Seventh International Conference on
Autonomous Agents and Multiagent Systems
AAMAS 2008
http://gaips.inesc-id.pt/aamas2008/

Sixth International Workshop

AT2AI-6:
From Agent Theory to Agent Implementation

http://www.ofai.at/research/agents/conf/at2ai6/

May 13, 2008, Estoril, Portugal (EU)
Chairs: Bernhard Jung, Fabien Michel, Alessandro Ricci and Paolo Petta

Working Notes
OFAI Technical Report 2008-01

This volume is available as OFAI Technical Report 2008-01 at:
http://www.ofai.at/cgi-bin/tr-online?number+2008-01

AT2AI-6: From Agent Theory to Agent Implementation

Preface

Bernhard Jung
Austrian Research Institute for Artificial Intelligence (OFAI) *
Freyung 6/6
1010 Vienna, Austria, EU
bernhard.jung@ofai.at

Fabien Michel
CReSTIC / LERI
IUT de Reims-Châlons-Charleville
Rue des crayères BP 1035
51 687 Reims CEDEX 2, France, EU
fabien.michel@univ-reims.fr

Alessandro Ricci
DEIS, ALMA MATER STUDIORUM
Università di Bologna
Via Venezia, 52
47023 Cesena, Italy, EU
a.ricci@unibo.it

Paolo Petta
Austrian Research Institute for Artificial Intelligence (OFAI)
Freyung 6/6
1010 Vienna, Austria, EU
paolo.petta@ofai.at

Since its first edition in 1998, the workshop series "From Agent Theory to Agent Implementation" has been not only documenting the progress in development and practical deployment of agent-related technologies, but also managed to contribute itself to the rapid development of this area. AT2AI promotes actively the exchange of ideas and experiences between researchers and practitioners working on the whole range of theoretical and application-oriented issues of agent technology. It encompasses both the micro and macro aspects of agent-oriented design, and discusses the relations of drawing boards and partly idealised models to modelling tools and frameworks to deployment, management and maintenance of implementations. The focus of AT2AI lies in the discussion of direct experience reports from all stakeholders, so as to remain well aware of the actual target domains while using the language of current agent terminology.

Of particular relevance to the workshop are reflections that share insights about experiences and lessons learnt when applying specific agent theories or architectures to application problems, or, from the recipients' end, when contracting agent technologies to provide a service envisioned. Such discussions and critiques may be aimed at conceptual vocabulary, methods, methodologies, good and bad management practices, and other tools and activities: anything that may be of value for system designers to improve the mapping of their agent-oriented toolbox to application needs, and for other stakeholders to better understand the available potential and current challenges of agent-oriented systems.

Previous editions of AT2AI produced a first blueprint of a layered ecology of technologies for the development of agent based applications[1] This perspective considers middleware, tools, off-the shelf platforms, integrated development environments (IDEs), and the like, with respect to their practical value to improve the application performance delivered. The qualities of these support technologies can in turn be improved and better exploited with the design of architectural frameworks and the deployment of standards. The evolution of these in turn can be assisted by the development of sound theoretical foundations and related formal methods. Methodologies are considered as working know-that and know-how, capturing and maintaining the best practises how to identify, align, and process application- and environment-derived (bottom-up) and support technology related (top-down) requirements and options.

Subsequently, AT2AI compiled an updated inventory against the maturing agent field: the status of logic-based approaches was addressed in particular; but evidence was also provided for how routine consideration of a multitude of perspectives is finally starting to meet the requirements posed by serious application needs, including e.g. issues of privacy and flexible access right management. The delineation of the scope of our domain of analysis is in need of constant further revising and updating, with recent developments being exemplified by the integration of agent-based contributions into the overall technology toolbox for distributed systems and increasing autonomy of technical MAS at the structural macro level.

This is the first edition of AT2AI being co-located with AAMAS and thus held outside the nurturing environment of the European Meeting on Cybernetics and Systems research that has played a decisive contribution for the establishing of this workshop series. We were all the more happy to see how well this call for papers was received. With the help of the serious and timely effort of the programme committee, out of 35 submissions the twenty papers collected in these working notes were selected. As explained in the call, the evaluation criteria emphasised a contribution's potential of being of value for the broad community of agent-based technology researchers, developers, and users: be it in terms of innovative perspectives, be it in terms of insights gained with real-world deployments. As much of the precious limited time at the workshop as possible will be used for in-

*The Austrian Research Institute for Artificial Intelligence is supported by the Austrian Federal Ministry for Science and Research and the Austrian Federal Ministry for Transport, Innovation and Technology.

[1] Petta P., Müller J.P.: Editorial: Engineering Agent Systems, Applied Artificial Intelligence, Best of "From Agent Theory to Agent Implementation 3", **16**(9-10):671-676, 2002.

teraction among the participants, setting out from topics addressed in the accepted papers. The discussion slots will involve authors of multiple papers, with these working notes providing supporting background material. Due to this approach taken, it may well be the case (and we actually do hope) that single papers are referred to at multiple times during the working day.

AT2AI-6 entertains strong links to two other AAMAS workshops: Agent Oriented Software Engineering (AOSE) and Programming Multi-Agent Systems (ProMAS): Immediately following AT2AI-6, there will be a joint evening session of these three workshops.

Acknowledgements

We have to thank Jörg P. Müller in at least twofold ways: for co-initiating the AT2AI series, and for suggesting to us to place a bid to organise AT2AI-6 with AAMAS 2008. Our thanks go also to Robert Trappl, for supporting the birth and development of the AT2AI workshop within EMCSR. As Workshop Chair, Juan Antonio Rodríguez-Aguilar has provided reliable support throughout the preparation process, enduring all our special quirks and related enquiries. And of course we also wish to thank the local organisers, headed by Ana Paiva and Luis Antunes, for their substantial efforts.

Programme Committee

Ardissono, Liliana
Bauer, Bernhard
Bergenti, Federico
Boissier, Olivier
Bordini, Rafael
Coelho, Helder
Demazeau, Yves
Dastani, Mehdi
Dickinson, Ian
El Fallah Seghrouchni, Amal
Ferber, Jacques
Giorgini, Paolo
Gomez Sanz, Jorge J.
Gouaïch, Abdelkader
Gustavsson, Rune
Hanachi, Chihab
Holvoet, Tom
Hubner, Jomi Fred
Leite, João
Marinier III, Robert P.
Omicini, Andrea
Platon, Eric
van Riemsdijk, Birna
Schumacher, Michael Ignaz
Simonin, Olivier
Vizzari, Giuseppe
Weiss, Gerhard
Weyns, Danny

We are also grateful to the following additional reviewers:

Amara, Nejla
Asnar, Yudistira
Bryl, Volha
Crepin, Ludivine
Dalpiaz, Fabiano
Huget, Marc-Philippe

Table of Contents

Keeping Balance up Sisyphus Path: Thoughts on Bringing Innovation in BPM through Agent Technology

Giovanni Rimassa
Whitestein Technologies AG
Pestalozzistrasse, 24
CH-8032 Zurich
+41 44 256 5000
gri@whitestein.com

ABSTRACT

This talk presents and discusses an overall perspective on the challenges, opportunities, and constraints faced by Whitestein while trying to leverage Agent Technologies ideas and techniques to bring substantial innovation to the Business Process Management market through the novel approach of *goal-oriented, autonomic BPM*.

Faithfully to the original spirit of the AT2AI workshop series, special care is devoted to highlight the interplay of matters theoretical vs. practical, as well as business vs. technical, novel vs. established, and other relevant dialectic pairs. Moreover, the overall Whitestein activity in the area is placed in the context of the conceptual framework proposed by the current edition of the workshop.

As a concrete token that seeded most of the considerations, the *Living Systems® Autonomic Business Process Management* (*LS/ABPM*) product is used. Customer communication, business partnerships, product conception, product development all presented valuable lessons to be learned, some of which are reported here as a hopefully stimulating contribution to the discussion.

Categories and Subject Descriptors

I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Intelligent agents Multiagent system.* H.4.1 [**Information Systems Application**]: Office Automation – *Workflow management.*

General Terms

Management, Design, Economics, Standardization, Languages.

Keywords

Business Process Management, Technology Transfer, Business Innovation, Theory Application.

TALK OUTLINE

A company that focuses on being an innovative technology provider while keeping a sustainable business model faces many challenges that can be expressed as keeping proper equilibrium between opposite excesses while continuously struggling to push new approaches through their technology life cycle curve (and being ultimately prepared to render its own technology obsolete in the long term through the introduction of even further advancements).

The first part of the talk considers the concrete case of Whitestein, whose long lasting connection to Agent Technology need be considered in the ever-changing business landscape. From the company mission to the target markets of choice, to the product portfolio and business partnerships, all these factors contribute to depict a strategy seeking a delicate and multi-faceted balance to deploy successful business through technological innovation.

The second part of the talk zooms in on the latest addition to Whitestein's product offering. LS/ABPM is an integrated environment to support goal-oriented autonomic BPM, based on Agent Technology. The characteristics of the Business Process Management technical challenges, the market structure as well as the strengths and weaknesses of the currently leading commercial offers, all suggest ideas on the product scope, features, and most importantly the differentiating and most innovative product traits, and what is the role of Agent Technology in enabling and supporting all the above.

Finally, a series of thoughts is presented, which were inspired by the activity around LS/ABPM and the Whitestein Autonomic Technology Platform (the overall software infrastructure that also includes the LS/TS agent middleware) at large. These thoughts, partly lessons learned, partly empirically motivated conjectures, and partly wishes for the future, will be submitted to the audience as input for reflection and discussion.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

From a Generic MultiAgent Architecture to MultiAgent Information Retrieval Systems

Andrea Addis
University of Cagliari
Piazza d'Armi,
I-09123, Cagliari, Italy
addis@diee.unica.it

Giuliano Armano
University of Cagliari
Piazza d'Armi,
I-09123, Cagliari, Italy
armano@diee.unica.it

Eloisa Vargiu
University of Cagliari
Piazza d'Armi,
I-09123, Cagliari, Italy
vargiu@diee.unica.it

ABSTRACT

The continuous growth of documents in digital form, together with the corresponding volume of daily updated contents, makes the problem of retrieving information a challenging task. In this paper we present X.MAS, a generic multiagent architecture explicitly devoted to implement information retrieval tasks. The proposed architecture has been adopted in several applications. To put into evidence how to bridge the gap from theory to practice, we illustrate and discuss three relevant applications of X.MAS.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval, Systems and Software

Keywords

Information Retrieval, MultiAgent Systems, Applications

1. INTRODUCTION

In the last fifteen years, AI researchers have concentrated their efforts in the field of intelligent autonomous agents, i.e., systems capable of autonomous sensing, reasoning and acting in complex environments. Suitable single-agent architectures have been devised to overcome the complexity problems that arise while trying to give agents a flexible behavior [14], [13], [27], [1]. Let us briefly recall that an agent architecture is essentially a map of the internals of an agent, i.e., its data structures, the operations that may be performed on them, and the corresponding control flows [36]. Furthermore, to allow cooperation and to implement suitable multiagent systems, several multiagent architectures have been devised. From a general perspective, we identify two kinds of these architectures: general-purpose [18], [26], [8] and application-oriented [11], [34], [35].

Although generic guidelines to build general-purpose architectures are required, in our view it is important to concentrate the efforts in a specific application field in order to bridge the gap between theoretical and pragmatical issues. To this end, in this paper we present a generic multiagent architecture explicitly devised to implement information retrieval tasks.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

The remainder of the paper is organized as follows: Section 2 gives a brief introduction about agents in information retrieval (*theoretical foundations*). Section 3 presents the architecture from a conceptual (*standard architectures*) and a technological point of view (*off-the-shelf platforms*). Then, Section 4 illustrates three actual systems built upon the proposed architecture (*applications*). Finally, Section 5 draws conclusions and points to future work.

2. FUNDAMENTALS ON AGENTS AND INFORMATION RETRIEVAL

Due to the increased availability of documents in digital form and the consequential need to access them in a flexible way, automated content-based document management tasks have gained a main task in the information systems field [30]. In particular, web information retrieval is highly popular and presents a technical challenge due to the heterogeneity and size of the web, which is continuously growing (see [17], for a survey).

In our opinion, it is very difficult for users to select contents according to their personal interests, especially when contents are frequently updated (e.g., news, newspaper articles, reuters, rss feeds, wikis, and blogs). Supporting users in handling the enormous and widespread amount of information (especially the one provided by the web) is becoming a primary issue. To this aim, several online services have been proposed that provide a personalization mechanism based on keywords, which is often inadequate to express what the user is really searching for. Moreover, users must often refine by hand the results provided by the service.

Agents have been widely proposed as a solution to these problems. An information agent is an agent that has access to one or more information sources, and is able to store and process information obtained from these sources in order to answer queries posed by users and other information agents [37]. The information sources may be of many types, including web services, web sites, RSS-feeds, and traditional databases.

In the literature, several centralized agent-based architectures aimed to perform information retrieval tasks have been proposed. Among others, let us recall NewT [32], Letizia [21], WebWatcher [3], and SoftBots [12].

NewT [32] has been designed as a society of information-filtering interface agents, which learn user preferences and act on her/his behalf. These information agents use a keyword-based filtering algorithm, whereas adaptive techniques are relevance feedback and genetic algorithms. Letizia [21] is an intelligent user interface agent able to assist a user while

browsing the Web. The search for information results as a cooperative venture between the user and the software agent: both browse the same search space of linked web documents, looking for interesting ones. WebWatcher [3] is an information search agent that follows web hyperlinks according to user interests, returning a list of links deemed interesting. In contrast to systems for assisted browsing or information retrieval, SoftBots [12] accept high-level user goals and dynamically synthesize the appropriate sequence of Internet commands according to a suitable ad-hoc language.

Despite the fact that a centralized approach could have some advantages, in information retrieval tasks it may encompass several problems, in particular how to scale up the architectures to large numbers of users, how to provide high availability in case of constant demand of the involved services, and how to provide high trustability in case of sensitive information, such as personal data. To this end suitable multiagent systems devoted to perform information retrieval tasks have been proposed. In particular, Sycara [33] proposed Retsina, a multiagent system infrastructure applied in many domains, has been presented. Retsina is an open MAS infrastructure that supports communities of heterogeneous agents. Three types of agents have been defined: interface agents, able to display the information to the users; task agents, able to assist the user in the management of her/his information; and information agents, able to gather relevant information from the selected sources.

Apart from Retsina, in the literature, several multiagent systems have been proposed and implemented. Among others, let us recall IR agents [19], CEMAS [6], and the cooperative multiagent system for web information retrieval proposed in [31].

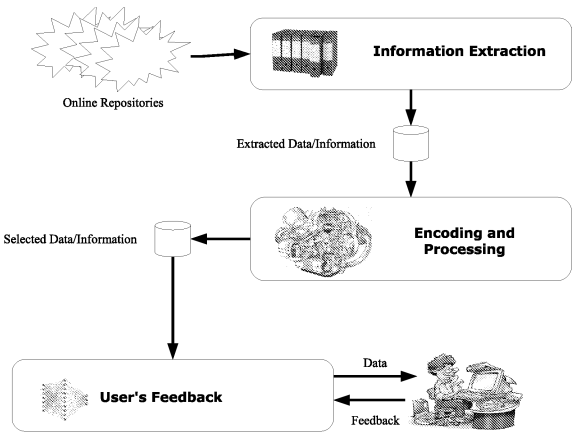


Figure 1: The abstract architecture.

IR agents [19] implement an XML-based multiagents model for information retrieval. The corresponding framework is composed of three kinds of agents: (i) managing agents, aimed to extract the semantics of information and to perform the actual tasks imposed by coordinator agents, (ii) interface agents, devised to interact with the users, and (iii) search agents, aimed to discover relevant information on the web. IR agents do not take into account personalization, while providing information in a structured form without the adoption of specific classification mechanisms. In CEMAS (Concept Exchanging Multi-Agent System) [6] the basic idea is to provide specialized agents for each main task, the main tasks being: (i) exchanging concepts and links, (ii) representing the user, (iii) searching for new relevant documents matching existing concepts, and (iv) supporting agent coordination. Although CEMAS provides personalization and classification mechanisms based on a semantic approach, the main drawback is that it is not generic, being mainly aimed to support scientists while looking for comprehensive information about their topic area. Finally, in [31] the underlying idea is to adopt intelligent agents that mimic everyday-life activities of information seekers. To this end, agents are also able to profile the user in order to anticipate and achieve her/his preferred goals. Although the approach is quite interesting, it is mainly focused on cooperation among agents rather then on information retrieval issues.

3. X.MAS: THE PROPOSED GENERIC ARCHITECTURE

Focusing on the role of software agents, the following categories can be identified in a context of information retrieval: (i) *information agents*, tailored to extract and handle information while accessing information sources[24], (ii) *filter agents*, able to transform information according to user preferences [23], (iii) *task agents*, able to help users to perform tasks typically in cooperation with other agents [15], (iv) *interface agents*, in charge of interacting with the user such that s/he interacts with other agents throughout them [22], and (v) *middle agents*, devised to establish communication among requesters and providers [10].

3.1 The Abstract Architecture

From a theoretical point of view, an information retrieval task involves three main activities: (i) extracting the required information, (ii) encoding and processing it according to the specific application, and (iii) providing suitable feedback mechanisms to improve the overall performances. Figure 1 shows a generic architecture able to perform these activities.

The information extraction module is aimed to extract data from information sources through specialized wrappers. In general, given an information source S, a specific wrapper W_S must be implemented, able to map each data D_S, designed according to the constraints imposed by S, to a suitable description O, which contains relevant information in a structured form –such as title, author(s), description, and images.

The encoding-and-processing module is aimed to encode information that flows from external sources (i.e., the selected information sources) and to progressively filter it to the end user by retaining only relevant data. The actual encoding strictly depends on the specific application (preprocessing activities, such as feature selection, could be performed to prepare the data to be processed). Data are processed according to high-level procedures, which are independent from the specific user. If needed, a personalized process can be performed according to user needs and preferences.

The user feedback module is devoted to deal with any feedback optionally provided by the end-user. In general, trivial –though effective– solutions can been implemented, e.g. based on artificial neural networks (ANNs) or a k-NN

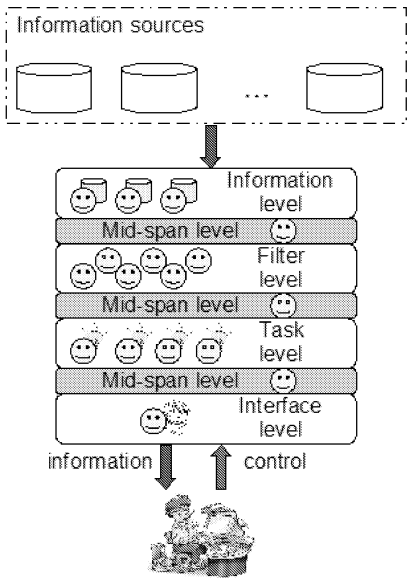


Figure 2: The concrete architecture.

classifiers.

3.2 The Concrete Architecture

An information retrieval system must take into account several issues, the most relevant being: (i) how to deal with different information sources and to integrate new information sources without re-writing significant parts of it, (ii) how to suitably encode data in order to put into evidence the informative content useful to discriminate among categories, (iii) how to control the unbalance between relevant and non relevant articles (the latter being usually much more numerous than the former), (iv) how to allow the user to specify her/his preferences, and (v) how to exploit the user feedback to improve the overall performance of the system.

The above problems are typically strongly interdependent in state-of-the-art systems. To better concentrate on these aspects separately, we adopted a layered multiagent architecture, able to promote the decoupling among all aspects deemed relevant. In particular, the functionalities of the abstract architecture, described in the previous section, have been implemented exploiting the X.MAS architecture. X.MAS is a generic multiagent architecture, aimed to retrieve, filter and reorganize information according to user interests. The *X* in X.MAS points out the generic nature of the architecture, playing the role of a wildcard to be substituted with an identifier specific of the corresponding application. The X.MAS generic architecture has been implemented on top of the well known JADE [5] agent-based infrastructure.

3.2.1 Macro-Architecture

The X.MAS architecture (depicted in Figure 2) encompasses four main levels: information, filter, task, and interface. The communication between adjacent levels is achieved through suitable middle agents, which form a corresponding mid-span level.

At the *information level*, agents are entrusted with extracting data from the information sources. Each information agent is associated to one information source, playing the role of wrapper.

At the *filter level*, agents are aimed to select information deemed relevant to the users, and to cooperate to prevent information from being overloaded and/or redundant. In general, two filtering strategies can be adopted: generic and personalized. The former applies the same rules to all users; whereas the latter is applied when a customized behavior is required for a specific user.

At the *task level*, agents arrange data according to users personal needs and preferences. In a sense, they can be considered as the core of the architecture. In fact, they are devoted to achieve user goals by cooperating together and adapting themselves to the changes of the underlying environment.

At the *interface level*, a suitable interface agent is associated with each different user interface. In fact, a user can generally interact with an application through several interfaces and devices (e.g., pc, pda, mobile phones, etc.).

At *mid-span levels*, agents are aimed to establish communication among requesters and providers. Agents at these architectural levels can be implemented as matchmakers or brokers, depending on the specific application [10].

3.2.2 Micro-Architecture

X.MAS agents can implement several capabilities, depending on the actual application and on their specific role. In particular, X.MAS agents, other than autonomy and flexibility, can provide any subset of the following capabilities: (i) *personalization*, to fulfill user interests and preferences, (ii) *adaptation*, to adapt to the underlying environment, (iii) *cooperation*, to interact with other agents in order to achieve a common goal, (iv) *deliberative capability*, to reason about the world model and to engage planning and negotiation, possibly in coordination with other agents; and (v) *mobility*, to migrate from node to node in a local- or wide-area network.

X.MAS agents are JADE agents capable of (i) interacting exchanging FIPA-ACL messages, (ii) sharing a common ontology in accordance with the actual application, and (iii) exhibiting a specific behavior according to their role. As for agent internals, Figure 3 shows the micro-architecture for agents belonging to each architectural level. Let us note that each agent encompasses a scheduler aimed to control the information flow between adjacent levels. Information and interface agents embody information sources and a specific devices, respectively. Filter and task agents encompass an actuator that depends on the actual application. Middle agents contain a dispatcher aimed to handle interactions among requesters and providers.

4. BUILDING INFORMATION RETRIEVAL SYSTEMS BY USING X.MAS

In order to highlight how to bridge the gap from theory to practice by adopting X.MAS, three relevant systems are presented. The first is concerned with the problem of classifying Wikipedia contents according to a predefined set of classes, the second is focused on giving a support to professors and students while interacting with a media asset management system, and the third is devoted to address the problem of monitoring boats in a marine reserve.

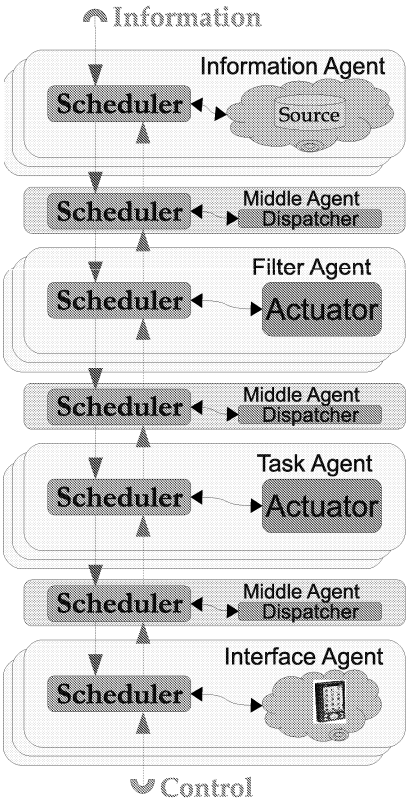


Figure 3: Agent internals.

4.1 WIKI.MAS: X.MAS for Classifying Wikipedia Contents

4.1.1 The Scenario.

As already pointed out in Section 2, supporting users in handling the enormous and widespread amount of web information is becoming a primary issue. Currently, the most overshadowing and noteworthy web information sources are developed according to the collaborative-web paradigm [7], also known as Web 2.0. It represents a paradigm shift in the way users approach the web. Users (also called prosumers) are no longer passive consumers of published content, but become involved, implicitly and explicitly, as they cooperate by providing their own content in an "architecture of participation" [28].

Among others, Wikipedia [1], an online encyclopedia based on the notion that an entry can be added/edited by any web user, has became an important benchmark for all Internet users interested in searching for definitions and/or references. Unfortunately, Wikipedia search engine allows users to choose their interests among macro-areas (e.g. *Arts*, *History*, and *Science*), which is often inadequate to express what the user is really interested in. Moreover, such search engine does not provide a feedback mechanism able to allow the user to specify non-relevant items –with the goal of progressively adapting the system to her/his actual interests.

Using X.MAS, we developed a system explicitly aimed to retrieve and classify Wikipedia contents according to a predefined set of classes, i.e., those belonging to WordNet Domains [25].

[1] http://www.wikipedia.org/

4.1.2 The Implementation.

To implement this specific application, X.MAS has been customized as follows:

- *Information level.* Agents at this level are aimed to deal with the huge information source provided by Wikipedia. To this end a suitable wrapper has been implemented, able to handle the structure of a document by saving the informations about the corresponding metadata (e.g. title, abstract, keywords, section headers) and by surfing across the whole reference links through the cooperation with other information agents.

- *Filter level.* Filter agents are aimed to select information deemed relevant to the users, and to cooperate to prevent information from being overloaded and redundant. A suitable encoding of the text content has been enforced at this level to facilitate the work of agents belonging to the task level. In particular, all non-informative words such as prepositions, conjunctions, pronouns and very common verbs are removed using a stop-word list. After that, a standard stemming algorithm [29] removes the most common morphological and inflexional suffixes. Then, for each category, feature selection, based on the information-gain heuristics [38], has been adopted to reduce the dimensionality of the feature space.

- *Task level.* Task agents are devoted to identify relevant Wikipedia documents, depending on user interests. Agents belonging to this architectural level are aimed to perform two kinds of actions: classify any given input in accordance with the selected set of classes, and decide whether it may be of interest to the user or not. Each task agent has been trained by resorting to state-of-the-art algorithms that implement the k-NN technique, in its *"weighted"* variant [9]. Furthermore, to express what the user is really interested in, we implemented suitable composition strategies by using extended boolean models [20]. In fact, typically, the user is not directly concerned with "generic" topics that coincide with the selected classes (such as *Arts*, *History*, or *Science*). Rather, a set of arguments of interest can be obtained by composing these generic topics with suitable logical operators (i.e., *and*, *or*, and *not*). In the proposed system, we adopted a quite general soft boolean perspective, in which the combination is evaluated using P-norms [16].

- *Interface level.* Information agents are aimed to perform the feedback from the user –which can be exploited to improve the overall ability of discriminating relevant from non relevant inputs. So far, a simple solution based on the k-NN technology has been implemented and experimented to deal with the problem of supporting the user feedback. When a non-relevant article is evidenced by the user, it is immediately embedded in the training set of the k-NN classifier that implements the feedback. A check performed on this training set after inserting the negative example allows to trigger a procedure entrusted with keeping the

number of negative and positive examples balanced. In particular, when the ratio between negative and positive examples exceeds a given threshold (by default set to 1.1), some examples are randomly extracted from the set of "true" positive examples and embedded in the above training set.

4.2 MAM.MAS: X.MAS for a Media Asset Management System

4.2.1 The Scenario.

E-learning differentiates from the traditional learning in its ability to train anyone, anytime, and anywhere, thanks to the openness of the Internet. Without the temporal and spatial limitation, one can have an independent and individual learning space. Currently, several Digital Asset Management (DAM) systems, also called Media Asset Management (MAM), have been proposed and devised [4]. As for e-learning, such systems are aimed to store, manage, and organize course materials, bibliography, and teacher notes.

Among other provided services, MAM systems must supply suitable support during the insertion phase. In particular, classification techniques might be devised to improve such systems to suitably organize contents and to help users in managing such data.

Using X.MAS, we developed a system aimed to support users in inserting multimedia contents in a MAM system. Being interested in handling university courses, typically organized in a hierarchy of classes, suitable hierarchical classification techniques has been studied. In particular, the current version of the system implements a hierarchical text categorization approach and is able to deal with text documents. Let us note that any multimedia document could be processed if a suitable textual description is given. In our experiments the system classifies data according to the taxonomy related to university courses, i.e., the one concerned with the bachelor's degree in electronic engineering. Figure 4 illustrates a portion of the adopted taxonomy.

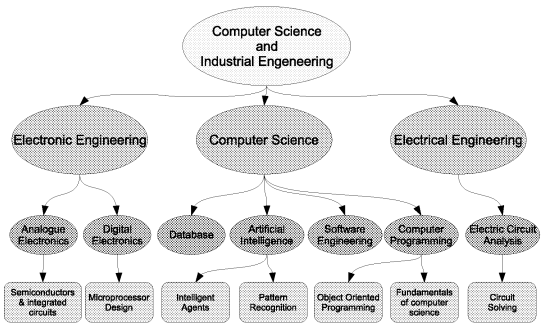


Figure 4: A portion of the adopted taxonomy.

4.2.2 The Implementation.

As for the implementation, X.MAS has been customized as follows:

- *Information level.* Input documents are the files provided by teachers during the insertion phase. Such documents can be simple-text-, formatted-, pdf-files, or multimedia contents together with their textual description. Information agents are able to handle all these kinds of documents by extracting the corresponding information.
- *Filter level.* As in WIKI.MAS, filter agents are aimed to select information deemed relevant to the users, and to cooperate to prevent information from being overloaded and redundant. Suitable encoding techniques have been enforced: all non-informative words are removed using a stop-word list; a standard stemming algorithm removes the most common morphological and inflexional suffixes; and, for each category, feature selection, based on the information-gain heuristics, has been adopted to reduce the dimensionality of the feature space.
- *Task level.* Task agents perform the hierarchical text categorization, resorting to a progressive filtering technique as described in [2]. In particular, each task agent has been trained by resorting to state-of-the-art algorithms that implement the *wk*-NN technique, whereas the progressive filtering is provided by the cooperation of the corresponding agents.
- *Interface level.* Interface agents are devoted to interact with the user within the MAM in order to support her/him while inserting documents. Further agents are aimed to perform user feedback. So far, a simple solution based on an ANN has been implemented. This solution consists of training an ANN with a set of examples classified as "of interest to the user". When the amount of feedback provided by the user has trespassed a given threshold, the ANN is trained again –after updating the previous training set with the information provided by the user.

4.3 SEA.MAS: X.MAS for Monitoring Boats in Marine Reserves

4.3.1 The Scenario.

Setting up a marine reserve involves issues concerning how to enforce access monitoring, with the goal of avoiding intrusions by not authorized boats, also considering that typically marine reserves are located in areas not easily accessible.

Currently, intrusion detection in marine reserves is carried out by adopting radar systems or by resorting to suitable cameras activated by movement sensors.

Ee are currently developing with X.MAS a system aimed to monitor boats in marine reserves. The corresponding scenario involves authorized boats, equipped with GPS+GSM devices, as well as not authorized boats. Boats are tracked by a digital radar that detects their positions. In this way, not authorized boats are easily identified comparing radar signals with those received from GPS+GSM devices.

4.3.2 The Implementation.

As for the implementation, X.MAS is being customized as follows:

- *Information level.* Each information agent encapsulates a specific information source. Two kinds of wrappers will be implemented to retrieve data from the devices and the radar, respectively.
- *Filter level.* Filter agents are devoted to collect the data retrieved by the information agents. In particular, they disregard redundant information, as signals

detected more than once by the same device (caching) or from different devices (information overloading).

- *Task level.* Each task agent corresponds to a boat. Information provided by filter agents allows to know exactly boat positions. In so doing, authorized and not authorized boats can be easily detected. The main tasks of the agents belonging to this architectural level are: (i) to follow a boat position during its navigation, also dealing with any temporary lack of signal; (ii) to promptly identify not authorized boats alerting the interface agents; and (iii) to handle messages coming from the interface level in order to notify the involved devices.

- *Interface level.* A suitable interface agent allows users to interact with the system. Final users are the system administrator and staff operators.

5. CONCLUSIONS AND FUTURE WORK

In this paper X.MAS, a generic architecture designed to support the implementation of applications g manage information among different and heterogeneous sources, has been presented. To put into evidence how to bridge the gap from theory to practice by adopting X.MAS, three relevant applications have been briefly described: the first concerned with the problem of classifying Wikipedia contents according to a predefined set of classes, the second focused on giving a support to professors and students while interacting with a media asset management system, and the third devoted to monitor boats in a marine reserve.

As for the future work, we are investigating how to improve the intelligent capabilities of agents with more complex forms of proactive and deliberative capabilities. Moreover, the possibility to implement further applications using X.MAS is currently under study.

Acknowledgments

The case study related to the implementation of a Wikipedia information retrieval system has been supported by the Italian Ministry of Education, under the project “DART - Distributed Architecture for Semantic Search and Personalized Content Retrieval”.

The case study related to the implementation of a support for a multimedia asset management system has been supported by Regione Autonoma della Sardegna, under the project “SOFIA - Online Services of University Courses and Education”.

The case study related to the implementation of a system for monitoring boats in marine reserves is supported by Regione Autonoma della Sardegna, under the project “A Multiagent System for Monitoring Intrusions in Marine Reserves”.

6. REFERENCES

[1] G. Armano, G. Cherchi, and E. Vargiu. An agent architecture for planning in a dynamic environment. *Lecture Notes in Computer Science*, 2175:388–394, 2001.

[2] G. Armano, F. Mascia, and E. Vargiu. Using taxonomic domain knowledge in text categorization tasks. *International Journal of Intelligent Control and Systems*, 12(2):150–157, 2007. Special Issue on Distributed Intelligent Systems.

[3] R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring Symposium on Information Gathering*, pages 6–12, 1995.

[4] D. Austerberry. *Digital Asset Management*. Focal Press, 2006.

[5] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology)*. John Wiley and Sons, 2007.

[6] M. Bleyer. *Multi-Agent Systems for Information Retrieval on the World Wide Web*. PhD thesis, University of Ulm, Germany, 1998.

[7] J. Burdman. *Collaborative Web Development: Strategies and Best Practices for Web Teams*. Addison-Wesley Longman Ltd., 1999.

[8] D. Camacho, R. Aler, D. Borrajo, and J. Molina. A multi-agent architecture for intelligent gathering systems. *AI Communications, The European Journal on Artificial Intelligence*, 18(1):1–Ű19, 2005.

[9] W. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78, 1993.

[10] K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 97)*, pages 578–583, 1997.

[11] F. Dorca, C. Lopes, and M. Fernandes. A multiagent architecture for distance education systems. In *Proceedings of the 3rd IEEE International Conference on Advanced Learning Technologies*, pages 368–369, 2003.

[12] O. Etzioni and D. Weld. Intelligent agents on the internet: fact, fiction and forecast. *IEEE Expert*, 10(4):44–49, 1995.

[13] I. Ferguson. *TouringMachines: An Architecture for Dynamic, Rational, Mobile Agents*. PhD thesis, lare Hall, University of Cambridge, UK, 1992.

[14] M. Georgeff and A. Lansky. Reactive reasoning and planning. In *Proceedings of the Sixth National on Artificial Intelligence (AAAI-87)*, pages 677–682, 1987.

[15] J. Giampapa, K. Sycara, A. Fath, A. Steinfeld, and D. Siewiorek. A multi-agent system for automatically resolving network interoperability problems. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1462–1463, 2004.

[16] G. Golub and C. V. Loan. *Matrix Computations*. Baltimore: The Johns Hopkins University Press, 1996.

[17] L. Huang. A survey on web information retrieval technologies. In *Technical report, ECSL*, 2000.

[18] C. A. Iglesias, J. C. Gonzalez, and J. R. Velasco. Mix: A general purpose multiagent architecture. In *In M. Wooldridge, J. P. Muller, and M. Tambe, editors,* Intelligent Agents II, *Springer-Verlag*, pages 251–266, 1995.

[19] W. Jirapanthong and T. Sunetnanta. An xml-based multi-agents model for information retrieval on www. In *Proceedings of the 4th National Computer Science*

and Engineering Conference (NCSEC2000), 2000.

[20] J. Lee. Properties of extended boolean models in information retrieval. In *Proceedings of the 17th Annual international ACM SIGIR Conference on Research and Development in information Retrieval*, pages 182–190, 1994.

[21] H. Lieberman. Letizia: An agent that assists web browsing. In C. S. Mellish, editor, *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 924–929, Montreal, Quebec, Canada, 1995. Morgan Kaufmann publishers Inc.: San Mateo, CA, USA.

[22] H. Lieberman. Autonomous interface agents. In *Proceedings of the ACM Conference on Computers and Human Interface (CHI-97)*, pages 67–74, 1997.

[23] E. Lutz, H. Kleist-Retzow, and K. Hoernig. Mafia an active mail-filter-agent for an intelligent document processing support. *ACM SIGOIS Bulletin*, 11(4):16–32, 1990.

[24] P. Maes. Agents that reduce work and information overload. *Communications of the ACM*, 37(7):31–40, 1994.

[25] B. Magnini and G. Cavaglià. Integrating subject field codes into wordnet. In *Gavrilidou M., Crayannis G., Markantonatu S., Piperidis S. and Stainhaouer G. (Eds.)* Proceedings of LREC-2000, Second International Conference on Language Resources and Evaluation, pages 1413–1418, 2000.

[26] P. Marcenac and S. Giroux. Geamas: A generic architecture for agent-oriented simulations of complex processes. *International Journal of Applied Intelligence*, 1998.

[27] J. Mueller. A cooperation model for autonomous agents. In *In J.P Mueller, M. Wooldridge and N.R. Jennings (eds)* Intelligent Agents III*, LNAI, Vol. 1193. Springer-Verlag, Berlin Heidelberg New York*, pages 245–26, 1997.

[28] T. O'Reilly. *What is Web 2.0, Design Patterns and Business Models for the Next Generation of Software*. O' Reilly, 2005.

[29] M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[30] F. Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys (CSUR)*, 34(1):1–55, 2002.

[31] K. Shaban, O. Basir, and M. Kamel. Team consensus in web multi-agents information retrieval system. In *Team Consensus in Web Multi-agents Information Retrieval System*, pages 68–73, 2004.

[32] B. Sheth and P. Maes. Evolving agents for personalized information filtering. In *Proceedings of the 9th Conference on Artificial Intelligence for Applications (CAIA-93)*, pages 345–352, 1993.

[33] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. Technical Report CMU-RI-TR-01-05, Robotics Institute Technical Report, Carnegie Mellon, 2001.

[34] S. Walczak. A multiagent architecture for developing medical information retrieval agents. *Journal of Medical Systems*, 27(5):479–498, 2003.

[35] D. Wei and A. B. Abrahams. A multiagent architecture for semantic web resources. In *Proceedings of IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 289–292, 2005.

[36] M. Wooldridge. Intelligent agents. In *G.ăWeiss (ed)* Multiagent Systems*, MIT Press, Berlin Heidelberg New York*, 1999.

[37] M. Woolridge. *Introduction to Multiagent Systems*. John Wiley and Sons, Inc., 2001.

[38] Y. Yang and J. O. Pedersen. Feature selection in statistical learning of text categorization. In *Proceedings of the 14th International Conference of Machine Learning ICML9*, pages 412–420, 1997.

Comparative Efficiency and Implementation Issues of Itinerant Agent Language on Different Agent Platforms

Abdullah Almuhaideb[1,3] Kutila Gunasekera[1] Arkady Zaslavsky[1] Seng Wai Loke[2]
[1]Faculty of Information Technology
Monash University
Melbourne, Australia
[2]Department of Computer Science and Computer Engineering
La Trobe University
Melbourne, Australia

aalmuhaidob@kfu.edu.sa, {kutila.gunasekera, arkady.zaslavsky}@infotech.monash.edu.au, S.Loke@latrobe.edu.au

ABSTRACT
An itinerary scripting language provides a tool to accelerate the development of mobile agent applications. The main purpose of this paper is to discuss research challenges of itinerant mobile agents (ITAG) and describe the migration of ITAG to JADE, the popular FIPA-compliant agent platform. The migration process was based on two major steps: the migration of agent communications and agent tasks (behaviors). The ITAG Engine has been extracted to provide an easy migration to any new agent platform. In our experiments the results show that the current ITAGIII (using JADE) has better performance and stability compared to ITAGII (using Grasshopper). The paper discusses the experiments and comparisons in detail.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems*; C.2.4 [**Computer-Communication Networks**]: Distributed Systems

General Terms
Algorithms, Measurement, Performance, Design, Languages, Experimentation, Theory

Keywords
Itinerary Language, Itinerary Agent (ITAG), JADE, Grasshopper, Agent Platforms, Scalability, Applications, Agent Communication, Agent, Messaging

1. INTRODUCTION
Mobile agents are defined as a future framework of distributed electronic services and are used in data mining, electronic commerce and network management [1]. All stages of a business transaction, such as negotiating and signing contracts can be carried out using mobile agents. A mobile agent can be described as a software entity which is capable of moving from one location to another and continuing its execution. ITAG (ITinerary AGents) is a scripting language and prototype system previously developed by us [2, 3, 4]. An itinerary scripting language (for mobile agents) aims to make developing agent applications easier. It allows the developer to script together mobile agents from existing components by specifying what the agent should do, at which location and when.

The purpose of this paper is to extend the Itinerary Language and port the current ITAG system which runs on Grasshopper agent platform to the popular FIPA (Foundation for Intelligent Physical Agents)-compliant agent platform JADE [5]. We also developed a visual tool which allows users to easily create and execute their own test cases and demonstration environments for ITAG.

There are a number of reasons behind choosing JADE. One of these reasons is that JADE is open-source, continuously maintained and well supported [6]. Since ITAG aims to support mobile devices, the existence of JADE-LEAP (Lightweight Extensible Agent Platform) [6], a lightweight release of JADE for mobile devices, was also a consideration.

The rest of this paper is organized as follows. In section 2, an overview of the ITAG system will be given. Section 3 will discuss the migration process from Grasshopper to JADE. In section 4, we will present new features of ITAGIII. In section 5, the experiments and result will be evaluated. Some related work will be discussed in section 6. Finally, conclusions and future work of this paper will be presented in section 7.

2. ITAG SYSTEM OVERVIEW

2.1 ITAG: The Itinerary Scripting Language
ITAG is an executable implementation of the itinerary algebra in the form of a scripting language described in [2, 4]. We first briefly outline the algebra below. We assume an object-oriented model of agents, where an agent is an instance of a class given roughly by:

mobile agent = state + action + mobility

We assume that agents have the capability of cloning, which is, creating copies of themselves with the same state and code. Also, agents can communicate to synchronize their movements, and the agent's code can execute at each location it visits.

[3]Abdullah Almuhaideb is a staff member at the Department of Network Computing, King Faisal University in AlAhsa, Saudi Arabia. This work was done while he was visiting Monash University.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,*May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

Let **A**, **O** and **P** be finite sets of agent, action and place symbols, respectively. Itineraries (denoted by *I*) are now formed as follows representing the null activity, atomic activity, parallel, sequential, nondeterministic, conditional nondeterministic behavior, and have the following syntax:

$$I :: 0 \mid A^a_p \mid (I \Vert_{\oplus} I) \mid (I \,.\, I) \mid (I \mid I) \mid (I :_{\Pi} I)$$

where $A \in A$, $a \in O$, $p \in P$, $\oplus$ is an operator which, after a parallel operation causing cloning, recombines an agent with its clone to form one agent, and Π is an operator which returns a boolean value to model conditional behavior. We specify how $\oplus$ and Π are used but we assume that their definitions are application-specific.

We assume that all agents in an itinerary have a starting place (which we call the agent's home) denoted by $h \in P$. Given an itinerary *I*, we shall use *agents*(*I*) to refer to the agents mentioned in *I*.

Agent Movement (A^a_p). A^a_p means "move agent *A* to place *p* and perform action *a*". This expression is the smallest granularity mobility abstraction. It involves one agent, one move and one action at the destination.

Parallel Composition ("‖"). Two expressions composed by "‖" are executed in parallel. For instance, ($A^a_p \Vert B^b_q$) means that agents A and B are executed concurrently. Parallelism may imply cloning of agents. For instance, to execute the expression ($A^a_p \Vert A^b_q$), where $p \neq q$, cloning is needed since agent *A* has to perform actions at both *p* and *q* in parallel. When cloning has occurred, decloning is needed, i.e. clones are combined using an associated application specific operator (denoted by $\oplus$ as mentioned earlier).

Sequential Composition (".."). Two expressions composed by the operator "." are executed sequentially. For example, (A^a_p . A^b_q) means move agent *A* to place *p* to perform action *a* and then to place *q* to perform action *b*.

Independent Nondeterminism ("|"). An itinerary of the form (*I* | *J*) is used to express nondeterministic choice: "I don't care which but perform one of *I* or *J*". If *agents(I)* ∩ *agents(J)* ≠ 0, no clones are assumed, i.e. *I* and *J* are treated independently. It is an implementation decision whether to perform both actions concurrently terminating when either one succeeds (which might involve cloning but clones are destroyed once a result is obtained), or trying one at a time (in which case order may matter).

Conditional Nondeterminism (":"). Independent nondeterminism does not specify any dependencies between its alternatives. We introduce conditional nondeterminism which is similar to short-circuit evaluation of boolean expressions in programming languages such as C. An itinerary of the form $I :_{\Pi} J$ means first perform I , and then evaluate Π on the state of the agents. If Π evaluates to true, then the itinerary is completed. If Π evaluates to false, the itinerary J is performed (i.e., in effect, we perform *I* . *J*). The semantics of conditional nondeterminism depends on some given Π.

2.2 Itinerary Language Examples and Implementation

We give an example using agents to vote. An agent V, starting from home, carries a list of candidates from host to host visiting each voting party. Once each party has voted, the agent goes home to tabulate results (assuming that home provides the resources and details about how to tabulate), and then announces the results to all voters in parallel (and cloning itself as it does so). Assuming four voters (at places *p, q, r,* and *s*), vote is an action accepting a vote (e.g., by displaying a graphical user interface), tabulate is the action of tabulating results, and announce is the action of displaying results; the mobility behavior is as follows:

$$V^{vote}_{p} \cdot V^{vote}_{q} \cdot V^{vote}_{r} \cdot V^{vote}_{s} \cdot V^{tabulate}_{h} .(V^{announce}_{p} \Vert V^{announce}_{q} \Vert V^{announce}_{r} \Vert V^{announce}_{s})$$

Implementation: To allow the programmer to type the itinerary expressions into the computer, we provide an ASCII syntax and a Controlled English (limited natural language) version. The translations are given in Table 1. When the operators are used without op, we assume a pre-specified system default one, i.e. using op is an optional clause. $A^a_p \,.\, A^b_q \,.\, A^c_r$ can be described as follows: "(move A to a do p) then (move A to b do q) then (move A to c do r)." Apart from the above basic elements of the language, we define the following five phrases that map down to more complex expressions:

1. A^a_h is translated as `return A do a.`

2. $A^a_p \,.\, A^a_q \,.\, A^a_r \,.\, A^a_s$ is translated as

`tour A to p, q, r, s in series do a.`

3. $A^a_p \Vert A^a_q \Vert A^a_r \Vert A^a_s$ is translated as

`tour A to p, q, r, s in parallel do a.`

4. $A^a_p \mid A^a_q \mid A^a_r \mid A^a_s$ is translated as

`tour A to one of p, q, r, s do a.`

5. $A^a_p : A^a_q : A^a_r : A^a_s$ is translated as

`tour A if needed to p, q, r, s do a.`

Table 1. Translations

Symbol	ASCII	Controlled English
A^a_p	[A,p,a]	Move A to P do a
.	.	Then
$:_{\Pi}$	:{op}	Otherwise using op
|	|	Or
$\Vert_{\oplus}$	#{op}	In parallel with using op

Similarly, we also have $A^a_p :_{\Pi} A^a_q :_{\Pi} A^a_r :_{\Pi} A^a_s$ translated as `tour` *A* `if needed to` *p, q, r, s* `do` *a* `using` Π.

Using the phrases, the voting itinerary can be concisely described as follows:

```
(tour V to p,q,r,s in series do vote)
 then (return V do tabulate)
```

```
then (tour V to p,q,r,s in parallel do announce)
```

ITAG implementation is in the Java programming language, previously built on top of Grasshopper (ITAG and ITAGII) and now JADE (ITAGIII) agent toolkits. In all implementations, the user first types in itinerary scripts into an applet (running in a Web browser). Then, the itinerary script is parsed into a binary tree representation and executed by an interpreter. Execution is as follows: the interpreter translates the actions specified in the script into commands which are then forwarded to agents which are initially at a place (the home). These agents on receiving the commands are then launched into the agent network to do their work. Figure 1 shows an example of the ITAG demo user interface. We explain the ITAG system architecture in the next section.

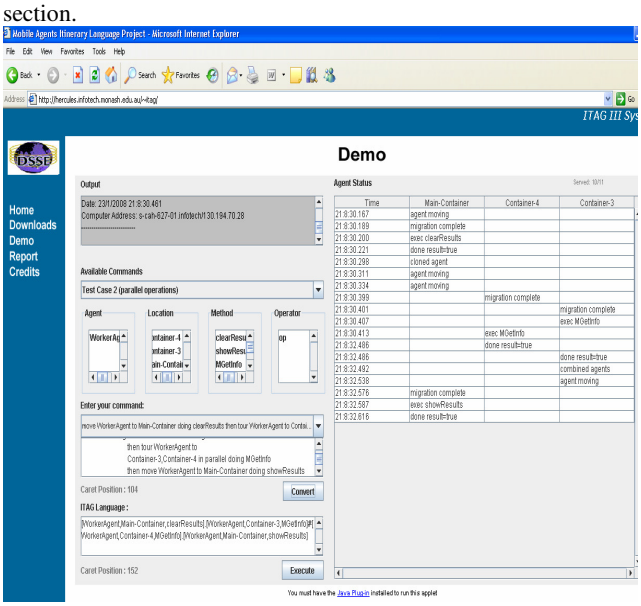


Figure 1. An example of ITAG demo user interface in action

2.3 ITAG System Architecture

The ITAG system is broken down into a number of distinguishable modules that handle different functionalities of the system. Figure 2 below illustrates these different modules and their interactions with each other.

Users of the ITAG system interact with it through the User Interface. The UI allows users to configure an itinerary using a limited natural language format. The UI also contains a separate area which shows the results of itinerary execution (Agent status and Output panes in Figure1). The UI itself does not understand the ITAG language and cannot execute an itinerary. A user can configure an itinerary in the limited natural language format in the User Interface and have it converted to an itinerary language statement. This conversion is done by the *ITAG Parser* module.

When the user requests the itinerary to be executed, the itinerary is passed on to the Controller Agent for execution.

ITAG Parser is part of the ITAG API and is independent of the underlying agent platform. It parses a user configured itinerary from its user-friendly limited natural language format to the itinerary language format (ASCII format).

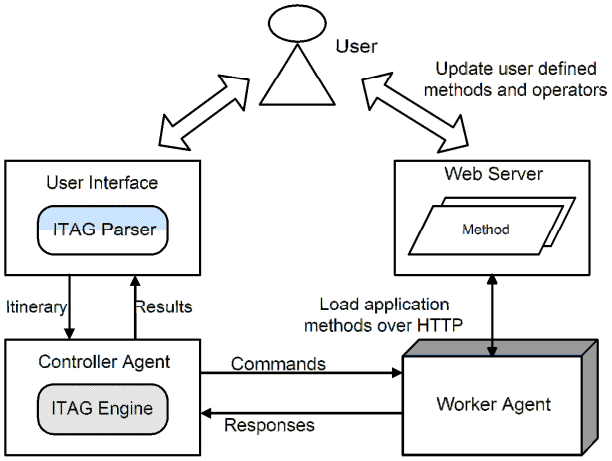


Figure 2. System Architecture

The *ITAG Engine* is the core of ITAG and contains the logic of the Itinerary Language. It also forms part of the ITAG API and is independent of the underlying agent platform. The engine provides the necessary components to execute an itinerary and the Controller Agent uses the *ITAG Engine* to understand and execute an itinerary. The Engine takes as input an itinerary. As output the engine makes method calls to carry out the tasks specified in the itinerary.

The ITAG system consists of two types of agents: *Controller Agents* and *Worker Agents*. Worker Agents are created and controlled by Controller Agents. During the execution of an itinerary Worker Agents maybe created and destroyed afterwards. A system may consist of multiple Controller Agents. These two types of agents are described below.

The *Controller Agent* gets as input an itinerary from the User Interface. Its main functionality is executing this itinerary by driving worker agents accordingly with the help of the ITAG Engine. The Controller Agent is the controller and executor of an itinerary.

The *Worker Agents* carry out application-specific useful functionality requested by users from the ITAG system. A worker is a simple agent controlled by another (controller) agent. It only responds to commands from the controller agent and is not aware of itineraries or the ITAG language. The worker agent is able to do the following tasks when requested by its controller:

- move or copy itself to different locations
- execute a method (by downloading its class files from a pre-defined location)
- store any results it gathers by executing various methods in its "pocket"
- combine "pockets" with other worker agents
- destroy itself

The web server provides a place to host the User Interface (e.g. applet) and the user defined method and operator classes. Methods are application specific code to be executed by ITAG agents (i.e. worker agents) which are represented as *actions* in the itinerary language. They contain useful functionality written by application developers and added to the system dynamically.

Operators (represented by ⊕ and Π in the itinerary language) can also be dynamically added to the web server for use by the ITAG system.

3. MIGRATION PROCESS FROM GRASSHOPPER TO JADE PLATFORM

The migration of ITAG from Grasshopper to JADE consists of two main stages. The first is porting the agent communication and the second, agent tasks (behaviors). The communication and behavior mechanisms of JADE lead to ease of development and better performance in comparison to Grasshopper.

3.1 Agent Communication

Agent communication which is a fundamental feature of an agent platform describes how two agents converse. In Grasshopper, the communication between agents is through their proxies. But in JADE, agents communicate through message passing as asynchronous agent messages [5]. Proxies do not exist in JADE; instead, an agent searches the current location of its target by querying the AMS (Agent Management Service) according to the FIPA specifications. The Agent Management Service gives JADE a better communication hub compared to Grasshopper. The region server in Grasshopper could become a bottleneck, as it must update every proxy just before being used [7]. Based on our implementation we found that JADE has an excellent control over cloned agents in terms of keeping references of these agents and destroying them at the end of their tasks whereas in Grasshopper some cloned agents could be left without being killed after task completion.

3.2 Agent Tasks (Behaviors)

In JADE, the agent is allowed to have just a single Java thread per-agent [5]. Inside this thread multiple behaviors can be added using a round-robin non-preemptive scheduling policy [8]. Grasshopper makes use of the more complicated alternative of implementing a multi-threaded system to handle, for example, socket connections and communication processes. The JADE behaviors improve agent performance as the switching between behaviors is far faster than switching between Java threads. Another advantage of the JADE behaviors against multi-threaded systems is its elimination of all synchronization issues between parallel behaviors accessing the same resources since all behaviors are executed by the same Java thread which result in a performance enhancement as well. [5, 8, 9]

4. ITAGIII NEW FEATURES

With ITAGIII, an improved GUI has been introduced to provide simplicity to the system. The first screen that a user encounters in the demo system lists the existing applications in the server that an ITAG agent is capable of executing. An application, in ITAG, is a collection of methods that collaborate together to provide a complete service to the user. Also an intermediate page has been introduced as an option in case an application may require further user input to run this application. For example, an intermediate page has been introduced in "make an appointment" application in order to collect an appointment preferred times of the user.

Also in ITAGIII, the problem of nondeterministic activity ("|") implementation has been fully solved and full control of the clone agents has been achieved. The implemented semantics carry out all activities in parallel and select the first one to finish (others are discarded). The complexity of killing other agents and threads was the reason for non-implementation in the previous version of ITAG under Grasshopper. (i.e. The original agent could be killed under nondeterministic activity but with the clone agents left alive).

Another feature of ITAGIII is the ITAG Engine which has been extracted from ITAGII version. The ITAG Engine provides for easy development of an ITAG system on any new mobile agent platform. The Controller Agent for the new agent platform has to implement an interface `itagIII.engine.AgentDealings` which contains methods representing the output commands of the ITAG Engine. We give the AgentDealings interface below followed by figure 3 which shows the ITAG Engine and its relation to other components.

```
public interface AgentDealings {
   public boolean go(String agentId,
      String tempLocation) throws Throwable;
   public boolean exec(String agentId,
      String method, String
      dynamicpath) throws Throwable;
   public String cloning(String agentId)
      throws Exception;
   public void combine(String source,
      String destination, int ativityNum)
      throws Exception;
   public Vector getAgentPocket(
      String source)throws Exception;
   public void showAgentPocket(
      String agentId)throws Exception;
   public void removeAgent(String agentName)
      throws Exception;
   public String yourLocation(
      String agentName) throws Exception;
   public boolean existAgent(
     String agentId);
   public boolean existLocation(
     String agentId);
}
```

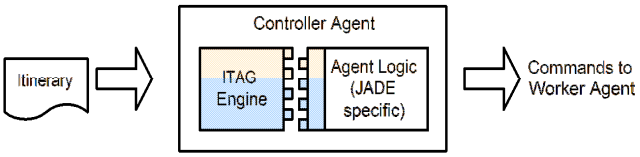


Figure 3. ITAG Engine and its relation to other ITAG system components

5. COMPARISON OF ITAGII AND ITAGIII PERFORMANCES

The aim of this experiment is to compare the performance of the two versions of ITAG system on JADE (ITAGIII) and Grasshopper (ITAGII) agent platforms.

5.1 Experiment Set-up

5.1.1 Test Scenarios

The basic test scenario is explained in this section. First, we give below an itinerary encapsulating all the test cases.

```
move WorkerAgent to Home doing clearResults
  then tour WorkerAgent to [one of]
  Location-1, Location-2, Location-3 ...
```

```
[in series|in parallel|if needed]
    doing getInfo
then move WorkerAgent to Home doing
    showResults
```

The controller agent communicates first with the worker agent to return home and run the clearResults method, which will clear the worker agent's pocket. Then the controller agent communicates with the worker agent to move to Location-1, Location-2 and Location-3 running the getInfo method. getInfo method will retrieve the date and time of the location of the agent as well as the machine name and IP address. Also getInfo method returns true or false randomly in order to test nondeterministic and conditional nondeterministic activities. Finally the controller agent communicates with the worker agent to return home running the showResults method. This method will display the agent's pocket in the user interface as well as writing the result to a log file.

Different test scenarios are realized through change of parameters, which are explained below.

- *Number of destinations* - The most likely scenario for an agent is to travel to a number of locations. Therefore, in our test we examine the worker agent in a different number of locations starting with two and increasing up to six locations.
- *Type of activities* – The four types of ITAG activities to be tested are: sequential, parallel, nondeterministic and conditional nondeterministic.

5.1.2 Test Environment and Measurement

The experiment is repeated for every activity, with the locations on the same host and on different hosts. Due to limitations with ITAGII (under Grasshopper), the system was not tested for nondeterministic activity, and also the system could not be tested with the different locations on different hosts. The testing environments for a number of locations located on different hosts were four PC's: one hosted the Controller Agent and the JADE main container and the other three hosted the six locations with two containers for every host. Table 2 shows hardware and software configuration of the test PC's. All PC's were connected to a 100Mbps network. To gain accurate results, each experiment was repeated ten times and the average taken. The performance was measured for every activity based on the number of locations and the average time taken to complete the test.

Table 2. Hardware and Software of the testing environment

Processor	Intel Pentium 4 CPU 3.00 GHz
Memory	1GB (SDRAM)
Operating System	Windows XP Professional Version 2002 Service Pack 2
Java version	Sun JDK 1.5.0_12
JADE version	3.5
Grasshopper version	2.2.4

5.2 Results

This section presents the results based on ITAG system activities. Each activity section will contain two figures. The first one will compare ITAGII and ITAGIII by plotting the average time versus number of locations in the itinerary. All locations for this test are on the same host. The second figure shows a similar graph but compares ITAGIII (JADE based agents) with the locations physically distributed on four different hosts and on the same host.

5.2.1 Sequential Activity

The result of this experiment shows that ITAGIII has a better performance than ITAGII. As figure 4 illustrates, an ITAGIII agent was consistently able to finish its itinerary task faster than ITAGII. The performance of ITAGII degrades rapidly with increasing number of locations compared to ITAGIII. For example, with four locations we see ITAGIII is seven times faster and with six locations it is six times faster.

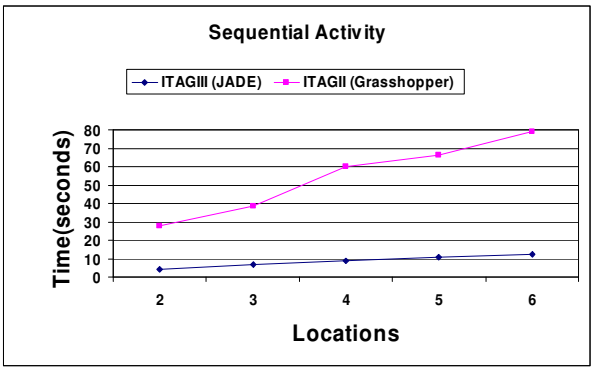


Figure 4. Sequential activity, comparing ITAG system with JADE and Grasshopper based agents on the same host

When comparing ITAGIII under JADE based agents with different number of locations on the same host and on different hosts (Figure 5) we see no significant variation in the time taken. Since the experiments were conducted on a high-speed LAN with low traffic, communication overheads were negligible. This indicates that ITAGIII does not incur an extra overhead when the agents communicate and move between multiple machines. As future work we intend to conduct experiments in heterogeneous wide-area network environments using agents with larger workloads.

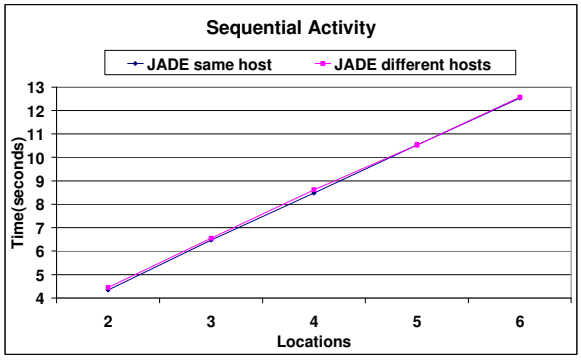


Figure 5. Sequential activity, comparing ITAG system with JADE based agents on the same host and on different hosts

5.2.2 *Parallel Activity*

The parallel activity scenario test in figure 6 shows that, as in the sequential activity, that ITAGIII has a better performance than ITAGII (six times faster on 3 locations). However, on ITAGII the agent was unable to continue its tasks with more than 3 locations to be visited in parallel. This is due to ITAGII's issues with managing multiple clones which was previously explained. Also this indicates that ITAGIII is more stable, scalable and efficient than ITAGII.

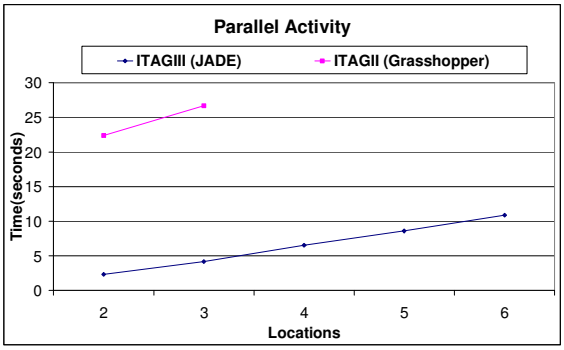


Figure 6. Parallel activity, comparing ITAG system with JADE and Grasshopper based agents on the same host

Figure 7 shows that ITAGIII has the same performance with the locations on the same host and on different hosts, which support the same conclusion from the sequential activity.

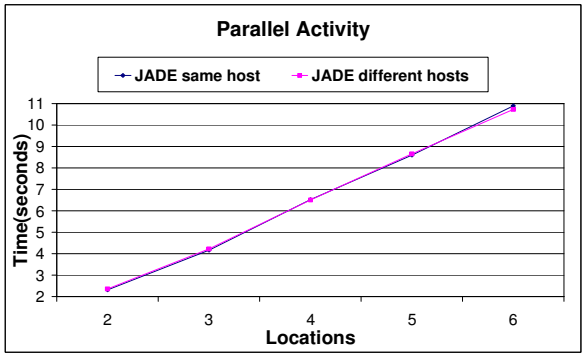


Figure 7. Parallel activity, comparing ITAG system with JADE based agents on the same host and on different hosts

5.2.3 *Nondeterministic Activity*

In this test, the worker agent clones itself based on the number of locations and sends them to the different locations to execute the tasks. When one of the agents finishes its task that agent's result is taken and all the clones destroyed. We can see from figure 8 that ITAGIII on different machines has a slightly better and stable performance than on the same machine. We believe this behaviour could be due to the distribution of the computation load between multiple CPUs with a fast network connecting them. A similar argument is put forward in [8].

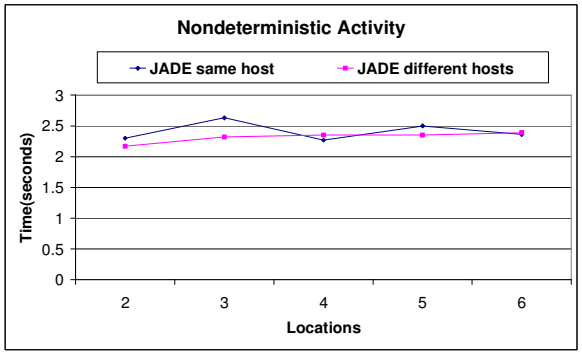


Figure 8. Nondeterministic activity, comparing ITAG system with JADE based agents on the same host and on different hosts

5.2.4 *Conditional Nondeterministic Activity*

When running this test multiple times the number of locations the worker agent travels to (in sequence) fluctuates based on the result returned by the getInfo method. For example, the agent travels from one location to another as long as the getInfo method returns false, otherwise the agent will skip the rest of locations and return home to show the result. In the graphs below we show the average times taken to complete the test. Figure 9 supports the previous test results and shows that ITAGIII has better performance than ITAGII.

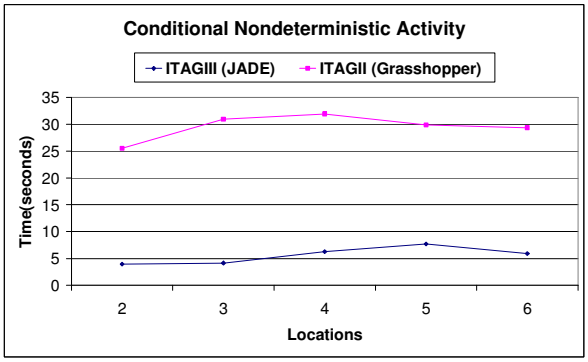


Figure 9. Conditional Nondeterministic activity, comparing ITAG system with JADE and Grasshopper based agents on the same host

In previous tests we saw that the performance of JADE on same host versus multiple hosts is similar, but in figure 10 we do not see this relationship because of the random behavior explained above.

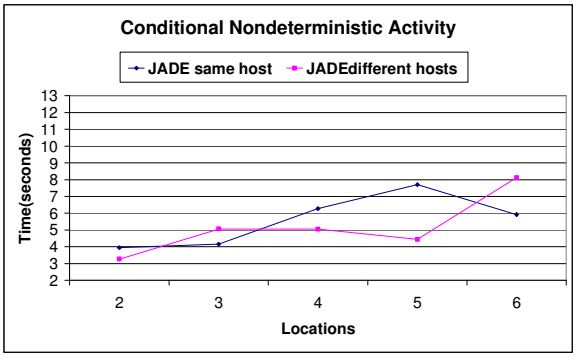


Figure 10. Conditional Nondeterministic activity, comparing ITAG system with JADE based agents on the same host and on different hosts

6. RELATED WORK

In this section we first briefly look at several itinerary languages found in literature. Lu and Xu defined a mobile agent itinerary language (MAIL) [10] which has been implemented as a feature in the Naplet [11] mobile agent system. Their work however is limited to a single mobile agent platform (Naplet) while ITAG can be easily implemented on any mobile agent platform by using the ITAG Engine. Performance figures for MAIL based agents were also not available for comparison with ITAG.

Rech et al. in [12, 13, 14] describe a flexible itinerary that can be adjusted at run-time to ensure greater fault tolerance. In ITAG with the activity A^{a}_{l} the agents have some flexibility by specifying the place of execution as a result of some function "*l*". Also, a branch of ITAG [15] with the concept of "Goals" introduces a degree of flexibility to the itinerary.

Finally, we consider the Itinerary Graph [16] system. In their work the migration strategies which are used to perform the mobile agent's actions are *sequential*, *parallel* and *selective*. The *selective* strategy is similar to ITAG's *Conditional Non-determinism*. The lack of a strategy equivalent to *Independent Non-determinism* is a limitation in Itinerary Graphs.

In terms of platform comparisons, we find that previous work such as [7, 8, 9] give general discussions of the performance and efficiency of JADE and Grasshopper. However, in this paper we compare them in a more specific manner through the implementation of ITAG and its four behaviours, namely *sequential, parallel, Independent Non-determinism* and *Conditional Non-determinism*.

7. CONCLUSIONS AND FUTUREWORK

This paper describes an implementation of ITAG (ITinerary AGent) system based on the theory of itinerary scripting language which aims to minimize the effort in mobile agent applications development. A description of the itinerary language has been given with examples as well as descriptions of the implementation of ITAG on JADE. The ITAG Engine is the fundamental component of the ITAG implementation which can be ported to any mobile agent platform. The experimental results demonstrate an evidently better performance of the new ITAGIII under JADE platform compared with ITAGII under Grasshopper. Under the tested situations, there were no noticeable differences in performance between JADE based agents on the same host and on different hosts. In our future work, we aim to extend the itinerary language with more behaviors as well as enhancing the ITAG demo system with more applications.

8. ACKNOWLEDGMENTS

The work reported in this paper has been funded in part by the Australian Research Council's Research Network on Enterprise Information Infrastructure through the Taskforce on Context-Aware Computing (EII-CAC). We also acknowledge the travel assistance by King Faisal University, AlAhsa, Saudi Arabia.

9. REFERENCES

[1] Esparza, O., Fernandez, M. and Soriano, M. 2003. Protecting mobile agents by using traceability techniques. In Proceedings of the International Conference on Information Technology: Research and Education 2003, 264-268.

[2] Loke, S.W., Schmidt, H. and Zaslavsky, A. 1999. Programming the Mobility Behaviour of Agents by Composing Itineraries. In The 5th Asian Computer Science Conference (ASIAN'99), (Phuket, Thailand), Springer-Verlag, 214–226.

[3] Loke, S.W., Zaslavsky, A., Yap, B. and Fonseka, J.R. 2001. An Itinerary Scripting Language for Mobile Agents in Enterprise Applications. In Proceedings of the 2nd Asia-Pacific Conference on Intelligent Agent Technology (IAT 2001), (Maebashi, Japan), 124-128.

[4] Yap, B. and Fonseka, J.R. 2001. ITAG: Itinerary Agent, DSTC, Monash University, 29.

[5] Bellifemine, F.L., Caire, G. and Greenwood, D. 2007. Developing multi-agent systems with JADE. John Wiley, Chichester, England ; Hoboken, NJ.

[6] Leszczyna, R. 2004. Evaluation of agent platforms, Technical report, European Commission, Joint Research Centre,Institute for the Protection and security of the Citizen, Ispra, Italy

[7] Trillo, R., Ilarri, S. and Mena, E. 2007. Comparison and Performance Evaluation of Mobile Agent Platforms. In Third International Conference on Autonomic and Autonomous Systems (ICAS'07), IEEE Computer Society, 41.

[8] Burbeck, K., Garpe, D. and Nadjm-Tehrani, S. 2004. Scale-up and performance studies of three agent platforms. In IEEE International Conference on Performance, Computing, and Communications, 857-863.

[9] Kusek, K.J.G.J.M. 2006. A Performance Analysis of Multi-Agent Systems. International Transactions on Systems Science and Applications (ITSSA), *1, No. 4*. 335 – 341.

[10] Lu, S. and Xu, C. 2005. A formal framework for agent itinerary specification, security reasoning and logic analysis. In 25th IEEE International Conference on Distributed Computing Systems Workshops. 580-586.

[11] Cheng-Zhong, X. 2002. Naplet: a flexible mobile agent framework for network-centric applications. In Proceedings International, IPDPS 2002, Parallel and Distributed Processing Symposium, 219-226.

[12] Rech, L., Montez, C. and de Oliveira, R. 2006. A New Model for the Itinerary Definition of Real-Time Imprecise Mobile Agents. In 2006 IEEE International Conference on Information Reuse and Integration, 123-126.

[13] Rech, L., Montez, C. and de Oliveira, R. 2006. A Clone-Pair Approach for the Determination of the Itinerary of Imprecise Mobile Agents with Firm Deadlines. In ETFA '06. IEEE Conference on Emerging Technologies and Factory Automation, 9-15.

[14] Rech, L., de Oliveira, R.S. and Montez, C. 2005. Dynamic determination of the itinerary of mobile agents with timing constraints. In IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 45-50.

[15] Toan, P., Loke, S.W. and Harland, J. 2003. Adding flexibility using structured goals: the case of itinerant mobile agents. In IEEE/WIC International Conference on Intelligent Agent Technology. IAT 2003, 562-565.

[16] Bo, Y., Da-You, L., Kun, Y. and Wang, S.-S. 2003. Strategically migrating agents in itinerary graph. In International Conference on Machine Learning and Cybernetics, 1871-1876.

A Universal Criteria Catalog for Evaluation of Heterogeneous Agent Development Artifacts

Lars Braubach Alexander Pokahr Winfried Lamersdorf

Distributed Systems and Information Systems
Computer Science Department, University of Hamburg
Vogt-Kölln-Str. 30, D-22527 Hamburg, Germany
{braubach|pokahr|lamersdorf}@informatik.uni-hamburg.de

ABSTRACT

The research discipline of multi-agent systems is characterized by a high degree of heterogeneity. This heterogeneity leads to a vast amount of options (e.g. different architectures and languages) how to employ agent technology but is also one major source of difficulties for its adoption. People interested in using multi-agent systems depend on solid survey articles, which clarify and evaluate these different options and explain in which situations which choices should be made. A survey should also propose viable classification means for helping readers to understand which development artifacts broadly exhibit similar properties. To date, in most cases multi-agent system surveys do without classifications and only address one specific type of artifact such as agent languages or tools. Often, only the characteristics of the representatives are described without evaluating them. In this work a universal criteria catalog will be presented that has been defined abstractly enough for being usable for a wide variety of agent development artifacts. It will be shown how this abstract catalog can be further refined with respect to the chosen area of investigation. In addition to the catalog its general usage as part of a survey will be explained and a blueprint for survey conduction will be presented. To demonstrate its usefulness cutouts of extensive evaluations, performed in the areas of agent architectures, languages, methodologies, tools and platforms, will be presented.

1. INTRODUCTION

The high heterogeneity of the multi-agent systems (MAS) research field leads to many options for realizing agent applications. As many of the available solutions (be it methodologies, platforms or other things) are suitable only in specific application contexts it is very important to have guidelines at hand for the selection of the right option with respect to the given problem [9]. One viable instrument for people interested in agent technology consists in studying surveys and evaluations about specific agent artifacts such as agent architectures or languages. Regrettably, most existing surveys do not contain evaluations of the described artifacts and available comparisons of artifacts suffer from ad-hoc classifications resp. selections as well as from non-standardized evaluation criteria. Especially, divergent criteria make it hard to appraise and compare evaluation results, because it remains unclear if the considered criteria are relevant and if there are others not discussed at all.

To improve this situation in this paper a *universal criteria catalog* is presented that has been deduced from established standards and is sufficiently generic for being utilized for the evaluation of arbitrary agent artifacts. The usage of the catalog fosters several important aspects. Firstly, evaluations of the same artifact type become comparable making visible the advances in the multi-agent research field, e.g. platform surveys from nowadays and from 5 years ago could show in which areas (e.g. operating ability) progress has been achieved. Secondly, evaluations of different artifact types become comparable. This will allow identifying how the state-of-the-art with respect to different artifacts is related and e.g. in which area research should be urged. Thirdly, the criteria catalog has been conceived to be usable in different scenarios. This only requires conceiving a suitable weighting scheme, which emphasizes the different criteria according to their importance with respect to the overall evaluation objective.

Besides the criteria catalog itself it is also sketched what else needs to be done to obtain significant evaluation results. Therefore, a survey blueprint is presented highlighting all important aspects that a survey (including an evaluation part) should possess and also in which order they should roughly be performed. Additionally, an evaluation process is proposed, that describes, how to apply the criteria catalog to a concrete evaluation setting, e.g. considering the artifact type that is to be evaluated.

The rest of this paper is structured as follows. In section 2, we give an overview over types of artifacts related to the development of agent-based systems. Section 3 introduces the universal criteria catalog, which is proposed for the evaluation of such artifacts. An abstract evaluation process is described in section 4. To illustrate the catalog usage, excerpts of evaluations, performed in the area of agent architectures and programming languages, are presented in sections 5 and 6, respectively. In section 7 we discuss the pros and cons of the presented approach. Section 8 concludes the paper with a summary and an outlook.

2. ARTIFACTS FOR MAS DEVELOPMENT

Agent technology has been applied to and further devel-

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

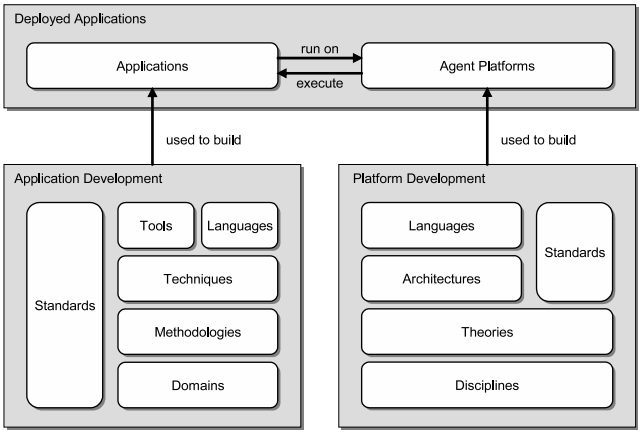


Figure 1: MAS Development artifacts

oped in many entirely different fields, such as robotics, business process management, or social simulation. The various unrelated backgrounds of people involved in agent research have led to a high diversity of the research area, which is at the same time a strength and a weakness of agent technology as a whole. The motivation for the work presented in this paper is rooted in the availability of numerous artifacts – conceptual (e.g. design approaches) as well as tangible (e.g. software products) – for supporting the development of a vast variety of agent-based software systems. Due to the diversity of the field, it is commonly not apparent to an agent developer, which types of artifacts are required for a given software development problem and how to select suitable candidates from the available representatives of each required artifact type.

To illustrate the landscape of potentially relevant artifacts, an overview of the most important artifact types playing a role in MAS development is shown in Fig. 1 (from [8, 23]). In this figure it is assumed that an agent application runs on an agent platform, which in turn is used to execute agent applications (see *Deployed Applications*). For the development of both artifacts different kinds of additional artifacts are necessary (see *Application Development* vs. *Platform Development*). For the application development especially methodologies, modelling techniques, tools and programming langauges are essential whereas the platform development relies on agent theories that are concretized to architectures and agent languages. Moreover, application development should be rooted in some kind of application or problem domain and the development of agent platforms is often based on ideas from other disciplines, such as philosophy or biology. Finally, different kinds of standards apply to different artifact types and ensure e.g. that agent platforms can communicate with each other or that applications follow the guidelines of a specific domain.

Given such a landscape with many choices for each of the artifact types, agent developers as well as agent researchers need suitable instruments for finding well-performing solutions that fit to their individual software project context. The universal criteria catalog proposed in this paper is one such instrument, as it not only allows to compare representatives of the same artifact type, but also to contrast evaluation results regarding different artifacts.

3. CRITERIA CATALOG

In this section it is described what comprises the proposed universal criteria catalog, how it was deduced from existing standards and how it relates to other work in the field of agent surveys and evaluations.

In general there are two opposing requirements for the criteria catalog. On the one hand, the catalog should be universal and apply to all possible kinds of agent development artifacts (cf. 2). On the other hand, the catalog should be highly concrete to allow meaningful results, when evaluating specific types of artifacts for specific settings. The first requirement, i.e. the universality of the catalog, is essential for achieving comparability of evaluation results by providing a stable set of criteria, which can also be used in future investigations. Moreover, applying a single universal set of criteria to many different types of artifacts is the best way to ensure the completeness of the criteria catalog.

On the other hand, having concrete criteria, as mandated by the second requirement, is the precondition for obtaining profound and accurate evaluation results. Detailed and clearly defined criteria contribute to the objectiveness of an evaluation, as they prevent unconscious or ad-hoc assessments. A concrete and detailed criteria catalog also allows for adapting the evaluation to a specific setting, by weighting different criteria according to their need. When e.g. an agent platform is searched as a teaching vehicle, one would apply different ratings to criteria as one would do in an industrial software project.

3.1 Foundations

Before introducing our approach allowing to adequately meet both opposing requirements, we give a short overview of related and preceding work in the area of evaluation criteria and evaluating agent development artifacts. Our work is based on existing standards such as ISO 9126 and ISO 9241 [17, 18] and on evaluations of agent-oriented artifacts such as development methodologies or programming environments [14, 33, 11, 9]. The ISO 9126-1 standard defines six general criteria for evaluating software product quality: *functionality*, *reliability*, *usability*, *efficiency*, *maintainability*, *portability*. These criteria (called characteristics) are subdivided into more specific subcharacteristics (e.g. *fault-tolerance* as part of *reliability*). The standard motivates, that subcharacteristics are further refined by attributes, which can be verified or measured. Those attributes are specific to concrete types of software and not defined in the standard. Moreover, the set of ISO 9241 specifications describes guidelines for the ergonomics of human system interaction that can e.g. be used for the further refinement of certain (sub)characteristics.

Besides existing standards, another source for relevant evaluation criteria are the existing surveys in the area of agent-oriented development artifacts. Unfortunately, most of these surveys focus on one specific type of artifact and therefore do not consider the universality of the applied criteria. Nevertheless, the surveys are helpful for identifying criteria that are considered relevant by agent developers. E.g. Sturm and Shehory [33] propose the criteria categories *concepts and properties*, *notations and modeling techniques*, *process*, and *pragmatics*, refined into numerous subcriteria, for evaluating agent-oriented methodologies. For evaluating agent development software, such as agent platforms or development environments, Eiter and Mascardi [14] introduce the categories *agent attitudes*, *software engineering*

support, *agent and MAS implementation*, *technical issues*, and *economical aspects*, together with specific characteristics for each category. This work forms the basis of our criteria catalog as described next.

3.2 Criteria Catalog Proposal

To meet the requirements of a universal yet concrete criteria catalog, we adopt the ISO approach of broad criteria categories, refined into more detailed, but still universal subcriteria. Basically, the catalog proposal distinguishes between *functional* and *non-functional* requirements. Functional requirements are directly related to the scope of operation, whereas non-functional requirements consider the quality of a given artifact and its provided functions. In the criteria catalog non-functional requirements have been further decomposed into the categories *usability*, *operating ability* and *pragmatics*, thereby subsuming and generalizing the six main criteria of ISO 9126-1. Similarly, the subcriteria are obtained from a unification and generalization of criteria found in standards and existing surveys. In the following the top-level and subcriteria of the catalog will be presented in more detail:

Function: The function encompasses all functional properties of an artifact. Which concrete functional requirements are important highly depends on the kind of researched artifact, e.g. in the context of programming languages other aspects are considered than in the context of architectures. The detailed criteria can be deduced asking the following typical questions:

- Which are the base concepts of the artifact?
- What does the artifact enable and which restrictions exist (power, missing concepts, capabilities)?
- Is the function of the artifact adequate in the given context (for which contexts was it conceived/is it usable)?
- Is the function of the artifact adequate with respect to the developer's knowledge and capabilities (suitable for beginners/experts only)?

Usability: The usability of an artifact refers to its suitability for the construction of agent applications. The degree of usability is evaluated according to the following aspects:

- **Simplicity/intuitivity:** How simple are the underlying artifact's mechanisms (simplicity vs. power)? Is the artifact intuitive, i.e. can the developer use the artifact in an understandable and anticipatable way?
- **Learnability/familarity:** What learning curve has the artifact? Is the developer already familiar with the artifact or its underlying mechanisms from another context?
- **Individualization:** Can the artifact be tailored towards the user and/or the context?
- **Extensibility:** Can the artifact be extended with new functionalities?
- **Software engineering principles:** Does the artifact respect well-known principles such as modularization, refactoring, verification and/or does it facilitate the reusability of elements in other contexts?

Operating ability: The operating ability of an artifact encompasses all aspects that are relevant while the artifact is executed (used), i.e. which properties does the artifact exhibit during its operation? The operating ability's quality is measured using the following aspects:

- **Performance:** How efficient is the artifact with respect to space- and/or time-critical operations?
- **Robustness:** How tolerant is the artifact with respect to (partial) breakdowns?
- **Stability:** How does the artifact behave if executed during a longer time period?
- **Scalability:** How does the artifact behave when applied to varying problem sizes?

Pragmatics: Pragmatic aspects refer to external factors that are neither related to the construction nor to the operation of the artifact. Nonetheless, pragmatic aspects can exert an important influence on the evaluation of the artifact under consideration (e.g. a software), as they determine to a high degree if an artifact can be used in practice. In detail, pragmatic aspects are subdivided into the following criteria:

- **Installation/adoption:** How easily can the artifact be installed?
- **Documentation/examples/support:** Is documentation, example code and support available for the artifact? What quality do they have?
- **Popularity:** How big is the user community? Are field reports available?
- **Maturity:** How mature is the artifact conceptually as well as technologically?
- **Technical boundary conditions:** How easily can the artifact be embedded into existing IT landscapes? How well does the artifact fit to the technological mainstream?
- **Costs:** Which costs are related with the artifact (purchase, construction, operation, working time, training, etc.)?

To conclude, this section has presented a universal criteria catalog, which is based on established ISO-standards and consists of the four main categories: *function, usability, operating ability* and *pragmatics*. It is general enough to be used for the evaluation of all important agent development artifacts and provides enough detailed requirements for producing meaningful and comparable evaluations as explained in the next sections.

4. USAGE OF THE CATALOG

The criteria catalog has been used as a foundation for extensive research in the field of multi-agent systems and was specifically used by the authors to describe and evaluate agent architectures, languages, methodologies, platforms and tools [8, 23]. In this section it will be shown how the catalog can be embedded as part of a survey with respect to a specific development artifact. Therefore, a blueprint of such a survey will be sketched and later on example surveys in the selected areas of agent architectures and languages will be presented. Conducting a survey naturally requires incorporating related work as much as possible with respect

to other surveys as a whole and also with respect to all important parts of a survey as explained next.

The survey blueprint we propose consists of the following logical steps: *artifact definition*, *classification*, *selection*, *catalog refinement*, *evaluation* and *summarizing results*. Starting point of each survey should be a discussion of the meaning of the selected artifact. In this respect the important definitions from literature should be discussed and it should be made clear (if the notion is not unambiguously defined) why a certain definition forms the foundation of the further explorations. Thereafter, the state-of-the-art should be described. For this purpose, in a first step classifications have to be taken from literature or newly conceived and evaluated with respect to the research objective.

A good classification serves two purposes: firstly, it should help identifying structures in the research field giving the unfamiliar reader an initial orientation and secondly it should facilitate the selection of specific representatives for a detailed evaluation as representatives within the same category have similar properties. Given that a lot of individual representatives are usually available a detailed discussion of all of them is often neither feasible nor desired and it is sufficient to select prototypical representatives from each important category. As last preparing step for the description of the state-of-the-art it is necessary to perform the already mentioned catalog refinement with respect to the given research artifact. In many cases it should be sufficient to elaborate the meaning of the functional criteria of the catalog as the non-functional aspects are quite stable and rather independent of the artifact type. Thereafter, the selected individual representatives should be explained and evaluated with respect to the criteria catalog. Finally, the individual results should be condensed by abstracting away from details and calculating key data. The formula for aggregating the results can vary according to the research objective and should be defined in beforehand of the evaluation (e.g. as part of the catalog refinement step).

The proposed survey blueprint can be tailored to a concrete evaluation scenario by introducing specific evaluation scales. In the following, as an example a generic scheme will be explained that is easy to apply and additionally fosters objective evaluation results. The proposed scheme exhibits a neutral weighting with regard to the criteria, and intends that the evaluation will be done in three steps. Firstly, each individual subcharacteristic of the criteria catalog (such as simplicity/intuitivity) should be rated on a coarse scale (+1/-1/0 standing for yes/no/undefined resp. good/bad/neutral). Such a coarse evaluation scale (e.g. instead of values from 0 to 10) speeds up the evaluation process and also improves objectiveness, because the decision if a criterion is fulfilled or not is less prone to be influenced by personal taste. To add further details, each decision should be justified by a meaningful textual description of the relevant reasons, which allows readers of the evaluation to follow or scrutinize the decisions.

In a second step, the rating of a main characteristic (such as usability) can be calculated as a sum of the individual results. As the number of subcriteria differs for the main characteristics, these summary values should be normalized to a common scale (e.g. from -2 to +2 meaning from very weak to very strong). As final step, the overall evaluation result can be determined as mean of the main characteristics, assuming for a generic evaluation, that all main characteristics are equally important.

5. ARCHITECTURE SURVEY

In the following a cutout of the agent architecture evaluation from [8] will be presented. The presentation here will highlight how the proposed blueprint in general and criteria catalog in particular can be used for conducting a survey.

Even though there are several surveys that target agent architectures such as [30, 36] nearly all of them focus on a detailed description and possibly classification of the individual representatives, whereas none focuses on an exhaustive evaluation and comparison of the representatives (in [30] at least some selected properties are compared). One reason for this might be that architectures are abstract and in connection with the development of MAS it seems more obvious to evaluate agent platforms, because these artifacts will directly be used. However, agent platforms employ agent architectures and therefore inherit a good deal of their properties. This means that architecture evaluations can e.g. help to identify the underlying conceptual limitations of agent platforms.

5.1 Architecture Survey: Artifact Definition

Bass et al. [2] define: "The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." It therefore has the basic task of making structures of elements and their relationships visible.

In the area of multi-agent systems it is broadly distinguished between *internal agent architectures* and *multi-agent (or social) architectures*. An internal agent architecture tackles the question of what comprises an agent, i.e. which building blocks can be used for its construction and how these concepts are related. On the other hand, a social agent architecture tries to describe how coordination between agents, e.g. in the sense of teamwork, can be conceptualized.

5.2 Architecture Survey: Classification

Different kinds of classifications have been proposed for internal architectures. The most influential scheme is the one of Wooldridge and Jennings [36], who introduced a distinction between *reactive*, *deliberative* and *hybrid* architectures. In our work [8] we preferred a more general classification scheme that relates architectures to the discipline and theory from which they originated (see Fig. 2). In general, architectures have not been invented per se but rely on a more abstract description in form of an agent theory. This means the scheme basically distinguishes four different categories according to the disciplines philosophy, psychology, biology and sociology from which they have been influenced. These base categories are further refined towards the relatively abstract descriptions in the form of agent theories such as the Unified Theories of Cognition (UTC) [22] and the Belief-Desire-Intention Model (BDI) [7]. Most architectures have been conceived as an interpretation and concretization of such a theory.

5.3 Architecture Survey: Selection

The selection of representatives then excluded social architectures and also those, which are not intended as foundation for application construction (e.g. psychologically inspired architectures such as ACT-R that are used for experimentation only). Concretely, the Subsumption [10], BDI (here

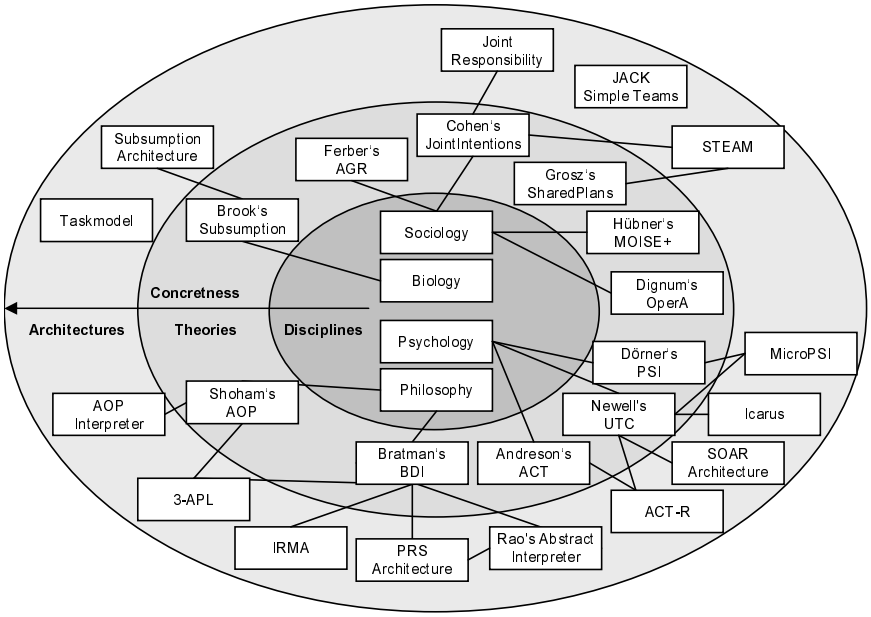


Figure 2: Architecture overview

specifically PRS) [26], Soar [19], AOP [32], 3APL [12] and Taskmodel (e.g. [13]) architectures have been evaluated in the context of the criteria catalog.

5.4 Architecture Survey: Catalog Refinement

The catalog refinement with respect to internal agent architectures needs to define what functionality is required from an agent. Therefore, it is necessary to agree upon a generally accepted agent definition. We used the *strong notion of agency* as defined in [36] for that purpose. It stresses the following agent properties: *autonomy, reactivity, proactivity, social abilities* and *mentalistic notions*. Other abilities that go beyond this definition (such as learning) are not considered as required in our evaluation and can only influence it positively.

5.5 Architecture Survey: Evaluation

The descriptions of the individual architectures and their evaluation with regard to the criteria catalog cannot be presented in full length here due to the space limitations (see [8] for the whole survey). To get an impression of how the catalog is used for the evaluation of an artifact instance the detailed evaluation of one representative with respect to one main characteristic will be further illustrated. For this purpose the *usability* of the Soar architecture will be discussed:

Simplicity/intuitivity (=): The basic principles of the Soar architecture are simple and have been formulated in terms of a few hypotheses (esp. the physical symbol system and the problem space hypothesis). Foundation of the behavior control is the search within problem spaces, which relies on the intuitive concepts of beliefs and goals. The intuitivity is yet reduced by the fact that only one solution context can be active at the same time which is not in line with the parallel goal achievement behavior of humans.

Learnability/familarity (-): The learnability of the architecture is rather low because the architecture concepts cannot directly be used for the realization of agents, i.e. instead of goals the developer has to deal with different kinds of low-level production rules. Developers with a solid background on rule-based approaches will have advantages in learning Soar.

Individualization (-): The architecture does not allow being tailored neither towards the user nor to the context.

Extensibility (-): Extensibility has not been integrated at the architectural level of Soar. Therefore, extensions have to be built at the application level, which e.g. has been successfully done for a teamwork approach in [34]. Extensions that aim at enhancing the available base mechanisms such as the knowledge representation or learning method are far more difficult to realize and require a deep understanding of the architecture.

Software engineering principles (=): Soar supports the modular software development by separation of the overall problem into individual problem spaces. This allows for constructing the different functionalities of an application rather independently of each other. Nevertheless, it does not provide true reusability of software artifacts among different application contexts as tight coupling between functionalities exists due to direct connections of problem spaces via common beliefs.

The rating for the usability of the Soar architecture is therefore calculated as weak (-)[1] and underlines that improvements in this direction could further enhance the acceptance of the approach.

5.6 Architecture Survey: Summarizing Results

The result table is depicted in Fig. 3 and shows the evaluations of the four main characteristics for all analyzed architectures as well as their overall results. It reveals that none of the architectures was able to achieve very good valuations in all of the criteria. With respect to the different criteria

[1]Calculated as normalized sum: (0+(-1)+(-1)+(-1)+0)/5*2=-1.2

Internal Agent Architectures

Evaluation Criteria	Subsumption	BDI	Soar	AOP	3APL	Taskmodel
Function	-	+	+	-	+	-
Usability	+	+	-	-	=	=
Operating Ability	++	++	++	+	+	+
Pragmatics	-	+	+	-	-	++
Result	= (0,3)	+ (1,3)	+ (0,8)	- (-0,5)	= (0,3)	+ (0,5)

Legend: ++ very strong + strong = neutral - weak -- very weak

Figure 3: Architecture evaluation results

it is conspicuous that the *operating ability* of all architectures is high or even very high meaning that all architectures can be implemented efficiently. In contrast, the other criteria have produced mixed results. In most architectures the *function* can be further improved and exhibits weaknesses with respect to the required properties of the strong notion of agency. The same applies for the *usability* which is not as good as it is often claimed for the agent paradigm, i.e. even though the paradigm provides intuitive metaphors, it does not automatically lead to understandable and software-technically sound architectures. Pragmatic aspects mainly depend on the number and quality of available software implementations, which is best for widely used approaches such as the Taskmodel, BDI and Soar.

Considering the overall results, BDI, Soar and Taskmodel architectures have attained good evaluations, whereby the BDI architecture is the only architecture without obvious weaknesses. Nonetheless, all architectures can be improved especially with respect to the function and usability.

6. LANGUAGE SURVEY

This section recapitulates an evaluation of agent-oriented programming languages performed in [23]. Existing surveys of agent-oriented programming languages such as [36, 5, 4, 24] have been taken into account primarily for reviewing existing categorizations, e.g. with regard to the language type (e.g. logic-based vs. procedural) [5, 4], the design approach (new language vs. extension of existing language) [24], or aspects of the agent architecture (e.g. deductive, practical, reactive or hybrid reasoning) [36]. This review made sure that the scope of the planned survey would be broad enough to incorporate all relevant work in the area. Moreover specific evaluations, e.g. of logic based [21] or BDI-style languages [20] were investigated as part of the process of finding suitable refinements of the criteria catalog (i.e. the refinement should make sure that aspects of existing evaluations are considered).

6.1 Language Survey: Artifact Definition

In general, a programming language is commonly agreed to be an artificial language that can be used to instruct machines. E.g. the ACM SIGPLAN group [1] defines programming languages as "[...] languages that permit the specification of a variety of different computations, thereby providing the user with significant control (immediate or delayed) over the computer's operation." In the area of agent-based systems, programming languages are used for a number of different purposes. We adopt the scheme proposed by Ferber [15], which introduces subgroups of agent-oriented languages, such as communication languages and knowledge representation languages. In our evaluation, we focused on languages for describing the behavior of agent systems.

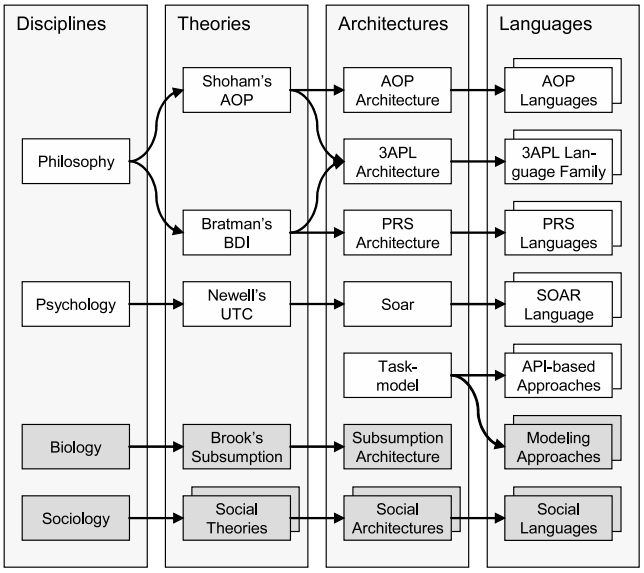


Figure 4: Language overview

6.2 Language Survey: Classification

The classification of agent languages is straightforward, as it is based on the previously presented classification of agent architectures with regard to their origin (see Fig. 4). E.g. the PRS language category is introduced for languages implementing a PRS-like architecture as originating from the philosophical BDI model. It is noteworthy that the language type "API-based approach" has been identified as the common way to implement the Taskmodel architecture, where "API-based" means that instead of conceiving a new language only a programming library for an existing language (e.g. Java) is provided.

6.3 Language Survey: Selection

During the selection process, social languages have been excluded (c.f. Fig. 4), as the focus of our investigation was on programming constructs for individual agents. Moreover, the Subsumption architecture category has no correspondence in the language survey. Moreover, graphical modeling approaches (e.g. [16]) were excluded, restricting the scope of the survey to text-based programming languages. From each of the remaining five categories – AOP, 3APL, PRS (=BDI), Soar, API-based (=Taskmodel) – the most prominent representatives, based on academic as well as industrial recognition, have been selected for further investigation. For some categories such as PRS languages, where a large number of representatives exists, a pre-analysis of the field has been performed to identify existing relationships (cf. Fig. 5).

Investigated PRS languages are AgentSpeak(L) as implemented in the academic Jason interpreter [6] and the languages of the commercial JACK agent toolkit (JAL) [35] and the academic Jadex framework [25]. In the area of AOP languages, the original work on Agent-0 [32] has been considered, as well as the languages employed in the commercial AgentBuilder (RADL) [27] and the academic Agent Factory

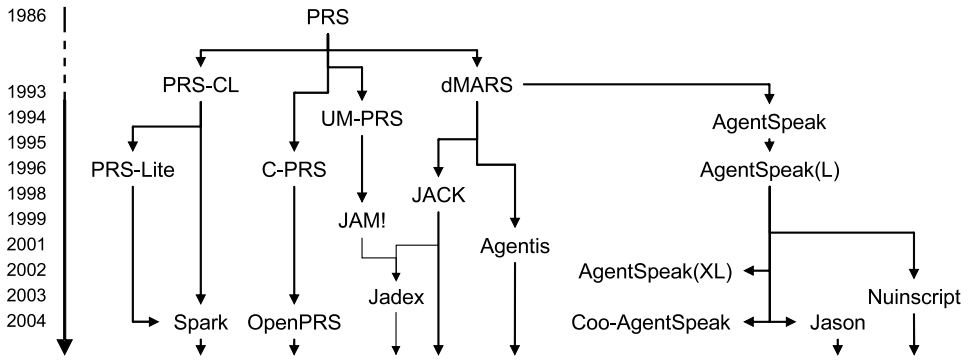


Figure 5: PRS-like systems and relations

toolkits (AF-APL) [29]. The JADE platform [3] and the Living Systems Technology Suite (LS/TS) from Whitestein Technologies [28] have been selected as representatives of API-based approaches. For 3APL and Soar, the most recent incarnations (at that time), as described in [12] and [19], have been evaluated.

6.4 Language Survey: Catalog Refinement

The refinement of the criteria catalog introduces relevant areas of functionality for programming languages. The functionality of a programming language is represented by its programming constructs. Based on general programming language literature (e.g. [31]) and existing evaluations of agent-oriented programming languages (e.g. [21, 20]) three basic functional criteria have been identified: *concept abstraction*, *control flow abstraction*, and *safety*. These criteria are briefly explained in the following. For a more detailed explanation see [23].

Concept Abstraction: Concept abstraction means the introduction of programming constructs for some kind of high-level concept (e.g. abstract data types, procedures or – in the agent world – goals or commitments). Concept abstraction is subdivided into *data abstraction*, *behavior abstraction*, and *functionality abstraction*, which respectively denote the availability of programming language constructs for representing data (e.g. objects), behavior (e.g. activities), and functionality (e.g. modules).

Control Flow Abstraction: One main requirement for an agent-oriented programming language is to provide constructs for specifying independent or interrelated sequences of activities, i.e. control flow abstraction. Subcriteria for control flow abstraction are *concurrency*, *dynamic behavior*, and *dynamic execution*. Concurrency constructs (e.g. threads and semaphores) allow to specify (quasi-) parallel activities and synchronization points between them. Dynamic behavior means the ability of an agent to dynamically select among different actions when requested to perform some task. Finally, with concepts for dynamic execution (e.g. event or exception handlers), behaviors can be activated in a dynamic (runtime-dependent) way.

Safety: Safety subsumes built-in functionality of a programming language aimed at reducing the number of programming errors and can be subdivided into *error prevention* and *error recognition* functionality. Error prevention reduces possible sources for errors by design (e.g. local name spaces prevent name conflicts, automatic garbage collection reduces memory management errors). Error recognition functionality, on the other hand, aims at detecting errors early in the development process. E.g. with static typing, many errors can be detected at compile time, which would otherwise occur at runtime, and asserts allow to easily find errors at runtime, which might otherwise stay undetected while leading to incorrect results.

6.5 Language Survey: Evaluation

The evaluation process for languages follows the generic evaluation scheme already applied to the architectures, i.e. coming to yes / no decisions about individual subcriteria, which are then summarized to produce the overall score. The results for each of the subcriteria have been obtained in analytical and empirical evaluations. To evaluate criteria such as simplicity/intuitivity, example applications have been implemented with all of the languages. Among these applications, some simple benchmarks, initially implemented for evaluating platform performance (e.g. memory consumption and time needed for creating/destroying a given number of agents) were also useful for evaluating the language constructs available for these common tasks. As an example, the evaluation of the functionality of the JACK agent language (JAL) is presented next:

Concept Abstraction (+): JAL is based on Java and therefore allows data to be represented in an object-oriented fashion. A slight drawback is the fact, that the internal reasoning of JACK requires some data to be represented in a simple relational model, which does not fit well with the object-oriented abstraction. Behavior abstraction is nicely supported by the notion of plans, as well as functionality abstraction is provided by the so called capability concept.

Control Flow Abstraction (=): The control flow of a BDI agent is driven by the practical reasoning process consisting of the steps goal deliberation and means-end reasoning. The PRS approach, as implemented in JACK, focuses on means-end reasoning, i.e. selecting plans for achieving goals or reacting to events. While the language therefore provides very good control flow abstraction in this area, the first step of practical reasoning, i.e. the selection of goals to pursue, is not directly supported. This part of the agent behavior therefore needs to be implemented in a manual, ad-hoc fashion by the agent developer.

Safety (+): The new constructs that JAL introduces as extensions to the Java language have been designed to allow for compile-time consistency checks and therefore support

Programming Languages

Evaluation Criteria	AOP	PRS	3APL	Soar	API-based
Function	-	+	=	=	+
Usability	=	+	=	-	=
Operating Ability	+	++	=	++	+
Pragmatics	=	+	=	+	+
Result	= (0,0)	+ (1,3)	= (0,0)	+ (0,5)	+ (0,8)

Legend: ++ very strong, + strong, = neutral, - weak, -- very weak

Figure 6: Language evaluation results

the safety of the language. Moreover, Java constructs for runtime checks, such as asserts, can be used in JAL as well.

As a result, JAL is attested a good (+) functionality score,[2] while identifying also some areas for improvement.

6.6 Language Survey: Summarizing Results

One aim of the survey was comparing programming approaches abstracting away from concrete language implementations. Therefore, the results of the individual representatives, such as AgentSpeak or JAL, have been used to compose a unified result for each approach. To avoid that deficiencies of a single representative weaken the score of an approach as a whole, for each main criteria such as function or usability, the best value has been selected instead of the average.

The accumulated scores of the evaluation are shown in Fig. 3. PRS languages are rated best, followed by API-based approaches and Soar. This result might also be influenced by the fact that these are the language families, where ongoing commercial development takes place. Another indication for the importance of commercial development is that the winners achieve their best scores in the area of operating ability. Nevertheless, it can be seen that still many areas for improvement exist, e.g. the continued integration of agent-oriented concepts into useful programming language constructs to further improve the functionality of agent-oriented programming languages.

7. DISCUSSION

The criteria catalog proposed in this paper has proven useful in a number of evaluations (see [8, 23]) and in this respect could be used in isolation for aiding some future evaluations. Nevertheless, to be of value to the research community as a whole and to achieve the ultimate goal of comparable evaluations, the criteria catalog needs to be picked up by agent researchers. Hence, in this section we give an overview of the pros and cons of the approach and argue in favor of its adoption.

The main objectives of the universal criteria catalog are twofold. For individual analysts, the availability of a widely accepted criteria catalog should ensure the completeness of the investigations and largely simplify the task of producing meaningful evaluation results. For the research community as a whole, the adoption of a universal catalog would lead to a better comparability of independently conducted evaluations. This comparability would on the one hand apply to evaluations concerning similar artifact types. For example, evaluations of agent systems such as JADE, Jadex, Jason, etc. often reflect a special viewpoint of the investigator as some researcher might be more interested in agent architectures, while others are interested in programming languages or execution platforms. Applying universal criteria in all of these evaluations would therefore facilitate comparisions to a great extent. On the other hand, also the comparability of evaluations concerning quite different artifact types (e.g. methodologies vs. platforms) would contribute to an assessment of the level of maturity of the different areas of the research field.[3]

The approach taken in this proposal follows a multi-staged process, in which 16 universal criteria (grouped in the areas *function*, *usability*, *operating ability* and *pragmatics*) can be refined or interpreted with respect to other criteria, that are specific to an evaluation. This approach allows to recast criteria of existing evaluations by subordinating each specific criterion to an adequate universial criterion and then comparing the cumulated results of different evaluations. Moreover, the approach does not exclude any criteria that are considered relevant by some researcher, because the catalog does not prescribe the usage of some subset of commonly used criteria, but instead the idea is that any relevant criterion can be used as an interpretation or refinement of some more abstract criterion. Finally, in addition to the catalog itself, the proposed steps for an evaluation (*definition*, *classification*, *selection*, *refinement*, *evaluation*, *summary*) foster different forms of reusability across evaluations. Besides the criteria refinement, e.g. an existing classification could be reused and applied to a new (or extended) set of artifacts or an existing evaluation can be newly summarzied by using a different weighting scheme according to a new problem setting.

We believe that, even though our proposed selection of universal criteria might leave room for discussion, the general idea is the only way to meet the objectives outlined above. As an alternative one might consider using separate criteria catalogs for each artifact type or for every single evaluation, but this would impair the desired comparability. Moreover, this alternative does not offer many advantages, as the universal catalog is meant to be general enough to cover arbitrarily detailed refinements of criteria. Another alternative would be not to have only a few universal criteria, but to have a catalog of virtually all possible criteria. We think that such an approach, although perfectly viable in higly standardized domains, can not be applied to the area of agent technology, due to the high diversity of the field. For example, some researchers might be interested in details (like commitment strategies) that are not available in some examined artifacts (e.g. the subsumption architecture) leading to exceedingly oversized evaluations cluttered with many "does not apply" ratings. Again, for specialized evaluations, such detailed criteria can also be incorporated in our approach by performing a suitable criteria refinement.

During our work, we considered a number of objections against the approach, which we would like to rebut in the following. Objections can relate to the catalog as a whole

[2]Calculated as normalized sum: (1+0+1)/3*2=+1.3

[3]We e.g. found in our evaluations that the methodologies could not achieve as high overall ratings as agent architectures, which indicates that agent architectures are more mature than agent methodologies.

(incomplete, not enough focused on agents), to the criteria itself (too abstract), and to the steps of the evaluation process (refinement leading to arbitrariness or divergence, cumulated ratings omit important information).

Catalog completeness. Thanks to an extensive analysis of available standards and existing surveys, we think that our criteria catalog is quite complete in the sense that all criteria that we found could be subordinated to some of our abstract criteria and that the identified criteria apply to all kinds of artifacts.[4] Nevertheless, we do not claim the total completeness of the approach and would appreciate sensible extensions or enhancements in order to adapt the catalog to further artifact types.[5]

Catalog focus. Although devised for the area of agent technology, the catalog indeed does not contain criteria that exclusively focus on agents. As stated above, this is a result of the diversity of the research field and has to be remedied by suitable catalog refinements for the subarea of interest. On the other hand, this can also be seen as an advantage, because it would allow to compare artifacts from agent technology to other solutions from competing areas, such as object orientation, SOA and web services. Nevertheless, since the catalog has not (yet) been applied to these areas, we do not make claims regarding the significance of such cross-area comparisons in this paper.

Criteria abstractness. It is true that the quest for universality harbors the danger of criteria becoming too abstract and therefore meaningless. In our work, we have applied the presented criteria catalog to five different artifact types and a total of 27 representatives (4 methodologies, 6 architectures, 6 programming languages, 6 development tools, 5 platforms). In these evaluations, we found all criteria quite helpful. Even though they are indeed partly abstract, they encourage investigators to think in a systematic way about the qualities of the examined artifacts and lead to new insights and useful results about the objects of investigation.

Evaluation comparability. Regarding the steps of the evaluation process, the different taylorings, refinements and choices that are necessary in each step (e.g. regarding the artifact classification or the evaluation function) bear some potential for divergence and arbitrariness. But this is also (and even more so) true for any other evaluation that is e.g. performed on the basis of ad-hoc selections. Using the proposed process, comparability can be at least partially established, because the choices during the process are made explicit as part of the evaluation and should therefore be well motivated instead of being of implicit, ad-hoc nature.

Evaluation scheme. The proposed evaluation scale has been conceived for a general evaluation of agent artifacts without consideration of some special (e.g. application) focus. Hence, the resulting values are only general indications of an artifacts maturity and for any concrete setting a different evaluation scheme should be applied, which focuses on those qualities that are required for the problem in question.

[4]The refinement of the criteria catalog was most challenging with respect to methodologies, as usability and operating ability here relate purely to the construction of a MAS and not to its operation. For the dintinction of both, different viewpoints have been introduced [8]. Usability is evaluated from an individual perspective of a developer where it is mainly of interest how good he can use the artifact, whereas operating ability is considered from a organizational perspective where it is relevent which advantages and disadvantages for a company exist when applying the methodology in their projects.

[5]For example, we have considered *security* (i.e. robustness against attacks) as a separate criterion in the *operating ability* area. As we found that nearly none of the researched agent artifact types (besides a few agent platforms) addresses security we decided to exclude this aspect in most of our investigations.

8. CONCLUSION

This paper has tackled the question how a developer can choose among the many development options when implementing an agent application. One key aspect here is to understand that agent technology currently offers many problem-specific solutions that address only certain types of application domains. We argue that one important foundation for making accurate choices is the availability of well-defined and comparable surveys and evaluations of artifacts such as platforms or methodologies. Therefore, we have proposed a new universal criteria catalog for evaluating many different kinds of agent artifacts. This is possible because the criteria catalog is two-staged, consisting of an abstract artifact-agnostic stage and additionally an artifact-specific stage, which needs to be refined with respect to the concrete artifact type. The general applicability of the criteria catalog has been proven by cutouts of two extensive evaluations in the area of agent architectures and programming languages, but was also successfully employed for methodologies, platforms and tools [8, 23].

Besides the criteria catalog it has also been shown how it can be used as one part in a complete survey. For this purpose the integral ingredients of a survey have been identified and their coarse ordering has been defined. In a first step the *definition of the artifact* has to be explained for establishing a common discussion basis. Thereafter an overview of the available representatives should be given and a *selection* of artifact instances should be performed by applying a meaningful *classification*. The main part of the survey should then present the selected representatives and *evaluate* them against the in beforehand *refined criteria catalog*. Finally, the detailed results should be *summarized* and coarsened to deduce also globally valid statements.

In future work we want to employ the criteria catalog to perform individualized surveys that try to reason about the degree of usefulness of artifact instances with respect to specific application scenarios (e.g. which agent platforms are specifically suited for the transportation domain and why). Additionally, we plan to investigate the applicability of the presented criteria catalog regarding other non agent-related artifact types (such as component-based approaches). Finally, a main objective and hope of the paper is that other researchers pick-up the universal criteria catalog for their planned investigations. This would lead to many positive effects especially with respect to the completeness of the used criteria and the comparability with other evaluations.

9. REFERENCES

[1] ACM SIGPLAN. Bylaws of the special interest group on programming languages of the association for computing machinery. ACM, 2003.

[2] L. Bass, P. Clements, and R. Kazman. *Software architecture in practice*. Addison-Wesley, 2005.

[3] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE - A Java Agent Development Framework. [5], pages 125–147.

[4] R. Bordini, L. Braubach, M. Dastani, A. El Fallah Seghrouchni, J. Gomez-Sanz, J. Leite, G. O'Hare, A. Pokahr, and A. Ricci. A survey of programming languages and platforms for multi-agent systems. *Informatica*, 30:33–44, 2006.

[5] R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.

[6] R. Bordini, J. F. Hübner, and R. Vieira. Jason and the Golden Fleece of Agent-Oriented Programming. [5], pages 3–37.

[7] M. Bratman. *Intention, Plans, and Practical Reason*. Harvard Univ. Press, 1987.

[8] L. Braubach. *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme*. PhD thesis, Univ. Hamburg, 2007.

[9] L. Braubach, A. Pokahr, and W. Lamersdorf. Tools and Standards. In S. Kirn, O. Herzog, P. Lockemann, and O. Spaniol, editors, *Multiagent Systems. Intelligent Applications and Flexible Solutions*, pages 503–530. Springer, 2006.

[10] R. A. Brooks. How to build complete creatures rather than isolated cognitive simulators. In *Architectures for Intelligence*. Lawrence Erlbaum Associates, 1989.

[11] M. Casagni and M. Lyell. Comparison of two component frameworks. In *Proc. of ICSE'03*, pages 341–351. IEEE Computer Society, 2003.

[12] M. Dastani, B. van Riemsdijk, and J. J. Meyer. Programming Multi-Agent Systems in 3APL. [5], pages 39–67.

[13] S. A. DeLoach. Specifying agent behavior as concurrent tasks. In *AGENTS'01*. ACM Press, 2001.

[14] T. Eiter and V. Mascardi. Comparing environments for developing software agents. *The European Journal on Artificial Intelligence*, pages 169–197, 2002.

[15] J. Ferber. *Multi-Agents Systems - An Introduction to Distributed Artificial Intelligence*. Addison-Wesley, 1999.

[16] M. Griss, S. Fonseca, R. Cowan, and R. Kessler. Using UML State Machine Models for More Precise and Flexible JADE Agent Behaviors. In *AOSE*. Springer, 2003.

[17] International Organization for Standadization (ISO). *Software engineering – Product quality – Part 1: Quality model*, ISO/IEC 9126-1:2001 edition, 2001.

[18] International Organization for Standadization (ISO). *Ergonomics of Human-System Interaction- Part 110: Dialogue Principles*, ISO 9241-110:2006 edition, 2006.

[19] J. F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition. Technical report, University of Michigan, 2006.

[20] V. Mascardi, D. Demergasso, and D. Ancona. Languages for programming BDI-style agents: an overview. In *Proc. of WOA'05*. Pitagora Editrice, 2005.

[21] V. Mascardi, M. Martelli, and L. Sterling. Logic-based specification languages for intelligent software agents. *CoRR*, cs.AI/0311024, 2003.

[22] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[23] A. Pokahr. *Programmiersprachen und Werkzeuge zur Entwicklung verteilter agentenorientierter Softwaresysteme*. PhD thesis, Univ. Hamburg, 2007.

[24] A. Pokahr, L. Braubach, and W. Lamersdorf. Agenten: Technologie für den mainstream? In *it - Information Technology*. Oldenbourg, 11 2005.

[25] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. [5].

[26] A. Rao and M. Georgeff. BDI Agents: from theory to practice. In *Proc. of ICMAS'95*, pages 312–319. MIT Press, 1995.

[27] Reticular Systems. *AgentBuilder User's Guide*, version 1.3 edition, 2000.

[28] G. Rimassa, D. Greenwood, and M. E. Kernland. The Living Systems Technology Suite. In *In Proc. ICAS'06*, 2006.

[29] R. Ross, R. Collier, and G. O'Hare. AF-APL - Bridging Principles and Practice in Agent Oriented Languages. In *Proc. of ProMAS'04*, pages 66–88. Springer, 2005.

[30] M. Scheutz and V. Andronache. The apoc framework for the comparison and evaluation of agent architectures. In *AAAI Workshop*. AAAI Press, 2004.

[31] R. Sebesta. *Concepts of Programming Languages*. Addison Wesley, 2005.

[32] Y. Shoham. Agent-Oriented Programming. *Artificial Intelligence*, 60(1), 1993.

[33] A. Sturm and O. Shehory. A Framework for Evaluating Agent-Oriented Methodologies. In *Proc. of AOIS'03*, pages 94–109. Springer, 2004.

[34] M. Tambe. Towards Flexible Teamwork. *Art. Intelligence Research*, 7, 1997.

[35] M. Winikoff. JACK Intelligent Agents: An Industrial Strength Platform. [5].

[36] M. Wooldridge and N. Jennings. Agent theories, architectures, and languages: A survey. In *Proc. of ATAL 1994*, pages 1–39. Springer Verlag, 1995.

Implementing reactive BDI agents with user-given constraints and objectives

Aniruddha Dasgupta
Decision Systems Lab
School of Computer Science and Software Engineering
University of Wollongong
Wollongong, NSW 2522, Australia
ad844@uow.edu.au

Aditya K.Ghose
Decision Systems Lab
School of Computer Science and Software Engineering
University of Wollongong
Wollongong, NSW 2522, Australia
aditya@uow.edu.au

ABSTRACT

CASO is an agent-oriented programming language based on AgentSpeak(L), one of the most influential abstract languages based on the BDI (Beliefs-Desires-Intentions) architecture. For many applications, it is more convenient to let the user provide in real time, a more elaborate specification consisting of constraints and preferences over possible goal states. Then, let the system discover a plan for the most desirable among the feasible goal states. CASO incorporates constraints and objectives into the symbolic approach of reactive BDI model which lead to better expressive capabilities as well as more efficient computation. Jason is a fully-fledged interpreter for a much improved version of AgentSpeak(L). In this work we modify Jason to incorporate the operational semantics of CASO. CASO also uses ECLiPSe, an open source constraint solver, to apply constraint solving techniques. Our preliminary results show that CASO can be used as a powerful multi agent programming language in solving problems in complex application domains.

1. INTRODUCTION

Agent-oriented programming is highly suited for applications which are embedded in complex dynamic environments, and is based on human concepts, such as beliefs, goals and plans. This allows a natural specification of sophisticated software systems in terms that are similar to human understanding, thus permitting programmers to concentrate on the critical properties of the application rather than getting absorbed in the intricate detail of a complicated environment. One of the most popular and successful framework for Agent technology is that of Rao and Georgeff [11], in which the notions of *Belief*, *Desire* and *Intention* or *BDI* are central. Beliefs represent the agent's current knowledge about the world, including information about the current state of the environment inferred from perception devices and messages from other agents, as well as internal information. Desires represent a state which the agent is trying to achieve. Intentions are the chosen means to achieve the agent's desires, and are generally implemented as plans and post-conditions. As in general an agent may have multiple desires, an agent can have a number of intentions active at any one time. These intentions may be thought of as running concurrently, with one chosen intention active at any one time. Besides these components, the BDI model includes a plan library, namely a set of "recipes" representing the procedural knowledge of the agent, and an event queue where both events (either perceived from the environment or generated by the agent itself to notify an update of its belief base) and internal subgoals (generated by the agent itself while trying to achieve a desire) are stored. Usually, BDI-style agents do no adopt first principles planning at all, as all plans must be generated by the agent programmer at design time. The planning done by agents consists entirely of context-sensitive subgoal expansion, which is deferred until a point in time at which the subgoal is selected for execution. The BDI model provides that all the knowledge of a rational agent about the world is organized in statements that are its beliefs. An agent's desires depict some states of the world that the agent "would like" to be realized. In the *multi-agent systems* (MAS) community each agent is given the mandate to achieve defined goals. To do this, it autonomously selects appropriate actions, depending on the prevailing conditions in the environment, based on its own capabilities and means until it succeeds, fails, needs decisions or new instructions or is stopped by its owner.

In this paper present an implementation our design of CASO agent [7], discussing in more detail: (a) the requirements specification with respect to CASO; (b) an implementation of the CASO design; (c) some example exploring the CASO design and and finally (d) the strengths and weaknesses of our design. From an implementation point of view the existence of special libraries or dedicated programming languages that provide data and control structures for manipulating agent specific properties allows for an easy implementation of agent models. The remainder of this article is organized as follows. Section 2 gives a brief introduction of popular BDI language AgentSpeak(L) [10] as well as talks briefly about Jason [5], an interpreter of AgentSpeak(L); section 3 discusses ECLiPSe [2], a constraint programming toolkit respectively; section 4 describes the language CASO and gives an overview of its operational semantics; section 5 describes the implementation details; section 6 gives an example of using CASO and section 7 provides some experimental results. Finally, the last section describes some related work and gives concluding remarks.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

2. AGENTSPEAK(L)

AgentSpeak(L) is an agent framework/language with explicit representations of beliefs and intentions for agents. This agent programming language was initially introduced by Rao [10]. AgentSpeak(L) is a programming language based on a restricted first-order language with events and actions. The behaviour of the agent (i.e., its interaction with the environment) is dictated by the programs written in AgentSpeak(L). The beliefs, desires, and intentions of the agent are not explicitly represented as modal formulas. Instead, designers can ascribe these notions to agents written in AgentSpeak(L). The current state of the agent, which is a model of itself, its environment, and other agents, can be viewed as its current belief state; states which the agent wants to bring about based on its external or internal stimuli can be viewed as desires; and the adoption of programs to satisfy such stimuli can be viewed as intentions.
AgentSpeak(L) agent is described as a set of
$\langle E, B, P, I, A, S_E, S_O, S_I \rangle$ where:

- E is a set of events.
- B is a set of base beliefs.
- P is a set of plans.
- I is a set of intentions.
- A is a set of atomic actions.
- S_E selects an event from the set E.
- S_O selects a plan from the set P.
- S_I selects an intention from the set I.

The *alphabet* of the formal language consists of variables, constants, function symbols, predicate symbols, action symbols, connectives, quantifiers, and punctuation symbols. Apart from firstorder connectives, AgentSpeak(L) uses ! (for achievement), ? (for test), ; (for sequencing), and $\leftarrow$ (for implication). There are two types of goals in AgentSpeak(L). An "achievement goal" (a predicate prefixed with "!"), states that the agent wishes to achieve a state of the world in which the associated predicate is true. A "test goal" (a predicate prefixed with "?"), states that the agent wishes to test if the associated predicate is a true. Events in AgentSpeak(L) might be external or internal. External events represent the changes in the state of the world that should be handled by the agent. On the other hand, internal events are triggered from within the agent as a result of executing a plan. An agent must have pre-designed plans in its plan library to handle the incoming internal or external events. Plans are the central concept to the abilities of an agent. They are means that enable an agent to respond to the changes in its' environment.
One of the most popular fully-fledged interpreter of AgentSpeak(L) is **Jason**. Jason has many extensions making up for a very expressive programming language for cognitive agents. It implements the operational semantics of that language, and provides a platform for the development of multi-agent systems, with many user-customisable features. Jason is implemented in Java (thus multi-platform) and is available Open Source, distributed under GNU LGPL.

3. ECLIPSE: A CONSTRAINT SOLVER

Constraint satisfaction is a powerful computational paradigm which proposes techniques to find assignments for problem variables subject to constraints on which only certain combinations of values are acceptable. The success and the increasing application of this paradigm in various domains mainly derive by the fact that many combinatorial problems can be expressed in a natural way as a Constraint Satisfaction Problem (CSP), and can subsequently be solved by applying powerful CSP techniques.
ECLiPSe is a software system for the cost-effective development and deployment of constraint programming applications, e.g. in the areas of planning, scheduling, resource allocation, timetabling, transport etc. It is also ideal for teaching most aspects of combinatorial problem solving, e.g. problem modelling, constraint programming, mathematical programming, and search techniques. It contains several constraint solver libraries, a high-level modelling and control language, interfaces to third-party solvers, an integrated development environment and interfaces for embedding into host environments.

4. CASO: A REACTIVE BDI LANGUAGE

The concept of using constraints and explicit objectives in a high-level agent specification language like Agentspeak(L), yields significant advantages in terms of both expressivity and efficiency as shown in our previous work in [8]. The improvised technique applies constraint and objective directed solving on the context section of a BDI agent's plan specification in order to determine an application plan to fire. CASO (**C**onstraint **A**gent**S**peak(L) with **O**bjective) is a programming language based on the popular BDI language AgentSpeak(L)which incorporates constraints and objectives into the symbolic approach of BDI model. CASO incorporates Constraint Solving and Optimization (CSOP) techniques where the optimization is based on the objective function (softgoal).
In CASO, one can express agents' goals quantitatively - for example, agents can have some utility (objective) function which needs to be maximized. Incorporating constraints into a reactive BDI agent programming language can lead to better expressive capabilities as well as more efficient computation (in some instances). More interestingly, the use of constraint-based representations can make it possible to deal with explicit agent objectives (as distinct from agent goals) that express the things that an agent may seek to optimize at any given point in time. CASO also incorporates efficient option selection (selecting the best plan to use to deal with the current event) with parametric look-ahead techniques, i.e., techniques where the extent of look-ahead style deliberation can be adjusted. The typical CASO execution cycle is characterized by the following steps:

1. observe the world and the agent's internal state, and update the event queue consequently;
2. generate possible new plan instances whose trigger event matches an event in the event queue (relevant plan instances) and whose precondition (beliefs and constraints in plan body) is satisfied (applicable plan instances); plan selection is based on the satisfiability of the current set of constraints as well as the one which maximizes the current objective(using look-ahead techniques);

3. select for execution one instance from the set of applicable plan instances;
4. push the selected instance onto an existing or new intention stack, according to whether or not the event is a (sub)goal;
5. select an intention stack, take the topmost plan instance and execute the next step of this current instance: if the step is an action, perform it, otherwise, if it is a subgoal, insert it on the event queue.

Informally, an agent program in CASO consists of a set of beliefs B, a set of constraints C, an objective function O, a set of events E, a set of intention I, a plan library P, a constraint store CS, an objective store OS and three selection functions S_E, S_P, S_I to select an event , a plan and an intention respectively to process and n_p and n_i are the two parameters which denote the number of steps to look-ahead for plan and intentions selection respectively.

Definition: *A CASO agent program is a tuple* $\{B, P, E, I, C, O, S_O, S_E, S_I, n_p, n_i, CS, OS\}$ *where*

- *B is a set of Beliefs.*
- *P is agent plan repository, a library of agent plans.*
- *E is set of events (including external and internal).*
- *I is a set of intentions.*
- *C is a set of constraints.*
- *O is an objective function.*
- S_E *is a selection function which selects an event to process from set E of events.*
- S_O *is a selection function which selects an applicable plan to a trigger t from set P of plans.*
- S_I *is a selection function which selects an intention to execute from set I of intentions*
- *CS is a constraint store which stores constraints which come as events.*
- *OS is an objective store which stores the objective function which comes as an event.*
- n_p *is an integer which denotes the number of steps required to look-ahead for plan selection.*
- n_i *is an integer which denotes the number of steps required to look-ahead for intention selection.*

In CASO, a constraint directed improvisation is incorporated into the computation strategy employed during the interpretation process. Constraint logic programming (CLP) combines the flexibility of logic with the power of search to provide high-level constructs for solving computationally hard problems such as resource allocation.
Formally, a language CLP(X) is defined by

- *constraint domain X,*
- *solver for the constraint domain X and*
- *simplifier for the constraint domain X*

A CASO ***plan*** *p is of the form* $t : b_1 \wedge b_2 \wedge \cdots \wedge b_n \wedge c_1 \wedge c_2 \wedge \cdots \wedge c_m \leftarrow sg_1, sg_2, \cdots, sg_k$ *where* t *is the trigger; each* b_i *refers to a belief; each* c_i *is an atomic constraint; each* s_g *is either an atomic action or a subgoal.*
For brevity we will use *BContext(p)* to denote the belief context of plan p. Thus

$$BContext(p) \equiv b_1 \wedge b_2 \wedge \cdots \wedge b_n$$

Similarly, we will use *CContext(p)* to denote the constraint context of plan p. Thus

$$CContext(p) \equiv c_1 \wedge c_2 \wedge \cdots \wedge c_m$$

Transition of agent program to process events depends on the event triggers. An event trigger, *t*, can be addition(+) or removal(-) of an achievement goal($\pm!g_i$) or a belief($\pm b_i$).

4.1 Informal Semantics

The CASO interpreter manages set of events, a constraint store, a objective store and a set of intentions with three selection functions. Intentions are particular courses of actions to which an agent has committed in order to handle certain events. Each intention is a stack of partially instantiated plans. Events, which may start off the execution of plans that have relevant triggering events, can be external when originating from perception of the agentsŠ environment (i.e., addition and deletion of beliefs based on perception are external events) ; or internal, when generated form the agentŠs own execution of a plan (i.e., as subgoal in a plan generates an event of the type addition of an achievement goal).
In the latter case, the event is accompanied with the intention which generated it (as the plan chosen for that event will be pushed on top of that intention). External events create new intentions, representing separated focuses of attention for the agentŠs acting on the environment.
The constraint store is initialized by the relevant constraints whenever a trigger contains a constraint in its context. At every cycle of the interpreter, the constraint store is enhanced with new constraints when applicable selected plan is executed. These incremental constraints collecting process eventually leads to a final consistent constraints set. Constraint solving is applied to the context of each plan to determine applicable plans as well as to generate solutions for subsequent actions. Similarly, the objective store contains the set of objective functions that need to be maximized (or minimized) which are part of the event context and is similarly updated at each cycle. *Plan Selection* is described in detail in the next subsection.
At every interpretation cycle of an agent program, CASO updates a list of events, which may be generated from perception of the environment, or from the execution of intentions (when subgoals are specified in the body of plans). It is assumed that beliefs are updated from perception and whenever there are changes in the agents beliefs, this implies the insertion of an event in the set of events. On top of the selected intention there is a plan, and the formula in the beginning of its body is taken for execution. This implies that either a basic action is performed by the agent on its environment, an internal event is generated (in case the selected formula is an achievement goal denoted by $!g_i$), or a test goal is performed (which means that the set of beliefs has to be checked). If the intention is to perform a basic action or a test goal denoted by $?g_i$, the set of intentions needs to be updated. In the case of a test goal, the

belief base will be searched for a belief atom that unifies with the predicate in the test goal. If that search succeeds, further variable instantiation will occur in the partially instantiated plan which contained that test goal (and the test goal itself is removed from the intention from which it was taken). In the case where a basic action is selected, the necessary updating of the set of intentions is simply to remove that action from the intention (the interpreter informs to the architecture component responsible for the agent effectors what action is required). When all formulae in the body of a plan have been removed (i.e., have been executed), the whole plan is removed from the intention, and so is the achievement goal that generated it (if that was the case).
This ends a cycle of execution, and CASO starts all over again, checking the state of the environment after agents have acted upon it, generating the relevant events, and so forth.

4.2 Plan selection with parametric look-ahead

After S_E has selected an event, CASO has to unify that event with triggering events in the heads of plans. This generates a set of all *relevant plans*. The constraints (if any) that are included in the constraint part of the context are put in the *constraint store*. The context part of the plans is unified against the agents beliefs. Constraint solving is now performed on these *relevant plans* to determine whether the constraint(s) in the context of the plan is (are) consistent with the constraints already collected in the constraint store . This results in a set of *applicable plans*(plans that can actually be used at that moment for handling the chosen event).
The objective store maintains a set of objective function which may be present in the event context. At each interpreter cycle, the objective store is also updated with an *objective function* for maximizing (or minimizing).

Given plans p_1 and p_2 in the plan library, and given a current constraint store C and a current objective store O, $p_1 \leq_{opt} p_2$ if and only if: $OptSol(C \cup CContext(p_1), OS) \geq OptSol(C \cup CContext(p_2), O)$.

$OptSol(Constraints, Objective)$ denotes the value of the objective function when applied to the optimal solution to the problem denoted by the pair (Constraints, Objective). We assume of course that $C \cup CContext(p_1)$ and $C \cup CContext(p_2)$ are solvable.
Optimization techniques are now applied by the optimizer to each of the applicable plan to determine an optimal solution. In effect we are solving a 'Constraint Satisfaction Optimisation Problem' (CSOP) which consists of a standard 'Constraint Satisfaction Problem' (CSP) and an optimisation function that maps every solution (complete labelling of variables) to a numerical value. S_O now chooses this optimal solution from that set, which becomes the intended means for handling that event, and either pushes that plan on the top of an existing intention (if the event was an internal one), or creates a new intention in the set of intentions (if the event was external, i.e., generated from perception of the environment). One of the properties of CASO is that since CSOP is solved at various steps using a solver, all the beliefs and constraints must be *global* variables. Plan selection is defined as follows:

Given a trigger t and a set of applicable plans $AppPlans(t)$ for t, a plan $p \in AppPlans(t)$ is referred to as an O-preferred plan if and only if: $p \leq_{opt} p_i$ for all $p_i \in AppPlans(t)$.

The agent program is also responsible for making sure that the objective store is consistent at any point of time. During each cycle of the interpreter, new objectives are added into the objective store and hence a consistency checker is used to maintain consistency. Formally a *consistent objective store* is defined as below.

Given an objective store OS and a new objective f, the result of augmenting OS with f, denoted by OS^*_f, is defined as $\gamma(MaxCons(OS \cup f))$ where γ is a choice function and $MaxCons(X)$ is the set of all $x \subseteq X$ such that

1. *x is consistent and*
2. *there exists no x' such that $x \subset x' \subseteq X$ and x' is consistent*

The new objective store is now given by $\gamma(MaxCons(OS \cup \overline{O}) \cap OS)$ where γ is the choice function, OS is the objective store and $\overline{O}$ is the negation of the objective O.
Selection of O-preferred plan can be further enhanced by using n_p the look-ahead parameter form plan selection. In case n_p=0, no look-ahead is performed and maximizing the objective function on the set of *applicable plans* would result in an *O-preferred* plan as described earlier. However, if $n_p > 0$ then a look-ahead algorithm (used for choosing the next move in a two-player game) is performed to select the O-preferred plan.
We assume that the agent is trying maximize its objective function and the environment may change in the worst possible way which would minimize the objective function. The goal of the agent would be to select a plan which would maximize the minimum value of the objective function resulting from the selection of plans which may occur due to the set of new possible events that may come from the environment.
We follow the definition of *goal-plan tree* given in [13] to decompose the set of plans into a tree structure. In CASO, goals are achieved by executing plans and each goal has at least one plan, if not many, that can be used to satisfy the goal. Each plan can include sub-goals, but need not have any. The leaf nodes of the tree are plan-nodes with no children (i.e., no sub-goals).
Each goal-plan tree consists of - a number of 'AND' nodes which are subgoals that must be executed sequentially for the goal to succeed; and a number of 'OR' nodes which are subgoals any one of which must be executed for the goal to succeed. Given a set of applicable plans, an agent would always try to achieve this objective at every decision step. However, there could be unforeseen situations which may result in the agent changing its normal course of action at any of these decision points. Thus the strategy for the agent is to compute in advance the worst case scenario that may occur due to the change in the highly dynamic environment. Figure 1 shows the tree decomposition for plan P depicting all possible choices (OR nodes). The numbers corresponding to the leaf nodes are the values of the optimization function (say, f) which we are trying to maximize. Using the *LookAheadPlanSelection* look-ahead algorithm shown in Algorithm 1, we obtain the value of 3 at the root node which suggest that the agent should follow plan P2.

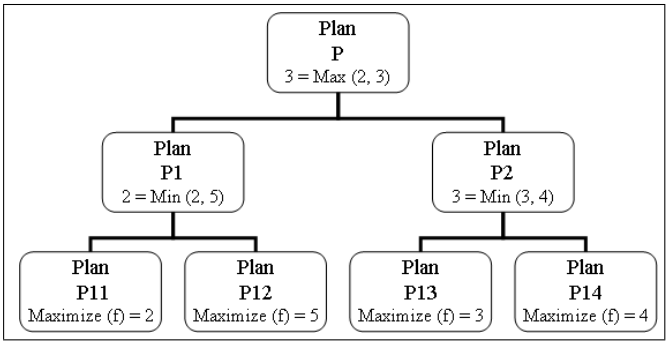


Figure 1: Plan Tree

5. IMPLEMENTING CASO

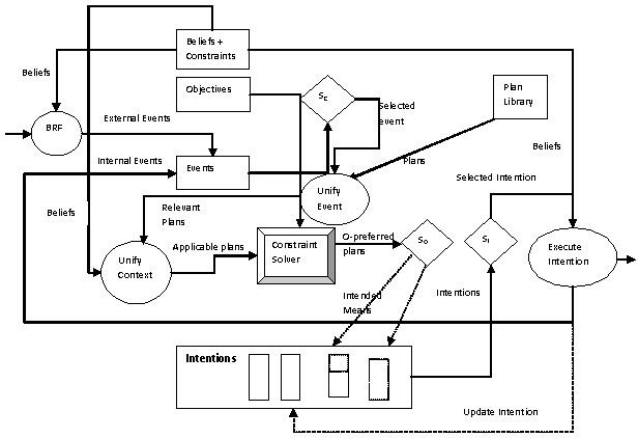


Figure 2: Operational Semantics of CASO

In this section we give some more details on the implementation of a CASO interpreter, which is clearly depicted in Figure 2 (modified from [5]). The pictorial description of such interpreter, greatly facilitates the understanding of the interpreter. In the figure, sets (of beliefs, events, plans, and intentions) are represented as rectangles. Diamonds represent selection (of one element from a set). Circles represent some of the processing involved in the interpretation of CASO programs. The 3-d box represents the ECLiPSe CLP solver that is plugged into the system which is responsible for the option selection function based on the set of objectives and beliefs and/or constraints.

5.1 ECLiPSe plug-in for option selection function

As mentioned in earlier sections, we use ECLiPSe for option selection function S_O. A CASO agent has a set of constraints/beliefs in its belief base and a set of objective functions at any point in time during the execution cycle. When an agent tries to select a plan from a possible set of *applicable plans*, it invokes the ECLiPSe constraint solver to determine the *O-preferred plan*. The ability for an user to add and remove objectives is a unique feature of a CASO agent which is not embedded inside the selection functions. Beliefs in CASO are written as ECLiPSe CLP programs

Algorithm 1 LookAheadPlanSelection(int n, state S, ObjectiveStore OS, ConstraintStore CS)

1: Generate *goal-plan tree* up to n levels from current state S comprising of subgoals of AND and OR nodes with subplans.
2: Start from the root node.
3: Let constraint store at node $p = c_p$.
4: Let o_p denote the value of objective function at node p.
5: For each node p in the goal plan tree set $c_p \leftarrow CS$
6: **if** node p has child nodes $p1, p2 \cdots, pk$ in an AND structure **then**
7: Apply constraint solving at each pi with the current constraint store c_{pi} and the set of constraints for pi to obtain o_{pi}.
8: Set $c_{pi+1} \leftarrow c_{pi}$ for all $i \geq 1$.
9: Initialize constraint store for all child nodes of each p_i with c_{pi}.
10: **end if**
11: **if** node p has child nodes $p1, p2 \cdots, pk$ in an OR structure **then**
12: Compute the objective function and update the constraint store for each pi.
13: Initialize constraint store for all child nodes of each p_i with c_{pi}.
14: **end if**
15: **while** $n \neq 1$ **do**
16: Propagate minimum value of objective function up to each parent node starting from the leaf node.
17: $n = n - 1$.
18: **end while**
19: Propagate the maximum value of its children for state S.
20: At state S, the best plan is the child with the maximum value.

as shown in figure 3 below where *Vars* represent the set of variables. Constraints are defined on the variables as shown below. In the example below, let us assume that the belief *resource_available()* is a part of the agent belief base. As per CASO, this belief is stored as in a file *resource_available.ecl*. In this example, when the predicate *resource_available()* is executed as a query, the solver would solve the optimization as per the objective shown in the figure and generate a solution. The solver first calls the *eplex* (External CPLEX Solver Interface) library which allows an external Mathematical Programming (MP) solver to be used by ECLiPSe. *eplex* is one of the most widely distributed, scaleable and efficient packages incorporating a linear constraint solver. A problem in ECLiPSe is modeled by a set of simultaneous equations: an objective function that is to be minimized or maximized, subject to a set of constraints on the problem variables, expressed as equalities and inequalities. The *eplex* library allows for the user to write programs that combines MP's global algorithmic solving techniques with the local propagation techniques of Constraint Logic Programming. The result of the objective function along with the instantiated variables (Vars) is stored in an output file *resource_available.txt*. A CASO agent may not have any objective function initially in which case there would be no function to minimize and the solver may choose to instantiate the variables with a possible set of variable instantiation based on the set of constraints. In

case the objective comes externally from the environment, the file *resource_available.ecl* is modified and specific objective function is written into the file. When the objective changes, i.e. a new objective comes to the objective store, this file is re-written with the new objective function. The text file (*resource_available.txt*), containing the output from the ECLiPSe solver is read by S_O and if there are several applicable plans to pursue, S_O would choose the plan which produces the highest value of the objective function and is then pushed as an intention. In case there are actions associated with plan whose parameters are part of the ECLiPSe CLP, the variable values are also pushed along with the intention.

***Belief*:**
resource_available() :-
Vars = [A1,A2,A3,B1,B2,B3,C1,C2,C3,D1,D2,D3],
Vars :: 0..inf,
integers(Vars),
(A1 + A2 + A3 = 21),
(B1 + B2 + B3 = 40),
(C1 + C2 + C3 = 34),
(D1 + D2 + D3 = 10),
(A1 + B1 + C1 + D1 =< 50),
(A2 + B2 + C2 + D2 =< 30),
(A3 + B3 + C3 + D3 =< 40).

Optimization function:
optimize(max(1*A1 + 7*A2 + 200*A3 + 8*B1
+ 5*B2 + 2*B3 + 5*C1 + 5*C2
+ 1*C3 + 6*D1 + 4*D2 + 1*D3))

Figure 3: CASO Belief and Objective function written in ECLiPSe CLP style

5.2 Modifying Jason

We are not going into the details of our modifications but we only describe the important changes to Jason in this section. Since we have kept the essence of Jason interpreter, the only notable change we did has been with regards to the new operational semantics that has been described earlier. In particular, our main modifications to Jason are the following:

- CLP-style beliefs (with constraints and objectives) are now written in a separate file that can be modified by an external event when a new objective is added or deleted. The ECLiPSe solver reads this file and generates output.

- *TransitionSystem.java* which is part of the *asSemantics* package on Jason, is modified to call the external ECLiPSe solver and does the new file handling operation. It also implements the look-ahead function for option selection which can be added as a parameter.

Any application that can be deployed in Jason can also be deployed in CASO with the added benefit of application of an objective function that can be user defined which can change with every interpretation of CASO execution cycle by an external event.

6. EXAMPLE: USING CASO IN REAL-TIME DECISION MAKING FOR BIOMASS SUPPLY CHAIN

Our implementation of CASO as described earlier, can be shown by the following example where we try to describe the benefit of using CLP in agent paradigm. The example is chosen from a supply chain system where we describe the optimizations carried out by a single agent.
The supply chain considerations and costs of using biomass fuel on a large scale for electricity generation at power stations is quite complex and is made up of a range of different activities which is described in detail in [1]. The activities can include ground preparation and planting, cultivation, harvesting, handling, storage, in-field/forest transport, road transport and utilisation of the fuel at the power station. Moreover, the options for supplying the end user with biomass fuel of the right specification, in the right quantity at the right time from resources which are typically diverse and often seasonally dependent. Also, given the typical locations for biomass fuel sources (i.e. on farms and in forests) the transport infrastructure is usually such that road transport will be the only potential mode for collection of the fuel. In order to supply biomass from its point of production to a power station the following activities are necessary:

- Harvesting of the biomass in the field/forest.

- In-field/forest handling and transport to move the biomass to a point where road transport vehicles can be used

- Storage of the biomass. Many types of biomass will be harvested at a specific time of year but will be required at the power station on a year round basis, it will therefore be necessary to store them. The storage point can be located on the farm/forest, at the power station or at an intermediate site.

- Loading and unloading road transport vehicles. Once the biomass has been moved to the roadside it will need to be transferred to road transport vehicles for conveyance to the power station. At the power station the biomass will need to be unloaded from the vehicles.

- Transport by road transport vehicle using heavy goods vehicles for transport to the power station is used due to the average distance from farms to power station, and the carrying capacity and road speed of such vehicles.

- Processing of the biomass to improve its handling efficiency and the quantity that can be transported. This can involve increasing the bulk density of the biomass (e.g. processing forest fuel or coppice stems into wood chips) or unitising the biomass (e.g. processing straw or miscanthus in the swath into bales). Processing can occur at any stage in the supply chain but will often precede road transport and is generally cheapest when integrated with the harvesting

Figure 4 describes the various options faced in the Biomass SCM. The figure shows that there are several decision points in the SCM which affect the final outcome. As an example, trees grown on farmland on a short rotation coppice basis can either be harvested as five meter whole sticks or cut and immediately processed into wood chips. Different harvesting

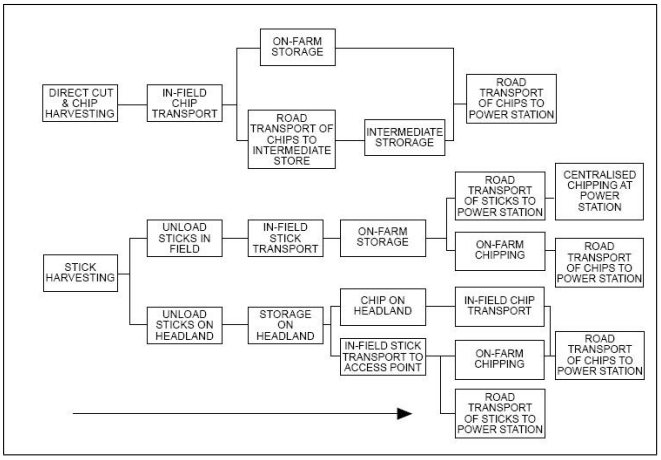


Figure 4: Decision making in Biomass Supply Chain

machinery is required for each of these harvesting systems. While sticks can be stored on the edge of fields without experiencing decomposition, chips need to be stored on at least a hard standing, and in a covered environment if decomposition is to be prevented. Therefore, the storage requirements for the two products are very different. Similarly, sticks and chips require very different transport systems in terms of both handling (i.e. loading and unloading vehicles) and suitable transport vehicle bodies for carrying them.. The easiest and most cost effective harvesting system can result in the need for expensive storage systems and can even lead to an inability to supply fuel of the desired quality to the power station.

The SCM as shown in the figure can be represented by a CASO agent program (Figure 5) which has several plans at its disposal and can choose one of the possible alternate plans based on the current set of constraints as well as the current objective that the SCM is trying to optimize. If we look into Plan#1, we can see that one of the subgoals of stick harvesting is to either chip on field or transport the sticks as denoted by *fieldChipOrTransport()*. These two subgoals are further elaborated in plans Plan#2 and Plan#3 where two possible course of action could take place depending on whether transport to power station is available or chipping on farm is possible. These two possible options are denoted by *transportAvailableToPowerStationFromField()* and *chippingPossibleOnFarm()* as two beliefs which can be written in ECLiPSe style CLP and is shown in Figure 6. For the sake of simplicity, let us assume that the objective function (minimize cost) is given by:

*optimize(min(50*StickDeliveryTrucksReqd + 60*LoadingRobotReqd + 75*StickAsseblersReqd + 50*ChipDeliveryTrucksReqd + 70*ChippingMachineReqd + 65*ChipLoadersReqd)).*

The numbers denote the *$ cost* and the variables (shown in figure 6) denote the quantity of each resource required. The first 3 variables are related to stick transport and the last 3 are related to chipping on field. If we look carefully into the two beliefs, we see that when the agent tries to select either of plans Plan#2 and Plan#3 in figure 5, it evaluates the context of each plan as given by beliefs Belief#1 and Belief#2 in figure 6, and finds out that both plans can equally be selected as all constraints are satisfied. However, once the

Plan #1:
+!stickHarvesting : fieldAvailableToUnload()
←unloadInField(); inFieldTransport();onFarmStorage();
!fieldChipOrTransport().
Plan #2:
+! fieldChipOrTransport():
transportAvailableToPowerStationFromField()
←tranportSticksToPowerStationFromField();
centralizedChipping();
Plan #3:
+! fieldChipOrTransport(): chippingPossibleOnFarm()
←onFarmChipping();
transportChipsToPowerStationFromField().
Plan #4:
+!stickHarvesting : headlandAvailableToUnload()
←unloadOnHeadland(); storeOnHeadland();
!headlandChipOrTransport().
Plan #5:
+!headlandChipOrTransport() : chipPossibleOnHeadland()
←chipOnHeadland(); inFieldChipTransport();
transportChipsToPowerStationFromHeadland().
Plan #6:
+!headlandChipOrTransport() :
transportAvailableToAccessPoint()
←inFieldStickTransportToAccessPoint();
!accessPointChippingOrTransport().
Plan #7:
+!accessPointChippingOrTransport():
chipPossibleAtAccessPoint()
←onFarmChipping();
transportChipsToPowerStationFromAccessPoint().
Plan #8:
+!accessPointChippingOrTransport():
transportAvailableToPowerStation()
←transportSticksToPowerStationFromAccessPoint().

Figure 5: Plans related to a CASO Stick Harvesting Agent

objective function is taken into consideration, the solution obtained from evaluating Belief#1 gives a value of 245 of the objective function and that from Belief#2 gives a value of 185. Based on the current scenario, the agent would choose Plan#1 as it gives lower cost. Tables 1 and 2 show the value of the objective function together with the value of the variables. Now, if the objective function changes at any point, the result obtained may be different and the agent would then choose a different plan based on the circumstances. As an example, the current objective is a function of *cost* but it can equally be made into a function of *time*. Thus, if the consideration is to minimize the amount of *time* that is required to carry out either of the two plans without any regards to the *cost*, a new objective function (minimize) could be written which is a function of *time*. The value of the variables are passed to the intention stack in the CASO interpreter cycle, and are used to initialize the parameters as described earlier. Also, the above example showed only 1-step look-ahead - however, one can easily go to reasonable desired depth of the decision tree (AND/OR goal-plan tree), and obtain all possible values of the objective function. The agent would then choose the plan that would be the best out

BELIEF #1
transportAvailableToPowerStationFromField():-
% integer variables
Vars = [StickDeliveryTrucksReqd, LoadingRobotReqd, StickAsseblersReqd],
integers(Vars),

%constants
StickDeliveryTrucksAvail=2,
LoadingRobotAvail =2,
StickAsseblersAvail =2,

%inequality constraints
%enough trucks for transportation
StickDeliveryTrucksReqd <=StickDeliveryTrucksAvail,

%enough robots for loading sticks onto truck
LoadingRobotReqd <=LoadingRobotAvail,

%enough persons to assemble the sticks
StickAsseblersReqd <=StickAsseblersAvail,

%at least 3 robots + stick assemblers are required
LoadingRobotReqd + StickAsseblersReqd >=3,

%at least 1 delivery truck is required
StickDeliveryTrucksReqd >=1,

%equality constraints:*total number of resources required*
StickDeliveryTrucksReqd + LoadingRobotReqd + StickAsseblersReqd =4

BELIEF #2
chippingPossibleOnFarm():-
%integer variables
Vars = [ChipDeliveryTrucksReqd, ChippingMachineReqd, ChipLoadersReqd],
integers(Vars),

%constants
ChipDeliveryTrucksAvail =2,
ChippingMachineAvail =2,
ChipLoadersAvail =2,

%inequality constraints
%enough trucks for transportation
ChipDeliveryTrucksReqd <=ChipDeliveryTrucksAvail,
%enough chipping machines
ChippingMachineReqd <=ChippingMachineAvail,
%enough persons to load the sticks into chipping machine
ChipLoadersReqd <=ChipLoadersAvail,
%at least 1 machine, 1 loader and 1 delivery truck are required
ChippingMachineReqd => 1,
ChipLoadersReqd >=1,
ChipDeliveryTrucksReqd >=1,

%equality constraints :*total number of resources required*
ChipDeliveryTrucksReqd + ChippingMachineReqd + ChipLoadersReqd =4

Figure 6: Partial Set of beliefs related to a CASO Stick Harvesting Agent

Belief#1	
StickDeliveryTrucksReqd	=1
LoadingRobotReqd	=2
StickAsseblersReqd	=1
Objective	=245

Table 1: Value of variables and objective function for Belief#1

Belief#2	
ChipDeliveryTrucksReqd	=1
ChippingMachineReqd	=1
ChipLoadersReqd	=1
Objective	=185

Table 2: Value of variables and objective function for Belief #2

of all possible worst cases as described in earlier section. It should also be noted here that the we are currently depicting only one agent in a multi-agent scenario which is doing decision making. However, this can easily be extended to a fully fledged MAS where several agents interact with each other and take their own decisions for optimizing their own objectives.

7. EXPERIMENTAL RESULTS

We ran a series of experiments to find out how the *quickly* the system could find the optimal plan. Our goal is to show that with a reasonable number of look-ahead steps and with moderate number of plans/actions, the CASO agent would be *reactive* enough (i.e., perform plan selection in real-time). The experiments were conducted on Intel dual-core machine using complex set of constraints with linear objective functions which were basically solved by the ECLiPSe solver.
For any given CASO program, the following parameters can greatly affect the way plan selection is done:

1. The branching factor of the goal-plan tree (i.e., the number of OR nodes that are present for each plan).
2. The look-ahead depth (or level) of the goal-plan tree up to which CSOP technique will be applied.
3. The number of constraints for each plan.
4. The number of variables in the CASO program. Note that the list of variables in the program has to be globally defined.

We randomly generated CASO programs and tested the plan selection function by fixing some parameters and varying other parameters as given above.

Experiment 1 Given a plan in a CASO program with multiple subplans, we calculated the time taken to find the optimum plan among the choices by fixing the branching factor, number of constraints per plan and the number of variables for a given objective function and varying the *depth* of look-ahead. We set *branching factor = 3, number of constraints/plan = 2* and *number of variables =5.*

Experiment 2 Given a plan in a CASO program with multiple subplans, we calculated the time taken to find the optimum plan among the choices by fixing the look-ahead depth, the branching factor, number of constraints per plan variables and varying the number of *variables*. Note that for this experiment, we generated a number of CASO programs with the same set of plans having same head and body, but different context (different set of variables). Also, we used similar objective function with lesser variables. We set *branching factor = 3*, *number of constraints/plan = 2* and *look-ahead depth =3*.

Experiment 3 Given a plan in a CASO program with multiple subplans, we calculated the time taken to find the optimum plan among the choices by fixing the look-ahead depth, the branching factor, number of variables and the same objective function and varying the number of *constraints per plan*. Note that for this experiment, we generated a number of CASO programs with the same set of plans having same head and body, but different context (different number of constraints per plan). We set *branching factor = 3*, *look-ahead depth = 3* and *number of variables =5*.

Experiment 4 Given a plan in a CASO program with multiple subplans, we calculated the time taken to find the optimum plan among the choices by fixing the look-ahead depth, number of variables, the number of constraints per plan and the same objective function and varied the *branching factor*. Note that for this experiment, we generated a number of CASO programs with different set of plans. We set *look-ahead depth = 3*, *number of variables =5* and *number of constraints/plan = 2*.

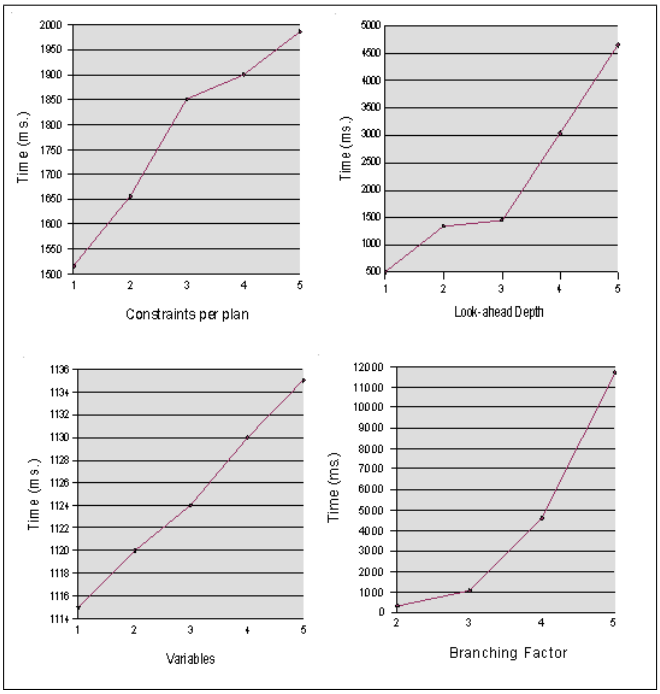


Figure 7: Graphs showing experimental results

Figure 7 depict the results for each of the above experiments. As we can see that with increasing *depth* for the same set of plans, the time taken to find the optimal plan increases. Similar trend is noted when the number of number of *variables* or *constraints per plan* is increased although the difference is not that significant as with varying depth. Finally, if branching factor is increased the time taken to find the optimal plan increases as more combinations have to be generated. It is to be noted here though that for every run, we are generating a different set of plans (a different CASO program), solving LP for each of these programs is quite different each time and there is no consistency among them. Thus, for some CASO programs, it might be such that finding a solution may be faster with a higher branching factor than one with a lower factor. for this reason, we randomly generated 100 CASO programs with different branching factors keeping all other parameters constant and found that on an average, with an increase in the branching factor, the time taken to find the optimal plan also increases (Figure 7).

Overall we can see that the times take (in ms.) are quite small and we can select an optimal very quickly using ECLiPSe together with Jason in our implementation of CASO.

8. RELATED WORK AND CONCLUSION

Decision agents can be designed to provide interactive decision aids for end-users by eliciting their preferences and then recommending matching products. In [6] constraint logic programming and data model approach is used within BDI agent framework. However, this work speaks of BDI agents in general and does not integrate with any BDI programming language. AgentSpeak(XL) programming language [4] integrates AgentSpeak (L) with the TAEMS scheduler in order to generate the intention selection function. It also describes a precise mechanism for allowing programmers to use events in order to handle plan failures which is not included in AgentSpeak(L). This work, however, adds priority to the tasks. Some related theoretical work on selecting new plans in the context of existing plans is presented in [9]. Another related work on detecting and resolving conflicts between plans in BDI agents is presented in [14]. The "degree of boldness" of an agent is defined in [12] which represents he maximum number of plan steps the agent executes before re-considering its intentions. However in this case it is assumed that the agent would backtrack if the environment changes *after* it has started executing the plans.

Our implementation of CASO provides the user with the flexibility of adding explicit objectives and constraints to achieve final goals. CASO uses a modified version of Jason, the well-known BDI AgentSpeak(L) interpreter, together with another open-source constraint solver ECLiPSe thereby combining reactive combining agent programming with constraint solving techniques. CASO is based on the strong theoretical foundations of BDI and in the simple example described in earlier section, we can see that CASO can indeed be deployed in many agent application domains like supply chain, health care etc. as well as used in the design and simulation of such applications where several types of decision making and optimizations may be required. Moreover, the time taken to select a particular plan in real-time is very small (with a reasonable look-ahead depth) and is only

depended on the constraint-solver that we use. In future we plan to extend CASO to incorporate user preferences as c-semiring [3] and implement the design to create a more robust and powerful MAS which can be deployed in complex applications.

9. REFERENCES

[1] J. Allen, M. Browne, A. Hunter, J. Boyd, and H. Palmer. *Logistics management and costs of biomass fuel supply*. MCB UP Ltd., 1998.

[2] K. R. Apt and M. Wallace. *Constraint Logic Programming using Eclipse*. Cambridge University Press, 2007.

[3] S. Bistarelli, U. Montanary, and F. Rossi. Semiring-based constraint satisfaction and optimisation. In *Journal of ACM*. ACM Press, 1997.

[4] R. Bordini, A. Bazzan, R. Jannone, D. Basso, R. Vicari, and V. Lesser. *AgentSpeak(XL):Efficient intention selection in BDI agents via decision-theoretic task scheduling*. ACM Press, 2002.

[5] R. Bordini, J. Hubner, and M. Wooldridge. *Programming Multi-Agent Systems in AgentSpeak Using Jason*. John Wiley & Sons, Ltd, 2007.

[6] S. Chalmers and P. M. D. Gray. Bdi agents and constraint logic. In *AISB Journal Special Issue on Agent Technology*, 2001.

[7] A. Dasgupta and A. K. Ghose. Dealing with objectives in a constraint-based extension to agentspeak(l). In *Proc. of the Pacific Rim International Workshop on Multi-Agents*, 2005.

[8] A. Dasgupta and A. K. Ghose. Caso: A framework for dealing with objectives in a constraint-based extension to agentspeak(l). In *Proc. of the 2006 Australasian Computer Science Conference*, 2006.

[9] J. Horty and M. Pollack. Evaluating new options in the context of existing plans. In *Artifical Intelligence*, 2001.

[10] A. Rao. Agentspeak(l): Bdi agents speak out in a logical computable language. In *Agents Breaking Away: Proceedings of the 7th European WS on Modelling Autonomous Agents in a Multi-Agent World*. Springer-Verlag: Heidelberg,Germany, 1996.

[11] A. Rao and M. Georgeff. *BDI Agents: from theory to practice*. San Fransisco, USA, 1995.

[12] M. Schut and M. Wooldridge. Intention reconsideration in complex environments. In *Proceedings of International Conference on Autonomous Agents*, Varcelona, Spain, 2000.

[13] J. Thangarajah. Managing the concurrent execution of goals in intelligent agents. In *Phd. Thesis*. RMIT, 2004.

[14] J. Thangarajah, L. Padhgam, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *G. Gottlob and T. Walsh, editors, Proceedings of the International Joint Conference on Artificial Intelligence*. Academic Press, 2003.

Modeling Multi-Agent Systems through Event-driven Lightweight DSC-based Agents

Giancarlo Fortino
DEIS
Università della Calabria,
Via P. Bucci cubo 41c
87036 Rende (CS) Italy,
+39.0984.494063
g.fortino@unical.it

Alfredo Garro
DEIS
Università della Calabria,
Via P. Bucci cubo 41c
87036 Rende (CS) Italy,
+39.0984.494795
garro@unical.it

Samuele Mascillaro
DEIS
Università della Calabria,
Via P. Bucci cubo 41c
87036 Rende (CS) Italy,
+39.0984.494754
s.mascillaro@unical.it

Wilma Russo
DEIS
Università della Calabria,
Via P. Bucci cubo 41c
87036 Rende (CS) Italy,
+39.0984.494691
w.russo@unical.it

ABSTRACT
To date several agent models and related programming frameworks have been introduced for developing distributed applications in terms of multi-agent systems in open and dynamic environments. Among them, those based on lightweight architectures, asynchronous messages/events and state-based programming such as Jade, Bond and Actors have demonstrated great effectiveness for modeling open and distributed software systems. In this paper, we propose the Event-driven Lightweight Distilled StateCharts-based Agent (ELDA) model which is based on the same basics of the aforementioned agent models and frameworks, and further enables a more effective design through (i) Statecharts-based specification of the agent behavior, (ii) multiple coordination spaces for local/remote inter-agent and agent/non-agent-component interactions, and (iii) a coarse-grained strong agent mobility. A MAS based on the ELDA model can be easily designed through the ELDA meta-model and programmed through the ELDAFramework, a Java-based implementation of the meta-model. MAS programming is supported by the ELDATool, an Eclipse-based visual tool which also automates code generation. A simple yet effective case study is provided to exemplify the proposed model and its related tools.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *State diagrams.* D.2.6 [**Software Engineering**]: Programming Environments – *Integrated environments.* I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *Multiagent systems.*

General Terms
Design, Languages.

Keywords
Agent Models, Statecharts, Events, Multi-coordination, Mobility.

1. INTRODUCTION
The agent paradigm is one of the mainstream paradigms for the modeling and implementation of complex software systems, especially in open and dynamic environments. As advertised by a manifesto for agent technology [18] agents can be seen as both a design metaphor and a source of technology. In particular, from the design metaphor perspective, several agent models and related frameworks have been to date introduced [26]. Such agent models basically can be classified in two large groups: (i) models based on intelligent agent architectures [18, 21] ranging from reactive agents (e.g. Brook's subsumption architecture) to deliberative agents (e.g. BDI agents); (ii) models based on the mobile active object concept encompassing mobile agent architectures [5, 24]. Models of the first group are mainly oriented to problem-solving, planning and reasoning systems whereas models of the second group are more oriented to distributed computation in open and dynamic environments like the Internet.

In the context of the Internet computing, agent models and frameworks based on lightweight architectures, asynchronous messages/events and state-based programming such as Jade [3], Bond [4], and Actors [2], have demonstrated great effectiveness for modeling and programming agent-based distributed applications.

This paper proposes the Event-driven Lightweight Distilled StateCharts-based Agent (ELDA) model which aims at modeling multi-agent systems (MASs) in the context of open and dynamic computing environments. The ELDA model is based on the same fundamentals of the aforementioned models/frameworks and also introduces new enabling features which allow for a more effective development of MASs. These new features result as an enhancement of basic characteristics provided by two previously defined models (the MAO model [8] and the MC model [11]) and their integration. In particular with reference to the three models (Behavioral, Interaction and Mobility) on which the ELDA model is centered:

- The Behavioral and Mobility models are mainly based on characteristics derived from the MAO (Mobile Active Object) model, which allows for a multi paradigm approach to the construction of distributed applications in highly dynamic distributed environments. In particular, modeling of the agent behavior is based on the Distilled StateCharts (DSC) formalism and agent migration relies on a coarse grain strong mobility model.
- The Interaction model is an extension of the MC (Multi-Coordination) model, which is based on high-level events unifying access to and exploitation of underlying coordination spaces and agent server resources.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,*
May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

Moreover, the paper presents the ELDA meta-model which effectively supports the modeling of multi-agent systems (MAS) based on the ELDA model. A MAS designed through the proposed meta-model can be seamlessly coded by using the ELDAFramework, a Java implementation of the meta-model. The coding phase is facilitated by the ELDATool, an Eclipse-based visual tool which supports graphical programming of the agent behavior and automatic generation of code according to the ELDAFramework. Finally, to exemplify agent programming a case study concerning distributed information retrieval based on coordinated set of agents is also described.

The remainder of this paper is organized as follows. Section 2 presents the ELDA model by describing the Behavioral, Interaction, and Mobility models, and discusses its distinctive features. In section 3 the ELDA meta-model is described by using a views-based approach. Section 4 describes the case study whereas section 5 discusses related work. Finally conclusions are drawn and on-going research briefly delineated.

2. THE ELDA MODEL

The Event-driven Lightweight Distilled Statecharts-based Agent (ELDA) model is based on the concept of event-driven lightweight agent which is a single-threaded autonomous entity interacting through asynchronous events, executing upon reaction, and capable of migration. In particular, an event-driven lightweight agent is represented by the following tuple: <Id, Beh, DS, TC, EQ>, where, Id is the unique identifier of the agent, Beh is the agent behavior, DS is the data space or world knowledge of the agent, TC is the single thread of control supporting agent execution, and EQ is the event queue containing the incoming events targeting the agent.

The ELDA model relies on the Behavioral, Interaction and Mobility models. The Behavioral model allows for the specification of the agent behavior through the definition of agent states, transitions among states, and agent reactions (i.e. atomic actions attached to transitions). In particular, the agent behavior is defined to react to a specific set of events and a reaction can produce computations, and/or generation of one or more events, or a migration. The Interaction model, which is based on asynchronous events, enables multi-coordination among agents and between agents and non-agent components through the exploitation of multiple coordination structures. The Mobility model is based on a coarse grain strong mobility model which allows for agent transparent migration (both autonomous and passive) and easy programming of the migration points.

These models are founded on the Distilled StateCharts (DSCs) formalism [12] which is derived from the Statecharts formalism [15], a visual formalism that gained valuable success in the Software Engineering community mainly due to its appealing graphical features and the means it offers for the modeling of complex software systems. Statecharts, formerly introduced by Harel, were included in UML [22] and currently are the most used formalism for modeling the behavior of object-oriented reactive systems [16]. In the following sections these models are presented in detail.

2.1 The Behavioral Model

The behavior of ELDA agents is specified through DSCs [12] which are obtained from Statecharts as follows: (i) deriving some basic and advanced characteristics from Statecharts (deriving process), (ii) imposing some constraints on Statecharts (constraining process), and (iii) augmenting Statecharts with some features (augmenting process). In particular:

- *Deriving process*. DSCs derive the following characteristics from Statecharts:
  - *Structure based on a higraph* consisting of rounded rectilinear blobs representing states, linked together with transitions.
  - *Transitions based on the ECA rule*: E[C]/A, when E(vent) occurs and C(ondition) holds, the transition fires and A(ction) is atomically executed.
  - *OR decomposition of states* in hierarchies of states. The enclosing state is called composite state,the nested states are called substates and a state without nested states is called simple state.
  - *Inter-level state transitions* that can originate from or lead to nested states on any level of the hierarchy.
  - *History entrance pseudostates* allow entering the substate which was most recently visited. With respect to the composite state on which the pseudostates appear, *shallow history* indicates that history is applied only at the level of the composite state whereas *deep history* applies the same rule recursively to all levels of the state hierarchy of the composite state.
  - *Default entrances* indicate the substate of a composite state to be entered when a transition targets its border.
  - *Default history entrances* indicate the substate of a composite state to be entered in absence of history.
- *Constraining process*. DSCs impose the following constrains:
  - Each DSC has an enclosing top state.
  - States do not include activity, entry and exit actions. So activity is only carried out under the form of atomic actions labeling transitions.
  - Transitions (apart from default entrances and default history entrances) are always labeled by an event.
  - Each composite state has an initial pseudostate from which the default entrance originates, which can only be labeled by an action.
  - Run-to-completion execution semantics: an event can be processed only if the processing of the previous event has been fully completed. The sequence of operations which starts from fetching an event from the event queue to its complete processing is called *run-to-completion step*.
- *Augmentation process*. DSCs augment Statecharts with the following features:
  - Events are implicitly and asynchronously received through an event queue.
  - To explicitly and asynchronously emit events the action language provides the primitive `generate(<event>(<parameters>))`, where `event` is an event instance and `parameters` is the list of formal parameters of `event` including the event sender, the event target, and (possibly) a list of specific event parameters (see Section 2.2).

- Variables can be declared in each state and inside the actions to form a hierarchical data space.

STATES:
Composite States = {TopState, ACTIVE, ADSC},
Simple States = {INITIATED, TRANSIT, SUSPENDED, WAITING},
Final State=FS, Deep History Pseudostate=DHS,
Initial State=TopState(INITIATED)

TRANSITIONS:

t_1 : TopState(INITIATED) $\xrightarrow{\text{Invoke}}$ TopState(ACTIVE(DHS))
t_2 : TopState(ACTIVE(x)) $\xrightarrow{\text{Move}}$ TopState(TRANSIT)
t_3 : TopState(TRANSIT) $\xrightarrow{\text{Execute}}$ TopState(ACTIVE(DHS))
t_4 : TopState(ACTIVE(x)) $\xrightarrow{\text{Suspend}}$ TopState(SUSPENDED)
t_5 : TopState(SUSPENDED) $\xrightarrow{\text{Resume}}$ TopState(ACTIVE(DHS))
t_6 : TopState(ACTIVE(x)) $\xrightarrow{\text{Wait}}$ TopState(WAITING)
t_7 : TopState(WAITING) $\xrightarrow{\text{WakeUP}}$ TopState(ACTIVE(DHS))
t_8 : TopState(x) $\xrightarrow{\text{Destroy}}$ TopState(FS)
t_9 : TopState(x) $\xrightarrow{\text{Quit}}$ TopState(FS)

Default Entrance of DHS = ADSC;
notation: A(B) = A encloses B, A(x) = any state inside A

Figure 1. The FIPA-based template of the agent behavior.

Each ELDA behavior is forged according to an extended version of the FIPA agent lifecycle template [13] in which the ACTIVE state is always entered through a deep history pseudostate (DHS) to restore the agent execution state after agent migration and, in general, after agent suspension (see Section 2.3). In particular, the ACTIVE state contains the active DSC (ADSC) composite state to which the default entrance of the DHS points. The active agent behavior can be obtained by refining the ADSC. The resulting FIPA template of an ELDA agent is shown in Figure 1 by using a term-rewriting formalism [19].

2.2 The Interaction Model

Interactions of ELDA agents are based on Events which formalize both self-triggering events (Internal events) and requests to or notifications from the local agent server (Management, Coordination and Exception events). Events are further classified into OUT-events which are generated by the agent and always target the local agent server and IN-events which are generated by local agent server and delivered to target agents.

2.2.1 Internal Events

Internal events are generated by agents for proactively driving their behavior. In particular, a generated internal event is placed into the event queue of the generating agent so an internal event can be considered as both OUT and IN.

2.2.2 Management Events

Management events (see Table 1) which include requests to and notifications from the local agent server are further classified with reference to the following functionalities/services: agent lifecycle management, timer setting, and resource access.

The *agent lifecycle management* events allow for the management of agent creation, cloning, migration, suspension and destruction. In particular:

- *agent creation* is supported by the OUT-event CREATE and the IN-event CREATENOTIFY, which respectively formalize the request for the creation of one or more agents and the creation notification (if requested);
- *agent cloning* is enabled by the OUT-event CLONE and the IN-event CLONENOTIFY, which respectively formalize the request for cloning of an agent and the cloning notification (if requested);
- *agent migration* is requested by the OUT-event MOVEREQUEST, which embodies the identifier of the agent to be migrated and the destination agent server location, and is actually carried out after delivering the IN-event MOVE to the agent; after migration the agent execution is resumed through the IN-event EXECUTE (see Section 2.3);
- *agent waiting*, *suspension*, and *quit* are respectively requested through the OUT-events WAITREQUEST, SUSPENDREQUEST, and QUITREQUEST, and actualized through the IN-events WAIT, SUSPEND and QUIT; a waiting agent is waken up through the IN-event WAKEUP whereas a suspended agent is resumed through the IN-event RESUME; finally, an agent is started and destroyed by the agent server through the IN-events INITIATE and DESTROY, respectively.

The *timer setting* events allow for timing agent activities. In particular, the OUT-events CREATETIMER, STARTTIMER, STOPTIMER, RESETTIMER, RELEASETIMER allow for the creation, start, stop, reset and release of timers. A created timer is notified through the IN-event TIMERNOTIFY whereas a timeout event (i.e. an event raised when the timeout expires) is derived from the IN-event TIMEOUTNOTIFY.

The *resource access* events allow for access to the resources of the agent server such as files, console, databases, and sensor/actuators. A resource is requested through the OUT-event RESOURCEREQUEST and granted through the IN-event RESOURCENOTIFY. An input operation on a resource is requested through the OUT-event RESOURCEINPUTREQUEST and the provided input is sent to the agent through the IN-event RESOURCEINPUT; an output operation on a resource is requested through the OUT-event RESOURCEOUTPUT; finally, a resource is released through the RESOURCERELEASE event.

Table 1. Classification of Management events

MANAGEMENT		
Class	*Event Type OUT*	*Event Type IN*
LIFECYCLE	CREATE	CREATENOTIFY
	CLONE	CLONENOTIFY
	MOVEREQUEST	MOVE, EXECUTE
	WAITREQUEST	WAIT, WAKEUP
	SUSPENDREQUEST	SUSPEND, RESUME
	QUITREQUEST	QUIT
		INITIATE
		DESTROY
TIMER	CREATETIMER	TIMERNOTIFY
	STARTTIMER	TIMEOUTNOTIFY
	STOPTIMER	
	RESETTIMER	
	RELEASETIMER	
RESOURCE	RESOURCEREQUEST	RESOURCENOTIFY
	RESOURCEOUTPUT	
	RESOURCEINPUTREQUEST	RESOURCEINPUT
	RESOURCERELEASE	

2.2.3 Coordination Events

Coordination events (see Table 2) enable coordination acts between agents and between agents and non-agent components (e.g. remote objects, web services) according to a specific coordination model. The considered inter-agent coordination models are the Direct (synchronous and asynchronous), the Tuple-based, and the Publish/Subscribe event-based models, whereas the considered agent/non-agent components interaction are a general RMI Object model and a Web Services model. In particular:

- The Direct model is supported by the OUT-event MSGREQUEST and the IN-event MSG for asynchronous message passing, and by the OUT-event RPCREQUEST and the IN-event RPCRESULT for synchronous message passing. MSGREQUEST formalizes a request for sending an asynchronous message and contains the actual message of the MSG type to be sent, whereas MSG contains the message content to be delivered to the target agent. RPCREQUEST formalizes a request for sending a synchronous message and contains the message of the MSG type to be delivered to the target agent along with the back event of the RPCRESULT type. When the receiving agent accomplishes the request, the return value is encapsulated in the RPCRESULT previously specified which is passed to the requesting agent.
- The Linda-like Tuple-based model is enabled by the OUT-events IN, OUT, and RD, and by the IN-event RETURNTUPLE. OUT, IN, and RD formalize the corresponding Linda primitives for insertion, extraction and reading of a tuple, respectively. IN and RD can be either synchronous or asynchronous whereas OUT is only asynchronous. RETURNTUPLE embodies the tuple/s associated to a previously submitted IN or RD event.
- The Publish/Subscribe event-based model is supported by the OUT-events SUBSCRIBE, UNSUBSCRIBE, and PUBLISH, and by the IN-event EVTNOTIFICATION. SUBSCRIBE and UNSUBSCRIBE respectively formalize subscription and unsubscription to given events/topics, PUBLISH embodies a generated event, and EVTNOTIFICATION, which is specified in a previously submitted SUBSCRIBE event, contains an event notification.
- The RMI Object model is supported by the OUT-event RMIINVOKE and the IN-event RMIRETURN for the invocation of methods on non-agent components. RMIINVOKE contains the information needed to invoke a remote method on a remote object along with the back event of the RMIRETURN type which will embody the return value, if any, of the invoked method.
- The Web Services model is supported by the OUT-event SERVICEDISCOVERY, WSDLREQUEST, SERVICEINVOKE and by the IN-event DISCOVERYRESULT, WSDLRESULT e SERVICERESULT. SERVICEDISCOVERY formalizes the service discovery request and the DISCOVERYRESULT, which is sent back to the agent, contains the list of discovered services. WSDLREQUEST formalizes the WSDL request of the chosen service and the corresponding reply is provided through the WSDLRESULT event. SERVICEINVOKE formalizes the service invocation request and a possible return value is sent back through the SERVICERESULT event.

Table 2. Classification of Coordination events

COORDINATION		
Model	*Event Type OUT*	*Event Type IN*
DIRECT	MSGREQUEST	MSG
	RPCREQUEST	RPCRESULT
TUPLE-BASED	RD, IN	RETURNTUPLE
	OUT	
P/S_EVENT-BASED	SUBSCRIBE	
	UNSUBSCRIBE	
	PUBLISH	EVTNOTIFICATION
RMI OBJECT	RMIINVOKE	RMIRETURN
WEBSERVICES	SERVICEDISCOVERY	DISCOVERYRESULT
	WSDLREQUEST	WSDLRESULT
	SERVICEINVOKE	SERVICERESULT

2.2.4 Exception Events

Exception events are modeled as IN-events which are sent from the local agent server to agents to notify the impossibility to execute a service which was requested through the generation of an OUT-event. An exception is defined per each OUT-event and includes the description of the raised exception and its typology. An exception also contains the causingEvent, i.e. the instance of the event which has not been served by the local agent server and caused the exception. The exceptions are organized into a hierarchy which mirrors that of the Management and Coordination OUT-events.

2.3 The Mobility Model

The mobility model of ELDA agents is based on a strong mobility model which allows retaining the agent execution state. With respect to a fine-grain mobility type in which the agent migration can occur on a per-instruction basis the offered strong mobility model is of the coarse-grain type as ELDA agents can migrate on a per-action basis (i.e. after the execution on an action where an action is a set of instructions atomically executed). In particular the migration points of an ELDA agent match with the end of the run-to-completion step (see Section 2.1) and represent the only agent execution points in which it is possible to process Move events (see Section 2.2).

The migration of ELDA agents can be either autonomous (i.e. triggered by the agent itself) or passive (i.e. enforced by the system or induced by other agents) [25]. Specifically, migration points are known by the agent for autonomous migration as they are specified in the agent (DSC-based) behavior through an appropriate definition of states, events and transitions. In case of passive migration, migration points are not known in advance as they are induced by other agents or by the system; then, to obtain a behavior more reactive to migration could be necessary to program an ELDA agent with finer granularity of its actions.

The ELDA migration process is defined as follows. According to the FIPA template (see Figure 1 for the referred transitions), an ELDA agent after receiving the Move event passes into the Transit state (see t2) where it rests until the migration is completed; at the destination location the ELDA agent receives the Execute event, generated by the system, which brings the ELDA agent back into the state it was before the migration (see t3) by retaining the same execution state. State retaining is intrinsic due to the properties of the DSCs, particularly empty states and run-to-completion semantics, and to the structure of the

FIPA-based template, specifically the entrance with deep history in the ACTIVE state (see Section 2.1). In fact, after processing an event the execution state of an ELDA agent is automatically stored into its ACTIVE state so when the ELDA agent migrates it goes into the TRANSIT state without modifying its execution state as no exit action is allowed; after migration it is resumed and the ACTIVE state is re-entered through the deep history pseudostate which allows to set the current state to the state prior to migration without modifying the execution state as no entry action is allowed.

2.4 Distinctive features of the ELDA Model

The distinctive features of the ELDA model, which derive from the characteristics of the *Behavioral*, *Interaction* and *Mobility* models on which it is centered, can be summarized as follows:

- *Visual modeling*. The use of a visual language, based on the *UML* Statecharts [22], for modeling the behavior of ELDA agents, reduces the learning curve for their modeling due to the pervasive exploitation of UML in Industry and Academia, and increases the productivity of designers and programmers as it facilitates application development.
- *Executable specifications*. As the DSC-based behaviors of the ELDA agents are executable according to operational semantics derived from Statecharts, ELDA agents can be effectively verified (e.g. by means of formal methods or simulation) prior to their actual implementation and deployment.
- *Multi-coordination*. Coordination based on multiple models can facilitate application design, improve efficiency, and enable adaptability in dynamic and heterogeneous environments as it allows agents to choose among a variety of different coordination spaces and patterns which best fit their dynamic communication and synchronization needs.
- *Coarse-grain strong mobility*. The ELDA mobility model allows to easily identify and define the migration points of an agent so letting agent designers choose and control the granularity of the offered coarse-grain strong mobility. Moreover this mobility model can be easily implemented through any language which only provides native support to weak mobility like the Java language.

These distinctive features make the ELDA model appropriate for the modeling and implementation of distributed applications characterized by:

- complex computations to be performed on huge data sets (distributed data mining);
- tasks that are inherently parallel and distributed in nature (distributed workflow execution);
- search in huge data repositories, especially in presence of high network latency (distributed information retrieval);
- management and delivery of replicated content (content delivery networks);
- computation in dynamic and resource-constrained environments (wireless sensor networks).

Moreover, the ELDA model can be used to design other agent-based behavioral and interaction models. In particular:

- the ELDA *Behavioral* model based on states and events can support the design of agents ranging from reactive to BDI agents;
- the ELDA *Interaction* model based on multiple coordination paradigms can support the design of agent protocols ranging from simple agent-to-agent interactions to complex multi-agent negotiation protocols possibly based on specific ACL messages.

3. THE ELDA-BASED MAS META-MODEL

Multi agent systems (MASs) based on the ELDA model can be designed through the ELDA meta-model which provides all the modeling elements needed to the design phase. This meta-model is organized in six views correlated as shown in Figure 2:

- *Agent View*, which represents the structure of an ELDA agent and its relationships with the coordination and system spaces.
- *Event View*, which represents the structure of events.
- *SystemSpace View*, which represents the structure of the system space.
- *CoordinationSpace View*, which represents the hierarchy of the coordination spaces.
- *DSC View*, which represents the structure of a DSC.
- *FIPATemplate View*, which represents the structure of the FIPA template of the ELDA behavior.

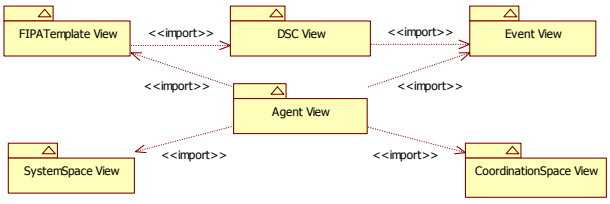


Figure 2. ELDA meta-model: Top-Level View

As shown in the Agent View (see Figure 3) an ELDA agent is composed of a single behavior which is specified through the FIPA template (see Figure 1) whose structure is contained in the FIPATemplate View (not reported for the sake of space). In particular, the FIPA template as well as the ADSC of an ELDA agent is modeled according to the DSC structure shown in the DSC View (see Figure 4). The Agent View also shows that an ELDA agent can interact with the System Space, which provides system services, through the ManagementOUT and ManagementIN events and with the Coordination Space, which provides coordination services, through the CoordinationOUT and CoordinationIN events. These events along with Internal and Exception events, defined in Section 2.2, are included in the Event View (not reported for the sake of space).

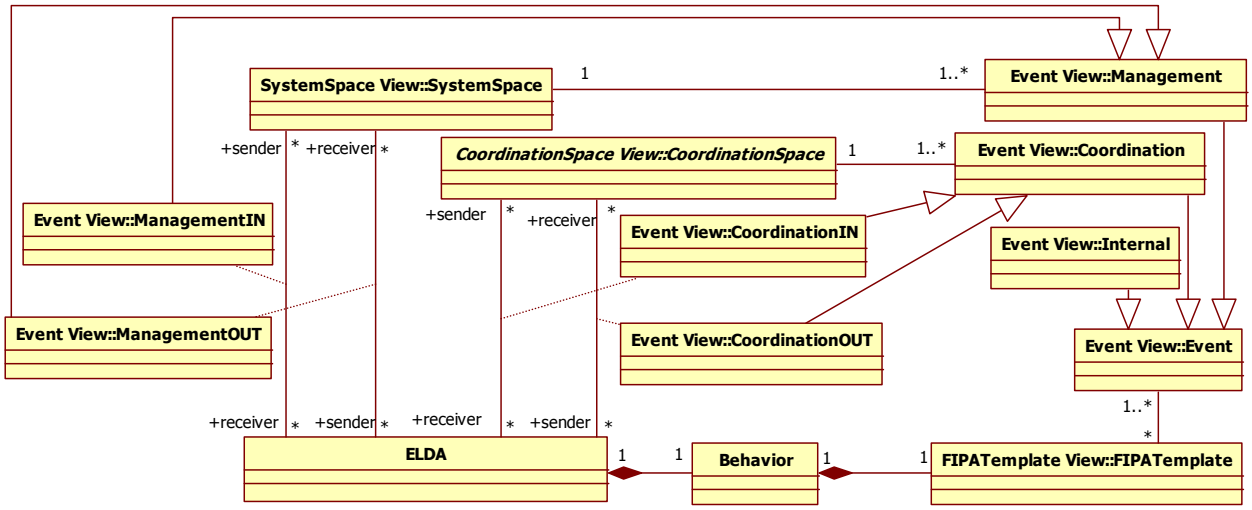


Figure 3. ELDA meta-model: Agent View

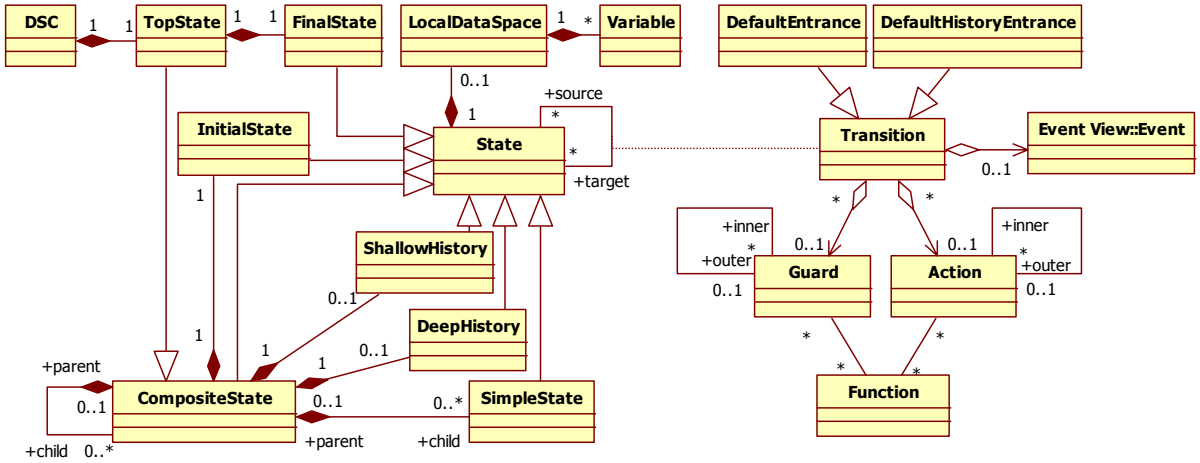


Figure 4. ELDA meta-model: DSC View

As shown in the SystemSpace View (see Figure 5), the System Space is composed of three basic managers, LifeCycleManager, TimerManager, and ResourceManager which handle the *Management* events of the *Lifecycle*, *Timer*, and *Resource* classes, respectively (see Table 1 and Section 2.2). It is worth noting that the ResourceManager provides access services to consoles, databases, files, sensors and other available local resources through associated sub-managers (ConsoleManager, DBManager, FileManager, SensorManager, etc) which handle such specific resources. Moreover, to extend the provided system services new special-purpose managers can be defined by the designer along with the related OUT- and IN-events.

The Coordination Space represents a local or global coordination structure based on a given coordination model through which agents interact. As shown in the CoordinationSpace View (see Figure 6), six coordination spaces are currently defined: DirectSpace (AsynchronousMsgSpace and SynchronousMsgSpace), TupleSpace, PublishSubscribeSpace, RMIObjectSpace, and WebServicesSpace. The interaction with these spaces is regulated by the *Coordination* events reported in Table 2 and described in Section 2.2. New coordination spaces can be introduced by defining new coordination space structures along with their related OUT/IN events.

To actually program ELDA-based MASs, the ELDA meta-model is currently implemented as a set of Java classes constituting the ELDAFramework. To enable rapid prototyping of MASs through the ELDAFramework, an Eclipse-based visual tool named ELDATool [9] is available which allows for visual programming of agent behaviors, graphical definition of events, and automatic generation of code (additional information about the ELDATool and the ELDAFramework can be found in [7]).

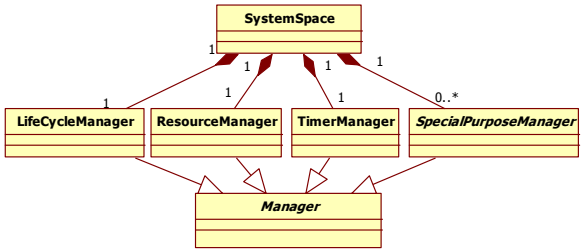


Figure 5. ELDA meta-model: SystemSpace View

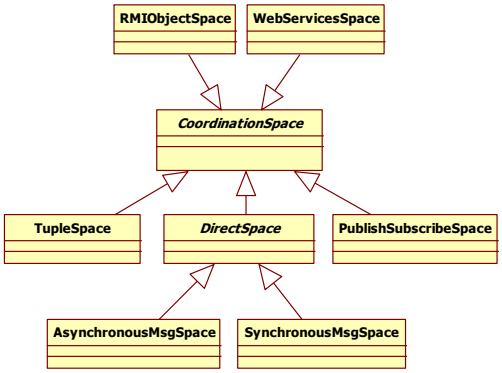


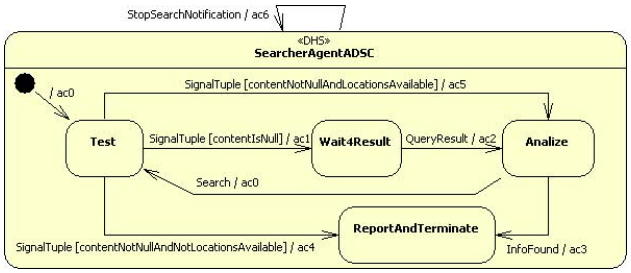
Figure 6. ELDA meta-model: CoordinationSpace View

4. A CASE STUDY

The proposed case study concerns with distributed information retrieval in a distributed computing system. In particular, a *User Agent* searches for specific information over networked federated locations by creating and launching a coordinated set of information *Searcher Agents* onto different locations. As soon as a *Searcher Agent* finds the desired information by locally interacting with distributed *Servant Agents*, stops all the other *Searcher Agents* and finally reports the found information to its owner *User Agent*. Owing to the multi-coordination features provided by the ELDA model, the application design choices are the following:

- *Searcher Agents* locally coordinate using a *local tuple space* to avoid duplicate search on the same location. Before searching for the information on a given location, the *Searcher Agent* checks for the presence of a signaling tuple in the *local tuple space* which is left by another *Searcher Agent* that has already visited the location. If the signaling tuple is not present, the *Searcher Agent* can search for the information; otherwise, it migrates to a new location, if a new location is available, or quits.
- *Searcher Agents* are stopped by exploiting an *event-based Publish/Subscribe* service. This allows a *Searcher Agent* to emit a searching stop event to easily stop all the other agents as soon as it finds the desired information.
- The *Searcher Agent* reports the found information to the *User Agent* through an *asynchronous message* instead of moving to the location of the *User Agent*. This is done to avoid the migration of the agent along with the found information, which would very likely take longer time.

In the following the behavior of the *Searcher Agent* (see Figure 7), programmed through the ELDATool [9], is fully described.




```
private void ac0 (ELDAEvent e) {
 generate(new ELDAEventRd(self(), signalTuple, false,
                  new SignalTuple(self())));
}
private void ac1 (ELDAEvent e) {
 if (firstHost)
  generate(new ELDAEventSubscribe(self(), stopSearch,
          new StopSearchNotification(self()) ));
 firstHost=false; servant=getServant();
 generate(new ELDAEventOut(self(), signalTuple,
          null, false));
 generate(new ELDAEventMSGRequest(self(),
          new QueryMsg(self(), servant, query)));
}
private void ac2 (ELDAEvent e) {
 Object result = ((QueryResult) e).getData();
 info=analize( result );
 if (info.found()) generate(new InfoFound(self()));
 else{
  locations.addLocations(info.getNeighbourLocations());
  if (locations.hasMoreLocations()) ac5(e);
  else ac6(e);
 }
}
private void ac3 (ELDAEvent e) {
 generate(new ELDAEventMSGRequest(self(),
          new Report(self(), owner, info)));
 generate(new ELDAEventPublish(self(), null,
          stopSearch));
 ac6(e);
}
private void ac4 (ELDAEvent e) {
 if (!firstHost)
  generate(new ELDAEventUnsubscribe(self(), stopSearch));
 generate(new ELDAEventQuitRequest(self()));
}
private void ac5 (ELDAEvent e) {
 generate(new ELDAEventMoveRequest(self(),
          locations.nextLocation()));
 generate(new Search(self()));
}
private void ac6 (ELDAEvent e) {
 generate(new ELDAEventUnsubscribe(self(), stopSearch));
 generate(new ELDAEventQuitRequest(self()));
}
```

Figure 7. The *Searcher Agent*: active behavior with the related action code

A *Searcher Agent*, once created and started on a given location, requests the reading of the `signalTuple` in the TupleSpace for checking if another *Searcher Agent* has already visited the location (see action `ac0`). In particular, the reading of the `signalTuple` is asynchronous and the back event to be delivered to the *Searcher Agent* is represented by the SIGNALTUPLE event (derived from RETURNTUPLE) which is handled as follows:

1) If the SIGNALTUPLE event has content null (i.e. no other *Searcher Agent* has searched in this location), the *Searcher Agent* subscribes to the `stopSearch` topic through the

SUBSCRIBE event, if the location visited is the first one (`firstHost`=true), inserts the signalling tuple through the OUT event, and sends to the `servantAgent` a query through the QUERYMSG event (derived from MSG) asking for the searched information (see action `ac1`). In particular, SUBSCRIBE embodies the STOPSEARCHNOTIFICATION event (derived from EVTNOTIFICATION), which is delivered to the *Searcher Agent* when another one finds the searched information.

2) If the SIGNALTUPLE event has content not null and other locations are available, the *Searcher Agent* generates a MOVEREQUEST to the next available location and the internal event SEARCH to keep searching (see action `ac5`). At the new location the migrated *Searcher Agent*, after receiving the internal SEARCH event, enters again into the Test state by executing `ac0`.
3) If the SIGNALTUPLE event has content not null and no other location is available, the *Searcher Agent* unsubscribes from the `stopSearch` topic through the UNSUBSCRIBE event, if `firstHost`=false, and generates a QUITREQUEST event (see action `ac4`).

Consequently to the case 1, as soon as the QUERYRESULT event (derived from MSG) sent by `servantAgent` is delivered to the *Searcher Agent*, its content is analyzed (see action `ac2`): if the searched information is found, the internal event INFOFOUND is generated; otherwise, possible new locations included in the obtained `info` are added to the list of available locations (`locations`) and, if this list has more locations, a MOVEREQUEST to the next available location and the internal event SEARCH are generated; else the *Searcher Agent* unsubscribes from the `stopSearch` topic and quits. When INFOFOUND is received by a *Searcher Agent*, the agent reports to its owner through the REPORT event (derived from MSG), publishes through the PUBLISH event a `stopSearch` topic, unsubscribes from the `stopSearch` topic and quits (see action `ac3`).

In whatever state the *Searcher Agent* is, when it receives STOPSEARCHNOTIFICATION, unsubscribes from the `stopSearch` topic and quits (see action `ac6`).

5. RELATED WORK

Several agent models and frameworks are related to the proposed ELDA model with respect to the three main dimensions of agent modeling: behavior, interaction and mobility. In the following, the comparison is restricted to those agent models which share the following basic features with the ELDA model: lightweight agent architectures, asynchronous agent interaction and state-based programming. In particular, we have considered the agent models on which Jade [3], Bond [4, 20] and Actors [1, 2] are based.

With reference to the agent behavior model, the behavioral model of ELDA agents is based on Distilled StateCharts, Jade offers, among different agent behavior types, an agent behavior (called FSMBehaviour) based on flat finite state machines (FSMs), an add-on of Jade (SmartAgent) [17] provides an extension of the Jade FSMBehaviour (named HSMBehaviour) based on hierarchical finite state machines (HSMs), Bond defines the agent behavior as a multi-plane state machine in which each plane is modeled as an FSM, and Actors are based on agents modeled as active objects with state variables and action methods. The execution semantics of the HSMBehaviour, the ELDA behavior, the Bond agent behavior and the actor behavior is very similar: a message/event triggers the execution of an action; when the action execution is terminated the next available message/event is fetched and processed. Conversely, the execution semantics of the Jade FSMBehaviour is not driven by messages/events but by action completions triggering transitions. The advantages of the ELDA behavior with respect to the other agent behaviors relies on the DSC formalism which (i) overtakes the limited features of the FSMBehaviour by also introducing hierarchy and history and of the HSMBehaviour which do not exploit history, (ii) derives from the well formalized Statecharts formalism whereas a well founded formalization and correlated tools for the Bond multi-plane state machine and the actor behavior are not yet available, (iii) lends itself to be easily supported by a visual tool.

With reference to the agent interaction model, the ELDA model provides the interesting notion of multi-coordination [11] which enables a holistic exploitation of multiple coordination spaces, each based on a different coordination model. This is strategic in the context of open and dynamic environments where agents to fulfill their goal should interact with other agents or with other components through different coordination models. Jade, Bond and Actors are mainly based on asynchronous message passing, even though Bond agents can also interact through synchronous message passing, a tuple space based on the IBM TSpace and a publish/subscribe event model. From the interaction perspective further and interesting related work is represented by the coordination infrastructures [23] such as reactive tuple spaces (e.g. TuCSon), environmental and organizational artifacts. These infrastructures/artifacts can be easily integrated and used as new coordination spaces (see Section 3) for ELDA agents.

With reference to the agent mobility model, Jade, Bond and Actors (in particular the implementation of Actors carried out in the ActorFoudry framework [1]) are based on a weak mobility model [14]. Conversely, the ELDA model is based on strong mobility of the programmable coarse-grain type which enables active and passive migration by simplifying the management code of the agent migration with respect to agents based on weak mobility whose programming is complicated by the explicit management (save and restore) of the agent execution state.

6. CONCLUSION

This paper has proposed the ELDA model and its exemplification through a case study developed by using ELDA-based design methods and programming tools. In particular, the ELDA model provides the following distinctive features: (i) DSC-centered behavioral specification which supports formal-driven and visual-based modeling of the agent behavior so enabling rapid prototyping due to both the graphical representation of the agent behavior and the availability of Statecharts-based formal tools for its validation; (ii) event-based interaction between agents and the hosting local agent server through an easily extensible system space which provides basic and advanced services for agent lifecycle management, timer handling and resource access; (iii) local/remote inter-agent and agents/non-agent-components interaction based on multiple coordination spaces which rely on both already available models (e.g. message passing, Linda-like tuple spaces, publish/subscribe, etc) and models to be purposely

defined; (iv) autonomous and passive agent migration based on strong mobility of the programmable coarse-grain type.

The aforementioned features make the ELDA model more effective than related agent models and frameworks currently available in the literature. Moreover, rapid prototyping of ELDA-based multi-agent systems is enabled by effective visual programming tools (ELDATool) and frameworks (ELDAFramework).

On the basis of the obtained results current research is focused on: (i) finalizing a simulated execution platform enabling functional and non-functional validation of ELDA-based MASs before their deployment stage; (ii) defining a full-fledged methodology supporting the development of ELDA-based MASs from analysis to implementation and validation; this research is based on the experiences gained by using PASSI [6] and GAIA [10] to drive the analysis and design phases in the development of MASs centered on Statecharts-based agents; (iii) formalizing the ELDA model through rewriting logic-based techniques.

7. REFERENCES

[1] Astley, M. 1999. Customization and Composition of Distributed Objects: Policy Management in Distributed Software Architectures. Doctoral Thesis, University of Illinois at Urbana-Champaign.

[2] Astley, M. and Agha, G. 1998. Customization and Composition of Distributed Objects: Middleware Abstractions for Policy Management. In Proceedings of ACM SIGSOFT 6th International Symposium on Foundations of Software Engineering (FSE-6 SIGSOFT'98, Orlando, FL, USA, 1998).

[3] Bellifemine, F., Poggi, and A., Rimassa, G. 2001. Developing multi agent systems with a FIPA-compliant agent framework. Software Practice And Experience 31, 103-128.

[4] Boloni, L. and Marinescu, D.C. 1999. A Multi-Plane State Machine Agent Model. Technical Report CSD-TR-99-027, Computer Science Department, Purdue University.

[5] Braun, P. and Rossak, W. 2005. Mobile Agents: basic concepts, mobility models, & the tracy toolkit. Morgan Kaufmann Pub., Heildelberg, Germany.

[6] Cossentino, M., Fortino, G., Garro, A., Mascillaro, S. and Russo, W. 2008. PASSIM: a simulation-based process for the development of multi-agent systems. Int. J. Agent-Oriented Software Engineering 2(2), 132-170.

[7] ELDATool documentation and software, http://lisdip.deis.unical.it/software/eldatool.

[8] Fortino, G., Frattolillo, F., Russo, W. and Zimeo, E.. 2002. Mobile Active Objects for highly dynamic distributed computing. In Proceedings of IEEE International Parallel and Distributed Processing Symposium (IPDPS), Workshop - Java for Parallel and Distributed Computing (JPDC'02, Fort Lauderdale, FL, Apr. 15-19, 2002).

[9] Fortino, G., Garro A., Mascillaro S., and Russo W. 2007. ELDATool: A Statecharts-based Tool for Prototyping Multi-Agent Systems. In Proceedings of Workshop on Objects and Agents (WOA'07, Genova, IT, Sept. 24-25, 2007), pp. 14-19.

[10] Fortino, G., Garro, A., and Russo, W. 2005. An Integrated Approach for the Development and Validation of Multi Agent Systems. Computer Systems Science & Engineering 20, 4, 94-107.

[11] Fortino, G. and Russo, W. 2005. Multi-coordination of Mobile Agents: a Model and a Component-based Architecture. In Proceedings of 20th Annual ACM Symposium on Applied Computing (SAC'05, Santa Fe, NM, USA, Mar. 13-17, 2005), Special Track on Coordination Models, Languages and Applications, vol. 1, pp. 443-450.

[12] Fortino, G., Russo, W. and Zimeo, E. 2004. Statecharts-based Software Development Process for Mobile Agents. Information and Software Technology 46, 13, 907-921.

[13] Foundation for Intelligent Physical Agents: Agent Management Support for Mobility Specification, DC00087C, 2002/05/10, http://www.fipa.org

[14] Fuggetta, A., Picco, G.P., and Vigna, G. 1998. Understanding Code Mobility. IEEE Trans. on Software Engineering 24, 5, 342-361.

[15] Harel, D. 1987. Statecharts: a visual formalism for complex systems. Science of Computer Programming 8, 231-274.

[16] Harel, D. and Gery, E. 1997. Executable Object Modelling with Statecharts. IEEE Computer 30, 7, 31-42.

[17] Kessler, R., Griss, M., Remick, B. and Delucchi, R. 2004. A hierarchical state machine using JADE behaviours with animation visualization. In Proceedings of Int'l Joint Conference on Autonomous Agents and Multi Agents Systems (AAMAS'04, New York City, NY, USA, July, 2004).

[18] Luck, M., McBurney, P., and Preist, C. 2004. A manifesto for agent technology: towards next generation computing. Autonomous Agents and Multi-Agent Systems 9, 3, 2004, 203-252.

[19] Lilius, J. and Paltor, I. P. 1999. The semantics of UML State Machines. Technical Report N. 273. Turku Centre of Computer Science (TUCS).

[20] Marinescu, D.C. 2002. Internet-based Workflow Management. John Wiley & Sons, Inc., New York.

[21] Nwana, H.S. 1996. Software Agents: an overview. Knowledge Engineering Review 11, 3, 205-244.

[22] Object Management Group.2005. Unified Modelling Language Specification (N. formal/2005-07-05) v. 2.0.

[23] Omicini, A., Ossowsky, S., and Ricci, A. 2004. Coordination infrastructures in the engineering of multi-agent systems. In Proceedings of Methodologies and Software Engineering for Agent Systems. (Kluwer, New York, 2004).

[24] Silva, A.R., Romao, A., Deugo, and D., Mira da Silva, M. 2001. Towards a reference model for surveying mobile agent systems. Autonomous Agent and Multi-Agent Systems 4, 3, 187-231.

[25] Xu, D., Yin, J., Deng, Y., and Ding, J. 2003. A Formal Architectural Model for Logical Agent Mobility. IEEE Trans. Software Eng. 29, 1, 31-45.

[26] Zambonelli, F. and Omicini, A. 2004. Challenges and research directions in agent oriented software engineering. Autonomous Agents and Multi-Agent Systems 9, 3, 253-284.

An Executable Activity Theory Based Framework for Early Requirements Analysis

Rubén Fuentes
ruben@fdi.ucm.es

Jorge J. Gomez-Sanz
jjgomez@sip.ucm.es

Eva Ullán
evah@sip.ucm.es

Facultad de Informática
Universidad Complutense de Madrid
28040 Madrid, Spain

ABSTRACT

Gathering requirements in a domain problem is a challenging task for which several agent-oriented solutions have been devised. The high-level abstraction of agent concepts and their associated semantics eases this elicitation. Nevertheless, this does not ensure that the specification reflects what the customer demands. A frequent validation technique consists in building a prototype so that the customer can appreciate the result of the conversations with the analysts. The creation of such system requires time and money, and thus the industry would appreciate means of reducing these costs. This paper addresses this problem trying to prevent unnecessary developments. The proposal consists in capturing requirements with a framework based on the analysis of human societies named SCAT. This framework is generic enough to be domain independent and uses concepts similar to those in Agent-Oriented Software Engineering. Specifications captured with SCAT are described at low cost, since only general principles are required, and get the extra advantages of being both executable and verifiable. Besides, the resulting instantiation of the framework can be translated later onto a specific methodology if the required transformation rules are developed.

Categories and Subject Descriptors

D.2.1 [**Software Engineering**]: Requirements/Specifications—*elicitation methods, languages, tools*

General Terms

Documentation, Languages, Verification

Keywords

Social properties, Simulation, Formal proof, Activity Theory, Requirements elicitation, Agent-Oriented Software Engineering

1. INTRODUCTION

Defining valid and correct requirement specifications remains a challenging task for software projects. Given a domain problem, a development team may realize too late that the original requirements were wrong since they did not capture what the customer wanted. Any delay in this detection have an important impact in the project, as the later the mistake is found, the greater is the cost of fixing it. The application of agent concepts has provided some aids in the early detection of mistakes in these specifications.

The goal-oriented modelling promotes goals as the key concept in requirements. Lamsweerde reviews some of its approaches in [20], being two relevant examples Tropos [10] and KAOS [4]. Despite of their focus on goals, both of them require producing detailed specifications of the problem in order to evaluate the validity of the requirements. These extended specifications are far beyond the mere use of goals and they are described with some kind of formalism, whose use requires a high-level of expertise. Although either KAOS or Tropos could generate a prototype in the traditional sense, they provide also other alternative means to check the requirements without formal methods. These alternatives demand less effort to produce specifications than the formal options, but they are also less powerful analytical tools.

Non goal-oriented methodologies, like Passi [3] or ADELFE [1], deal with requirements by means of use cases. The way to proceed once use cases have been identified differs from one methodology to another, but all of them require a quite complete multi-agent system specification to be able to validate the requirements by execution. That is, these approaches manually validate the requirements using the prototypes.

Both goal and use case oriented approaches can benefit of automated code generation facilities. Examples of these processes can be found in Executable UML (Executable Unified Modelling Language) [13] and INGENIAS [11]. Executable UML (xUML) is an extension of UML 2.0 intended for model driven development with transformations. As xUML is object-oriented, it must be extended and adapted for its with agent concepts. This is the case of the work with protocols in Agent UML reported in [7]. INGENIAS also permits to quickly develop a system from a partially completed specification. In this case, the departure modelling language is agent-oriented and the prototypes are built by instantiation of code templates with information from models. This kind of model-driven developments reduces the cost of later modifications in the requirements, as an important part of the implementation is automatically made. Nevertheless, it is subject of the same restrictions of the previous approaches, as there are no specialized means of

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

capturing and validating requirements others than carrying out an agent-oriented development.

Related with both the formal analysis and the use of prototypes, this paper proposes validating a requirement specification by means of its execution, the inspection of its traces, and the verification of developer-defined properties. Proposing the direct execution of specifications to ensure their validity is not new. For instance, Gravell and Henderson [12] defend the benefits of executable specifications and recommend complementing them with methods like manual or automatic inspection of the specifications. In order to be useful in this setting, the effort to produce these specifications should imply a lower cost than producing a prototype or a formal specification. Note that this line is different from that of code generation. The xUML or INGENIAS models are transformed to generate prototypes that are the object of the verification while, in this case, checking happens over the original specifications.

The approach in this paper addresses requirement specification with the SCAT (Situation Calculus for Activity Theory) framework for the development of Artificial Societies based on the Activity Theory. The Activity Theory (AT) [21] is a paradigm from the Social Sciences that analyzes human societies focusing on their contextualized activities. An Artificial Society (AS) [18] is a synthetic representation of a society. AS are related with the computational study of societies but not particularly with AT. The operational semantics of SCAT is an AT extension of the Situation Calculus according to the ConGolog [5] implementation. The advantage of this formalism is that its implementation enables executable specifications. A SCAT specification uses high-level AT concepts with semantics close to agent-oriented ones [8]. This allows transferring [9] the acquired knowledge to an agent-oriented methodology, which is better suited for the engineering of the system. When compared with other alternatives, SCAT offers some features that suggest it can trim down the cost of checking requirements through execution:

- SCAT handles only concepts of high level of abstraction. Hence, a SCAT specification is not blurred by low level details.
- The number of elements needed to describe a meaningful specification is low. It suffices declaring the participating entities, dependencies, and what is going to be run to perform validation tests.
- SCAT predicates semantics are intuitive. They are based on common concepts of everyday life used by AT to explain the behaviour of human societies.
- SCAT does not compel to incorporate time reasoning in the specification. This prevents effort on behalf the developer.

These advantages imply in exchange certain tradeoffs in the capabilities of SCAT. The most relevant one is that the level of detail that SCAT can capture is reduced if compared to what other formalisms [4, 10] allow. However, it must be noticed that SCAT is not intended to capture formally the requirements, but to provide an inexpensive way of early detecting mistakes in the specification, so that the real development can start in better conditions.

The rest of the paper is structured as follows. Section 2 briefly presents AT. The SCAT framework is introduced in Sect. 3, which describes the language and its grammar, and Sect. 4, which reports its implementation using ConGolog [5] and some hints about its execution. Section 5 shows an example of requirements elicitation and specification with SCAT. Section 6 describes how a SCAT program can be translated to a software methodology, INGENIAS [11] in this case. INGENIAS was chosen due to its extensive tool support for visual modelling, code generation, and reporting facilities. Related work regarding the execution of specifications as means to increase the knowledge about requirements is discussed in Sect. 7. Finally, we present some conclusions about the approach.

2. ACTIVITY THEORY

The Activity Theory (AT) [21] is a framework for the analysis of human groups focused on their contextualized acts. These acts are called *activities* and constitute the minimal meaningful unit to understand human actions. Their context comprehends the socio-physical environment and their historical development. This section introduces the main AT concepts through examples from the case study of this paper.

The term activity [6] refers to a process intended to transform some object into the outcome that satisfies the needs of the subject of that activity. For instance, a researcher working in a department do research transforming the available knowledge in papers that contain the solution for a given problem. The subject is the participant or group whose agency is chosen as the point of view in the analysis. Potential subjects are the agents, people using a software system, or their societies. In our example, the researcher is the subject. The objects of the activity can be raw materials or mind structures. Our researcher transforms knowledge to generate new knowledge for a paper. The subject's needs that motivate the performance of the activity are their objectives. Some objectives for our researcher can be contributing to mankind knowledge or obtaining merits for a job post.

The relations between subject and object are not direct but mediated by several kinds of elements. The subject always acts over the object through tools. Tools can be physical or symbolic, external or internal, and crystallizes the experience of the group in the process. Following our example, the researcher uses as tools previous experiences writing papers and devices like the laptop or the word processor. The community includes those subjects related in any way (e.g. by use, ownership, or awareness of the existence) with the objects of the activity. Communities in the research activity can be that of the researchers in the same field, the department where the researcher works, or the family and friends, as all of them may have some kind of influence over the activity. A community is moulded by the division of labour and the rules. The division of labour establishes the role of the actors in the community, the power that they hold, or the tasks they are responsible of as related with the transformation process. The rules, norms, and conventions of the communities that constrain the activity are encompassed under the concept rules. These mediating entities and their relations make explicit the influence of the environment in which the activity happens and the historical and social development of it.

All the previous elements (i.e. an activity and its subjects, objects, outcomes, tools, objectives, communities, rules, and divisions of labour) constitute an activity system, that is,

Program	=	Neighbourhood$^+$ Entities$^+$ Relations$^+$ ATDeclaration* Instance$^+$ Universe$^+$
Neighbourhood	=	**neighbourhood**(*ID*, [IDList]).
Entities	=	**entities**(*ID*, IDList, [ElemList]).
ElemList	=	**elem**(*ID*, ATRole) | **elem**(*ID*, ATRole), ElemList
ATRole	=	**activity** | **subject** | **object** | **tool** | **outcome** | **objective** | **community** | **rules** | **divisionOfLabour** | **action** | **operation**
Relations	=	**relations**(*ID*, [IDList], [RelList]).
RelList	=	**rel**(*ID*, ATRel, *ID*, *ID*) | **rel**(*ID*, ATRel, *ID*, *ID*), RelList
ATRel	=	**executedBy** | **transform** | **use** | **produce** | **try** | **accomplishedBy** | **ruledBy** | **organizedBy**
Instance	=	**instance**(*ID*, *ID*, [IDList]).
ATDeclaration	=	**pursue**(*ID*, *ID*). | **decompose**(*ID*, [IDList]). | **duration**(*ID*, *Int*).
Universe	=	**universe**(*ID*, [IDList], [IDList]).
IDList	=	*ID* | *ID*, IDList

Figure 1: SCAT grammar

the context of the activity. Such a context is a mandatory prerequisite to understand truly the whole system and its ruling principles, and to improve its engineering [6].

At the same time, an activity system is made up of, and embedded into, neighbourhoods of nested activities, actions, and operations, all of which could be conceived as separate activity systems depending on the observer's perspective. An action is a task that pursues goals non-relevant individually but that contribute to a primary goal. For instance, writing the paper is part of the research and, although required to complete the activity, does not satisfy by itself the objective of generating knowledge. An operation is a concrete realization of an action for specific circumstances. For instance, writing the paper can be done in a word processor or by hand. In these neighbourhoods, factors that affect the current activity system could be the outcome of other activity systems. Typical examples of these situations are activities that produce the components of the activity systems of other activities or even new activities. For instance, the researcher produce some experimental results that are part of the knowledge transformed as objects in the writing of the paper. For the remaining discussion, the term task stands for activities, actions, and operations when the differences between them are not relevant. The term artifact refers to any concept of an activity system.

3. SCAT GRAMMAR AND INFORMAL SEMANTICS

SCAT (Situation Calculus for Activity Theory) is a modelling, simulation, and verification framework for the analysis of societies. It is intended for practitioners of Social Sciences through a domain specific language [19]. The language allows specifying AS following the AT guidelines seen in Sect. 2. SCAT specifies AS with universes made up of neighbourhoods describing general patterns of behaviour and instances of them that characterize actual behaviours in the system.

Figure 1 presents the main rules of the grammar of SCAT with an EBNF-like notation. Terminal symbols are highlighted using a bold font. All the identifiers of elements are unique in the specification, although an element (but not a relation) can be declared several times with different roles. For instance, the same element can be the outcome of an activity and be used as a tool in other activity. Notice that following the Prolog notation used in ConGolog [5], square brackets do not indicate optional elements but are the terminal symbols for lists. For instance, *[ElemList]* corresponds to a SCAT list where *ElemList* defines its elements separated by commas.

A SCAT program comprehends declarations about several aspects, being some of them optional. A *neighbourhood* defines a parameterized network of activity systems with its participants declared in the *entities* declaration and their dependencies in the *relations* declaration. *ATDeclarations* are primitives adding extra information about individual entities. The *instances* define the actual activity systems by grounding some of the neighbourhoods. The *universe* is the set of instances that are going to be studied along with the actual participants that exists previously to the execution of any activity.

A *neighbourhood* declaration associates an identifier *ID* to a network of linked activity systems. It allows declaring as parameters the entities for the neighbourhood that can be externally set with the argument *IDList*. This way, the neighbourhood declaration can be reused with different actors.

The entities in the neighbourhood need to be associated to the roles they play within its activity systems. The *entities* declaration has as arguments, in order of appearance, the *ID* of the neighbourhood previously defined, a list of parameters that is the same as in the corresponding neighbourhood, and a list of elements. Each *elem* description includes an *ID* for its element, which can match a particular entity of the world or one of the parameters, and the role that it plays in that neighbourhood according to the AT. The different roles are extracted from the AT formalization presented in [8]. They correspond to the entities used to depict an UML-AT diagram. For more information, we invite readers to review the original work.

The *relations* declaration describes the dependencies between entities according to the AT. The first two arguments are the same as for an *entities* declaration. The last argument is the list of relationships among the entities previously defined for the neighbourhood. A *rel* declaration has four arguments: an *ID* for the relationship, the kind of AT relationship, and the entities it connects. The different kinds of relations are extracted as well from [8]. According to that definition, the elements connected by the relationship must play concrete AT roles. For instance, an **executedBy** rela-

tion indicates that a subject is responsible of the execution of an activity. Hence, it declares a capability of the subject. Another example is **produce** that establishes the concrete outcome generated by the execution of an activity.

An *instance* declaration establishes a parameterization of a neighbourhood. Its first two arguments are the identifiers of the instance and the associated neighbourhood, and the third one is a list of values that grounds the parameters of the neighbourhood.

Besides the general declarations about activity systems, SCAT allows making statements about individual elements with *ATDeclarations*. The declaration **pursue** establishes that a subject tries to achieve an objective. A declaration **decompose** describes how an artifact is obtained composing other artifacts (this is the case for activities realized through actions and actions through operations). Finally, **duration** expresses how many units of time take the execution of an operation. Without an explicit declaration, an operation needs one unit of time.

Finally, a *universe* refers to a set of instances whose evolution along time has to be observed. They correspond to scenarios where the different actors are tested. The arguments of this declaration are the identifier of the universe, since there can be several under inspection, its list of instance identifiers, and a list of elements initially available. All the elements existing in the universe must appear in this declaration or be produced by some activity.

4. SCAT IMPLEMENTATION

The predicates in the previous section characterize universes independently of their execution model. This allows certain types of static verification of the systems (like finding contradictions in the specifications in the line of [8]), but it does not allow checking their dynamic behaviour. For this purpose, SCAT needs an operational semantics that determines how universes evolve over time with the execution of tasks and the way in which subjects select the tasks to execute. This evolution has two important requisites. First, interruption of tasks must be allowed at any moment, as subjects can be unable, or do not want, to finish them. Second and to ease the analysis, the semantics must provide a sorted account of the events in the execution over a timescale. The issue of the universe evolution is addressed through an extension of ConGolog [5]; a standard implementation of the choice of tasks is provided following the Rationality Principle [15].

The introduction already points out that the operational semantics of SCAT is built upon ConGolog, which is a Prolog implementation of the Situation Calculus. The Situation Calculus (SC) deals with the evolution of a system, whose initial state is a term `s0`, by means of actions performed over states (the ConGolog term `do(action, state)`). The pre and post conditions of these actions are determined through customized frame axioms. These axioms determine what changes as a consequence of the execution of the actions and also explicitly state what remains unaltered after an action execution.

There are other extensions of SC different from ConGolog. Concretely, Pinto and Reiter [16] or Zimmerbraun and Scherl [22] deal with the integration of time into SC, but the results do not meet the needs of this concrete development. Pinto and Reiter [16] do not permit interruptions in the execution of activities and Zimmerbraun and Scherl [22] do not indicate when events appear over an absolute timescale. The solution applied in SCAT uses flags similar to those of [22] to mark the beginning and the end of the different tasks, as well as absolute marks of time in the different states, similar to those of the Event Calculus [14].

SCAT over ConGolog

Roughly, a universe is regarded as a set of ConGolog systems where multiple subjects simultaneously perform tasks. These tasks affect to their activity systems and have collateral effects in other systems through shared elements. The tasks need a time for their execution whose account demands an explicit representation of time. For this purpose, SCAT codifies its own quanta of time. Every universe includes a special non-declared subject called `cronos` that is responsible of time passing. It executes an atomic activity called `tick` dividing the quanta. A basic operation may begin and end its execution at the same or different quanta depending on its duration. To point out these facts in the SCAT framework, the basic unit of operation becomes the `doing` term:

```
doing(Context, ID, Flag, Subject, Task, Params)
```

It corresponds to the begin or the end of the execution of a task (i.e., AT activity, action, or operation). Its parameters are identifiers of elements in a neighbourhood plus some additional elements for the management of the doings. A `Subject` executes the doing related with a `Task`. The `Flag` takes values among three terms: `shot`, `begin`, and `end`. The value `shot` is only used with environment events that take zero units of times (i.e., they happen in just one quantum); `begin` and `end` indicate the start or the end of the execution of a task respectively, and are used in all the other cases. Each task execution in a run has an identifier `ID`, that is unique for a doing term with a shot flag, but shared by the terms for the same execution of a task with flags begin and end.

Following the SC, the evolution of a universe is described as a sequence of doing terms from an initial state `s0`. With the exception of the subject cronos, whose choices of doings are part of the execution environment, all the other subjects are able to decide when they want to execute a doing. These choices are considered with the `chooseDoing` predicate:

```
chooseDoing(Subject, State, Doing)
```

The predicate takes the identifier of the subject that has to choose a doing at a given setting indicated by `State`. This state is a sequence of doings beginning in `s0`, or the special term `now` to indicate the current state as in ConGolog.

Analysis of Requirements

SCAT programs built upon ConGolog get benefit of the standard resolution algorithm of Prolog, although it affects its scalability. Concretely, they can return every possible result (i.e. sequence of doing terms) that satisfies a given universe. SCAT programs are executed looking for the equilibrium of the society or the possibility of reaching a state that satisfies some constraints. The concept of equilibrium refers to a state where some society features do not change or they change in cycles. Examples are those of a universe completed or one where a subject always performs the same activity without achieving the desired effect in the environment. The second group of tests refers to the existence of

```
neighbourhood(research,
  [Researcher, Paper]).
entities(research, [Researcher, Paper],
  [elem(Researcher, subject),
  elem(write, activity),
  elem(createKnowledge, objective),
  elem(Paper, outcome)]).
relations(research, [Researcher, Paper],
  [rel(r1, try, write, createKnowledge),
  rel(r2, executedBy, write, Researcher),
  rel(r3, produce, write, Paper)]).
duration(write, 2).

neighbourhood(teaching, [Teacher]).
entities(teaching, [Teacher],
  [elem(Teacher, subject),
  elem(instruct, activity),
  elem(transmitKnowledge, objective)]).
relations(teach, [Teacher],
  [rel(r4, try, instruct, transmitKnowledge),
  rel(r5, executedBy, instruct, Teacher)]).

neighbourhood(job, [Employee, Post]).
entities(job, [Employee, Post],
  [elem(Employee, subject),
  elem(apply, activity),
  elem(getJob, objective),
  elem(Post, object),
  elem(rule(unique, [Post]), rules)].
relations(job, [Employee, Post],
  [rel(r6, try, apply, getJob),
  rel(r7, executedBy, apply, Employee),
  rel(r8, transform, apply, Post),
  rel(r9, ruledBy, apply, unique)]).
rule(unique, [Post]) : −
  constraint([], [instance(_, job, [_, Post])]).
rule(exist, [Employee, Post]) : −
  constraint([instance(_, research, [Employee, _])],
  [instance(_, job, [_, Post])]).
decompose(apply, [instruct, ask]).
decompose(apply, [instruct, write, ask]).

pursue(john, createKnowledge).
instance(disia1, research, [john, scatPaper]).
instance(disia2, research, [paul, scatPaper]).
pursue(john, transmitKnowledge).
pursue(paul, transmitKnowledge).
instance(disia3, teaching, [john]).
instance(disia4, teaching, [paul]).
pursue(john, getJob).
pursue(paul, getJob).
instance(fdi1, job, [john, lecturer]).
instance(fdi2, job, [paul, lecturer]).
universe(ucm1, [disia1, disia2],
  [john, paul, scatPaper, createKnowledge]).
universe(ucm2,
  [disia1, disia2, disia3, disia4, fdi1, fdi2],
  [john, paul, scatPaper, lecturer,
  createKnowledge, transmitKnowledge, getJob]).
```

Figure 2: SCAT program for the life in the department.

a given sequence of tasks in the universe that, if performed by their subjects, generates a state where some given properties holds. To perform the previous tests, SCAT provides the predicate `deriveSequence`:

```
deriveSequence(State, Sequence, PosConstraints,
                                NegConstraints)
```

`State` represents a specific initial setting of interest in the considered universe. `PosConstraints` is a list of AT entities (see Sect. 2) or instances of neighbourhoods that have to appear in the evolution of the system from the initial state to the final one. In the case of activities, they have to be executed and completed (satisfying or not their objectives); objectives have to be achieved; the other AT entities have to exist in `State` or being produced by some completed activity of the sequence; instances have to achieve all the objectives in their activity systems. `NegConstraints` is a list of elements that must not appear or be satisfied in the sequence. Last, `Sequence` is a sequence of doings that makes evolve the initial `State` to one that satisfies the constraints. A query about simulation ending just imposes empty lists of constraints, both positive and negative. An example of a query about the reachability of a state is:

```
deriveSequence(s0, Sequence, [elem(think, action),
          elem(explain, action),
          elem(scatPaper, outcome)], []).
```

The query tries to obtain a `Sequence` of doings from the initial state `s0` that leads to a state where the outcome `scatPaper` has been produced, and the actions `think` and `explain` have been completed.

5. EXAMPLES OF SCAT PROGRAMS: LIFE WITHIN THE ACADEMY

This section introduces part of a case study about life in a university whose goal is to construct a simulator for that setting. The current example focuses on how a person gains merits to obtain a job in a university department. There are two alternatives: to do research and teaching; or just teaching. Departments expect that all their applicants adopt the first option as it gives them a higher self-esteem and prestige to their departments. John and Paul are two applicants for a job. John has in his CV teaching and researching activities, and Paul has only teaching. The simulation refers to who will gain the position and how. Figure 2 shows the SCAT program with three neighbourhoods (for doing research, teaching, and applying to a job), and the instances and universes for their analysis.

The neighbourhood *research* explains how and why a researcher writes papers. Essentially, a *researcher* intends to *create knowledge* by an activity called *write* that produces a *Paper*. The activity write has a duration of two units of time. There is also a simpler neighbourhood for *teaching* with just one activity. The last neighbourhood explains how the job can be gained. An *employee* makes merits and then triggers the action *ask*. These actions make up the activity *apply*. The only applicant who complete the activity gets the job. The uniqueness is determined by the rule *unique*.

The two universes at the end of Fig. 2 represent potential scenarios for the system. The universe *ucm1* will tell who actually writes the paper when John and Paul are able to

?- deriveSequence(s0, Sequence, [elem(disia1, instance)], [elem(fdi1, instance)]).

Sequence = do(doing(nil, 12, end, paul, wait, [defining(wait, paul)]),
do(doing(nil, 12, begin, paul, wait, [defining(wait, paul)]),
do(doing(nil, 11, end, john, wait, [defining(wait, john)]),
do(doing(nil, 11, begin, john, wait, [defining(wait, john)]),
do(doing(nil, 10, shot, cronos, tick, [time(3)]),
do(doing(nil, 4, end, paul, apply, [defining(job, paul, lecturer)]),
do(doing(4, 7, end, paul, ask, [defining(job, paul, lecturer)]),
do(doing(2, 9, begin, john, ask, [defining(job, john, lecturer)]),
do(doing(nil, 1, end, john, write, [defining(research, john, scatPaper)]),
do(doing(nil, 8, shot, cronos, tick, [time(2)]),
do(doing(4, 7, begin, paul, ask, [defining(job, paul, lecturer)]),
do(doing(4, 5, end, paul, instruct, [defining(teaching, paul)]),
do(doing(2, 3, end, john, instruct, [defining(teching, john)]),
do(doing(nil, 6, shot, cronos, tick, [time(1)]),
do(doing(4, 5, begin, paul, instruct, [defining(teaching, paul)]),
do(doing(nil, 4, begin, paul, apply, [defining(job, paul, lecturer)]),
do(doing(2, 3, begin, john, instruct, [defining(teaching, john)]),
do(doing(nil, 2, begin, john, apply, [defining(job, john, lecturer)]),
do(doing(nil, 1, begin, john, write, [defining(research, john, scatPaper)]), s0)...)

Figure 3: Trace generated by the execution of the SCAT program

do it. That is, who will successfully complete one of the two instances *disia1* or *disia2*. The initial elements available at *ucm1* are the subjects, a paper to write, and the will of creating knowledge. The second universe, i.e., *ucm2*, mixes several instances where Paul and John write papers, give lectures, and pursue a job. It tests who gets the job as lecturer in the department by completing the instance *fdi1* or *fdi2*. The execution would start from the subjects, a paper to write, the post, the will of creating and transmitting knowledge, and the goal of getting a job.

Figure 3 includes a query about if it is possible that an applicant with research does not obtain the job in the department and a possible answer of the reasoning system. The activity systems use the default behaviour for the choice of doings. The fact that John made research corresponds to the satisfaction of the positive constraint about the instance *disia1*, while that John did not get the job is the negative constraint about the instance *fdi1*. Remember from Sect. 4 that a constraint about an instance is satisfied when all the objectives of its neighbourhoods are satisfied.

In this case, the system finds a solution for the query that also appears in Fig. 3. The subject Paul, who does not pursue the objective *createKnowledge* about doing research, is able to get the job, as one of the alternatives to achieve this goal just requires making teaching before asking the job. Paul begins

```
do(doing(4, 5, begin, paul, instruct,
   [defining(teaching, paul)])
```

in quantum 0 and finishes it in the next quantum, when he also begins the activity *ask*. In quantum 2, he gets the job as he satisfies all the requirements. As only one subject can obtain the job because of the applicable rule *unique* in the instance, the subject John, who teaches and does research, loses the job. The parameters just indicate the actual instantiation of the instances.

Another relevant issue is the flow of time. It appears as the doings of the activity tick carried out by the subject cronos. Finally, the execution is considered as finished when no subject (cronos excluded) can execute an activity different of wait, which does not produce any change in the universe. The subject Paul has satisfied his only objective with the activity *apply*. The subject John has made research but he cannot perform apply to satisfy the other objective because of the rule *unique*. Thus, both of them can only perform *wait* by quantum 4. This situation could be avoided substituting the rule *unique* by *exist*, which demands that the applicant do also research.

6. INTEGRATION WITH AGENT-ORIENTED METHODOLOGIES

Though executable, a SCAT program cannot be considered as a fully developed system. It is a resource to capture and execute activity systems at low cost, whose outcome is knowledge about the system to be. To have a complete running system, this knowledge must be integrated in the process of a software engineering methodology. Given that SCAT works on concepts from AT, choosing an agent-oriented methodology makes sense. The main reason is that the interpretation of intentionality embedded within SCAT resembles to the BDI [2] principles. SCAT considers in its default implementation, although it is not constrained to, subjects that accomplish activities in order to achieve objectives.

Applying the knowledge represented by a SCAT program in an agent-oriented methodology is done by producing a tentative system specification following the notation of the selected methodology. This problem has been already addressed by the authors in [8], where translations are provided from several methodologies to the AT based language UML-AT. Since SCAT borrows many concepts from UML-AT, it is natural to reuse those translation facilities.

Figure 4 shows some examples of the equivalence of concepts between AT and the agent-oriented methodology INGENIAS [11]. The different networks of AT concepts appear on the left and their correspondences in INGENIAS to the right. These AT concepts matches some of the ones pre-

AT	INGENIAS
Activity →/accomplishedBy/→subject →/transform/→object →/produce→outcome →/try/→objective	Agent Model: agent(subject) →/WFResponsible/→task(activity) →/GTPursues/→goal(objective) Tasks and Goals Model: task(activity) →/GTAffects/→fact(object) →/WFProduces/→fact(outcome) →/GTSatisfies/→goal(objective)
Community →/decompose/ →[subject1, subject2]	Organization Model: group(community)→/OHasMember/ →agent(subject1) →agent(subject2)

Figure 4: Some transformation rules from UML-AT to INGENIAS

sented when reviewing SCAT grammar in Sect. 3. These networks are represented as entities connected by relationships represented with arrows labelled within "/". In some cases, the same networks of elements can match several translating networks, both from AT to the agent-oriented methodology or vice versa. This ambiguity problem cannot be solved trivially. It requires human assistance to determine which translation is more appropriate in the current problem. To save some effort, for a concrete network of AT elements and agent-oriented methodology, the selection is made only once.

Following this mapping table, and using the developer intervention to disambiguate terms, the neighbourhood with identifier *job* shown in Fig. 2 can be translated to the INGENIAS diagram in Fig. 5. The transformation is not complete, since some elements from SCAT have no correspondence in INGENIAS, like the duration predicate. In this case, this predicate was translated as TextNotes entities, which are free text annotations included in the diagrams.

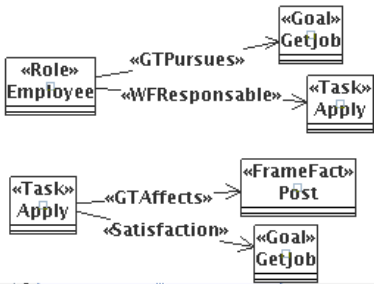


Figure 5: Partial transformation of the SCAT program

As expected, the size of the diagram is small since there is a high compatibility between BDI and AT concepts. To evaluate if the SCAT language really saves effort in the specification, we tried to make the program in Fig. 2 progressing towards a prototype with the INGENIAS Agent Framework (IAF). The first table, labelled as Result in Fig. 6, shows stats about the number of entities and relationships of the initial translation to INGENIAS. Notice that some original elements can be translated several times because different rules from the mapping provide different interpretations for them. This initial translation is not directly executable. It is a valid INGENIAS specification but its inconsistencies and the lack of certain elements make that the IAF cannot generate code for it. So a quick refactoring is made with the initial specification in order to remove all errors. In the initial translation to INGENIAS, the total amount of elements, entities or relationships, summed up to 28 elements. After making it IAF compatible, the specification size increased to 43 elements. Further application of the INGENIAS methodology - the definition of the deployment, external components for paper writing and application deliver, code associated to tasks - lead to 66 elements, an even greater increment. Given that INGENIAS has a quick implementation stage due to its code generation facilities, these figures and effort can be considered as a low limit to build a prototype for agent-oriented methodologies. It should be expected even wider differences with other methodologies less supported by tools.

Finally, remarking that though INGENIAS has been considered in this section, SCAT could be translated to other methodologies due to its roots in AT. Work made in agent model integration with AT in [9] justifies this statement.

7. RELATED WORK

The production of requirements specifications has been considered in the agent literature. This section focuses on two works highly related with early requirements specification, goal operationalization in KAOS and Formal Tropos.

Using the KAOS methodology, executing the requirements specification is studied as the operationalization of goals in [17]. This operationalization adds similar features to the work presented here with some remarks. A SCAT program is executable by itself. However, [17] requires defining additional information, like the preconditions and postconditions of the operations required to achieve the goals. On the other hand, the verification of properties in SCAT is made with a basic searching algorithm, whereas [17] bases on model checking tools. The animation of the execution is made graphically in [17] by means of goal state machines intended to express the operation descriptions. The research in this paper depends on the Prolog facilities to animate the specification. Therefore, the user can inspect the current evaluation of the different variables and the incorporation of new terms to clauses.

Formal Tropos [10] permits precise definitions of systems with time-based predicates about their behaviour. Model checking based tools are used to inspect this formal specifi-

Result	
Name	Times
Agent	2
FrameFact	2
GTAffects	1
GTPursues	1
GTSatisfies	2
Goal	2
Role	4
Task	4
TextNote	2
WFDecomposes	1
WFPlays	4
WFProduces	1
WFResponsable	2

Refactorized	
Name	Times
Agent	2
FrameFact	6
GTModifies	1
GTPursues	1
GTSatisfies	4
Goal	2
Role	4
Task	4
TextNote	2
WFConsumes	6
WFDecomposes	1
WFPlays	4
WFProduces	2
WFResponsable	4

Figure 6: Some transformation statistics for the overall universe

cation in order to verify the satisfaction of properties. On the other side, SCAT does not require including reasoning about time while specifying the system. Time is considered only during the execution. This makes a SCAT program simpler than a Formal Tropos specification. At the same time, it could be argued that Formal Tropos is more expressive than SCAT, at least in what refers to time aspects.

8. CONCLUSIONS

This paper has shown the application of SCAT, a framework for the analysis of activity systems, to early requirements gathering and validation. SCAT allows the quick specification of models of the systems under study. With little information, users are able to run simulations of the expected behaviour according to the AT, and checking hypothesis by means of their models.

Validation of requirements is achieved by checking the results obtained with simulations. These simulations can be used as well to determine if some states can be reached or not. It is assumed that the final validation happens as a result of confirming with customers the appropriateness of the simulation behaviour.

The SCAT specification can be later on translated to other methodologies for development. As an example, this paper has presented some results applied to the INGENIAS methodology. Nevertheless, SCAT can be translated to others, as proves some seminal work made in model integration with AT [9]. This translation has also been useful to show that SCAT specifications can provide simulation and verification with less effort then the required to build a prototype in an agent-oriented methodology, despite of the extensive tool support of INGENIAS.

The current model of SCAT has some limitations. The first one refers to its non-monotonic reasoning due to Prolog, which does not guarantee an answer to a query in finite time and hinders scalability of SCAT programs. To prevent this, model checking techniques are being studied. The idea is to transform a SCAT program into a suitable input for a model checking engine. Secondly, the specification of complex systems is still a tedious work subject to errors, as it needs many predicates. Here, we are working in the integration between SCAT and UML-AT modelling tools. Our purpose is to provide customized graphic modelling tools for SCAT. The last limitation emerges from the trace of the execution of universes. The state of a universe according to the SC is a chain of actions from an initial state. These traces become difficult to study very soon in the execution. New predicates will be added to SCAT to simplify the analysis of states in order to solve common queries. Besides, tools to animate specifications are being considered.

9. ACKNOWLEDGMENTS

This work has been funded by the Spanish Council for Science and Technology under grant TIN2005-08501-C03-01, the Dirección General de Universidades e Investigación de la Consejería de Educación of the Comunidad de Madrid, and the Universidad Complutense de Madrid (Research Group 921354).

10. REFERENCES

[1] C. Bernon, M. Gleizes, and G. Picard. Engineering Adaptive Multi-Agent Systems: The ADELFE Methodology. *Agent-oriented Methodologies*, 2005.

[2] M. E. Bratman. *Intention, Plans, and Practical Reason*. CSLI Publications, 1987.

[3] M. Cossentino and I. ICAR-CNR. From Requirements to Code with the PASSI Methodology. *Agent-oriented Methodologies*, 2005.

[4] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Selected Papers of the Sixth International Workshop on Software Specification and Design table of contents*, pages 3–50, 1993.

[5] G. De Giacomo, Y. Lespérance, and H. Levesque. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence*, 121(1-2):109–169, 2000.

[6] Y. Engeström. *Learning by Expanding: an Activity-Theoretical Approach to Developmental Research*. Orientakonsultit, 1987.

[7] J. O. et al., editor. *Representing Agent Interaction Protocols with Agent UML*, volume 3382 of *LNCS*. Springer, 2005.

[8] R. Fuentes, J. J. Gómez-Sanz, and J. Pavón. Activity theory for the analysis and design of multi-agent

systems. In P. Giorgini, J. P. Müller, and J. Odell, editors, *AOSE*, volume 2935 of *LNCS*, pages 110–122. Springer, 2003.

[9] R. Fuentes-Fernández, J. J. Gómez-Sanz, and J. Pavón. Model integration in agent-oriented development. *IJAOSE*, 1(1):2–27, 2007.

[10] A. Fuxman, L. Liu, J. Mylopoulos, M. Pistore, M. Roveri, and P. Traverso. Specifying and analyzing early requirements in Tropos. *Requirements Engineering*, 9(2):132–150, 2004.

[11] J. J. Gómez-Sanz, R. Fuentes, and J. Pavón. Enabling rapid prototyping using decoupling of code skeletons and code generation process. *InfoComp, Journal of Computer Science*, 2007.

[12] A. Gravell and P. Henderson. Executing formal specifications need not be harmful. *Software Engineering Journal*, 11(2):104–110, Mar 1996.

[13] S. J. Mellor, B. M., and J. I. *Executable UML: A oundation for Modl-Drive Architectures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[14] E. Mueller. Event Calculus Reasoning Through Satisfiability, 2004.

[15] A. Newell. The knowledge level (presidential address). *AI Magazine*, 2(2):1–20, 33, 1980.

[16] J. Pinto and R. Reiter. Reasoning about time in the situation calculus. *Annals of Mathematics and Artificial Intelligence*, 14(2):251–268, 1995.

[17] C. Ponsard, P. Massonet, A. Rifaut, J.-F. Molderez, A. van Lamsweerde, and H. T. Van. Early verification and validation of mission critical systems. *Electr. Notes Theor. Comput. Sci.*, 133:237–254, 2005.

[18] L. Steels. *The Artificial Life Route to Artificial Intelligence*, chapter Building Agents out of Autonomous Behavior Systems. Lawrence Arlbaum Associates, Inc., 1995.

[19] van Deursen A., P. Klint, and J. Visser. Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices*, 35(6):26–36, 2000.

[20] A. van Lamsweerde. Goal-oriented requirements engineering: a guided tour. *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 249–262, 2001.

[21] L. S. Vygotsky. *Mind and Society*. Harvard University Press, 1978.

[22] S. Zimmerbaum and R. B. Scherl. Sensing actions, time, and concurrency in the situation calculus. In C. Castelfranchi and Y. Lespérance, editors, *ATAL*, volume 1986 of *LNCS*, pages 31–45. Springer, 2000.

Issues for Organizational Multiagent Systems Development

Emilia Garcia
Department of Information Systems and Computation
Technical University of Valencia
Valencia, Spain
mgarcia@dsic.upv.es

Estefania Argente
Department of Information Systems and Computation
Technical University of Valencia
Valencia, Spain
eargente@dsic.upv.es

Adriana Giret
Department of Information Systems and Computation
Technical University of Valencia
Valencia, Spain
agiret@dsic.upv.es

Vicente Botti
Department of Information Systems and Computation
Technical University of Valencia
Valencia, Spain
vbotti@dsic.upv.es

ABSTRACT

Organizational multiagent system (OMAS) are rapidly emerging as a powerful paradigm for developing complex systems. Their development process implies specific requirements and software engineering tools. Recently a great number of methods and frameworks to develop OMAS have appeared. Each of them offers different functionality and they have distinct characteristics and perspectives. The main contribution of this paper is a comprehensive list of fundamental OMAS development issues. These issues will be a starting point to define a complete list of OMAS development requirements. From these requirements it could be possible to define an evaluation framework for OMAS development tools in order to help the developers in evaluating OMAS tools and applications.

Keywords

Multi-agent systems, MAS organizations, software engineering

1. INTRODUCTION

Organization multiagent systems (OMAS) have been usefully employed as a paradigm for developing agent systems [7, 17]. One of the advantages of organization development is that systems are modeled with a high level of abstraction, so the conceptual gap between real world and models is reduced. Also this kind of systems offers facilities to implement open systems and heterogeneous member participation [25].

OMAS development implies new requirements on traditional MAS models and technology, including the integration of organizational and individual perspectives and the dynamic adaptation to environmental changes [10]. Therefore, traditional MAS software engineering is not enough to develop OMAS. They need software engineering methods and frameworks that cover organizational concepts and offer the necessary technology. In the last years several OMAS methodologies [4], modeling techniques [22] and platforms [5, 15] have been developed. However, each approach uses its own terminology and offers different functionality, so it is very difficult for developers to choose between one or another. Many designers have doubts about how to translate organizational concepts into final execution entities for an application [6] and what platform and implementation requirements are needed. There is no common agreement on what should be the complete list of these "new" requirements for organization-oriented MAS systems. Moreover, there is no evaluation framework that helps in determining the correctness or completeness of a given OMAS method, tool or execution platform for this kind of system.

Some works like [13, 28] analyze and compare different agent-oriented frameworks to develop MAS. They show a list of the most important features that are needed to develop MAS and propose them to evaluate current methods and MAS development kits. However, this list of features is not complete enough for evaluating OMAS, because it does not deal with organizational concepts and related technology required for developing OMAS. The main goal of this paper is to contribute with a comprehensive list of fundamental OMAS development issues. These issues will be a starting point to define a complete list of OMAS development requirements. From these requirements it could be possible to define an evaluation framework for OMAS development tools in order to help developers in evaluating OMAS tools and applications.

In order to define the fundamental OMAS development issues we have made a detailed study of the state of the art methods and tools for OMAS. Section 2 overviews a set of the most representative ones. In this study we have analyzed their organizational concepts and technology associ-

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

ated. From this study we propose the list of fundamental OMAS development issues detailed in Section 3, we also present a brief discussion on the importance of these issues in OMAS development. Finally, Section 4 presents some conclusions and future work.

2. STATE OF THE ART OMAS DEVELOPMENT TOOLS

In this section we briefly present the most representative set of state of the art development tools for OMAS. We have studied this tools in order to figure out the OMAS fundamental issues required and already provided by them. The goal of this section is just to present an introduction of them. The detailed information on each tool can be found in the referenced literature. In the next section we present the insights of the OMAS development issues related with these tools.

AGR (Agent, Group, Role) [17] is an organizational methodology that is the evolution of the Aalaadin model [16]. In AGR agent, group and role are the primitive concepts. An agent is an active, communicating entity playing (multiple) roles within (several) groups.

The Tropos methodology, in the latest version [20], adopts an organizational viewpoint and explicitly studies the identification of the organizational structure. It proposes using generic multiagent structures that are based on human organizations.

GaiaEXOA (Gaia Extended with Organizational Abstractions) [31] is an evolving extension of the Gaia methodology for designing open MAS.

Ingenias [26] is a methodology for the development of OMAS that is supported by an integrated set of tools, the INGENIAS Development Kit (IDK). It integrates results from research in the area of agent technology with a well-established software development process, which in this case is the Rational Unified Process (RUP).

Electronic Institutions (EI) [1] is a framework that is focused on the design and implementation of an open MAS. It allows the definition of social norms and agent behavioral control. This framework provides several tools to define and implement electronic agent institutions.

Organizational Model for Normative Institutions (Omni) [11] is an integrated framework for norms, structure, interaction and ontologies for modeling organizations in MAS. It is a unification of two other models: the OperA and the HarmonIA framework.

Moise+ is an organizational model for Multi-Agent Systems based on notions like roles, groups, and missions. Moise+ models may be implemented in any platform. Authors present some software to work with Jason, Saci and S-Moise+ [23] which is developed by Moise+ authors.

Jack Teams [27] is an extension to JACK Intelligent Agents platform that provides a team-oriented modelling framework. As Jack, it is based on the BDI agent model and it is implemented in Java.

3. OMAS DEVELOPMENT ISSUES

OMAS software engineering is based on agent-oriented software engineering (AOSE). However, AOSE is not enough for developing this kind of systems as it does not consider the own characteristics of organization concepts and techniques [4].

The aim of this section is to show the most important software engineering issues for OMAS development. Because of the different perspectives and the terminological and conceptual confusion in OMAS [8], the identification of a set of independent, orthogonal features which completely characterize the OMAS development process seems infeasible. The purpose of this section is to make an overall analysis of the needs that arise when developing OMAS, relying more on the concepts than in a specific terminology. These issues are classified into three categories: (i) methodology and modeling language; (ii) development tool; and (iii) execution platform. Following, those categories are deeply discussed.

3.1 Methodology and modeling language

A well-defined methodology greatly helps developers to move from the requirements of the application to the final built system. A complete analysis of the most important features in agent methodologies can be found in [13]. This section is focused on the issues that must be considered when MAS organizational concepts are taken into account. Specifically, methodological features that need to be analyzed are:

- **Development process:** the methodology and the modeling language should be able to extract the organizational requirements; and model the organizational concepts and structure, and also the individual behaviours and objectives of agents. An analysis whether they cover the whole development process is needed. Most approaches, such as AGR, Roadmap and Tropos, only cover the analysis and the design stages. Despite this, approaches, such as Electronic Institutions, Ingenias and Omni, also cover the implementation stage.

- **Methodology concepts and modeling language relationship:** The gap between the concepts of the methodology and the modeling language should be as little as possible, so a complete organizational modeling language is needed [6]. This modeling language can be formal or informal. Most part of current OMAS methodologies offer an informal modeling language that cover completely the concepts and the relationships studied in the methodology. Despite this, there are approaches that introduce formal modeling. For example, all dimensions of Omni have a formal logical semantics, which ensures consistency and possibility of verification of the different aspects of the system.

- **Social patterns**: Patterns describe a problem commonly found in software design configurations and prescribe a flexible solution for the problem, so as to ease the reuse of that solution. This solution is repeatedly applied from one design to another, producing design structures that look quite similar across different applications [12]. They are very useful and reduce the developing time, but only few approaches integrate them. One of them is Tropos that integrates few social patterns in the organizational topologies. In [12] some of these patterns are explained (broker, matchmaker, mediator, monitor, embassy, wrapper, contract-net).

- **Domain dependence**: Methods should offer domain-independent organizational representations and domain-specific frameworks to capture particular relevant organizational characteristics [21].

On the other hand, there are specific organizational concepts that must also be modelled by an OMAS methodology, based on five dimensions [8, 4]: structural, dynamic, functional, normative and environment.

3.1.1 Structural Dimension.

It represents the structure of the organization and all the elements that persist in the organization independently of their members. The structure is defined by its roles, groups, and their dependencies and links. The issues related with the structural dimension are:

- *Topology Selection:* There are many organizational topologies (for example Tropos propose Flat-Structure, Pyramid Style, Chain of Values, Matrix, Structure-in-Five, Co-optation, Joint Venture, Bidding, Arm's Length y Hierarchical Contracting). Which typologies are supported by the methodology and by the modeling language is an important development issue. Most complete methodologies should provide a **guideline** to help developers in the decision of which structure is more appropriated. For example, Tropos offers some guidelines to determinate which is the most appropriate topology for each application. These guidelines are based on general aspects of the performance (fault tolerance, adaptability, coordinability) and they do not take into account organizational components like departments.

- *Composed organization:* This concept considers whether the approach allows the creation of an organization inside another organization. The most part of the methodologies allow it (AGR, GaiaEXOA, Moise+, etc.). Also whether an organization can be created from other pre-existing organizations.

- *Social relationships:* Methodologies might take into account and enable modeling role dependencies and the type of social relationship between the agents and with organizations [10]. Some role dependencies are: heritage, communication, compatibility, coordination, authority or power control. Some of relationship types between agents and organizations are: knowledge links (who can get information about other agents); communication links (who can interact with); authority links (who has control over others); etc.

3.1.2 Dynamic Dimension.

It represents the evolution of the organization, i. e., how agents go inside and outside of the organization and how they change their roles depending on their capabilities and objectives in each moment. The following concepts are needed:

- *Dynamical models:* The dinamicity of the system should be modeled, i. e., the modeling language should be able to represent how agents go in and out of the organization, how they change their roles and how the organizations are created and/or destroyed. Each approach represents the dinamicity of the system in a different way and given different functionality. For example: AGR models how agents change from one group to another. Omni uses contracts to specify when an agent can have a given role. Furthermore, approaches like EI or Moise include an entity which checks that the actions are valid before executing them.

- *Interactions:* The possibility to have open and dynamic systems that allow the interaction of heterogeneous agents is a desirable feature. To get this the definition of a *common ontology* is necessary. Also communicative acts should follow **interaction patterns** or **protocols**. It is necessary to define a set of the **valid illocutions** that agents can exchange and that satisfies a common ontology, a common communication language and knowledge representation language [2]. The most part of the approaches define interaction protocols, but only few of them, such as EI and Ingenias, are able to model a common ontology that completely specify the valid illocutions.

- *Context:* Some approaches specify the situations or context in which the organizations can be during their execution, and how an organization changes between one situation to another. The context specifies the state of the agents, which roles they play and the established norms [1]. Despite this is a very interesting feature, only few approaches like EI and Moise-Inst [18] (an extension of Moise) define contexts.

3.1.3 Functional Dimension.

It represents the organization goals and each component goals. Also it indicates how these goals are achieved, i. e., their decomposition in tasks and plans. Finally, it defines which functionality is offered by the organization and agents. In this case, the most relevant concepts to be taken into account are:

- *Goals:* It is important to not forget that agents which form an organization are autonomous and they have their own goals, behaviours and beliefs in addition to those from the organization. For that reason the approach should be able to model **individual goals**, global or **organizational goals** and how these global objectives are **decomposed** into individual goals in order to be achieved. The most part of the approaches support this feature.

- *Goal decomposition:* The decomposition of goals into tasks and plans should be modeled. Also the most part of the approaches support this feature.

- *Functionality:* The offered behaviour of any OMAS entity should be specified. Currently, there is no approach that specifies which functionality is offered by an organization.

3.1.4 Normative Dimension.

It represents the set of norms that control the organization. Norms facilitate the mechanisms to drive the behaviour of agents, specially in those cases when their behaviour affects other agents [24].

In organization-oriented methodologies two different trends can be observed when comparing several approaches [4]. On the one hand, methods such as Ingenias [26]. Agent-Group-Role [17] detail system roles, groups and relationships but they do not explicitly consider social norms. On the other hand, other methods and frameworks, such as Electronic Institutions [14], are focused on the social norms and explicitly define control policies to establish and reinforce them. Moreover, an extension of AGR to support norms is presented in [19].

Many kinds of norms can be distinguised, but not all methodologies and modeling languages allow the specification of all of them. Some of these types of norms are [24]: (1)Deontic (Obligations, Permissions and Prohibitions); (2) Legislatives, for creating, modifying or revoking norms; (3) Reinforcement, for controlling and penalizing; (4) Rewards.

3.1.5 Environment Dimension.

It represents all the elements of the environment that interact with the organization. The following concepts are needed:

- *Resources*: The mechanism to access resources (read, interact, modify, etc.) should be modeled [31, 26]. Some methodologies like Roadmap, GaiaEXOA and Omni, model the resources, their contextual relationships and how these resources are accessed.

- *Perceptors and Effectors:* Some approaches like [29] model observations as the ability of an entity to perceive the state of (or to receive a signal from) an observed entity by means of perceptors. Perceptor types are used to specify (by means of perceiving acts) the observations that agents can make. The specification of which entities can observe others is modeled with a **perceives dependency**. Different aspects of effecting interactions are modeled analogously, by means of **effectors**, effector types, effecting acts, and effect dependencies.

- *Stakeholders:* They represent the interaction links of the organization with its environment, detailing who takes benefit of the organizational results or who does the organization depend on [30]. Only few approaches model stakeholders, one example is Omni that model the entities that take benefit of the organization or that need it. Also identify which are their objectives and their dependences on the organization.

3.2 Development tool

Development tools are usually divided in two different kind: the specification tool that allows modeling the system, and the implementation tool that allows implementing the final code of the application.

Some tools try to integrate or connect both parts [1]. They add automatic code generation techniques that reduce significantly the implementation time and errors. Despite this, nowadays the gap between the model and the implementation is very high [28] and the most common situation is that a big part of the concepts defined in the models cannot be directly translated to the final implementation. Currently there is no tool that completely integrates the model and the final code.

Only the requirements that appear when the organizational concepts are being added to a MAS are taking into account in this paper. A complete analysis of the most important features of traditional MAS development kits can be found in [13].

3.2.1 Modeling tool.

It should offer modeling facilities by using the selected organizational modeling language. It should cover completely the modeling language and the methodology. It is convenient that some or all the guidelines offered by the methodology are integrated with the modeling tool, but current systems do not offer this facility.

3.2.2 Implementation tool.

The implementation tool should integrate the entire range of modeling features and should easiest the translation from these modeling features to the corresponding execution elements of a given agent execution platform.
Lots of methodologies and software engineering works only offer a theoric analysis and lots of them do not provide any development tool. Two of the most complete development tools are:
EI offers : Islander (a graphical tool that supports the specification and verification of the institutional rules); Simdei (a simulation tool to animate and analyze Islander specifications prior to the deployment stage); aBuilder (an agent development tool which given an Islader specification supports the generation of agent skeletons for that institution); Ameli (a software platform to run institutions); Monitoring tool (a tool which permits the monitoring of EI executions run by Ameli). Nevertheless, there are important lacks in EI tools: the model tool only takes into account organizational concepts; the development tool (aBuilder) only automatically generate agent skeletons and the code agent should be manually completed.
On the other hand, Ingenias is supported by an integrated set of tools, the Ingenias Development Kit (IDK). These tools include an editor to create and modify MAS models, and a set of modules for code generation and verification of properties from these models. This approach covers the entire development process in a basic way, but, it has important lacks in the transformation from models to the final implementation. It only offers a basic generation of code skeletons and does not provide an implementation environment.

3.3 Execution platform

Regarding agent platforms, the most well-known agent platforms (like Jade) offer basic functionalities to agents, such as AMS (Agent Management System) and DF (Directory. Facilitator) services; but designers must implement nearly all organizational features by themselves, like communication constraints imposed by the organization topology [3].

- **Organization representation:** Organizations can be materialized in the following ways [6]: (1) developers does not define the organization structure, although the observer can see an emergent organization; (2) the organization exists as a specified and formalized schema, made by a designer but agents do not known anything about it and do not reason about it; (3) each agent has an internal and local representation of cooperation patterns which it follows when deciding what to do; (4) agents have an explicit representation of the organization which has been defined. Thus an agent is able to reason about it and uses it in order to initiate cooperation with other agents . A good example of the 4 classification may be S-Moise+ or Ameli (EI platform). They have an explicit representation of the organization and both have similar architectures. They follow a three-layer architecture: the application layer that is formed by the autonomous agents of the application; the social layer that ensures that the interac-

tion follow the established norms; the communication layer that allow the communication between agents. Application agents are responsible for achieving organizational goals and using the agent proxy offered by the organization to interact with it.

The implementation tool should integrate the entire range of modeling features and should easiest the translation from these modeling features to the corresponding execution elements of a given agent execution platform.

- **Control mechanisms:** The platform should have control mechanisms that ensure the satisfaction of the organizational constraints. The most common architectures use **middlewares** between agents or between agents and the organization [15]. This middleware forces agents to respect the constraints of the organization. Also, this middleware allows that heterogeneous agents interact and that the organization changes dynamically. This feature is well supported byAmeli and S-Moise+. They act as a middleware between agents and the communication layer. Each agent has associated a proxy agent offered by the organization which control that all the norms and constraints are validated before the interaction.
- **Description of the organizations:** The organization should have an available description in a standard language. It allows external and internal agents to get some information about the organization at run-time. This last feature is not only useful in open systems, but also when considering a reorganization process. A good example of specification of the organization and its benefits can be found in the S-Moise+ platform.
- **AMS and DF extension:** The AMS and the DF offered by traditional MAS platforms should be improve. The AMS should have the information of the existing organizations and their members. The DF should publish the services offered by the agents individually and the services offered by an organization. It should have not only the name of the service offered, but also a description of it to allow open systems.
- **Communication layer:** The kind of communication layer used in the communicative acts is a very important feature. Some of them, such as FIPA-ACL (used by Ameli) and KQML (used by S-Moise+), are more suitable for open systems than TCP/IP, CORBA and RMI [6].
- **Monitorization:** The platform should offer a mechanism to monitorize the state of the agent and of the organizations.
- **Modeling concepts support:** The platform and the programming language should cover all these concepts (explained in Section 3.1). For example, which types of topologies support the platform, which kind of norms, etc. are very interesting features to analyze. No all the platform has a complete modeling concepts support, for example Ameli is focused on the management of rules and norms but do not support the definition of complex topologies. Jack Teams allows the creation of composed "Teams" but it do not offer support for other topologies.
- **API:** The platform should offer an API that allows [9, 5] to create, destroy and modify organizations; consult and modify the organization description, add, query and delete the agents of an organization; send messages to a whole organization, etc.

3.4 Discussion

OMAS development is a very complex task that implies new requirements on all the stages of MAS development process. As is shown, new methodologies, modeling languages, developing environments and platforms are needed. All these development tools requires specific issues to cover organizational characteristics.

The entire development process should be supported by OMAS software engineering tools. Generally, there are big gaps in the development process. The methodology defines concepts that are not supported by the modeling tool. Also, the modeling tool specifies some entities that do not have a corresponding implementation artifact. Moreover, there are few development tools that automatically generate implementation code for a given execution platform.

The study that we present in this paper, shows that there is no development tool that completely cover all the fundamental OMAS development issues. Furthermore, developers do not have any help to choose between one or another development tool. For this reason, we are convinced that there is a fundamental need of a complete evaluation framework in order to get a qualitative and quantative measurement of the completeness or correctness of a given OMAS development tool.

4. CONCLUSIONS AND FUTURE WORKS

In this work we have presented a comprehensive list of OMAS development issues. These issues were defined from a detailed study of the state of the art development methods and taking into account the new characteristics of OMAS related to traditional MAS. It is well known that OMAS are specially suited for open and large systems in which organizational structures are required in order to manage the system complexity. These facts makes compulsory to use Software Engineering principles, methods and techniques in the entire development cycle of this kind of systems. Many research efforts have been developed in this field. In this work we have shown that many of the fundamental OMAS issues are not completely covered by these works. Moreover, there is a fundamental need to have evaluation frameworks in order to get a qualitative and quantative measurement of the completeness or correctness of a given OMAS development tool. With such a framework a developer could select the more appropriate tool for the particular system to develop. The fundamental OMAS development issues presented in this work are a starting point to develop a complete requirement list for OMAS development. From this requirement list it should be possible to define a complete evaluation framework for OMAS development tools.

We are working on the definition of the OMAS development requirement list integrated with traditional MAS requirement, and issues from Service-Oriented MAS. The final goal of our research is the definition of a general qualitative and quantitative evaluation framework for MAS.

5. ACKNOWLEDGEMENTS

This work is partially supported by the PAID-06-07/3191, TIN2006-14630-C03-01 projects and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

6. ADDITIONAL AUTHORS

7. REFERENCES

[1] J. Arcos, M. Esteva, P. Noriega, J. A. Rodríguez, and C. Sierra. Environment Engineering for Multi Agent Systems. *Journal on Engineering Applications of Artificial Intelligence*, 18:191–204, 2005.

[2] J. L. Arcos, P. Noriega, J. A. Rodriguez-Aguilar, and C. Sierra. E4mas through electronic institutions. In D. Weyns, H. Parunak, and F. Michel, editors, *Environments for Multiagent Systems III*, volume 4389 of *Lecture Notes in Artificial Intelligence*, pages 184–202. Springer-Verlag, 2007.

[3] E. Argente, A. Giret, S. Valero, V. Julian, and V. Botti. Survey of MAS Methods and Platforms focusing on organizational concepts. In Vitria, J., Radeva, p. and Aguilo, I, editor, *Recent Advances in Artificial Intelligence Research and Development*, Frontiers in Artificial Intelligence and Applications, pages 309–316, 2004.

[4] E. Argente, V. Julian, and V. Botti. Multi-agent system development based on organizations. *Electronic Notes in Theoretical Computer Science*, 150:55–71, 2006.

[5] E. Argente, J. Palanca, G. Aranda, V. Julian, V. Botti, A. García-Fornes, and A. Espinosa. *Supporting Agent Organizations*, pages 236–245. 2007.

[6] O. Boissier, J. F. Hübner, and J. S. Sichman. Organization oriented programming: From closed to open organizations. *Engineering Societies in the Agents World VII*, 4457/2007:86–105, 2007.

[7] O. Boissier, J. Padget, V. Dignum, G. Lindemann, E. Matson, S. Ossowski, J. Sichman, and J. Vazquez-Salceda. *Coordination, Organizations, Institutions and Norms in Multi-Agent Systems*, volume 3913 of *LNCS (LNAI)*. 2006.

[8] L. Coutinho, J. Sichman, and O. Boissier. Modeling Organization in MAS: A Comparison of Models. In *First Workshop on Software Engineering for Agent-oriented Systems*, pages 1–10, 2005.

[9] N. Criado, E. Argente, V. Julian, and V. Botti. Organizational services for spade agent platform. In *IWPAAMS07*, 2007.

[10] V. Dignum and F. Dignum. A landscape of agent systems for the real world. Technical report 44-cs-2006-061, Institute of Information and Computing Sciences, Utrecht University, 2006.

[11] V. Dignum, J. Vazquez-Salceda, and F. Dignum. Omni: Introducing social structure, norms and ontologies into agent organizations. *LNAI 3346*, 2005.

[12] P. A. Do TT, Kolp M. Social patterns for designing multi-agent systems. In *Proceedings of SEKE-2003*, 2003.

[13] T. Eiter and V. Mascardi. Comparing environments for developing software agents. *AI Commun.*, 15(4):169–197, 2002.

[14] M. Esteva, J. Rodriguez-Aguilar, C. Sierra, J. Arcos, and P. Garcia. *On the Formal Specification of Electronic Institutions*, pages 126–147. Lecture Notes in Artificial Intelligence 1991. Springer-Verlag, 2001.

[15] M. Esteva, B. Rosell, J. A. Rodriguez, and J. L. Arcos. AMELI: An agent-based middleware for electronic institutions. In *In Proc. of AAMAS04*, pages 236–243, 2004.

[16] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS'98)*, pages 128–135. IEEE Computer Society, 1998.

[17] J. Ferber, O. Gutknecht, and F. Michel. From Agents to Organizations: an Organizational View of Multi-Agent Systems. In P. Giorgini, J. Muller, and J. Odell, editors, *Agent-Oriented Software Engineering VI*, volume LNCS 2935 of *Lecture Notes in Computer Science*, pages 214–230. Springer-Verlag, 2004.

[18] B. Gateau, O. Boissier, D. Khadraoui, and E. Dubois. Moiseinst: An organizational model for specifying rights and duties of autonomous agents. *Third European Workshop on Multi-Agent Systems (EUMAS 2005)*, pages 484–485, 2005.

[19] B. Gateau, O. Boissier, D. Khadraoui, and E. Dubois. Moiseinst: An organizational model for specifying rights and duties of autonomous agents. *Environments for Multi-Agent Systems III*, 4389:41–50, 2007.

[20] P. Giorgini, M. Kolp, and J. Mylopoulos. Multi-agent architectures as organizational structures. *Autonomous Agents and Multi-Agent Systems*, 13(1):3–25, 2006.

[21] B. Horling. *Quantitative organizational modeling and design for multi-agent systems.* PhD thesis, 2006.

[22] B. Horling and V. Lesser. Using odml to model multi-agent organizations. In *IAT '05: Proceedings of the IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, 2005.

[23] J. Hubner, J. Sichman, and O. Boissier. S-moise+: A middleware for developing organised multi-agent systems. In *In Proc.Int. Workshop on Organizations in Multi-Agent Systems, from Organizations to Organization Oriented Programming in MAS*, volume 3913 of *LNCS*, pages 64–78, 2006.

[24] F. Lopez, M. Luck, and M. d'Inverno. A normative framework for agent-based systems. *Computational and Mathematical Organization Theory*, 12:227–250, 2006.

[25] X. Mao and E. Yu. Organizational and social concepts in agent oriented software engineering. In *AOSE IV*, volume 3382 of *Lecture Notes in Artificial Intelligence*, pages 184–202, 2005.

[26] J. Pavon, J. Gomez-Sanz, and R. Fuentes. *The INGENIAS Methodology and Tools*, volume chapter IX, page 236Ů276. Henderson-Sellers, 2005.

[27] A. O. Software. Jack intelligent agents: Jack teams manual, release 4.1. 2004.

[28] J. Sudeikat, L. Braubach, A. Pokahr, and W. Lamersdorf. Evaluation of agent-oriented software methodologies examination of the gap between modeling and platform. *AOSE-2004 at AAMAS04*, 2004.

[29] I. Trencansky and R. Cervenka. Agent modelling

language (AML): A comprehensive approach to modelling mas. In *Informatica*, volume 29(4), pages 391–400, 2005.

[30] J. Vazquez-Salceda, V. Dignum, and F. Dignum. Organizing multiagent systems. Technical report uu-cs-2004-015, Institute of Information and Computing Sciences, Utrecht University, 2006.

[31] M. Wooldridge, N. R. Jennings, and D. Kinny. The Gaia Methodology for Agent-Oriented Analisys and Design. *Journal of Autonomous Agents and Multi-Agent Systems*, 15, 2000.

A Verification by Abstraction Framework for organizational Multi-Agent Systems

Nicolas Gaud
Multiagent Systems Group
System and Transport Laboratory
University of Technology of Belfort Montbéliard
90010 Belfort cedex, France
nicolas.gaud@utbm.fr

Vincent Hilaire
Multiagent Systems Group
System and Transport Laboratory
University of Technology of Belfort Montbéliard
90010 Belfort cedex, France
vincent.hilaire@utbm.fr

Stéphane Galland
Multiagent Systems Group
System and Transport Laboratory
University of Technology of Belfort Montbéliard
90010 Belfort cedex, France
stephane.galland@utbm.fr

Abderrafiâa Koukam
Multiagent Systems Group
System and Transport Laboratory
University of Technology of Belfort Montbéliard
90010 Belfort cedex, France
abder.koukam@utbm.fr

Massimo Cossentino
Istituto di Calcolo e Reti ad Alte Prestazioni
Consiglio Nazionale delle Ricerche
Palermo, Italy
cossentino@pa.icar.cnr.it

ABSTRACT

Software agents and multi-agent systems (MAS from now on) are recognized as both abstractions and effective technologies for modelling and building complex distributed applications. However, they are still difficult to engineer. The reason is that when massive number of autonomous components interact it is very difficult to predict that the emergent organizational structure fits the system goals or that the desired functionalities will be fulfilled. Verification approaches try to evaluate whether or not a product, service, or system complies with a specification. However verification approaches are limited by the state-space of the system under study. This paper proposes an approach based upon an organizational framework and specifically the capacity concept which enables to abstract a role know-how and to reduce the state space of the system under study. A formal framework based on multi-formalisms language and the specification approach are presented and illustrated through the specification of a part of the contract net protocol.

Keywords

Holonic and Multi-agent systems, Formal method, Verification, Abstraction

1. INTRODUCTION

Software agents and multi-agent systems (MAS from now on) are recognized as both abstractions and effective technologies for modelling and building complex distributed applications. However, they are still difficult to engineer. When massive number of autonomous components interact it is hard to predict if the emergent organizational structure will fit the system goals or if the desired functionalities will be fulfilled. Verification approaches try to evaluate whether or not a product, service, or system complies with a specification. However verification approaches are limited by the statespace of the system under study. In order to tackle this problem and to verify properties for large systems such as MAS, several techniques may be used. Verification by abstraction is one of these techniques. It consists in finding an abstraction relation and an abstract system that simulates the concrete one and that is manageable for algorithmic verification [5, 24].

The goal of this paper is to present a verification by abstraction approach dedicated to MAS and particularly organizational MAS and Holonic MAS (HMAS). This approach is based upon the abstraction of capacities of roles played by agents within organizations. Organizational approaches are now common within the MAS domain [16, 29, 4, 6] and propose organizational concepts for MAS and HMAS modelling. The framework presented in this paper, namely CRIO, is based upon four main concepts : Capacity, Role, Interaction and Organization. Agents play roles within organizations and interact between themselves. In order to be played by an agent, a role may require some capacities. A capacity is an abstraction of a know-how or a service. It is a very useful concept during the analysis and design of HMAS [26]. The verification by abstraction approach presented here is based upon this concept. Each capacity abstracts a part of role behaviours and separate it from its current implementation.

Each concept of the CRIO framework is specified using a formal language namely OZS [21]. This language composes two formalisms, Object-Z [14] and statecharts [22]. The formal semantics defined for this notation allows the verification of properties by using dedicated software environment such as SAL [9].

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

This paper is organized as follows, section 2 introduces OZS notation. Section 3 presents the CRIO framework, section 4 illustrates the framework and the abstraction approach using the contract net protocol. Eventually, section 5 concludes.

2. BACKGROUND

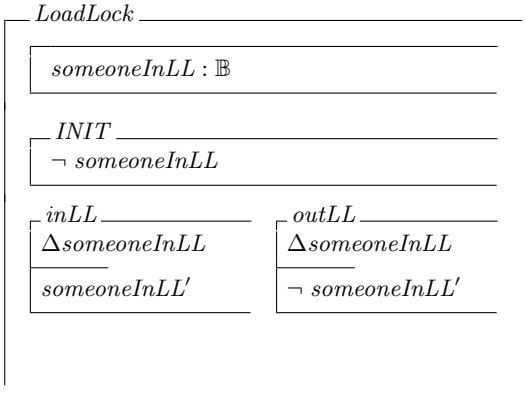
Many specification formalisms can be used to specify entire system but few, if any, are particularly suited to model all aspects of such systems. For large or complex systems, like MAS, the specification may use more than one formalism or extend existing formalism.

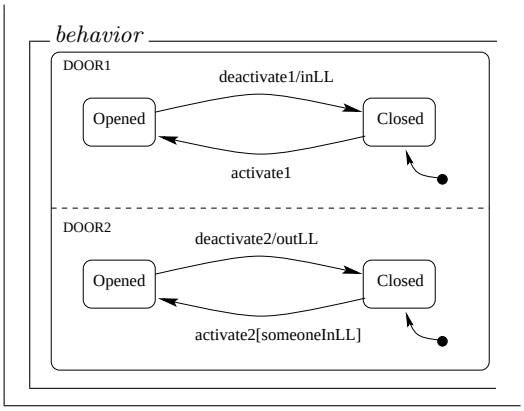
Our approach uses Object-Z to specify the transformational aspects and statecharts to specify the reactive aspects. Object-Z extends Z [25] with object-oriented specification support. The basic construct is the class which encapsulates state schema and operation schemas which may affect its variables.

Statecharts extend finite state automata with constructs for specifying parallelism, nested states and broadcast communication for events. Both languages have constructs which enable refinement of specification. Moreover, statecharts have an operational semantic which allows the execution of a specification.

We introduce a multi-formalisms notation that consists in integrating statecharts in Object-Z classes. The class describes the attributes and operations of the objects. This description is based upon set theory and first order predicates logic. The statechart describes the possible states of the object and events which may change these states. A statechart included in an Object-Z class can use attributes and operations of this class. The sharing mechanism is based on name identity. Moreover, we introduce basic types such as $[Event, Action, Attribute]$. *Event* is the set of events which trigger transitions in statecharts. *Action* is the set of statecharts actions and Object-Z classes operations. *Attribute* is the set of objects attributes.

The *LoadLock* class illustrates the integration of the two formalisms. It specifies a *LoadLock* composed of two doors which states evolve concurrently. Parallelism between the two doors is expressed by the dashed line between $DOOR1$ and $DOOR2$. The first door reacts to $activate1$ and $deactivate1$ events. When someone enters the *LoadLock* he first activates the first door enters the *LoadLock* and deactivates the first door. The transition triggered by $deactivate1$ event executes the *inLL* operation which sets the *someoneInLL* boolean to true. Someone which is between the first and the second door can activate the second door so as to open it.





The notation for attribute modification consists of the modified attributes which belongs to the Δ-list. In any operation sub-schema, attributes before their modification are noted by their names and attributes after the operation are suffixed by '.

The result of the composition of Object-Z and statecharts seems particularly well suited to specify MAS. Indeed, each formalism has constructs which enable complex structures specification. Moreover, aspects such as reactivity and concurrency can be easily dealt with.

3. THE CRIO METAMODEL

The CRIO metamodel presented in figure 1 is the basis of the framework we present in this paper. A more complete description of the metamodel related to a MAS methodology is given in [8]. As this metamodel is aimed at MAS and HMAS we consider that all agents are holons. Simple, non composed, holons are agent in the usual meaning. The metamodel introduces two different levels of abstraction.

The abstract level is concerned with the analysis of a problem in organizational terms. The analysis phase is based on four main concepts : role, interaction, organization and capacity. The adopted definition of role comes from [11]: "*Roles identify the activities and services necessary to achieve social objectives and enable to abstract from the specific individuals that will eventually perform them. From a society design perspective, roles provide the building blocks for agent systems that can perform the role, and from the agent design perspective, roles specify the expectations of the society with respect to the agent's activity in the society*". Moreover, the concept of roles and organization in CRIO is slightly different than the usual one. Indeed, in role is not just a specification of an expected behaviour but a rela building block that will be refined down to an implementation that will be used by agents. However, in order to obtain generic models of organizations, it is required to define a role without making any assumptions on the agent which will play this role. To deal with this issue the concept of capacity was defined [26]. A capacity is a pure description of a know-how and may consider as an interface between the role and associated entities. A role may require that individuals playing it have some specifics capacities to properly behave as defined. The role requires certain skills to define its behavior, which are modeled by capacity. The capacity can then be invoked in one of the tasks that comprise the behavior of the role. In return, an individual must provide a way of realizing all required capacities to play a role. Interactions

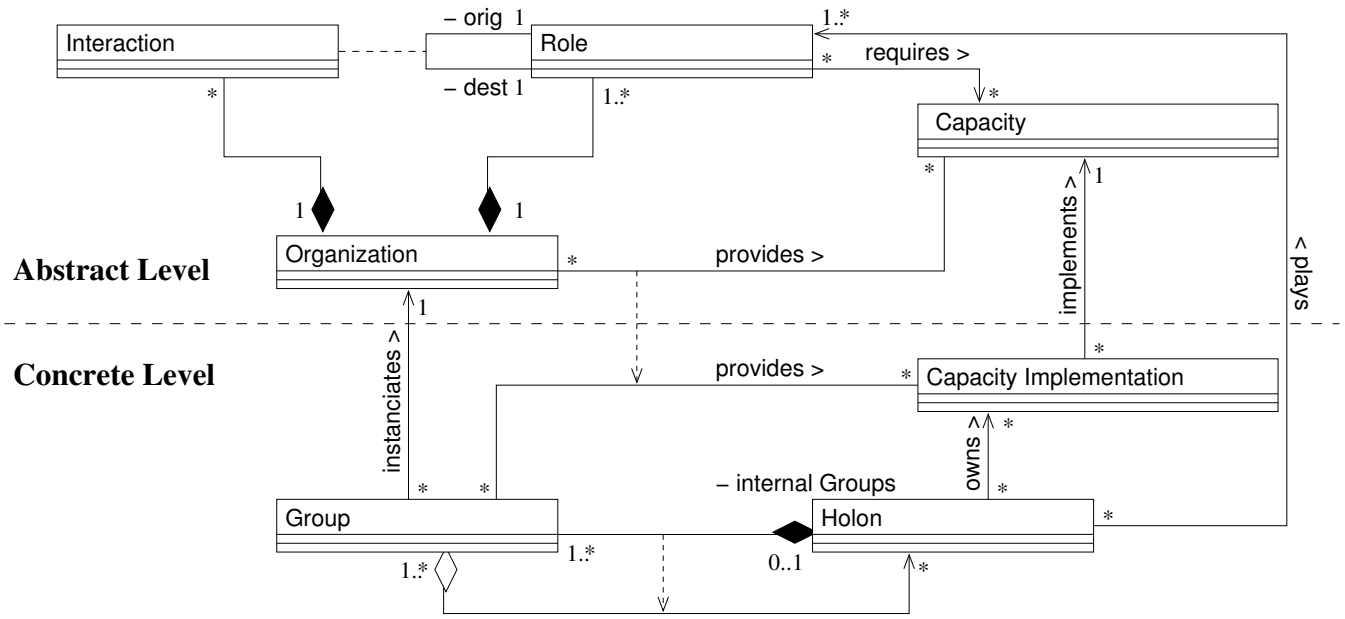


Figure 1: CRIO meta-model

are sequences of actions which consequences have influences over the future behaviours of roles. The context of these interactions is given by an organization. An organization is then a description of a set of roles and their interactions. They define a specific pattern of interaction.

The concrete level describes the solution in terms of groups instantiating organizations. Entities belonging to groups, agents or holons, have capacity implementations required by the played roles. For each concept of this metamodel, a formal description using the OZS notation is given. These specifications define a framework that can be used to formally describe a MAS model. In this paper we will not give the specifications of group and holon which are not necessary for the example. The following types are defined and have to be refined : [*Attribute*], [*Event*] and [*Action*]. These types define respectively the sets of attributes, stimulus and actions of roles. The first concept specified is the role. The role class defines an empty behavior schema and it has to be refined to specify the behavior of the role. It will be specified by using a statechart. A role is also composed of a set of *attributes*, a set of *events* it can react to and a set of *actions*. The role is also defined by a set of *capacities* required by the role and the conditions that have to be met in order to play and leave the role. The constraint states that whatever stimulus (resp. action) of the stimulus set (resp. actions) it must be present on at least one transition of the statechart defining the role behavior. The ρ symbol enables to access to the statechart included. This mechanism enables to manipulate the statechart concepts using the Object-Z language and is described in [20].

Role

behavior

$$
\begin{array}{|l}
\hline
attributes : \mathbb{P}\, Attribute \\
stimulus : \mathbb{P}\, Event \\
actions : \mathbb{P}\, Action \\
requiredCapacities : \mathbb{F}\, Capacity \\
obtainConditions, leaveConditions : Condition \\
\hline
\forall s \in stimulus, \exists e \in behavior.\rho \bullet \\
\quad (\exists t \in e.transitions \bullet t.label.event = s) \\
\forall e \in behavior.\rho \bullet \\
\quad (\forall t \in e.transitions \bullet t.label.action \subseteq actions) \\
\hline
\end{array}
$$

An interaction is specified by a couple of roles, *orig* and *dest*, which are respectively the origin and the destination of the interaction. The roles *orig* and *dest* interact by the way of operations op_1 and op_2. These operations are combined by the $\|$ operator which equates output of op_1 and input of op_2. In order to extend interaction to take into account more than two roles or more complex interactions involving plan exchange one has to inherit from *Interaction*.

Interaction

$$
\begin{array}{|l}
\hline
orig, dest : Role \\
op_1, op_2 : Action \\
\hline
op_1 \in orig.action \\
op_2 \in dest.action \\
\hline
\Diamond (orig.op_1 \| dest.op_2) \\
\hline
\end{array}
$$

Organization

$$
\begin{array}{|l}
\hline
roles : \mathbb{P}\, Role \\
interactions : \mathbb{P}\, Interaction \\
\hline
\forall i \in interactions \bullet \\
\quad (i.orig \in roles \\
\quad \wedge i.dest \in roles) \\
\forall r \in roles \bullet \\
\quad \exists i \in interactions \bullet \\
\qquad r \in i.orig \vee r \in i.dest \\
\hline
\end{array}
$$

An organization is specified by a set of roles and their interactions. Interactions happen between roles of the concerned organization. It is to say that for each interaction of the *interactions* set the roles of the interaction must belong

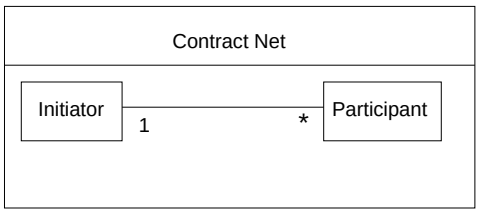


Figure 2: Contract Net organization

to *roles* set of the organization. Moreover, each role must be part of at least one interaction.
The capacity class specifies the concept of capacity. This concept is described by a set of attributes taken as *input* by the capacity and a set of *outputs* produced by the capacity. The *requires* and *ensures* sets of constraints specifies what must be true before the capacity can be called and after the capacity is called. This property is expressed with the constraint that whenever the capacity is called and the *requires* constraints are true then eventually the *ensures* constraint will be true.

Capacity

$inputs : \mathbb{F}\ Attribute$
$outputs : \mathbb{F}\ Attribute$
$requires : \mathbb{F}\ Constraint$
$ensures : \mathbb{F}\ Constraint$

$capacityCall(name) \wedge (\bigwedge_{r \in requires}) \Rightarrow \Diamond(\bigwedge_{e \in ensures} e)$

A capacity implementation is specified by the CapacityImpl class. This class has an *implements* attributes that specifies which capacity it implements. The behaviour schema specifies how the capacity is implemented.

CapacityImpl

$implements : Capacity$

behaviour

With this framework one can specify a MAS or HMAS solution using organizational concepts. The next section describes a part of the contract net protocol specified using this framework.

4. CONTRACT NET EXAMPLE

4.1 Specification

In this section the contract net protocol [27] is specified with the CRIO framework. We adopt the FIPA description of the contract net protocol [17]. The organization describing the contract net protocol is sketched in figure 2. This organization is composed of two roles : initiator and participant. The initiator is the manager who is interested in delegating a task. The participants are the members of the network which can receive the call for proposal and make propositions to the initiator.

The Initiator class specifies the Initiator role. It inherits from the role class of the CRIO framework and adds the following attributes : *proposals* which is a set of Proposal, *best* which is the best proposal selected by the initiator and *criteria* which is a set of functions which help to sort the different proposals. The role requires a capacity which is named *ChooseBestBid*. The behavior of the initiator role specified by the behavior schema consists of three states. The first and by default state is *idle*. Whenever the *taskToBeDistributed* event occurs, it means that initiator will delegate a task, the initiator sends a call for proposal ($cfp(t)$ action which is not described in this paper as it is a very simple communication) for a specific task t and enters the *waitingBids* state. In this state the initiator receives proposals and after a predefined timeout the initiator select among the bidders and send the corresponding answers. It then enters the *waitingResult* state waiting to receive a result from the chosen bidder. After the result is sent or a *timeout* has occurred the initiator returns to *idle* state.

The criterion used by an *Initiator* to choose a proposal are specified by a set of functions. Each function ranks with an integer a proposal as defined by the Criterion type. The *criteria* set is a set of such functions. It specifies a multi-criteria ranking for the proposals.

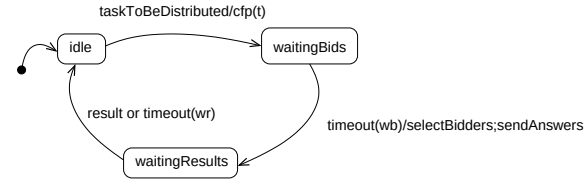
$Criterion == f : Proposal \rightarrow \mathbb{N}$

Initiator Role

$proposals : \mathbb{F}\ Proposal$
$best : Proposal$
$criteria : \mathbb{F}\ Criterion$

$ChooseBestBid \in requiredCapacities$

behaviour



selectBidders

$best' = capacityCall($
$\quad ChooseBestBid(proposals, criterion))$

sendAnswers

The *ChooseBestBid* capacity inherits from the Capacity class. Its *inputs* are a set of proposals and a sequence of criterion, namely criteria. This sequence of criterion can then be specified as $criteria == \text{seq}\ Criterion$. The notation to access to a specific function in this sequence consists in using it rank. For example, criteria(i,b) returns the value of the i^{th} criterion applied to the proposal b. It produces as *output* a proposal, which is the best according to all criterion, among the proposals in input. The proposals input set must not be empty in order to select one. It is the constraint stated in the *requires* set and the *ensures* set states that the best proposal is the best according to *criteria*. It means that it has to minimize the value of each *criterion*.

ChooseBestBid

Capacity

$$
\begin{array}{l}
inputs = \mathbb{F}\, Proposal, criteria \\
outputs = \{best : Proposal\} \\
requires : \{inputs \neq \varnothing\} \\
ensures : \nexists bid : inputs \bullet \\
\quad \exists\, i \in [1..\#criteria] \bullet \\
\quad criteria(i, bid) < criteria(i, best) \\
\quad \wedge\, \forall\, j \in [1..\#criteria] \setminus i \bullet \\
\qquad criteria(j, bid) = criteria(j, best)
\end{array}
$$

The *ChooseBestBidImpl* class specifies a possible implementation of the *ChooseBestBid* capacity. It inherits from *CapacityImpl* and has two attributes: a *proposal* which is the selected best proposal and *bids* which is a set of proposals. The behaviour schema specifies that at first the best proposal is initialized by the first proposal and after that each proposal is compared in sequence with the best found. If it is better than the current best according to the *min* operation it becomes the new best and the capacity implementation iterates through the bids sequence. The *min* operation returns the best proposal among two proposals. *head* and *tail* operations return respectively the first and the rest of the proposals sequence.

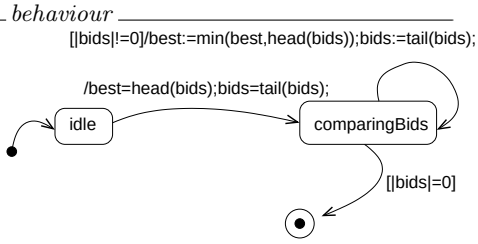
ChooseBestBidImpl

CapacityImpl

$best : Proposal$
$bids : \mathbb{F}\, Proposal$

$implements = \{ChooseBestBid\}$

behaviour



min

$prop_1?, prop_2?, res!Proposal$

$prop_1? \prec prop_2? \Rightarrow res! = prop_1?$
$prop_2? \precsim prop_1? \Rightarrow res! = prop_2?$

head

$sequence? : \text{seq}\, Proposal$
$res! : Proposal$

$res! = sequence?(1)$

tail

$sequence?, res! : \text{seq}\, Proposal$

$sequence? = head(sequence?) \cup res!$

4.2 Verification

The specification of the contract net example was given as input to the SAL environment [9] which is a suite of model checkers and theorem provers. It was compared with the same specification but without the capacity concept. It means that the initiator role integrates the behaviour that choose the best proposal. The SAL environment integrates a path finder which generates traces from the semantics of the specification. The basic behaviour is to generate a ten steps trace of the system. The first part of the table 1 (above the double line) sums up the time in seconds taken by the different computations. The first line corresponds to the construction of the structure used by the path-finder and the second line is the generation of a ten steps trace. One can see that, even on the simple example described in this paper, the version with capacity is more efficient than the version without capacity. Indeed, the version without abstraction is more than four times longer than the one with capacity.

The second part of the table 1 (below the double line) presents the experiment of theorem prooving with induction. The proven property is the first discussed in the end of this section. For the version with abstraction the results given are the sum of the times and numbers of nodes of the role and of the capacity implementation proofs. The construction of the proof structure corresponds to the generation of the data structure used for the proof. The version without capacity is about three times longer than the one with capacity. The proof line is the time taken by the proof, the ratio is about the same as the construction of the proof structure. The last lines is the number of nodes generated for each proof. We have chosen to compare the two specifications of the CNET protocol in time and space in order to support the claim that our approach of verification by abstraction leads to a specification which is more manageable for algorithmic verification than a complete specification. This state-space reduction is obtained by the abstraction of a part of the specification, here the capacity of chosing the best bid. In the version with abstraction this part of the specification is proven apart from the rest and the resulting theorem is taken as input for the verification of the whole system.

We were able to verify the following property for the *ChooseBestBidImpl* class.

$$
\begin{array}{l}
\nexists bid\text{: } inputs \bullet \\
\quad \exists\, i \in [1..\#criteria] \bullet \\
\quad criterion(i, bid) < criterion(i, best) \\
\quad \wedge\, \forall\, j \in [1..\#criteria] \setminus i \bullet \\
\qquad criteria(j, bid) = criteria(j, best)
\end{array}
$$

This property corresponds to the *ensures* set of the *ChooseBestBid* capacity. In order to verify this property we have used a k-induction scheme as described in [10]. It means that we have to prove that the property holds for initial states and is preserved under each transition. The SAL bounded model checker associated with induction proved this property. The *ChooseBestBidImpl* capacity implementation verifies then the *ChooseBestBid* capacity.

Concerning the specification of the *Initiator* role with the *ChooseBestBid* capacity we have proven the following property using the symbolic model checker.

$$behavior.state = waitingBids \Rightarrow \Diamond(behavior.state = idle)$$

We were then able to prove that the given specification satisfies the two properties that an *Initiator* always return

	Version with abstraction	Version without abstraction
Construction of the trace structure	0.15	0.63
Trace generation	0.23	1
Construction of the proof structure	1.36	3.75
Proof	5.16	14.6
Number of nodes	180391	474401

Table 1: Comparisons in time and space

to the idle state and the chosen proposal is allways the best one.

5. RELATED WORKS

Formal methods have been widely used in the MAS field see [1] for a short survey of formal methods in agent oriented software engineering and [19] for a more complete survey and roadmap on this topics. There are two common approaches for verification Model checking and automated theorem proving. Model checking is the process of checking whether a given structure is a model of a given logical formula. It carries out an exhaustive search through the state-space in order to produce a counter-example of the given property. Theorem proving consists in proving automatically or semi-automatically (with human interaction) that a given formula is a logical consequence of the specification.

In [3] model checking techniques were used for verifying multi-agent programs implemented with the AgentSpeak language. This approach is restricted to a subset of the AgentSpeak language, namely AgentSpeak(F), that produces finite state systems. The properties to be verified are expressed with a simplified BDI logic. In [2] the authors propose the use of slicing a technique to reduce the state-space for model checking. The principle of this approach is to simplify the specifications for eliminating details that are not relevant to the property to verify. Again this approach is limited to AgentSpeak program.

In [23] a compositional approach is used for the verification of MAS. Compositional approaches are based on the following principle : if each component behaves correctly in isolation, then it behaves correctly in concert with other components. One has thus to prove each component and then the composition relationship in order to prove properties concerning the whole system. The reported experience only concerns model checking and no evidence are given concerning the efficiency of the proposition.

For theorem proving many approaches use modal logics to specify and make proofs about MAS [18]. Proofs using modal logics theories can be non trivial. Moreover, deducing implementations from such specifications is not an easy task.

In the MAS field there are also some verification approaches which are restricted to a specific feature of agents such as communication protocols, see for example [15].

Organizational approaches are now common in the MAS field see [16, 29, 4, 6] for example. However, few among these approaches use formal methods. OMNI [13] is an integrated framework for norms, structure, interaction and ontologies for modeling organizations in MAS. It was preceded by OperA [12] and HarmonIA [28]. GAIA [29] is an analysis and design methodology for MAS. The main differences between these approaches and the one presented in this paper is that the approach presented in this paper enables the use of software tools to ease proofs and the organizational concepts are expressed in such a way that thay can be refined to an implementation.

6. CONCLUSION

The approach described in this paper considers organizations as blueprints that can be used to define a reusable and modular solution to a problem. The concept of capacity allow the definition of role without making any assumptions on the architecture of the agent that may play them. In this paper we have presented a framework of organizational concepts with a formal semantics which allow the use of abstraction during proofs. The abstraction is based on the capacity concept which abstracts a role know-how. The description of the capacity enables the abstraction of this know-how from the real implementations. The proofs of properties at the organization level are then less complex. This approach enables one to tackle the limitation of formal methods concerning the complexity of verification. These claims are illustrated through the contract net protocol specification example. The use of a random trace generator and a theorem proover on two versions of the contract net specification, one with the abstraction and aone without, shows that the one with abtraction is more tractable.

We have used an organizational framework which seems appropriate for MAS and HMAS modelling.

Future works will deals with the development of a software environment which will help the specifier in his tasks of building and verifying specifications. Moreover, we plan to integrate this formal verification approach within the ASPECS methodological process [7, 8] which enables the analysis and design of MAS and HMAS.

7. ADDITIONAL AUTHORS

8. REFERENCES

[1] Carole Bernon, Massimo Cossentino, and Juan Pavón. An overview of current trends in european AOSE research. *Informatica (Slovenia)*, 29(4):379–390, 2005.

[2] Rafael H. Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. State-space reduction techniques in agent verification. In *AAMAS*, pages 896–903. IEEE Computer Society, 2004.

[3] Rafael H. Bordini, Michael Fisher, Willem Visser, and Michael Wooldridge. Verifying multi-agent programs by model checking. *Autonomous Agents and Multi-Agent Systems*, 12(2):239–256, 2006.

[4] P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, and A. Perini. Tropos: An agent-oriented software development methodology. *Journal of Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[5] Edmund M. Clarke, Orna Grumberg, and David E. Long. Model checking and abstraction. In *POPL*, pages 342–354, 1992.

[6] M. Cossentino. *From Requirements to Code with the PASSI Methodology*, chapter IV, pages 79–106. Idea Group Inc., Hershey, PA, USA., 2005.

[7] Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. A holonic metamodel for agent-oriented analysis and design. In *HoloMAS'07*, 2007.

[8] Massimo Cossentino, Nicolas Gaud, Stéphane Galland, Vincent Hilaire, and Abderrafiâa Koukam. A metamodel and implementation platform for holonic multi-agent systems. In *EUMAS'07*, 2007.

[9] Leonardo de Moura, Sam Owre, Harald Rueß, John Rushby, N. Shankar, Maria Sorea, and Ashish Tiwari. SAL 2. In Rajeev Alur and Doron Peled, editors, *Computer-Aided Verification, CAV 2004*, volume 3114 of *Lecture Notes in Computer Science*, pages 496–500, Boston, MA, July 2004. Springer-Verlag.

[10] Leonardo Mendonça de Moura, Harald Rueß, and Maria Sorea. Bounded model checking and induction: From refutation to verification (extended abstract, category A). In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification, CAV 2003*, volume 2725 of *Lecture Notes in Computer Science*, pages 14–26, Boulder, CO, USA, July 8-12 2003. Springer.

[11] V. Dignum and F. Dignum. Coordinating tasks in agent organizations. or: Can we ask you to read this paper? In *Coordination, Organization, Institutions and Norms COIN@ECAI'06*, 2006.

[12] Virginia Dignum. *A Model for Organizational Interaction: Based on Agents, Founded in Logic*. PhD thesis, Universiteit Utrecht, 2004.

[13] Virginia Dignum, Javier Vázquez-Salceda, and Frank Dignum. OMNI: Introducing social structure, norms and ontologies into agent organizations. In *PROMAS*, volume 3346. Springer, 2004.

[14] Roger Duke, Paul King, Gordon Rose, and Graeme Smith. The Object-Z specification language. Technical report, Software Verification Research Center, Departement of Computer Science, University of Queensland, AUSTRALIA, 1991.

[15] Marc Esteva, Juan A. Rodríguez-Aguilar, Carles Sierra, Pere Garcia, and Josep Lluís Arcos. On the formal specifications of electronic institutions. In Frank Dignum and Carles Sierra, editors, *Agent Mediated Electronic Commerce, The European AgentLink Perspective*, volume 1991 of *Lecture Notes in Computer Science*, pages 126–147. Springer, 2001.

[16] Jacques Ferber, Olivier Gutknecht, and Fabien Michel. From agents to organizations: an organizational view of multi-agent systems. In *Agent-Oriented Software Engineering IV 4th International Workshop, (AOSE-2003@AAMAS 2003)*, volume 2935 of *LNCS*, pages 214–230, Melbourne, Australia, July 2003.

[17] FIPA. Fipa contract net interaction protocol specification. Technical Report SC00029H, FIPA, 2000.

[18] M. Fisher. Temporal semantics for concurrent MetateM. *JSC*, 22(5 and 6):627–648, November/December 1996.

[19] M Fisher, R. H Bordini, B Hirsch, and P. Torroni. Computational logics and agents: A roadmap of current technologies and future trends. *Computational Intelligence*, 1(23):61–91, 2007.

[20] P. Gruer, V. Hilaire, and Abder Koukam. Heterogeneous formal specification based on object-z and state charts: semantics and verification. *Journal of Systems and Software*, 70(1-2):95–105, 2004.

[21] Pablo Gruer, Vincent Hilaire, Abder Koukam, and P. Rovarini. Heterogeneous formal specification based on object-z and statecharts: semantics and verification. *Journal of Systems and Software*, 70(1-2):95–105, 2004.

[22] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[23] Catholijn M. Jonker and Jan Treur. Compositional verification of multi-agent systems: A formal analysis of pro-activeness and reactiveness. *Int. J. Cooperative Inf. Syst*, 11(1-2):51–91, 2002.

[24] C. Loiseaux, S. Graf, J. Sifakis, A. Bouajjani, and S. Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design: An International Journal*, 6(1):11–44, January 1995.

[25] Michael Luck and Mark d'Inverno. A formal framework for agency and autonomy. In Victor Lesser and Les Gasser, editors, *Proceedings of the First International Conference on Multi-Agent Systems*, pages 254–260. AAAI Press, 1995.

[26] Sebastian Rodriguez, Nicolas Gaud, Vincent Hilaire, Stéphane Galland, and Abder Koukam. An analysis and design concept for self-organization in holonic mas. In S Brueckner, S Hassas, M Jelasity, and D Yamins, editors, *Engineering Self-Organising Systems*, number 4335 in LNAI, pages 15–27. Springer, 2007.

[27] R. G. Smith. The contract net protocol : High-level communication and control in a distributed problem solver. *Morgan Kaufmann*, pages 357–366, 1988.

[28] Javier Vázquez-Salceda. The harmonIA framework. *KI*, 19(1):38, 2005.

[29] F. Zambonelli, N. Jennings, and M. Wooldridge. Developing multiagent systems: the gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), 2003.

MAMT: an environment for modeling and implementing mobile agents

Héla HACHICHA
SOIE
ISIMS, Institut Supérieur d'Informatique et de Multimédia, Sfax
hela.hachicha@fsegs.rnu.tn

Adlèn LOUKIL
SOIE
INSAT, Institut National des Sciences Appliquées et de Technologie, Tunis
adlen.loukil@insat.rnu.tn

Khaled GHEDIRA
SOIE
ENSI, Ecole Nationale des Sciences de l'informatique, Tunis
khaled.ghedira@isg.rnu.tn

ABSTRACT
This paper presents an approach to model and to implement mobile agents. This approach is materialized by a UML profile, called MA-UML for modeling mobile agents, and a software development environment that assists the specification, design and implementation stages of the agent system development lifecycle, called MAMT. The MAMT environment provides support for modeling multi-agent systems by using the MA-UML profile.

Keywords
Mobile Agent, UML, AUML, UML profile, mobile agent engineering.

1. INTRODUCTION
Mobile agents are software entities that can migrate autonomously throughout a network from host to host. This means they are not bounded to the platform they begin execution. Mobile Agents are emerging as an alternative programming-concept for the development of distributed applications.

So far, most of the work on the area of mobile agents has been focusing on the technology itself, and the development of agent frameworks to support mobility. However, few works have proposed to model mobile agent-based application and no formalism yet exists to sufficiently specify mobile agents.

In this context and in order to contribute towards to solve this problem, we have been working to propose an approach to model and to implement mobile agents. This approach is materialized first by the MA-UML (Mobile Agent UML) profile, which extends the UML language [1] and the AUML formalism [2]. Second, the proposed approach provides also the MAMT (Mobile Agent Modeling Tool) software CASE Tool, which supports the use of MA-UML profile and the generation of Java code from conceptual diagrams in order to implement mobile agent-based applications.

This paper is structured as follows. Section 2 describes the considerations to model mobile agents and reviews the previous approaches to model mobile-agent applications. In section 3, we describe an overview of the MA-UML profile. Section 4 shows the MAMT environment, describing its architecture and main features. Then, it presents our strategy for mapping conceptual specifications to Java code. Finally section 5 summarizes the paper and offers directions for future work.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

2. MOBILE AGENTS MODELING
2.1 Design considerations
We discuss in this section the basic concepts of mobile agent needed for its specification. According to the literature [3], the mobility of an agent is related to some concepts such as: itinerary, location, move action, remote cloning action, and security. In addition, a Mobile Agent (MA) must contain all of the following models: an agent model, a lifecycle model, a computational model, a security model, a communication model and a navigation model. Also MA must exist in a software environment (called, Mobile Agent Environment) in which it can execute.

Based on these issues, we have identified the following concepts to consider when modeling mobile agent:
Concept 1: the environment that describes entities of a mobile-agent application.
Concept 2: the internal structure which describes MA characteristics.
Concept 3: the itinerary which describes the list of locations to visit or to reside and the set of tasks to be performed in order to complete a specific mission.
Concept 4: the travel schema which describes the travel planning.
Concept 5: the tasks execution planning which describes the mapping of tasks to be performed on different locations.
Concept 6: the interactions which describe the communication acts.
Concept 7: the security mechanisms which describe the security properties needed to protect agent from malicious entities and to protect entities from malicious mobile agents.
Concept 8: the lifecycle model which describes MA's behavior.
Concept 9: the move action (weak or strong) and the cloning action.
Concept 10: the mobility paths which describes the different network nodes.

2.2 Mobile Agent modeling with UML
The literature defines three main categories of approaches to model mobile agents which are: the design pattern approach, the formal approach, and the semi-formal approach.

The semi-formal formalisms can be classified into two classes. The first class is the semi-formal approaches that propose their

own methodologies or extend their agent-oriented methodologies such as: the MaSE methodology [4] was extended to allow the analysis and the design of mobile agent. The m-Gaia [5] proposed extension of the Gaia Agent Oriented software Engineering (AOSE) methodology to model mobile agent systems. The second class of approaches aims to propose some extensions to UML or\and AUML notations.

In our work we are interested to the semi-formal approaches and particularly to formalisms which proposed extensions to UML or\and AUML notations. We mention hereafter some of them and the most relevant for our work.

The MAM-UML [6] profile has introduced new stereotyped classes, packages, relationships, and different tagged values to model entities of a mobile-agent application and their structural relationships. Additionally, it has proposed to describe the itinerary model using the UML interaction diagram (sequence or collaboration). In this diagram, authors have defined new stereotyped messages, using the dependency relationship (move, remote cloning), and constraints which associated to the mobility messages enable to specify why and when an agent moves. Mobile agent's behavior during its lifecycle has modeled using the UML statechart diagram and by defining new stereotyped actions (strong move, weak move, and remote cloning).

Moreover, a variant of an UML activity diagram is used to specify model of activities. Different swimlanes break the diagram into different types of places and different constraints associated to a transition between activities enable to specify why and when an agent moves. Interactions of mobile agents are modeled using the UML collaboration diagram as a template. Finally, the mobility view is modeled using the deployment and component diagrams.

The advantage of the MAM-UML approach is the covering of the most mobile agent concepts in the three phases of the development process (analyze, design and implementation). But, some limits can be identified in this work. In fact, the MAM-UML itinerary model describes the agent's mobility between locations and the interactions of MA with entities (stationary agents and resources). However, in the literature [7] "the itinerary typically consists of a list of tasks to be executed sequentially and the locations where the tasks are to be performed".

Also the travel planning designed in itinerary model is static which not considers the environment changes that can occurs and make agent incapable to reach some locations (e.g. the mobile agent platform on the destination address is not operational, the machine that the agent is moving from is isolated from the rest of the network). Moreover, the tasks execution planning designed with UML activity diagram is not dynamic (with a static set of locations). Finally, authors have not considered the design of mobile agent internal structure, security mechanisms, and mobility path.

Klein et al. [8] have proposed extensions to UML class diagram with new stereotypes and tagged values to model entities involved in a mobile agent application. In addition, stereotyped actions «move» and «remote execution» are introduced in the UML statechart diagram to show the relationship between agent state change and agent location change. Moreover, authors have proposed to specify agent mobility by extending the UML sequence diagram with the new elements and new stereotyped messages (move, remote execution, and clone).

This work represents an initial approach on modeling mobile agent and not considered the most mobile agent concepts.

Mouratidis et al. [9] have introduced extensions to the UML deployment and activity diagrams. These extensions have been integrated to the AUML formalism. The AUML deployment diagram allows developers to specify mobile agent, origin location, destination location, and static aspect of mobility paths. The AUML activity diagram allows developers to specify the dynamic aspect of the mobility path: the sequence of the movement, the detailed mobility path, and the decisions that drive the choice of particular intermediate nodes. The activity nodes model plan, while the transitions model events.

This work has the advantage of the modeling of the mobility paths of agent which considers the environment changes and the decisions which MA will take during its travel. Also, it allows specifying the travel planning of MA. However, this represents a static planning. Moreover, this approach not considers the other concepts of MA.

Kang et al. [10] have proposed extensions to activity diagram in UML 1.5 and UML 2.0 in order to model the dynamic behavior of mobile agents. Authors have introduced the new stereotype «host» with a parameter for a swimlane, which represents the location with a unique name (address). The activity "Go" is also introduced to model agent's movements. Authors have also modeled the exception handling which can occur and cause the failing to access to the destinations hosts.

This extended UML activity diagram allows to specify the tasks execution planning of mobile agent in different hosts. However, this represents a static planning. This work also has not considered the other mobile agent concepts.

Kusek et al. [11] have introduced extensions to UML sequence diagram to model agent mobility, current location, and location of agent creation. Four variants uses of sequence diagram are proposed to model agent mobility. In these diagrams different stereotypes are added: stereotype «agent» to model agent, stereotype «at» to model current location, and stereotype «move» to model agent's movements.

In these diagrams, vertical lines represent both places and instances of agents and arrows between these lines represent movements of agents between places. This makes diagrams so complex and particularly with an important number of places.

2.3 Discussion

All these approaches previously described are useful and interesting contributions. However, no formalism yet exists to sufficiently specify the basic concepts of MA described in section 2.1. Table 1 summarizes the principal concepts of mobile agent and the contributions of some existing approaches.

Table 1. Contributions of some existing approaches

	MAM-UML [6]	Mobile UML [8]	AUML [9]	Kang et al. [10]	Kusek et al. [11]
Environment	×	×			
MA Internal structure					
Itinerary	×	×			
Travel schema	×	×	×		×
Task planning	×			×	
Interaction	×		×		×
Lifecycle	×	×	×		
MA security					
Mobility Path			×		

After examining the presented approaches, some deficiencies can be identified to model some concepts; we summarize some limits in the following points:

The internal structure of MA is not well specified by these approaches.

The security properties needed for mobility is not addressed.

The travel planning of mobile agent modeled by some of these approaches [8] [11] [6] [9] is not flexible and not dynamic (with a static set of locations). In fact, if MA cannot reach some locations, then their related tasks cannot be performed. A mobile agent travel planning is considered one of the most important techniques for completing a given task efficiently. However, a static planning may not be the best approach in real network environments. So it is necessary to model a flexible and dynamic travel planning, which considers the environment changes.

The tasks execution planning modeled by some of these approaches [6] [10], using the activity diagram, is not dynamic (with a static set of locations). In fact, if MA cannot reach some locations for ever reason, then the list of tasks relevant to these locations cannot be performed and MA cannot reach its mission. So it is necessary to model a dynamic tasks execution planning.

In order to contribute towards to surmount some insufficiencies, we have been working to propose additional extensions to UML and AUML notations. This represents the MA-UML profile, presented in the next section.

3. THE PROPOSED MA-UML PROFILE

It is to be notice that our works focus on the modeling of mobile agent applications when there are only two kinds of entities: mobile agents and static locations; and they are not well suited for mobile computing modeling (laptops, mobile phones, PDAs).

There are seven diagrams in MA-UML profile classified into static and dynamic diagrams. MA-UML extends the UML and the AUML class diagrams and defines three new diagrams to model mobile agent structural / static aspects, which are environment diagram, mobile agent diagram and itinerary diagram. MA-UML extends the UML statechart diagram, the UML activity diagram, and the AUML sequence diagram to model mobile agent dynamic aspects, which are: lifecycle diagram, mobile agent activity diagram, mobile agent sequence diagram and navigation diagram. Figure 1 illustrates the relationships between these diagrams.

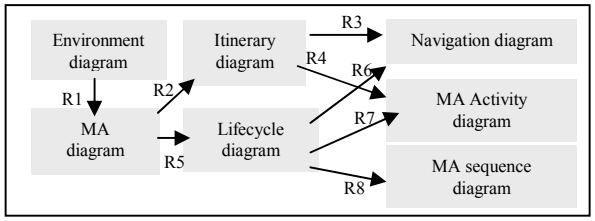


Figure 1. Inter-diagrams Relationships.

3.1 Environment Diagram

The purpose of this diagram is to model all entities involved in a mobile-agent application, their properties and their structural relationships. Typically, this application involves several agents (stationary and mobile) that interact and communicate, playing different roles. Also, it involves one or several region(s), a number of mobile-agent systems, some places, and several resources.

In order to model these entities and their structural relationships, we propose to introduce new stereotyped UML classes («place», «resource»), stereotyped AUML classes («mobile agent», «stationary agent»), stereotyped packages («region», «m-agent-system»), stereotyped associations («communicate», «reside», «manipulate», «home»), and new graphical notations. This extended diagram is called *Environment Diagram*. To illustrate these extensions, we present as an example the environment diagram designed for electronic commerce application (figure2).

Figure 2 illustrates an environment diagram for a simple mobile-agent application. In such application, a SearcherAgent (mobile agent) moves between different places provided by two different mobile-agent systems, in the context of a region. The InterfaceAgent (stationary agent) is responsible for the initialization of SearcherAgent with its mission. The SecurityAgent is responsible for authenticating the SercherAgent. The DataBase represents the resource manipulated by mobile agent.

After modeling the different entities, each mobile agent specified in this diagram must be specified in the mobile agent diagram in order to specify its internal structural and its characteristics (R1, Figure 1: Environment diagram → MA diagram).

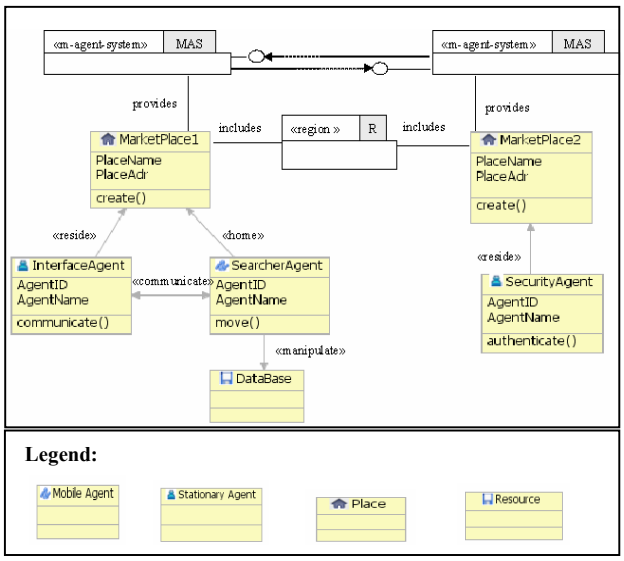


Figure 2. The environment diagram of a mobile-agent application

3.2 Mobile Agent Diagram

This diagram is responsible for modeling the internal structure and the characteristics of a Mobile Agent (MA). The properties of the internal structure of a MA depend on the requirements of applications (electronic commerce, telecommunication, etc.). But we believe that several properties must be identified and are available and needed for each application. Based on the literature [3], we propose to define additional properties in the internal structure of the stationary agent relevant to the mobile agent (e.g. authentication, history, itinerary).

In order to specify the properties relevant to the MA, we propose to extend the AUML agent class diagram by adding these properties as attributes in a separate AUML AgentBaseClass compartment. This extended diagram is called *Mobile Agent Diagram*; figure 3 shows the structure of the extended AUML class diagram.

After specifying the mobile agent internal structure, it is necessary to specify the itinerary diagram (R2, Figure1: mobile agent diagram → Itinerary diagram). Also, for each mobile agent it is necessary to specify the different states that agent can reach during its lifetime (R5, Figure 1: MA diagram → Lifecycle diagram).

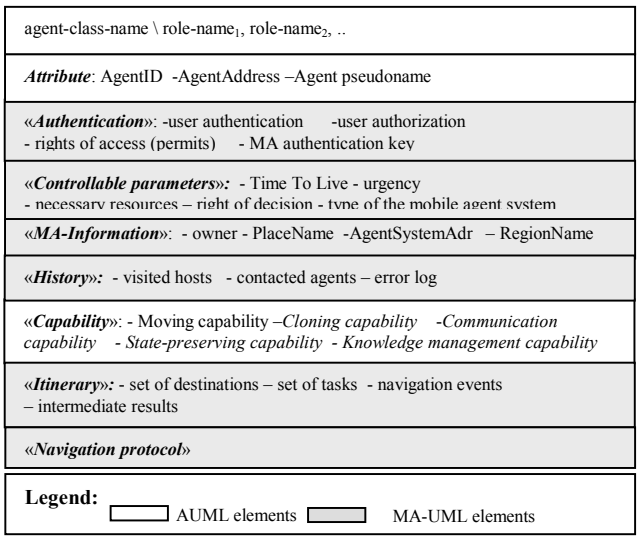


Figure 3. The internal structure of the mobile agent

3.3 Itinerary Diagram

An agent's itinerary describes the tasks of the agent and the locations where those tasks are to be performed [7]. The itinerary model defines the mobile agent travel planning. The travel planning can be determined either statically or dynamically. That is, it can be calculated either before the agent is dispatched or while the agent is migrating. Dynamic travel planning is more flexible, and can adapt to environmental changing in real time. However, since the travel planning is calculated on the fly, it also consumes more computation time and more power of the local sensor. On the other hand, although static travel schema cannot adapt to the network change, it is able to save both computation and power since the travel planning only needs to be calculated once. Computation-efficiency, power-efficiency, and flexibility are three parameters that cannot be satisfied at the same time.

In order to be able to specify a dynamic and flexible travel planning, which can adapt to environment and network changes, we propose that the developer predicts and introduces *Navigation Events* into the itinerary model. The navigation events represent the unexpected events that can be produced during the migration of mobile agent or having learned information from another entity. Moreover, the developer must define additional destinations places (*Equivalent Places*) in the itinerary model. Then, if a navigation event occurs and MA can not reach a given place, it must update its travel planning and move to equivalent place instead of the failure place in order to be able to perform the associated tasks.

In order to model the elements of the mobile agent itinerary model, we define a new diagram, called *Itinerary Diagram*. This diagram represents an extension of the UML class diagram by introducing new stereotyped classes and new graphical notations. Figure 4 describes the elements of the itinerary diagram.

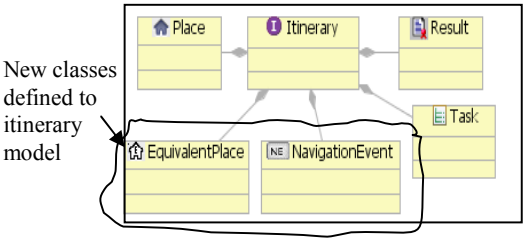


Figure 4. The Itinerary diagram

The set of navigation events that can occur and the equivalent places defined in the itinerary model can be updated by the MA during its travel, having learned any information or having met a problem. Figure 4 presents the static view of mobile agent itinerary model. The dynamic view of itinerary may be viewed as the specifications of the travel planning and the tasks execution planning. In order to model the dynamic view of the itinerary, we define two new diagrams: first the navigation diagram which specifies the travel planning between locations (R3, Figure 1: Itinerary diagram → Navigation diagram). Second, the mobile agent activity diagram which specifies the mapping of tasks to be performed on locations (R4, figure 1: Itinerary diagram → MA activity diagram).

3.4 Mobile Agent Diagram

This diagram is responsible for modeling the tasks execution planning among different places. In UML, activities (tasks) are specified with the activity diagram. In order to model the relationships between locations and tasks and the reuse aspect of an activity, we propose to introduce the concept of location in the UML activity diagram; we call this diagram *Mobile Agent Activity Diagram*. To specify the concept of location, we propose to attach parameters to each activity (parameterized activity). These parameters represent the list of places where this activity (task) needs to be performed. Figure 5 illustrates an example of a mobile agent activity diagram.

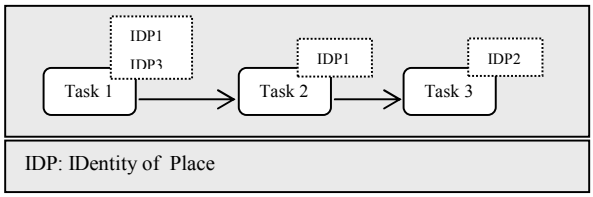


Figure 5. The mobile agent activity diagram

The parameters attached to each task make possible to the developer to specify the set of places (locations) where this task can be performed. In figure 5, task 1 for example is performed in place P1 and place P3. The parameters can be instantiated during the system execution process. That means that the set of places can be updated during the MA execution when it can not reach some places by the equivalent places specified in the itinerary diagram. Then, with the use of parameters it is possible to model a *dynamic tasks execution planning* and to specify the *reuse aspect* of an activity.

3.5 Navigation Diagram

The purpose of this diagram is to model the agent travel planning which describes its movements between places defined in its itinerary.

In order to specify a dynamic travel planning, we propose a variant use of the UML statechart; we call this diagram *Navigation Diagram*. We propose that the states model places, the transitions between states model the movements of an agent between places, and events which trigger the transitions between states model the navigation events which drive the movements of MA to the equivalent places. Figure 6 illustrates an example of the proposed navigation diagram.

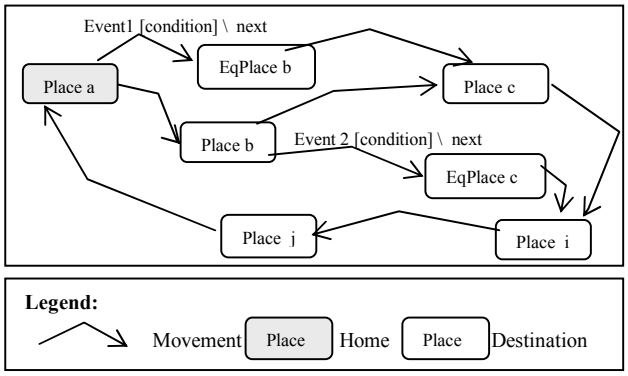


Figure 6. The navigation diagram

Figure 6 shows a navigation diagram that models the travel planning of the agent between different places. In this diagram, when MA is in "place a" and need to move, it passes automatically to the next place ("place b"), but if any navigation event occurs (event1), MA transit to the equivalent place of place b (Eqplace b).

With the use of the UML statechart diagram, it is possible to specify the navigation events that can be occurred during MA travel and to specify additional locations to be visited if a MA can not reach a given location. These issues allow to model a *flexible and dynamic travel planning* which adapt to the environment changes.

3.6 Lifecycle Diagram

The lifecycle diagram extends the UML statechart diagram by adding new stereotyped actions and new types of transitions.

During its lifetime and in order to achieve its mission, MA needs to communicate with other entities (agents, environment, user), to move from location to another, and to perform the assigned tasks. These issues allow to specify when and how mobile agent transits to one state to another. Then, in order to specify mobile agent change states, we believe it is necessary to specify relations with the interaction diagram, which specifies the communication acts, the navigation diagram, which specifies the travel planning, and the activity diagram, which specifies the tasks mapping.

In order to model these relationships, we propose to introduce new stereotyped actions and new types of transitions: activity, interaction, and navigation transitions. The activity transition triggers the execution of the mobile agent activity diagram in the

"Activated" state in order to perform tasks relevant to a given location (R7, Figure 1: Lifecycle diagram → MA Activity diagram). The interaction transition triggers the execution of the mobile agent sequence diagram in a given state in order to execute communication acts (R8, Figure 1: Lifecycle diagram → MA sequence diagram). The navigation transition triggers the execution of the navigation diagram in the "ChooseDest" state in order to determine and to decide to the next location to be visited (R6, Figure 1: Lifecycle diagram→ Navigation diagram). As an example, figure 7 illustrates the graphical notations and the use of navigation transition.

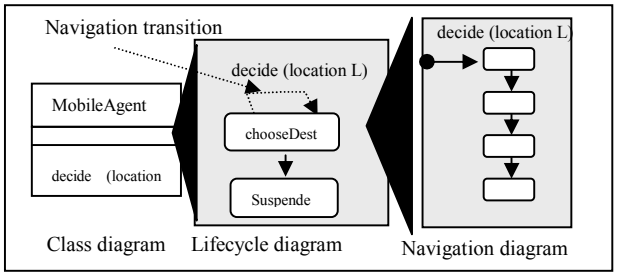


Figure 7. Granularity levels of the lifecycle diagram use: navigation transition

3.7 Mobile Agent Sequence Diagram

The AUML sequence diagram (protocol diagram) [2] presents a set of interactions between agents playing different roles. In order to model the different interactions between the new elements defined by MA-UML profile, we propose to introduce the new proposed elements in the AUML sequence diagram. We call this diagram *Mobile Agent Sequence Diagram*. Table 2 identifies the instances that may appear in the mobile agent sequence diagram and its associated diagram elements.

Table 2. The mobile agent sequence diagram elements

Instances	Mobile Agent	Stationary Agent	Place	Resource
Diagram element	Mobile Agent	Stationary Agent	Place	Resource
Instances example	SearcherAgent	InterfaceAgent	MarketPlace	DataBase

4. MAMT: AN ENVIRONMENT FOR MODELING AND IMPLEMENTING MOBILE AGENTS

In order to support the use of the MA-UML profile and to implement a system using MA-UML, it is necessary to create a software CASE Tool (Computer Aided Software Engineering Tool) and to refine the models and to generate Java code.

In the following sub-sections, we present the software CASE Tool we have developed. Then we describe the proposed strategy for mapping mobile agent specifications to Java code.

4.1 The MAMT environment

We developed MAMT (Mobile Agent Modeling Tool) that is a software development environment to support a mobile agent-based applications development process. The MAMT environment was developed as a set of plug-ins for the Eclipse Platform [12].

The MAMT environment consists of three plug-ins, which are:

The graphical editor tool: includes the MA-UML library. It is composed of the UML metamodel, the AUML metamodel and the MA-UML metamodel. The editor tool supports the seven MA-UML diagrams and allows the creation of a number of UML, AUML and MA-UML artefacts. The GEF plug-in was used to provide a powerful foundation for creating editors for visual editing of arbitrary models. The EMF plug-in was used to create and store UML, AUML and MA-UML models in the XMI format.

The translator tool: includes the transformation rules. It is responsible for the transformation of the MA-UML XMI file, which represents the output of the editor tool, to the UML XMI file.

The code generation tool: includes a Java library and a set of code generation rules. It is responsible to generate automatically Java code based on the UML XMI file. The code should be completed by the developer.

As a MAMT prototype, we have implemented the graphical editor tool using the UML2 2.0, EMF 2.2, GEF, GMF plug-ins of eclipse. This editor supports essentially the three static diagrams of the MA-UML profile which are: the environment diagram, the mobile agent diagram and the itinerary diagram. Also it supports the MA-UML navigation diagram. Figure 8 illustrates as an example the metamodel needed to implement the environment diagram editor.

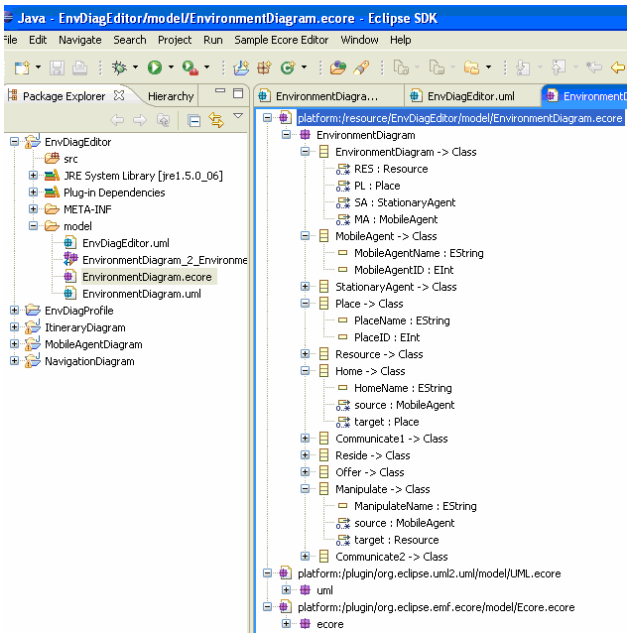


Figure 8. A metamodel part of the environment diagram

Figure 8 shows the metamodel of the environment diagram which describes the different metaclasses added to UML metamodel (e.g. mobile agent, resource, reside). The Editor tool corresponding to the environment diagram is illustrated in figure 9.

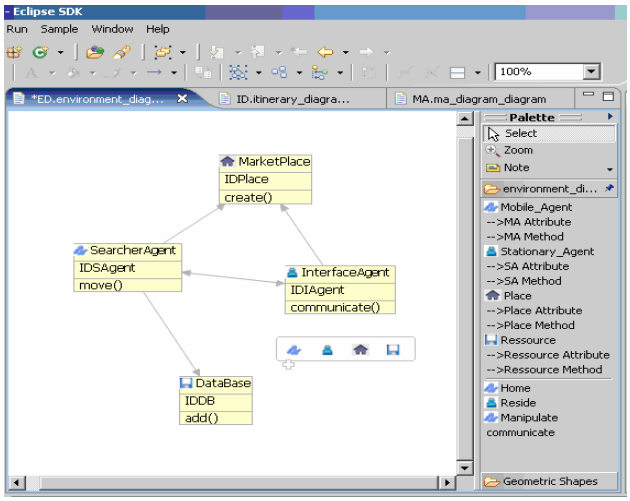


Figure 9. A snapshot of the MAMT tool showing the editing of an environment diagram

4.2 Mapping MA-UML models to Java code

The transformation process of MA-UML models to Java code is supported by the two MAMT plug-ins: the translator tool and the generator tool. In order to map MA-UML models to Java code, we propose to use the MDA approach [13]. The MDA defines a set of consecutive transformations that should be applied to the models in order to allow the transformation of high-level abstraction models into code. The MDA approach proposes four layers allow deriving code from the specifications, which are: CIM, PIM, PSM, and code. The proposed approach for mapping MA-UML specifications to Java code is illustrated in figure 10.

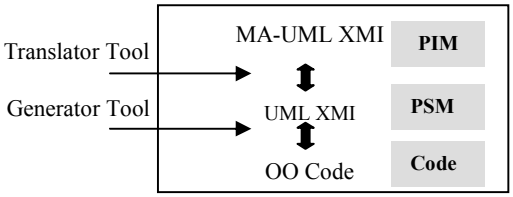


Figure 10. Transformation process based on MDA

The PIM layer. The MA-UML is a modelling language focused to model mobile agent-based application. The MA-UML models that describe an application are PIMs that are portable to diverse systems and can be used to generate different computational models by applying various implementation platforms.

Transforming PIMs into PSMs. After elaborating the MA-UML models of PIMs, these models should be transformed into PSMs. The transformation of MA-UML models into UML models occurs in two stages: (1) the first stage: the first stage consists of describing the all the MA-UML models in a textual description by using the XMI format. The XMI is used in our approach to represent the MA-UML models, to assist in the transformation from MA-UML models into UML models, to represent the UML models and to help the transformation from UML models into code. (2) The second stage: in this stage, the MA-UML XMI generated in the previous stage is converted into a UML XMI. This transformation is based on the transformation rules that we have defined.

The PSM layer. The UML XMI generated in the previous stage represents a PSM of the application. The UML XMI created in this stage is a UML class diagram that contains the classes of the application. All application entities, properties, relationships, and behaviour modelled by the MA-UML diagrams generate the UML class diagram of the application.

Transforming PSMs into code. In the final stage of the transformation process, the UML models are transformed into code. This kind of transformation corresponds to the last stage of the MDA approach that transforms PSMs into code. This transformation is based on the code generation rules that we have defined.

5. CONCLUSION

This paper presented an approach to support the modeling and the implementation of mobile agent. This approach consists first on the MA-UML profile which defines a set of seven diagrams that describes the static and the dynamic aspects of the mobile agent. Second, this approach consists on the MAMT that is a software development environment to support the use of MA-UML diagrams and transform specification models to Java code. The transformation process is based on the MDA approach. Our future works include two axes. In the first, we are looking to implement the other MA-UML dynamic diagrams. In the second axe, we are looking into the design and the implementation of a mobile agent-based application in the medical field.

6. REFERENCES

[1] Unified Modeling Language Specification, version 2.0, OMG, http://www.uml.org. Accessed in: December 10, 2004.

[2] Bauer, B., Müller, J. P., Odell, J. 2001. Agent UML: a Formalism for Specifying Multiagent Software Systems. In International Journal of Software Engineering and Knowledge Engineering, vol. 11, No. 3, pp.1-24, 2001.

[3] Tomoya, T., Tadanori, M., Takashi, W. 1998. A Model of Mobile Agent Services Enhanced for Resource Restrictions and Security. In International Conference on Parallel and Distributed Systems (ICPADS '98).

[4] Self, A., DeLoach, S. A. 2003. Designing and Specifying Mobility within the Multiagent Systems Engineering Methodology. Special Track on Agents, Interactions, Mobility, and Systems (AIMS) at The 18th ACM Symposium on Applied Computing (SAC 2003).

[5] Sutandiyo, W., Chhetri, M. B., Loke, S.W., Krishnaswamy, S. 2004. MGaia: Extending the Gaia Methodology to Model Mobile Agent Systems. In the Sixth International Conference on Enterprise Information Systems (ICEIS 2004), Porto, Portugal, April 14-17.

[6] Belloni, E., Marcos, C. 2003. Modeling of Mobile-Agent Applications with UML. In Proceedings of the Fourth

Argentine Symposium on Software Engineering (ASSE´2003). 32 JAIIO (Jornadas Argentinas de Informática e Investigación Operativa), Buenos Aires, Argentina. September 2003. ISSN 1666-1141, Volume 32.

[7] Ling, S., Loke, S. W. 2001. Verification of Itineraries for Mobile Agent Enabled Interorganizational Workflow. In Proceedings 4th International Workshop on Mobility in Databases and Distributed Systems (MDDS'2001), Munich, Germany, 582-586.

[8] Klein, C., Rausch, A., Sihlinh, M., Wen Z. 2001. Extension of the Unified Modeling Language for mobile agents. In Siau K. Halpin T. (Eds.): Unified Modeling Language. System Analysis, Design and Development Issues, chapter VIII. Idea Group Publishing, 2001.

[9] Mouratidis, H., Odell, J., Manson, G. 2002. Extending the Unified Modeling Language to Model Mobile Agents. Workshop on Agent-oriented methodologies. OOPSLA 2002, Seattle, USA, November (2002).

[10] Kang, M., Taguchi, K. 2004. Modeling Mobile Agent Applications by Extended UML Activity Diagram. In Proceedings of the 6th International Conference on Enterprise Information Systems (ICEIS)'04, Porto, Portugal, 519-522, April (2004).

[11] Kusek, M., Jezic G. 2006. Extending UML Sequence Diagrams to Model Agent Mobility. In Agent-Oriented Software Engineering, vol. 4405, pp. 51-63, 2006.

[12] Eclipse: Eclipse.org, v 3.0, http://www.eclipse.org/. Accessed on 05/2005.

[13] OMG MDA Guide. Version 1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf.

Component-based models and simulation experiments for multi-agent systems in James II

Jan Himmelspach
jh194@informatik.uni-rostock.de

Mathias Röhl
mroehl@informatik.uni-rostock.de

Adelinde M. Uhrmacher
lin@informatik.uni-rostock.de

Faculty of Computer Science and Electrical Engineering
University of Rostock
18059 Rostock, Germany

ABSTRACT

The architecture of the modelling and simulation framework JAMES II facilitates its reuse for a broad range of applications, including agent-based modelling and simulation. Simulation studies can be done by using component based models, which may have been defined in an external IDE and saved in an standardized way. We created a customisable middleware for JAMES II, which can be easily extended to read various experiments and models from different sources, to dynamically instrument the created structures, to execute models on different hardware infrastructures efficiently, to store simulation data into diverse data sinks, which can be used to create specialised IDEs, and which can be fully integrated into other applications.

1. INTRODUCTION

In multi-agent research, particularly in bridging the gap between conceptual modelling and implementation, simulation has played an important role from the very beginning and since then features central for those simulation environments to support an easy evaluating of multi-agent systems has been subject of discussion [10]. Early approaches of utilising simulation in the context of agent design have been characterised by ad-hoc implementations or by concentrating on one test scenario only, e.g. Tileworld [22]. Over the years the attention has turned towards exploiting state of the art modelling and simulation (m&s) methods, or entire frameworks for agent design. The list of existing simulation systems is rather long, they offer quite diverse features for supporting the design of diverse models. Some of these systems try to make models interoperable, e.g. MÖBIUS [2], some focus on an efficient distributed simulation, e.g. μsik [21], some on specific application areas (e.g. NS/2 [5], SWARM [19]), some on web-based simulation (e.g. D-SOL [14]), and some on certain formalisms or description languages, e.g. JDEVS [6] and SESAM [16]. Several of these systems are extensible, few have a clear separation between (declarative) model and simulator, and even less address the problem of a declarative experiment description.

Especially the translation process, from a declarative model representation to an executable model is handled quite differently by the available systems. Sometimes models are directly created in a general purpose programming language, an approach pursued in Repast, or they are transformed into source code (from a symbolic description), compiled and executed afterwards, others directly interpret the symbolic model. All techniques have advantages as well as disadvantages [9]. On the one hand, the expressiveness of a general purpose programming language is considered to be higher and the resulting code most often faster (but this heavily depends on the skills of the author). On the other hand, model languages may ease the creation of models, may prevent errors, and may even be better reusable. Compiling means speed up, interpretation eases observation and interaction.

Despite all this efforts, so far only few answers to the question on how to transform declarative models into efficient executable code and thus combining the benefits of a declarative, composite model definition with an efficient and sound execution, do exist. Whereas the benefits of an explicit model description appear obvious and most agent-oriented approaches distinguish between model and simulation, one other aspect has achieved too little attention so far: supporting the simulation experiment itself. However, an unambiguous description of experiments is a pre-requisite for a systematic experimentation with multi-agent systems and their repeatability. Again we find declarative descriptions and executions tightly connected. Thus, simulation systems face a set of interrelated challenges with respect to

Modelling research: Modelling research most often deals with the problem of how to describe a model. Thereby the focus is either on how to describe a problem (which most often leads to new formalisms) or on how to make models reusable and exchangeable. Especially the often neglected validation of models is of high importance for getting reliable results by simulation.

Simulation research: The development of simulation algorithms is strongly related to the modelling formalisms used and platforms to be supported. Thereby the achieved results are most often not comparable to results which have been previously achieved because most of these algorithms have strong interrelationships with the frameworks they are embedded in.

Simulation experiments: Quite many simulation systems have been developed for concrete application domains, or even for concrete simulation experiments [19], often by non-experts in the area of modelling and simulation. For being able to execute a broad range of different applications

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

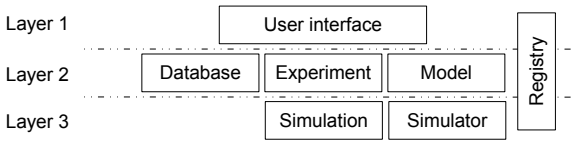


Figure 1: Modules of James II.

a variety of modelling formalisms and platforms have to be supported in an efficient manner and the software must be accessible for non m&s experts. In addition experiments and thus the computed results must be repeatable.

The two aspects (1) how to combine a composable model construction with efficient simulation engines and (2) to pursue the question what is an experiment, and how can it be supported shall form the nucleus of our paper. A plugin-based architecture provides a suitable base for our endeavour. In the following we will show the general experimentation process of James II and how symbolic model descriptions can be integrated. The proposed mechanism allows the integration of any model description language – as long as a so called *ModelReader* (and target model classes) exist in the system – but both can be easily extended as well. This process can be considered as a new way (for the generation of executable models) – symbolic models are read and mapped onto executable model components.

An example of a multi agent-based application will be used to illustrate the applicability and usability of the achieved solution in regards to component based modelling of multi agent systems and the usage of efficient distributed simulation for experimenting with the model.

2. BACKGROUND

A general modelling and simulation framework usable for different applications by various users has to be very flexible: all parts, and even sub parts, of the framework have to be exchangeable. For being reusable in the large James II has been split up into several modules (see Figure 1). Each of these modules realises a part of the functionality required for an m&s framework. These modules are clearly separated from each other – this ensures their interchangeability and reuse. The most apparent one may be the strict distinction between model and simulator. A model cannot directly call simulator functions – this makes it fairly easy to use different simulators, or even to exchange the simulator used during a simulation run – e.g. if another one is assumed to be more efficient. For being able to reuse a model, it is useful to have generic model descriptions which are independent from a concrete simulator implementation and can be converted into executable models on demand (e.g. [1]). These model components, sometimes called building blocks [30], can be described by, e.g., using XML.

The user interface encapsulates model creation, model execution (control + visualisation), data analysis, and so on. Up to now we only have a rudimentary graphical user interface (GUI), i.e. we only have a model view, a simple experiment editor, and very simple model editors. The GUI is only loosely coupled to the remaining simulation system and it is even possible to use James II without a GUI at all (either in a kind of batch mode or interactively) or to embed James II into another application. The GUI has been designed by using the model-view-controller paradigm

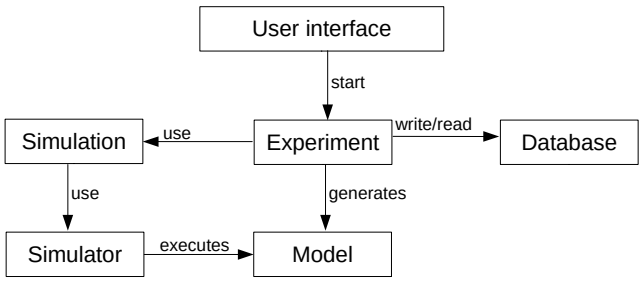


Figure 2: Relationship of the core modules of James II.

which allows, e.g., various views on the same model. The basic coupling between a running simulation and the GUI is realised by the Observer-pattern.

The data sink used during a simulation run may have to be able to handle many observations, such that the run can be analysed and visualised afterwards. The database interface makes no restriction in regards to the database to be used – thus everything starting from a plain ASCII file up to a modern database system can be used in principal. The interface is standardised, thus any sink implementing the interface can be used interchangeable for collecting runtime data.

The Model module encapsulates the modelling formalisms supported by James II. All executable models must be descendants of the base class `Model`, all externally visible functions should be placed in an interface on which the simulators can operate. These model classes can be directly used for creating models by coding in Java, for other model sources instances of these classes need to created while reading a model. Models are explicit and they are separated from the execution (simulation algorithms). The computer on which a model is computed does not matter, e.g. the mobility of agents is separated from moving models to other computers – if the latter is done, the reason is load balancing and not model logic.

The simulation module in James II contains the simulation management part. If a simulator executes a model this is called a simulation. In this package methods and classes for simulation setup (simulator selection and creation, partitioning, management of distributed computing resources), run time control (pause, ...) as well as simulation related run time jobs (e.g. load balancing) are located. Simulations in James II can either be executed sequentially on a single machine, several simulations can be executed in parallel on different machines, or simulation can be executed using parallel distributed algorithms. If they are to be executed in a parallel distributed manner, the integrated (and extensible) partitioning and load balancing sub packages come into play.

The Simulators are selected by and embedded into a simulation. This selection depends firstly on the model class of the model to be executed and secondly on other criteria. Currently the list of available criteria is extended by a new "intelligent" criterion, which tries to automatically select the best of the algorithms usable in principal. In James II several simulators for each of the different formalisms can coexist. E.g. currently we have six different simulation algorithms for PDEVS, including sequential and parallel variants [11].

Experiment is the central term in James II. Simulations

may be carried out with a focus either on conducting experiments with models or on experimenting with simulation algorithms. This package provides support for creating flexible experiments with support for a variety of simulation jobs, e.g. for parameter optimisation and validation.

An experiment is defined as the execution of a set of simulation runs for answering a concrete question. The experiment module has to combine the various, previously introduced modules (see Figure 2).

A PlugIn mechanism [13] allows the flexible extension of the simulation framework without the need to modify the code of the core later on. Due to the strict separation between models and simulators, simulation algorithms can be easily exchanged and thus evaluated. This makes the PlugIn mechanism a base for a reliable evaluation of new simulation algorithms. The adaptation of a modelling and simulation framework for certain user groups (especially of the user interface) is crucial for its usability. This adaptation can be easily done by the PlugIn mechanism or by embedding the complete JAMES II core (with all installed plugins) into another JAVA application [18]. Modelling and simulation applications whose individual requirements may even contradict (e.g. the use for demonstration purposes in the field of teaching simulation algorithms versus highly efficient implementations of algorithms for efficient experiments) can coexist in the framework [12, 11].

3. DIFFERENT MODEL SOURCES

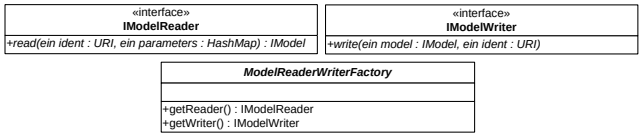


Figure 3: Interfaces and base factory class for reading / writing models.

JAMES II does not only allow the integration of different modelling formalisms / languages but also the usage of any description languages for them. I.e. in JAMES II symbolic and executable models are differentiated and only in the case of models coded in Java both might be the same. Interfaces defining implemented models are used for accessing executable models. This allows different implementations of the classes for executable models, and thereby allows efficient and adoptable model realisations. This is a highly required feature: different models may have completely different characteristics, such that different data structures in combination with different simulation algorithms might prove beneficial. For each description language a special reader has to be designed (a model reader) which maps a model description on instances of executable model classes. Depending on the reader any source (database, code files, ...) can be used for retrieving a model.

The interface, which is provided by the framework and has to be implemented by a model reader is shown in Figure 3. A model reader becomes accessible as a plugin by JAMES II by setting up a simple plugin description file. The plugin type description is given in Figure 4. If an XML plugin file for the ModelReader extension point is found, JAMES II automatically installs the plugin and if later on an experiment definition links to a model readable by the newly defined model reader, this plugin is automatically used.

```
<?xml version="1.0" encoding="UTF-8" ?>
<plugin xmlns="http://www.informatik.uni-rostock.de/
    mosi/cosa/plugintype">
  <id name="model reader/writer plugins" version="
      1.0" />
  <abstractfactory>james.core.data.model.
      AbstractModelReaderWriterFactory</
      abstractfactory>
  <basefactory>james.core.data.model.
      ModelReaderWriterFactory</basefactory>
  <description>Support of diverse model readers/
      writers.</description>
</plugin>
```

Figure 4: XML based Plugin type description file for model readers and writers (defines an extension point). Each reader/writer plugin must provide a factory which is a descendant of the specified basefactory.

JAMES II uses a "late" reading mechanism. Thus models are not read and instantiated if a new simulation configuration is created by the experiment but if the simulation is about to the started. Thereby the model reader is used on the computer the model must be instantiated on – this reduces network load (in distributed setups), and prevents the central (distributing) instance to become a bottleneck.

In addition to reading models JAMES II provides an interface for the integration of model writers (*ModelWriter*). Based on the interfaces of the model in memory a model description can be written to any supported target. This includes different storages and all description languages for which writers exist.

3.1 Component-oriented Modelling in XML

While models can always be directly described in Java it is possible to describe them by using a special XML-based syntax as well. The XML-based variant eases the import and export of models specified in a standard exchange format. Furthermore, XML descriptions enable us to simply generate and describe a broad range of experiments.

XML handling is based exclusively on entities that are bound to XML Schema Definitions [25]. Schema Definitions mainly define the syntax of an XML document and thereby provide the means for rendering XML documents valid or invalid. JAXB [28] is used to generate Java classes that conform to XML Schema Definitions. Thereby, the bound entities are able to transparently unmarshal XML documents to Java objects and marshal Java objects back to XML documents. This process has to be encapsulated into ModelReader and ModelWriter implementations for integrating the XML descriptions into JAMES II.

Execution of XML model components builds upon the simulators as described above. To this end, the declarative model definitions are automatically transformed to executable ones [25].

Based on XML-descriptions models can be specified in a component-oriented manner [23]. Provided and required interfaces of each model component have to be explicitly specified. Thereby internal details of a model are hidden and direct dependencies between models eliminated. A set of components may become customised and arranged to form a composition according to the aim of an experiment. Parameters set on component instances are evaluated and dependencies between components resolved.

3.2 Agents and James II

In JAMES II agents can either be modelled (i.e. JAMES II is used as a testbed for multi agent scenarios) [24], JAMES II can be used to create an environment in which "real" software agents can be plugged into and evaluated [8], or agent technology can be used to extend the simulation middleware of JAMES II.

JAMES II strictly separates between models and simulators. A model is a "pure" picture of the system under study and does not contain any simulation related information. Every agent model in JAMES II has always to be based on a supported modelling formalism. Agent specifics (e.g. communication protocols, migration issues) are either inherently supported by the modelling formalism used or they have to be explicitly modelled. I.e. in JAMES II agents and the environment they reside in are models. Agent models described outside of JAMES II have to be converted into an executable model of JAMES II before they can be simulated. The model reader schema allows the usage of arbitrary agent model repositories and it allows the usage of different agent modelling languages, if there is a translation into executable model classes of JAMES II. Later on we'll present an example of a multi agent simulation model, defined by using model components which are stored in XML (as described above).

JAMES II is well suited for the simulation of (large scale) multi agent models because the usage of different simulation algorithms for a model, e.g. different sequential and distributed ones, is supported. E.g., in JAMES II models can be executed by using a fine-grained parallel and distributed setup if the simulation of a model on a single machine is no longer possible (as long as there is an appropriate simulator plugin). If several simulations shall be executed in parallel (e.g. replications required because of stochastics) the model reader schema takes care of creating agent models directly on the hosts they shall be simulated on.

Thus the model reader schema enables flexibility for the simulation of multi agent systems in regards to model sources, computation schema used and up to a certain degree in regards to the modelling formalism used.

4. EXPERIMENTS IN JAMES II

Experimenting is a difficult, time consuming (sometimes event too time consuming [7]) and error prone task. Different types of experiments have to be conducted in a simulation study: runs for exploration are followed by validation experiments, for making sure that the model is valid. In addition, we might have to adjust some parameters by optimisation, or even re-design the entire model. Finally, we will execute the experiments to answer our initial questions we had in mind while creating the model. Consequently, a model must be independent from the type of experiment it is used in: we decided to create an explicit experiment description, which just links the model to be experimented with.

Reading/writing experiments.

JAMES II is not restricted to a concrete experiment definition language. Any experiment definition is fine as long as there is a suitable plugin for reading and converting the experiment definition. For the plugin to be accessible by JAMES II the interface *IExperimentReader* has to be implemented. A user interface may initiate the reading of an XML-based experiment definition from a database. The reader retrieves XML data and initialises the experiment instance according to the experiment definition. Changes to an experiment or a newly created experiment may be saved by JAMES II using a so called experiment writer, which implements the *IExperimentWriter* interface.

Job creation.

The support of different hardware infrastructures can reduce the overall time needed for an experiment if corresponding hardware is available. Consequently, an experiment definition in JAMES II is independent from the system an experiment will be executed on. An experiment definition usually comprises the parameters to be modified from simulation run to simulation run, how they shall be modified (e.g. by an optimisation method), the number of replications (e.g. for achieving statistical reliability), and so on. An experiment definition creates *simulation configurations* ("model and simulation parameter combinations") which are transferred to a *simulation runner*. Depending on the simulation runner used, the simulation configurations will be executed sequentially on a single host or by using a coarse- or fine-grained parallel simulation on any (supported) hardware infrastructure.

Simulation creation.

An executable model is created on the machine the simulation will be executed on by using the model reader schema. For creating a simulation from a job based on the XML model components as described above, a *ModelReader* retrieves the XML-data of the components from the database, unmarshals it, and configures the component(s) according to the provided model parameters (e.g. create 200 or 400 agent instances in the model). If a remote access to the model source (database) is possible, the model, or parts of it, might reside anywhere in the world. The executable model is created by using a ModelFactory which converts the model components into an executable instance of the target modelling formalism. Further details of the model creation process can be found in [26].

If the model has been created, an *instrumenter* will take care of attaching observers to the model. The selected data sink will be attached and the simulators will be created and instrumented. Afterwards the simulation is executed.

5. A MULTI AGENT EXAMPLE

Mobile ad-hoc networks (MANETs) are computer networks based on wireless communication. MANETs are characterised by dynamic network topologies. Nodes induce topology changes by appearing, disappearing, and moving in a spatial environment. To provide fast and reliable connections poses a severe challenge for MANETs, because MANETs lack central infrastructure and bandwidth as well as energy are strictly limited [3].

Network devices operating independently of the mains, typically have limited capabilities in terms of memory, battery and performance. Complex operations require cooperation among network nodes. To make cooperation possible, resources and capabilities of nodes, in the following referred to as services, have to be announced by providers and need to be locatable be a requester. From the perspective of human users, services should be accessible in a transparent

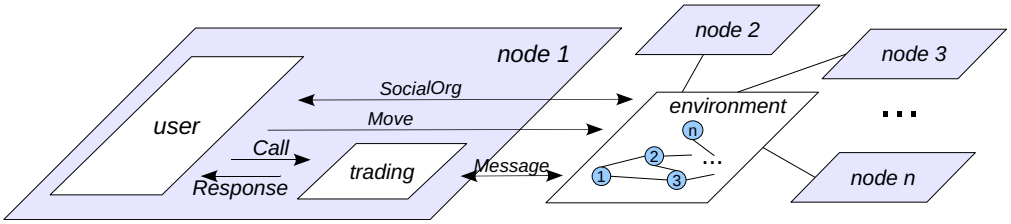


Figure 5: Conceptual model for evaluating service trading in mobile ad-hoc networks.

manner. Therefore, service descriptions, service matching algorithms, incentive schemes, and distributed reputation systems are developed in the project DIANE [4]. However, these mechanisms need to be thoroughly evaluated [15].

Conducting experiments to evaluate service trading in MANETs requires user models to represent network nodes as autonomous actors, which move in a spatial environment and announce and request services. Developers of service trading protocols are mainly interested in the cost-benefit ratio of protocols, if these are confronted with different kinds of user models. However, MANETs are typically simulated with special simulation systems, i.e. network simulators [17], which concentrate on the lower layers of the OSI protocol stack, e.g. routing protocols on the third layer. Within theses simulation systems, models for representing movement and service behaviour are simply calculated based on stochastic distributions [29]. As network traffic in MANETs is sensitive to local accumulation of nodes and temporal accumulation of network usage, more complex user models are needed, e.g. to allow a consistent modelling of motion and service behaviour and to incorporate social aspects. Thereby, users need not merely move individually but may form groups and move as clusters aiming at same destinations and partly synchronising their schedules.

Please note, that the purpose of this simulation scenario is not to test agents, but to use agent models for evaluating service trading protocols. Thereby, the agent models are part of the experimental setup.

Figure 5 shows a conceptual model for evaluating service trading in MANETs. The network consists of mobile nodes. Network connection between nodes depend on their positions. Each node comprises a user model and a service trading model such that the trading protocol mediates all network interactions and thereby provides transparent access to services available in the network. Models are supposed to exchange the following types of events:

Call A user initiates to announce, to revoke, or to search a certain service.

Response The trading protocol reacts, after having performed all necessary actions, to calls of the user with according responses.

Message Trading protocols communicate over the network with messages of arbitrary content.

Move Movement information in a two-dimensional spatial environment.

SocialOrg Social organisation requests and responses.

User models initiate communication by sending calls to the trading protocol. A call may indicate the publication of or search for services. Calls are passed to the protocol model, which answers user calls with *Response* events.

The concrete type of the user model and the trading protocol should be a variation point of the simulation model. Different kinds of user and protocol models should be exchangeable independently of each other within one node. For simulating service trading in MANETS with James II, user and protocol models have been realized as model components.

In the following we'll show how the experiment definition together with the model reader schema facilitates the flexible experimentation with this component-based model.

5.1 Defining composition structures

Figure 6 shows the composition structure of the Manet component. The *Manet* component can take parameters to initialise the number of nodes to be simulated, the type of the user model to be used, and the type of the protocol model to be used inside each node.

The spatial environment represents a certain geographical area containing streets and buildings and it keeps track of all user positions. The network component models the transport layer of the network. In real applications the whole OSI stack is part of each node. Since we are interested in higher level protocols the four lower OSI layers are pooled in a centralised network component, which delivers messages to nodes. The connectivity of each node is calculated according to its position with respect to other nodes.

Each node contains a *Protocol* component and a *User* component. From the point of view of the node component, the user and protocol components are black boxes whose couplings are defined by interfaces. The parameters *user* and *protocol*, which may be set on a node component, determine the type of sub components to be used. Each of both can be easily replaced by another one which provides the same interface. Three different versions of the user component have been implemented up to now. They can be composed into a node alternatively. We will now take a look at different implementations of user components and protocol components. All of these are realised themselves as composite components.

The simple user contains an *activity* sub component that manages login and logout behaviour. If logged in, the service component becomes notified to start publishing services and searching for services randomly according to a uniform distribution. Furthermore, the *activity* sub component selects destination points randomly and calculates routes to them. Routes are propagated to the *Motion* sub component, which executes them with a certain walking speed. After reaching a destination, a new route is requested from the *activity* component.

The second user model realises an activity-based user be-

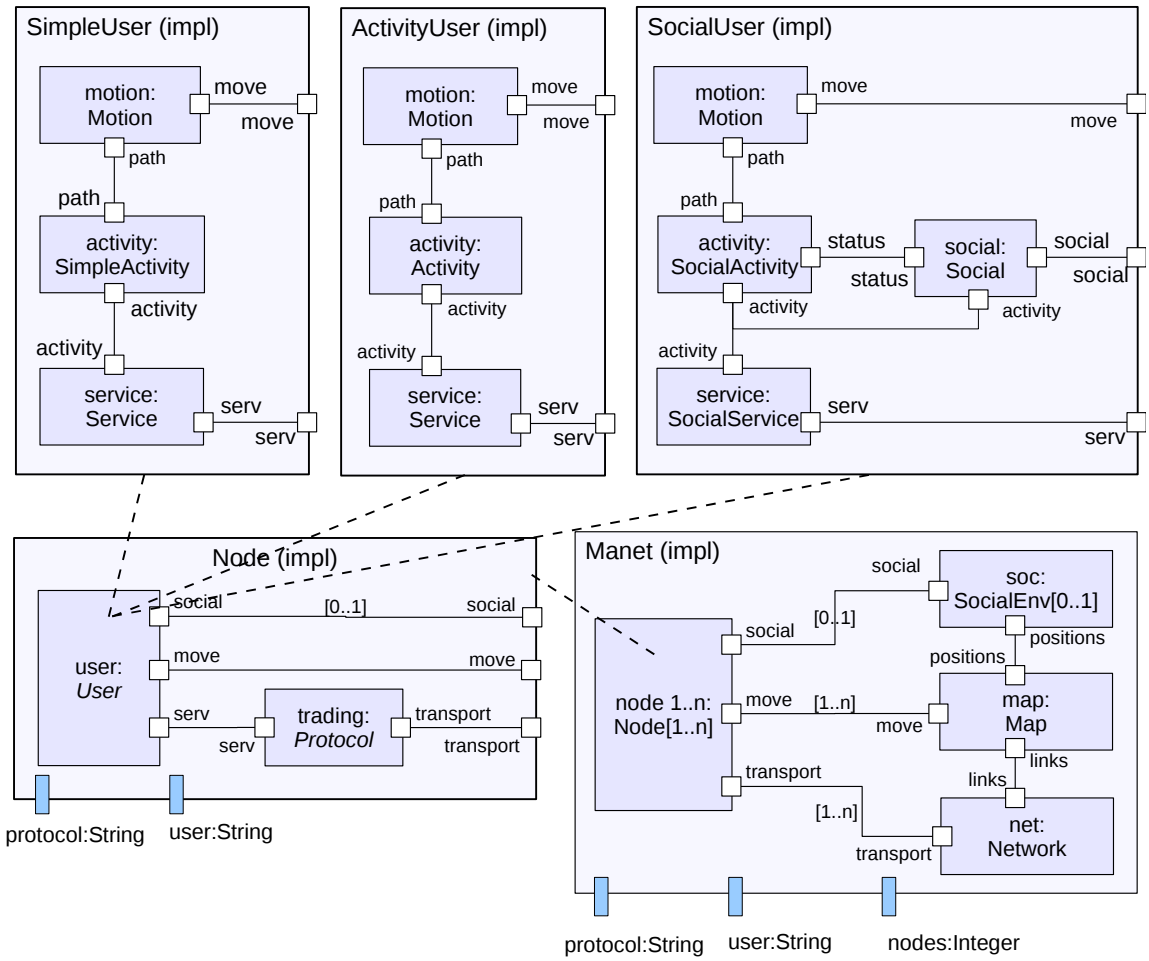


Figure 6: Structure of the manet model with three alternative user models.

haviour. Network usage and moving is not modelled independently of each other. Both depend on the activity a user is currently executing. The *Activity* model generates a schedule at the start of the day. The schedule contains fixed activities, e.g. attending a lecture as well as flexible activities, such as learning, with different priorities and durations. The current activity influence the *Motion* and *Service* sub component. The service component generates service offers and service requests according to the received activity. Activities are not independent of the spatial context, but each location is only suited for a certain set of activities. The current activity constrains the choice of the next destination and thereby the motion model. Thus, motion and service behaviour are both based on activities.

The third user model extends the activity-based model with social awareness. The sub component *Social* of each user announces planned activities to the social environment model. If users have planned similar activities and are spatially close the social environment forms a group and selects a group leader. The group leader chooses activities, which all group members may adopt. Because an activity does not uniquely define the location of performance, the group leader chooses a location out of a set of suited ones and communicates this choice to all group members. The group members are free to choose a path to the location. Thus, social users are synchronised with respect to the next joint activity and the location where this activity will be performed.

5.2 Defining model behaviour

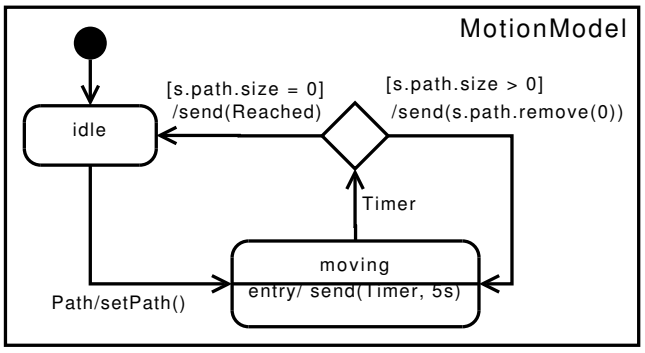


Figure 7: Motion model as a statechart.

Composition structures define the composition hierarchy of a model. The leaves of the model tree have to be equipped with behaviour, which in JAMES II may be done using different modelling formalisms. Statecharts are used in the

following to exemplify the definition of model behaviour.

Figure 7 visualizes the definition of the model behavior for the component *Motion* as a statechart. The model *Motion* starts in phase *idle* and waits for input. If path information is received, the model goes to phase *moving*. Each time phase *moving* is entered, a movement event is produced. After a certain time, triggered by a special send operation in phase *moving*, it is checked, whether the end of the path was reached. If this is not the case, the model enters phase *moving* again. If the end of a path is reached, the model produces an event of type *Reached* and returns to phase *idle*.

With SCXML [31] a proposal exists for representing statecharts in a style accessible not only to computers – like XMI, the exchange format for UML state machines [20] – but also to humans. Figure 8 lists the definition of the *Motion* model in SCXML.

```
<scxml
    xmlns="http://www.w3.org/2005/07/scxml"
    version="1.0" initialstate="idle">
 <datamodel>
  <data name="path"
      src="../PathProc:Path"/>
  <data name="posChange"
   src="../geometry:Position"/>
  <data name="move" src="../motion:Move"
      expr="create"/>
 </datamodel>
 <state id="idle">
  <transition event="../PathProc:Path"
      target="moving">
   <assign location="path"
       expr="_eventdata"/>
  </transition>
 </state>
 <state id="moving">
  <onentry> <send event="Timer"
      delay="5"/> </onentry>
  <transition event="Timer"
    cond="s.path.size() > 0"
        target="moving">
   <assign location="posChange"
       expr="s.path.remove(0)"/>
   <assign location="move"
       expr="setChange(s.posChange)"/>
   <send event="../motion:Move"
       namelist="move"/>
  </transition>
  <transition event="Timer"
   cond="s.path.size()==0" target="idle">
   <send event="../PathProc:Reached"/>
  </transition>
 </state>
</scxml>
```

Figure 8: Definition of the *Motion* model in SCXML.

5.3 Deriving Simulation Models

In the following 400 nodes are simulated with the trading protocol *Lanes* and different user components inside each network node. To simulate these different compositions, an experiment has to be defined. Figure 9 lists an according experiment definition in XML. The experiment induces three different parameter combinations ($|nodes| * |protocol| * |user|$). Each combination results in a separate simulation configuration.

```
<experiment
  xmlns="http://...de/cosa/experiment"
  xmlns:xsd="http://..org/2001/XMLSchema"
  xmlns:exp="unihro/diane/experiment"
  xmlns:net="unihro/diane/com/manet">
  <id>exp:experiment</id>
  <model>net:interface</model>
  <mparams>
    <param name="nodes">
      <value>400</value>
    </param>
    <param name="protocol">
      <value>Lanes</value>
    </param>
    <param name="user">
      <value>SimpleUser</value>
      <value>ActivityUser</value>
      <value>SocialUser</value>
    </param>
  </mparams>
  <targetFormalism>
    dynpdevs
   </targetFormalism>
  <platform>.../cosa/jamesii</platform>
  <observercfg>
    diane.experiments.v05.ObsVis
  </observercfg>
  <sparams>
    <startTime>0.0</startTime>
    <endTime>32400</endTime>
  </sparams>
</experiment>
```

Figure 9: Definition of an experiment in XML.

The experiment reader configures simulation job description, which are passed to an instance of the appropriate ModelReader, which reads and creates the model according to these.

Figure 10 shows the structure of a resulting simulation model that was derived using "400" nodes, the "Lanes" protocol and the "social user" component. The outcome is an executable PDEVS model [32], which was generated from the composition structures and model behaviours [27]. The parameter values are translated into a corresponding number of network nodes and are reflected in the internal structure of each node. After having created the model, observers for collecting experiment data are attached to the model.

Figure 11 shows three simulation runs, each with 400 nodes. Users appear uniformly distributed between 0 and 60 minutes and immediately log into the network. Subfigure 11 a) depicts the run which uses the "simple user" model for generating calls. For measuring the load of the network three different types of messages are distinguished. All messages that result from building up the lane structure are summarized by the *login* trajectory. Messages for keeping the lane structure valid are subsumed under *intra lanes* messages. Messages that are used to answer service calls of users are summarized within the *inter lane* message trajectory. Using simple user models results in a quite uniformly scattered number of service messages during the whole online period.

The trajectories in subfigure b) are produced using the "activity-based user" model. The effort for service related

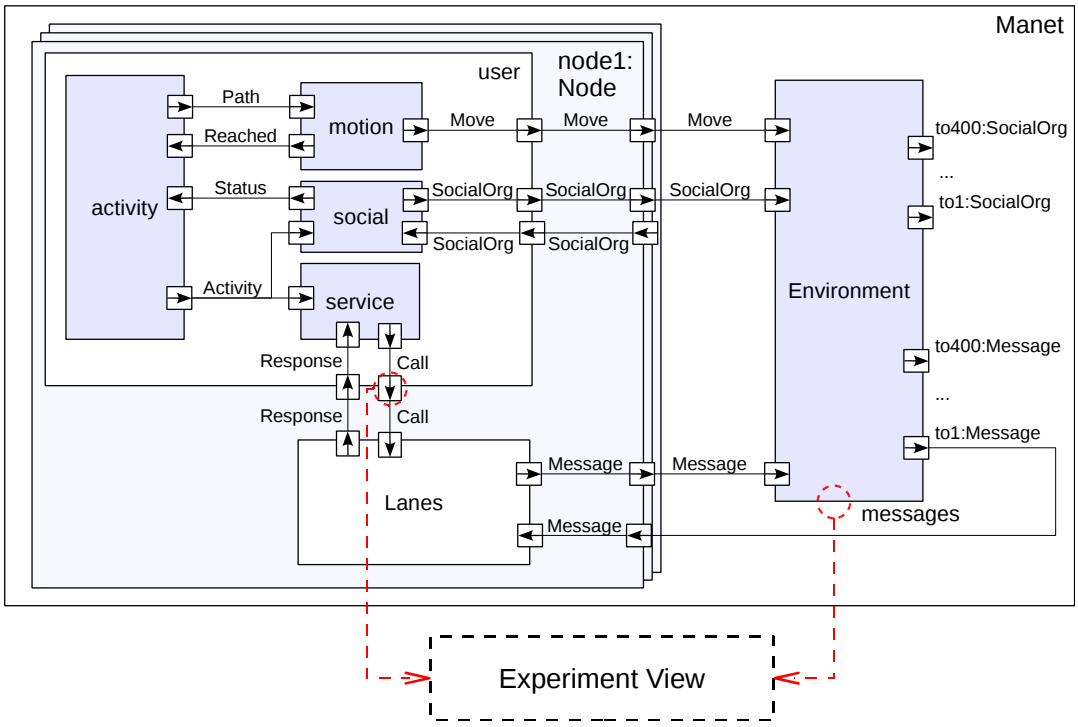


Figure 10: Structure of the derived simulation model.

messages varies not so much from minute to minute, but different periods with higher and lower numbers of service messages can be distinguished. The periods reflect globally scheduled user activities, e.g. attending a lecture. As indicated by subfigure c) "social user" models show less distinct global behaviour patterns. For further discussions of this experiment please refer to [24].

6. CONCLUSION

JAMES II realizes a plugin-based architecture that allows a flexible re-use of concepts in the area of modelling and simulation. The core of JAMES II provides standard functionality and the means to integrate additional functionality. If an experiment is conducted all those parts come together. Here we focused on the possibility to support different model descriptions in JAMES II, taking a component-based declarative modelling of a multi-agent system as an example. Even though the model is based on a declarative schema the execution is still quite efficient. This is achieved by mapping the declarative model (by a ModelReader) on executable model classes, which can be executed efficiently.

We have shown how "external" component-based model descriptions (here "COMO") can be embedded into JAMES II and that a component-based modelling of multi agent system is manageable and useful. This mechanism can be easily extended to integrate further different (component-based) model descriptions. In addition we voted for an explicit experiment definition and shortly sketched how an experiment definition can be converted into a JAMES II experiment. The strict separation of models (agents) and simulators eases the support of different hardware and thus allows the efficient execution of (many) small and large scale simulations, and thus helps to avoid errors as discovered in [7].

7. REFERENCES

[1] V. Balakrishnan, P. Frey, N. B. Abu-Ghazaleh, and P. A. Wilsey. A framework for performance analysis of parallel discrete event simulators. In *WSC '97: Proceedings of the 29th conference on Winter simulation*, pages 429–436, New York, NY, USA, 1997. ACM Press.

[2] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The möbius modeling tool. In *9th international Workshop on Petri Nets and Performance Models (PNPM'01)*, pages 241–250. IEEE, 2001.

[3] S. Corson and J. Macker. Mobile ad hoc networking (MANET): Routing protocol performance issues and evaluation considerations. `http://www.ietf.org/rfc/rfc2501.txt`, Jan. 1999. Network Working Group Memo (Request for Comments: 2501).

[4] Diane. Diane-projekt: Services in ad hoc networks (Dienste in Ad-Hoc-Netzen). `http://www.ipd.uni-karlsruhe.de/DIANE` (zugegriffen am 19. September 2007), 2007.

[5] K. Fall and K. Varadhan. *The ns Manual (formerly ns Notes and Documentation)*. The VINT Project, a collaboratoin between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PARC., 2008. `http://www.isi.edu/nsnam/ns/ns-documentation.html`.

[6] J.-B. Filippi and P. Bisgambiglia. JDEVS: an implementation of a DEVS based formal framework for environmental modelling. *Environmental Modelling and Software*, 19(3):261–274, March 2004. Elsevier Science.

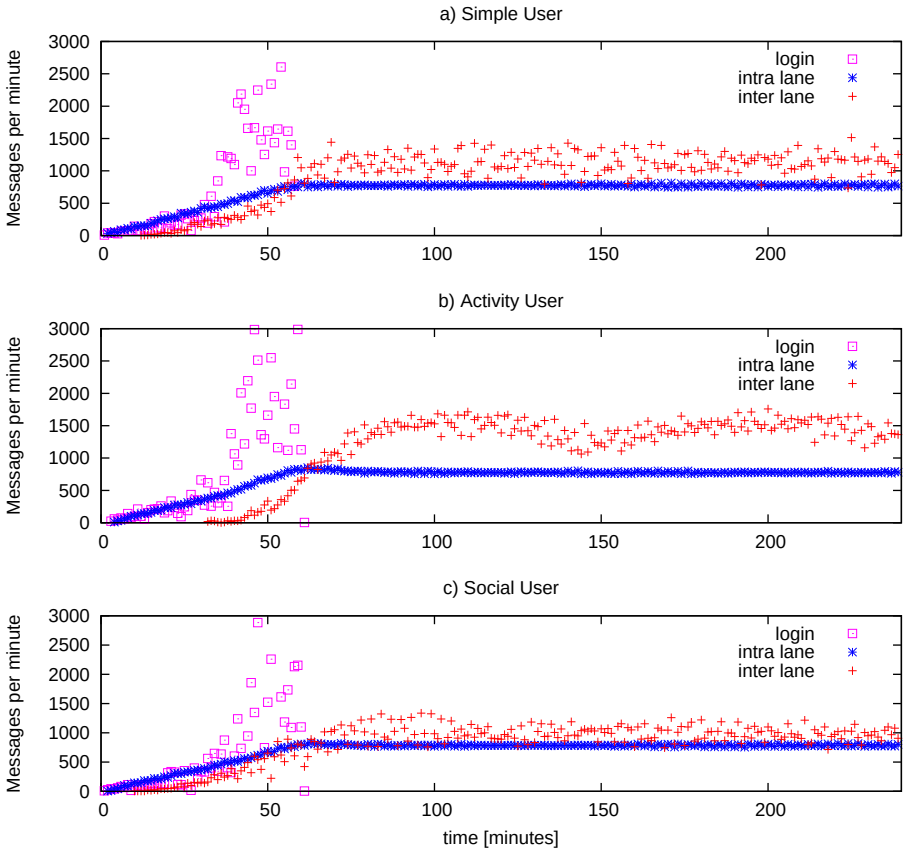


Figure 11: Number of network message with different user models.

[7] J. M. Galan and L. R. Izquierdo. Appearances can be deceiving: Lessons learned re-implementing axelrod's 'evolutionary approach to norms'. *Journal of Artificial Societies and Social Simulation*, 8(3), 2005.

[8] M. Gierke. Coupling Autominder and James. Diplomarbeit, Universität Rostock, Jan. 2005.

[9] G. N. Gilbert. Environments and languages to support social simulation. In *Social Science Microsimulation*, pages 457–458, 1995.

[10] S. Hanks, M. E. Pollack, and P. R. Cohen. Benchmarks, testbeds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):17–42, 1993.

[11] J. Himmelspach. *Konzeption, Realisierung und Verwendung eines allgemeinen Modellierungs-, Simulations und Experimentiersystems - Entwicklung und Evaluation effizienter Simulationsalgorithmen*. Reihe Informatik. Sierke Verlag, Göttingen, Dec. 2007.

[12] J. Himmelspach and A. M. Uhrmacher. Sequential processing of PDEVS models. In A. G. Bruzzone, A. Guasch, M. A. Piera, and J. Rozenblit, editors, *Proceedings of the 3rd EMSS*, pages 239–244, Barcelona, Spain, Oct 2006.

[13] J. Himmelspach and A. M. Uhrmacher. Plug'n simulate. In *Proceedings of the Spring Simulation Multiconference*. IEEE Computer Society, March 2007.

[14] P. H. M. Jacobs, N. A. Lang, and A. Verbraeck. Web-based simulation 1: D-sol; a distributed java based discrete event simulation architecture. In *WSC '02: Proceedings of the 34th conference on Winter simulation*, pages 793–800. Winter Simulation Conference, 2002.

[15] M. Klein, M. Hoffman, D. Matheis, and M. Müssig. Comparison of overlay mechanisms for service trading in ad hoc networks. Technical Report TR 2004-2, University of Karlsruhe, Oct. 2004. ISSN 1432-7864.

[16] F. Klügl and F. Puppe. The multi-agent simulation environment SeSAm. In H. K. Büning, editor, *Proceedings des Workshops Simulation in Knowledge-based Systems*, volume tr-ri-98-194 of *Reihe Informatik*, Paderborn, April 1998. Universität Paderborn.

[17] S. Kurkowski, T. Camp, and M. Colagrosso. MANET simulation studies: The incredibles. *ACM's Mobile Computing and Communications Review*, 9(4):50–61, 2005.

[18] A. Martens and J. Himmelspach. Combining intelligent tutoring and simulation systems. In P. Fishwick and B. Lok, editors, *Proceedings of the International Conference on Human-Computer Interface Advances for Modeling and Simulation (SIMCHI'05)*, pages 65–70, New Orleans, USA, Jan. 2005. SCS, The Society for Modeling and Simulation International.

[19] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The SWARM simulation system: a toolkit for building multi-agent simulations. Technical report, Santa Fe Institute, June 1996.

[20] OMG. Unified Modeling Language: Superstructure version 2.1 (document ptc/2006-04-02). `http://www.omg.org/cgi-bin/doc?ptc/2006-04-02`, Apr. 2006.

[21] K. S. Perumalla. μsik: A micro-kernel for parallel/distributed simulation systems. In *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pages 59–68, Washington, DC, USA, 2005. IEEE Computer Society.

[22] M. E. Pollack and M. Ringuette. Introducing the Tileworld: Experimentally Evaluating Agent Architectures. In *AAAI-90*, pages 183–189, Boston, MA, 1990.

[23] M. Röhl. Platform independent specification of simulation model components. In SCS, editor, *ECMS 2006*, pages 220–225, 2006.

[24] M. Röhl, B. König-Ries, and A. M. Uhrmacher. An experimental frame for evaluating service trading in mobile ad-hoc networks. In *Mobilität und Mobile Informationssysteme (MMS 2007)*, volume 104 of *Lect. Notes Inform.*, pages 37–48, 2007.

[25] M. Röhl and A. M. Uhrmacher. Flexible integration of XML into modeling and simulation systems. In *Proceedings of the 2005 Winter Simulation Conference*, pages 1813–1820, 2005.

[26] M. Röhl and A. M. Uhrmacher. Composing simulations from xml-specified model components. In *Proceedings of the Winter Simulation Conference 06*, pages 1083–1090. ACM, 2006.

[27] M. Röhl and A. M. Uhrmacher. Composing simulations from XML-specified model components. In *Proceedings of the Winter Simulation Conference*, pages 1083–1090. ACM, 2006.

[28] Sun. Java architecture for xml binding (JAXB). `http://java.sun.com/xml/jaxb`, 2005.

[29] D. S. Tan, S. Zhou, J.-M. Ho, J. S. Mehta, and H. Tanabe. Design and evaluation of an individually simulated mobility model in wireless ad hoc networks. In *Communication Networks and Distributed Systems Modeling and Simulation Conference 2002*, San Antonio, TX, 2002.

[30] A. Verbraeck. Component-based distributed simulations. the way forward? In *Proceedings of the 18th Workshop on Parallel and Distributed Simulation (PADS'04)*, pages 141–148, 2004.

[31] W3C. State chart XML (SCXML): State machine notation for control abstraction. `http://www.w3.org/TR/2007/WD-scxml-20070221`, 2007. W3C Working Draft 21 February 2007.

[32] B. Zeigler, H. Praehofer, and T. Kim. *Theory of Modeling and Simulation*. Academic Press, London, 2000.

Agent Programming in Practise — Experiences with the JIAC IV Agent Framework

Benjamin Hirsch
Benjamin.Hirsch@dai-labor.de

Stefan Fricke
Stefan.Fricke@dai-labor.de

Olaf Kroll-Peters
Olaf.Kroll-Peters@dai-labor.de

Thomas Konnerth
Thomas.Konnerth@dai-labor.de

DAI Labor
Technische Universität Berlin

ABSTRACT

This paper describes the agent framework JIAC IV, and the various projects it has been applied in. The projects were industry-funded as well as research-oriented. Our aim is to convey the particular requirements that followed from this link, and the consequences for the further development of the framework. We describe the projects and the impact that JIAC IV had on them, and conclude with an analysis and the current state of affairs.

1. INTRODUCTION

Agent technology has been around for a number of years now, and the still growing number and size of the various workshops and conferences in the field show that the importance of agents and related technologies is high. It is however also the case that even though a large number of researchers and practitioners work on agents, a much smaller number has managed to breach the wall between academia and industry. There are commercial frameworks available, and companies advertise the fact that they employ agent technology, but in general it can be said that the impact that agent technology has had on the market is less that it could have been. Reasons given for this failure — or even whether or not to call it a failure — vary wildly. We maintain that one of the reasons is the missing interaction between agent researchers and industry, sometimes leading to solutions that are not applicable or usable in "real" applications.

JIAC IV (Java Intelligent Agent Compontentware) is an agentframework that has initially been financed by Deutsche Telekom, and was from the very beginning designed to cope with industry requirements. In a number of projects of different domains, the framework has been adapted and further refined.

This paper gives an overview over the JIAC IV agent framework, with a high level view of the features (Section 2), and proceeds to describe various projects that have been implemented using JIAC IV (Section 3). The paper concludes with a discussion of the experience made and the lessons learned during those projects (Section 4).

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

2. JIAC IV

JIAC IV has first been developed in 1998 [7] and has seen a number of revisions since. Its focus was initially on supporting telecommunication applications, but it quickly got used in areas as different as information retrieval, telematic services, and personal information.

The JIAC IV agent framework supports the development of multi-agent systems (MAS) using BDI agents on FIPA [15, 16] compliant platforms. JIAC IV has been implemented using the Java programming language. Two building blocks constitute the basic agent architecture: the JIAC component system [25] and the JIAC Agent Description Language (JADL) [20]. The basic architecture of a JIAC-based application is summarised in the JIAC MAS meta-model, which is shown in Figure 1.

Based on this core architecture, the JIAC-framework also includes a list of enhanced features that cover the management and security aspects as well as the software lifecycle, i.e. engineering, deployment and runtime. All of these will be discussed in the following sections:

2.1 Agent and Service-model

As can be seen in the JIAC MAS meta-model (Figure 1), JIAC has explicit notions of goal, rule, plan, service, and protocol. Each JIAC agent constitutes it's own component system and is able to accommodate any number of different components that implement either the agents' behaviour or the agents' capabilities. While some of these components are implemented using Java-beans, our notion of components also covers concepts like ontologies or scripts which are written in the language JADL [20].

2.1.1 Knowledge

The central anchor for this component based approach is the agents factbase, which stores the agents world model and is accessible from all components. The knowledge stored in this factbase is represented by ontologies that describe the data and its relations. While these ontologies described in JADL are less expressive than OWL-lite, they can be mapped to a subset of OWL [22].

Furthermore, as JIAC-agents are intended to operate in open and dynamic environments, their knowledge representation is based on 3-valued logic [19] and therefore allow to reason with uncertainty. Consequently, JIAC-agents operate with an open world model.

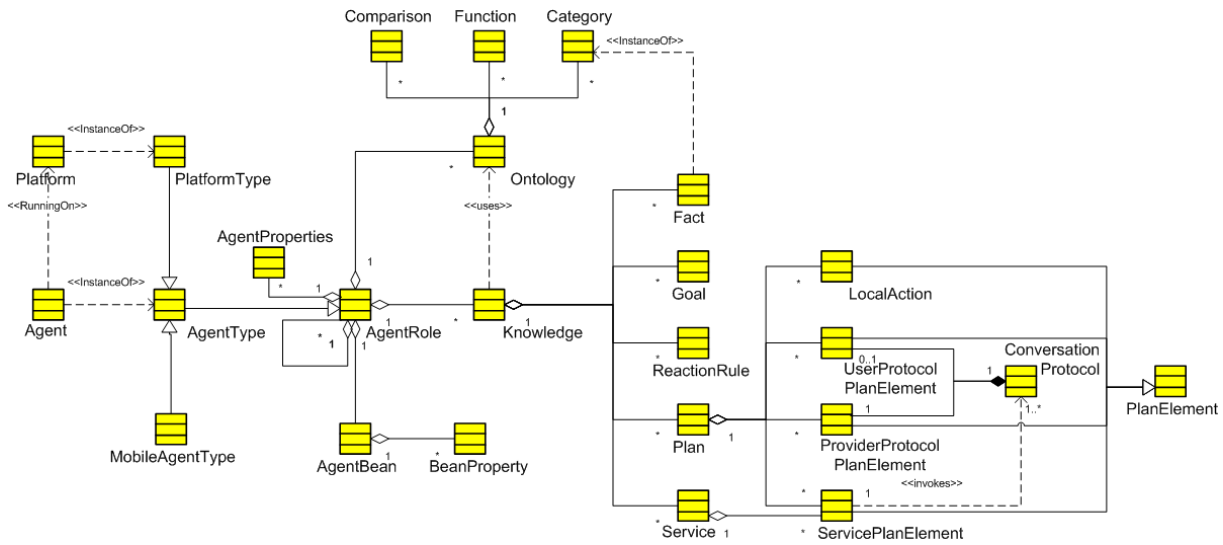


Figure 1: JIAC IV MAS meta-model

2.1.2 *Acting*

Based on this knowledge representation and its factbase, each agent employs a version of a BDI-cycle [8] in which its state-goals are evaluated against the factbase and appropriate capabilities are selected and triggered as necessary. These capabilities are called plan-elements and come in different versions. They can be local actions as well as services and they can be implemented by either a JavaBean or in JADL. Furthermore, plan-elements are allowed to invoke other plan-elements. However, as the system transparently looks up services from other agents if there is no local plan-element that can fulfil the goal, the difference between these plan-elements is more a matter of the preferred implementation language and has no effects on the runtime behaviour. Additionally, JIAC-agents can also deal with simple rules that provide a quick reaction to changes in the environment [20].

2.1.3 *Semantic service selection*

To support the dynamic and flexible selection of plan-elements, the action- and service-descriptions all contain semantic descriptions of preconditions and effects based on the JADL-ontologies. This allows the agent to verify both, whether an action is applicable and whether it reaches desired effect. Furthermore, it enables a semantic service matching, so that available services may be selected dynamically, without prior adaptation from the developer.

As an enhancement to this action selection and the BDI-cycle, JIAC agents support planning from first principles — using a UCPOP algorithm [24] they can, if no appropriate plan-element can be found, try and build a chain of services that will lead them to the desired state of affairs. Note that this goes further than just providing a number of pre-defined plans that the agent can choose from. The agent's execution will first try and find plans or services that match his goal, and only if it does not find any it will start planning.

2.1.4 *Communication*

As we have already mentioned, JIAC employs service-based interaction. More specifically, while JIAC uses FIPA compliant speech acts for communication, the communication between two JIAC-agents is always handled via service calls. Every service call is wrapped in a so called meta-protocol (which is essentially a modified *FIPA Request* protocol), which automatically takes care of security requirements, error handling, data exchange and negotiation protocols between agents. The actual exchange of messages between agents during one service session however is not restricted, as long as both agents can agree on a common protocol.

While service provision always occurs between two agents, the meta protocol also allows to precede the actual provision with a provider selection, where the service user can request a service from many agents and use any suitable protocol such as for example the contract net protocol [26] to select the most suitable provider. It can here use the semantic service description that each service has, including pre-condition, effect, and QoS information.

2.2 Infrastructure - Management and Security

Based on the core features we explained in the previous section, JIAC employs a list of advanced infrastructure functionalities which have proven differently useful in application development:

2.2.1 *Dynamic Components and Mobility*

One of the signature features of JIAC IV is its ability to exchange components during run-time. As stated earlier, components may be actions, services, rules, ontologies, but also so-called agent beans, i.e. Java components.

Furthermore, JIAC agents support strong mobility, meaning that they can be moved between hosts during run-time without having to stop execution — this presupposes however that the agents does not need software or services such as databases that are only available at certain hosts.

These two ability of the framework allows the quick adaptation of systems and applications, as the agents can automatically find and use new components, or can independently move to other hosts to e.g. reduce communication.

2.2.2 Security

While the dynamicity that follows from the component exchange and the strong mobility of agents can be quite powerful, it is also a potential security threat in real world systems. Therefore, JIAC contains elaborate security mechanisms on all levels. For example, it provides a public key infrastructure interface to authenticate agents, and supports encrypted communication between agents [9].

The Framework has been certified by the German Federal Office for Information Security (BSI) following the Common Criteria Level 3 [10]. This makes it worldwide the only agent framework to be security certified.

2.2.3 Accounting Features

As a further extension to the service mechanism, JIAC IV provides a management interface through which complex accounting and tariff schemes can be implemented [18]. This is important when agents move into the mainstream and offer services to other agents, and want to provide intelligent and dynamic billing methods. The accounting features have been tested in the course of the project Berlintainment [28].

2.3 Tools and Methodology

One last, but nevertheless important aspect of JIAC is its iterative methodology (as shown in Figure 2) that supports large projects by not only defining analysis and design but also including methods to deal with customers and team interaction. The methodology has been applied and refined in numerous projects that we have done at the DAI Labor with industrial and research partners.

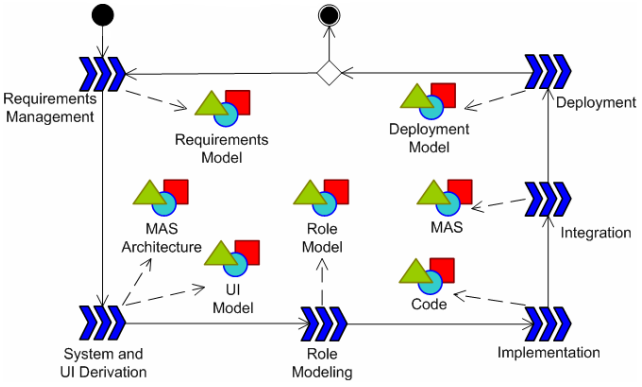


Figure 2: JIAC methodology - iterative and incremental process model in SPEM notation.

Supplementary with this methodology, JIAC comes with its own IDE which is implemented as Eclipse[1] plugin [27]. It provides textual and graphical editors for all parts of agent programming, including an ontology editor, an agent role editor, a knowledge builder, a repository for agents, and more. The IDE has extensive help functionality and has a built in tutorial to help the user get acquainted with JIAC and JADL.

3. PROJECTS

During last ten years we have used JIAC in many research projects in different domains such as personalised information services, security, healthcare, entertainment, among others. In the following, we describe some of these projects and the experiences we made.

[1] `http://www.eclipse.org`

3.1 URLAUB

The goal of the project URLAUB (German for holiday), running from September 2001 until August 2002, was a personalised system for for holiday planning. Unlike traditional booking systems, URLAUB created recommendations based on user preferences for climate, lodging, activities, and so on. These preferences were stored in a profile which was then used for providing future recommendations as well as for comparisons with other users' profiles. An agent-based approach was taken and combined with advanced filtering techniques in order to integrate a number of different travel agencies and related travel information sources, such as climate data, politics, and cultural or sports characteristics. Additionally, a device independent user interface, adaptive user management, the ability to pre-book, as well as feedback mechanisms set the system apart from standard travel booking sites.

3.1.1 Setup and Methodology

The task-oriented JIAC methodology was applied to the analysis and design, giving rise to seven task areas with a total of about 90 identified tasks, comprising, among others, the access and monitoring of remote sources, the managing of user profiles, user interfaces and devices, the processing of travel requests, information filtering as well as notification tasks. 17 roles were identified and subsequently mapped onto agent types. Each agent type combined a number of related tasks, identified through analysis of interaction between tasks. Non-functional requirements such as availability and reaction time was also taken into consideration. A total of 65 services where identified and assigned to 13 agent types. During deployment, several instances of each agent type were created in order to avoid bottlenecks, and critical functionality was distributed over several platforms running on different servers. In the first release, each user was assigned a unique user agent managing her preferences and recommendations.

3.1.2 Lessons Learned

The usage of agents to model the domain lead to a scalable and manageable system. New users are added by creating new user agents for them. The flexible service delivery schema made it easy to install redundant filtering agents for efficiency purposes being necessary after the number of users increased from 20 in the beginning to more than 200. However, with an increasing number of users we noticed a significant slow down in the overall answer behaviour of the system. This was mainly caused by a huge number of personal agents (one per user) that were running but inactive most of the time. The intuitive decision to provide one agent for each user proved to be a rather problematic design decision.

By suspending non-active agents automatically, the load on the system could be reduced to a minimum. The early overhead of modelling travel and profile data was compensated as soon as the system has been expanded with new features, travel agents, and information sources.

Another issue was the connection between user interfaces and the underlying system, as the user interface was event

oriented rather than goal oriented, and the synchronisation of the two models proved difficult.

3.2 PIA

PIA (Personal Information Agent) [4] is an ongoing project that started in 2003 with the aim to create a complete agent based solution for the personalised provision of information and news. Depending on an individual's personal interests and habits the systems provides relevant information at the right time. For example, it provides news in the morning and entertainment and event related information in the evening. Different filtering methods such as content-based and collaborative filtering [6] were implemented using agents. According to the particular needs, information is brokered to users using SMS, MMS, e-mail, or HTTP.

3.2.1 Setup and Methodology

The complexity was broken down into three tiers, one for the task of information extracting from different — typically external — sources, the other for information filtering, and the third for user personalisation and the provisioning of information. A highly distributed system with hundreds of agents — most of them being extractor agents — running on approximately 30 agent platforms was built. The most important part was the co-ordination filtering framework which resides on the filtering layer. Filtering agents utilising different methods of content-based and collaborative filtering techniques compete against each other in order to fulfill the users' information needs. So-called filter manager agents are monitoring, and controlling these filtering agents depending on information about the system state and estimations about the most promising filtering strategies. Additionally, rewards are computed on the basis of historical data, significance, as well as the users' feedback. As a result, the fitness of a filtering agent concerning a given information request changes over time. This yields to an adaptation of the system over time [5].

3.2.2 Lessons Learned

The PIA project is still ongoing, and has had a number of major revisions since its inception. The first project, while showing the power of using different agents to implement a variety of filtering and extraction functionalities, also showed the weaknesses of the system, some of which were conceptual, others related to the engineering approach. For example, extraction and filtering algorithms were computationally so expensive that they practically shut down agent communication on their nodes, leading to erratic behaviour as agents would not receive messages in time and therefore cancelled service calls. Different strategies of using the power of the agent paradigm, including using JIAC's planning capabilities to find good filtering strategies were working, but too slow for a production environment. The reaction time of the system grew with the number of users using the system, and was unacceptable even with a low number of concurrent users (a dozen). Another bottleneck was the directory facilitator (DF) implementation, as the amount of read and write calls lead to the system literally taking hours to start up. The problem of scalability and responsiveness could be traced back a combination of generic Java problems with prioritising threads on the one hand, and the design of JIAC IV to make heavy use of the DF as well as exchanging a lot of (often redundant) data within the meta protocol.

Another important problem was related to the operational availability and determinism. On the one hand the adaptation mechanisms made the system intentionally nondeterministic — leading to a high probability that an information request executed at two different time points would lead to different results. But often obscure results occurred as a consequence of unavailable content sources or crashed agents. In order to manage the large amount of agents, management functionalities were implemented and integrated into the agents, allowing for example automatic re-start in case of long timeouts. Also, administrators may manipulate these managed entities by use of a management console.

The current system, while still based on the idea of distributed filtering and extraction modules, is strongly limited in the number of agents. The dynamic delegation of filtering tasks to suitable agents in combination with agent mobility for load balancing purposes proved to be unsuitable for real-time behaviour. Therefore, once the co-ordination schemes were well understood, we concentrated related functionalities into one agent. Around 50 extractor agents perform the most crucial work of updating the content database which is now located on one server. All the filtering tasks are now controlled by one filtering agent which has the knowledge to decide which of the methods to apply for a given information request. I.e., all co-ordination that has been subject to inter-agent co-ordination mechanisms in the versions before, are now subject to internal control of an agent.

3.3 Common Criteria Certification

In order to establish whether the security features of JIAC are indeed industry grade, we did a security certification following the Common Criteria [1, 2, 3]. The Common Criteria are a worldwide accepted security standard in information technology. A defined process is used to check for security holes and issues. The security features are checked by independent reviewers and certified by an accredited institute. JIAC IV has been tested for security leaks by reviewers of T-Systems and certified by the*Bundesamt für Sicherheit in der Informationstechnik*, the federal office of IT security. The fact that the whole process took about two years and cost more than one million euros gives an idea of the scope of the certification process.

The security certification of JIAC IV was special in two respects. On the one hand, our research results had to be adapted to industry standards. On the other hand, it was a learning experience not only for us but also for the reviewers and certification institution because it was the first time that they had to certify a product with generic security features, rather than a concrete application.

3.3.1 Setup and Methodology

During a certification process, the reviewers examine the target product according to the criteria defined by the Common Criteria. They examine weaknesses within the functionality of the system, but also within the development environment and lifecycle of the product. The criteria are identical for all product classes and only differ in the evaluation level (also called EAL). The EAL defines the level of detail with which the criteria have been examined (For example, checking source code versus only checking use-cases). The choice of evaluation level reflects how detailed the evaluation has been, and has to be mentioned together with the certification. In the case of JIAC IV we chose EAL 3. This

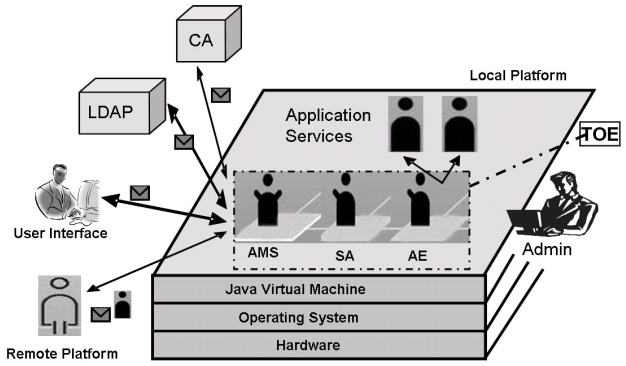


Figure 3: The target of evaluation

means that the reviewers executed tests independently from us, and checked the platform as well as the development environment for weaknesses on a very detailed level.

Because the certification process examines all parts of the product for weaknesses, we used a scaled down version of the agent framework, as shown in Figure 3. If we had not reduced the functionality of the framework, the effort needed to execute the certification would have been too much, not only for us but also for the reviewers and certification authority. This however was the result of the two-year long process of the certification, as we originally had planned to certify the whole framework. While the reduction was made in order to make the associated costs manageable, the effort involved in scaling down the framework was quite extensive too. For example, some documents describing features like high-level design, security targets, and more, had 300 iterations before the reviewers where content. The large amount of iterations between us and the reviewers was the result not only of the adaptation of the target platform, but also the problem of adapting the fairly high-level descriptions of the Common Criteria to our framework. Generally, only static security targets are certified, so that the dynamic aspect of JIAC IV was something that forced the reviewers to adapt the criteria.

3.3.2 Lessons Learned

There are a number of remarks that need to be made here. On the one hand, it appeared that the requirements of security targets in the "real world" are not easily adapted to a research environment. On the other hand, many processes generally used had to be adapted to cater for the concept of generic security, as they did not cover the special cases. For example, the requirement of having tests cover 100% of the code is simply not possible with generic security [17]. Generally however we can state that the security functionality of our framework has been certified according to internationally accepted standards. Also some of the requirements that are given in an industry setting but that do not necessarily make sense in a research environment have been adapted by our institute. For example, we learned that extensive tests do not necessarily mean an increase in time for the implementation, but can actually shorten the development time and make the product more stable.

In summary, we learned the following from the certification process:

- The JIAC IV agent framework can cope with international industry standards and requirements.
- The transfer of research results into industry was quite hard.
- Commercial security requirements can be met by agent frameworks.
- Common processes and best practices of quality assurance make sense and support a product design that has an industrial focus.

3.4 Nessi

A currently running project of our lab is the Network Security Simulator (Nessi) [11]. In the context of this project, devices within a large telecommunications network and possible threats are simulated. Using the simulator, the behaviour of attackers, threats, and possible counter measures can be evaluated.

3.4.1 Setup and Methodology

While the initial model used one agent per device for the simulation, the current implementation maps subnets onto agents. This change in the design was necessary to allow the system to scale to large networks of thousands of devices. Currently we simulate systems with about 100 subnets and about 3500 single devices.

One chief advantage of using agents in this project is the ability to simulate life cycles of the devices very easily. Furthermore, the abilities of agents support the requirements of the modelling of the simulated devices.

3.4.2 Lessons Learned

Depending on the target application, the "obvious" mapping of devices to agents is not necessarily the smartest one. Using aggregations allows to scale the system without loosing the advantage of agents or having less usable results.

4. CONCLUSION

There were a number of other projects implemented using JIAC IV, the ones above give a reasonable overview over the type of projects, as well as the experience we made. In this section we will try and summarise the pros and cons of using agents in general, and JIAC IV in particular.

One recurring theme that maybe did not come out clearly in the project descriptions was the disconnect between designing the system, and providing convincing and easy to use user interfaces (UI). There are two issues here that need to be addressed. The first is that for industrial projects, the presentation layer is often as important as the underlying framework — it is therefore not sufficient to provide functional but unpolished interfaces. This is nothing particular to agent based systems (other than sometimes the lack of responsiveness if a web-query requires different services to be executed. While the communication overhead does contribute to the high latency, it often comes with the territory of complex applications.) The second issue is more relevant to agents, and pertains to the actual human-computer interaction. Within a system here different agents interact and require user input in different situations, where users should take advantage of the flexibility of the system, providing a UI that allows for this flexibility while being easy and intuitive to use is a very difficult problem. Different paradigms,

namely the task-oriented versus the goal oriented paradigm clash, and there are no easy ways to combine the two.

Another common complaint was the need to use JADL to program agents. Developers that were not familiar with the agent concept tended to write trivial services and plans, and instead moved the system logic into agent beans, thereby effectively circumventing advantages that JIAC IV provided in lieu of being able to program in their familiar language Java. We have tried to make the use of JIAC as simple and intuitive as possible by providing powerful tools, but the tools do not take away the fact that developers need to understand the agent- and service paradigm and be familiar enough with logical expressions to design pre-conditions and effects.

When implementing projects with dozens or hundreds of agents running on a number of platforms, another issue showed. While the distribution of agents allowed for scalable and flexible systems, the problem of deployment — getting everything running in the first place — was and is an often neglected problem. In most of our projects we put a lot of effort into integration and deployment management to ensure that any system built would be executable on the target system. Typical issues include the provision of all relevant dependent libraries (often different developers would use slightly different and slightly incompatible versions of the same library), adapting the system to the target network (including the configuration of firewalls and subnets), and the compatibility across different operating systems. Another side effect of large systems was that it became a challenge to keep track of running and dead agents over different platforms. Eventually we implemented a platform monitor that allowed the monitoring of agents.

Another issue in the context of scalability is the application of methodologies to the problem at hand. While often it is straightforward and elegant to map entities directly to agents, we found that when scaling application to serve hundreds or thousands of users, or incorporate a large number of functionalities, scalability broke down (in terms of number of agents as well as communication overhead). In a number of projects we therefore ended up mapping aggregations of entities to agents in order to keep the system load manageable.

One very prominent feature of JIAC IV, the ability to exchange components of agents during run-time, was never used. Also the planning component, that allowed agents to build plans from first principle never made it into a running project, mainly because the requirements were known, and there was never enough time to implement alternative service solutions that would have been needed to actually find alternative plans. We did however employ the planning in small example projects.

It must be said however that in spite of all the mentioned issues and problems, we found that JIAC IV has met industry standards and has successfully been applied in many different domains, ranging from healthcare and entertainment to information retrieval and personalisation to simulation and high security applications.

4.1 JIAC TNG — the Next Step

Based on our experience of using JIAC IV in large industry-driven projects, we are currently working on the next major release. There, a number of changes are introduced. First of all, we defined a number of agent types of different complexities to cater for the wish of being able to use simple lightweight agents together with intelligent goal-oriented agents in one system. One of the reasons for performance issues in our projects was the fact that even for simple reactive tasks, a developer had to employ a goal oriented agent, which of course required more resources than necessary. Additionally, we will allow a simple message based communication as an alternative to the service metaphor, because even though the meta-protocol proved quite useful, it was too much overhead for certain situations (e.g. simple inform-type messages).

Furthermore, we want to rework some of the core implementations of JIAC in order to achieve better stability and performance. While we think that the concepts and ideas of JIAC IV are mostly still good and appropriate, much has happened on the technical level. There are for example new libraries for component systems or threading, and we think that these will greatly improve the stability of our agent platform.

Our experience in recent projects is that a lot of emphasis is put on the aspect of composability of functionality on a very abstract level. Following the service oriented architecture approach [14] and business process modelling (for an overview see e.g. [21]), we are working on a mapping from BPMN (Business Process Modelling Notation) [23] to our agent framework [13, 12] in order to allow for high-level abstract design of an agent system that later can automatically be transformed into a (more or less) running system. Three current projects explicitly focus on the "easy service creation" and the support of powerful tools to connect similarly powerful services to a workflow. Goal is to allow for a modelling of systems on a level that is abstract enough to be easily understandable yet flexible enough to be scalable and adaptive.

While we do emphasise the workflow aspect of applications, we do see the need to goal orientation, and are currently working on a language that allows the seamless integration of services, semantics, and goal oriented programming.

5. REFERENCES

[1] Common Criteria, part 1: Introduction and general model, version 2.1, Aug 1999.

[2] Common Criteria, part 2: Security functional requirements, version 2.1, Aug 1999.

[3] Common Criteria, part 3: Security assurance requirements, version 2.1, Aug 1999.

[4] S. Albayrak. The role of AI in shaping smart services and smart systems. In *KI 2007: Advances in Artificial Intelligence*, volume 4667 of *LNAI*, page 1. Springer, 2007.

[5] S. Albayrak and D. Milosevic. Situation-aware coordination in multi agent filtering framework. In C. Aykanat, T. Dayar, and I. Korpeoglu, editors, *The 19th International Symposium on Computer and Information Sciences (ISCIS 04), Antalya, Turkey*, volume 3280 of *LNCS*, pages 480–492. Springer, 2004.

[6] S. Albayrak and D. Milosevic. Strategy coordination approach for safe learning about novel filtering strategies in multi agent framework. In D. Yeung et al., editors, *Advances in Machine Learning and Cybernetics*, volume 3930 of *LNAI*, pages 30–42. Springer Berlin, 2006.

[7] S. Albayrak and D. Wieczorek. JIAC - an open and scalable agent architecture for telecommunication applications. In S. Albayrak, editor, *Intelligent Agents in Telecommunications Applications - Basics, Tools, Languages and Applications*. IOS Press, Amsterdam, 1998.

[8] M. E. Bratman. *Intentions, Plans, and Practical Reason*. Havard University Press, Cambridge, MA, 1987.

[9] K. Bsufka. *Public Key Infrastrukturen in Agentenarchitekturen zur Realisierung dienstbasierter Anwendungen*. Phd thesis, Technische Universität Berlin, 2006. `http://nbn-resolving.de/urn:nbn:de:kobv:83-opus-13536`.

[10] Bundesamt für Sicherheit in der Informationstechnik. BSI-DSZ-CC-0248-2005 for Java Intelligent Agent Componentware IV Version 4.3.11 from DAI-Labor Technische Universität Berlin. Certification Report BSI-DSZ-CC-0248-2005, Bundesamt für Sicherheit in der Informationstechnik, Jan 2005.

[11] R. Bye, S. Schmidt, K. Luther, and S. Albayrak. Application-level simulation for network security. In *First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SimoTools)*, 2008.

[12] H. Endert, B. Hirsch, T. Küster, and S. Albayrak. Towards a mapping from BPMN to agents. In J. Huang, R. Kowalczyk, Z. Maamar, D. Martin, I. Müller, S. Stoutenburg, and K. P. Sycara, editors, *Service-Oriented Computing: Agents, Semantics, and Engineering*, volume 4505 of *LNCS*, pages 92–106. Springer Berlin / Heidelberg, 2007.

[13] H. Endert, T. Küster, B. Hirsch, and S. Albayrak. Mapping BPMN to agents: An analysis. In M. Baldoni, C. Baroglio, and V. Mascardi, editors, *Agents, Web-Services, and Ontologies Integrated Methodologies*, pages 43–58, 2007.

[14] T. Erl. *Service-Oriented Architecture:Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall, Indiana, USA, August 2005.

[15] Foundation for Intelligent Physical Agents. FIPA Agent Communication Language Specifications, 2002.

[16] Foundation for Intelligent Physical Agents. Interaction Protocol Specifications, 2002.

[17] T. Geissler and O. Kroll-Peters. Applying security standards to multi agent systems. In *Proceedings of the First International Workshop on Safety and Security in Multiagent Systems, Sasamas'04, AAMAS'04*, 2004.

[18] J. Keiser, B. Hirsch, and S. Albayrak. Agents do it for money — accounting features in agents. In M. Dastani, A. E. F. Segrouchni, A. Ricci, and M. Winikoff, editors, *ProMAS 2007 Post-Proceedings*, volume 4908 of *LNAI*, pages 44–58. Springer Berlin/Heidelberg, 2008.

[19] S. C. Kleene. *Introduction to Metamathematics*. Wolters-Noordhoff Publishing and North-Holland Publishing Company, 1971. Written in 1953.

[20] T. Konnerth, B. Hirsch, and S. Albayrak. JADL — an agent description language for smart agents. In M. Baldoni and U. Endriss, editors, *Declarative Agent Languages and Technologies IV*, volume 4327 of *LNCS*, pages 141–155. Springer Berlin / Heidelberg, 2006.

[21] F. Lautenbacher and B. Bauer. A survey on workflow annotation & composition approaches. In M. Hepp, K. Hinkelmann, D. Karagiannis, R. Klein, and N. Stojanovic, editors, *Semantic Business Process and Product Lifecycle Management. Proceedings of the Workshop SBPM*, volume 251 of *CEUR-WS*, 2007.

[22] D. Martin, R. Hodgson, I. Horrocks, and P. Yendluri. Owl 1.1 web ontology language, 2006. `http://www.w3.org/Submission/2006/10/`.

[23] Object Management Group. Business Process Modeling Notation (BPMN) Specification. Final Adopted Specification dtc/06-02-01, OMG, 2006. `http://www.bpmn.org/Documents/OMGFinalAdoptedBPMN1-0Spec06-02-01.pdf`.

[24] J. S. Penberthy and D. Weld. UCPOP: A sound, complete, partial-order planner for ADL. In *Proceedings of Knowledge Review 92*, pages 103–114, Cambridge, MA, October 1992.

[25] R. Sesseler and S. Albayrak. JIAC IV - an open, scalable agent architecture for telecommunications applications. In *Proceedings of the First International NAISO Congress on Autonomous Intelligent Systems (ICAIS 2002)*. ICSC Interdisciplinary Research, 2002.

[26] R. G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1114, Dec 1980.

[27] E.-O. Tuguldur, A. Heßler, B. Hirsch, and S. Albayrak. Toolipse: An IDE for development of JIAC applications. In *Proceedings of PROMAS08: Programming Multi-Agent Systems*, 2008.

[28] J. Wohltorf, R. Cissée, and A. Rieger. Berlintainment: An agent-based context-aware entertainment planning system. *IEEE Communications Magazine*, 43(6):102–109, June 2005.

Resource Coordination Deployment for Physical Agents

Bianca Innocenti*
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
bianca.innocenti@udg.edu

Beatriz López
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
beatriz.lopez@udg.edu

Joaquim Salvi
Institut d'Informatica i Aplicacions
Universitat de Girona
Girona, Spain
joaquim.salvi@udg.edu

ABSTRACT

When developing a multi-agent architecture for controlling a single robot, as the one described here, agents share resources and a coordination mechanism is required to solve possible resource usage conflicts. Moreover, some the resources involved in a robot act in the physical world, and the agent exchange on the resource usage can cause some disruptions on the physical robot behavior. So, in addition to coordination method, a mechanism to effectively implement the resource exchange from one agent to another one is needed. In this paper we present a methodology to achieve a resource exchange among different agents that mitigates any possible disruption in the physical robot behavior. The methodology comprises a Sugeno fuzzy system to determine the time window interval required to perform an smoothing change. Our methodology has been tested in a Pioneer 2DX of ActivMedia Robotics.

Keywords

Distributed coordination, physical resource exchange, autonomous robot architecture

1. INTRODUCTION

There is an increasing interest on the use of Agent Technology for developing robots as physical agents [14, 15, 19, 23]. In the majority of the approaches a multi-agent system model a collection of robots. There are some examples in building multi-agent systems for controlling a single robot, too. For example, in [20] a multi-agent architecture is proposed to control a single robot in which two types of agents are distinguished: elemental agents, with basic skills, and high-level agents, responsible for integrating and coordinating various elemental agents. All the agents in the Neves approach deal with reactive robot global capacities. In [1], a multi-agent system is proposed for the navigation system, in which five agents (map manager, target tracker, risk manager, rescuer, and communicator) are coordinated by means of a bidding mechanism to determine the action to be carried out.

Multi-agent approaches facilitate the robot development with higher abstract level modelling capabilities than traditional single-agent, module-based approaches [23]. Today's robots still exhibit simple behaviors compared with humans, and their ability to perform high level reasoning and cooperation is limited [19]. AI has matured enough to offering planning techniques, adaptation and learning methods. Modelling all such cognitive capabilities in a single agent-based robot architecture seems an unfeasible task, and the multi-agent approach offers a new way of combining such AI techniques. A multi-agent architecture provides modularity, distributedness, flexibility and robustness: agents can be added, changed or modified without caring about the other agents [16].

As any robot architecture, resource usage is a key point. While most of the traditional single-agent approaches centralize their use, multi-agent technology offers both centralize and decentralize methods for resource sharing. For example, Neves [20] follows a centralized approach while we follow a decentralized one. In any case, a coordination mechanisms is required so that agents use the shared resources without conflict.

The particularity of the robot case, however, remains on the fact that we are dealing with physical resources, and special attention should be paid when the resource changes from one agent to another one. That is, let us suppose that an agent A is using the robot motor resource, maintaining the robot at a fast speed to the North coordinate. Then, a second agent B, after coordination, gets the robot motor resource. Agent's B goal is to move the robot towards the West and at a low speed. Thus, changing the motor resource from A to B could cause a disruption on the robot action. For example in the experiment shown in Figure 1 there are two agents governing the robot's movement, the avoid obstacle agent (avoid) and the go to a point agent (goto). The robot's initial position is $(x_i, y_i) = (0, 0)m$ and the destination point is $(x_g, y_g) = (5, 0)m$. The trajectory described by the robot is shown, as well as the agent that is controlling the robot movement during the trajectory execution. We can see an abrupt robot movement as a consequence of the resource exchange between the two agents of the architecture at $x = 2.4m$ approximately (see circle in the figure). Of course this could be necessary in dangerous situations, but probably this is not a desired behavior in most of the cases. More often, the agent resource exchange should not affect the robot performance.

*This work was partially supported by the Spanish MEC Project DPI2006-09370 and by the DURSI Automation Engineering and Distributed Systems Group, 00296.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

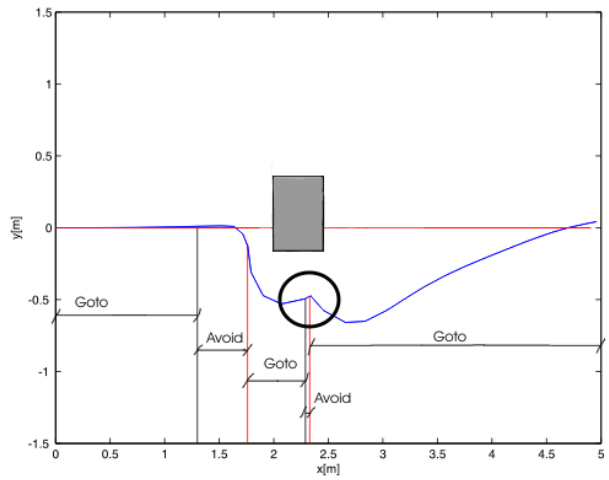


Figure 1: Example of abrupt resource control exchange.

From our understanding, there is a current gap between agent theory and agent implementation regarding this issue of deploying coordination agreements in the physical world. Particularly in [12] we present the multi-agent robot architecture that provides a coordination mechanism to deal with shared resources, but suffers from such a problem. The contribution of the current paper is to introduce a resource exchange methodology in order to cover this gap and that complements and improves our previous work. Regarding theory, we have followed the MaSE methodology to describe our architecture.

This paper is organized as follows. First, we describe in section 2 the multi-agent architecture we have used for physical agents as mobile robots. Then, in section 3 we explain the coordination deploy methodology to avoid abrupt robot behavior changes. In section 4 we show the results we have obtained when implementing the method on a multi-agent architecture for controlling on a Pioneer 2DX of ActivMedia Robotics. Finally, we end with some related work and conclusions.

2. AUTONOMOUS ROBOT MULTI-AGENT ARCHITECTURE WITH DISTRIBUTED COORDINATION

Our architecture, ARMADiCo, (Autonomous Robot Multi-agent Architecture with Distributed Coordination) has been designed to be a general purpose mobile robot architecture such that it can be applied to a wide range of mobile robots. In this paper we illustrate our architecture with the Pioneer 2DX robot but in a near future we will test it with other robots that are in our laboratory. The multi-agent framework allows an easy customization of the architecture to a given robot, by adding the appropriate agent that represents the real robot.

Agent coordination in the architecture is distributed, thus from the micro level (individual agents) emerge a macro level behavior (global system behavior) [3]. Engineering of emergence systems is still an open issue. Recent studies argue in favor of the use of a scientific methodology for their design [6]. This methodology distinguish two phases: theory and practice. Theory follows a top-down design: goals to roles and then to local behaviors. In this phase, current agent methodologies can help [3]. On the other hand, practice is a top-down process: from microscopic behavior, to phenomena and then to macroscopic behavior. Even though experimentation is the only known tool for this phase, several alternatives as information flows, design patterns and others have been proposed to understand the global system behavior [4].

From our experience, we have used the MaSE [5, 25] methodology in the first phase. MaSE is a UML based methodology that has some coincidences with the information flows approaches [4] in order to help in the second experimentation phase. MaSE proposes several steps in the definition of a multi-agent system. In Figure 2 the resulting agent class diagram is shown, as well as the messages sent from and received by each agent. This diagram does not include, for the sake of the figure complexity, the basic FIPA agents, as the Directory Facilitator (DF) agent, but they are considered in the architecture. Finally, when running our agents, we can check how, from the individual agent implementation arises the adequate robot behavior.

For the sake of length we focus on the remaining of this section on the description of two possible agent interactions with conflicting resources, and then on the explanation of the distributed coordination mechanism defined in the architecture.

2.1 Use cases

In order to illustrate the interactions of the agents in order to achieve a global robot behavior, we present two possible use case of the whole architecture, identifying some possible conflicting situations related to the utilization of shared resources.

2.1.1 *Case 1. Planning a robot movement.*

The task planning agent receives from the interface agent missions to achieve. In order to conduct them, it has a set of procedures similar to PRS [7], in which missions are decomposed into a set of tasks. Eventually, one of this tasks could involve the re-positioning of the robot. So, the task planning agent request to the path planning agent the optimal (in some aspect) trajectory from the current location to the desired one (see *RequestPath* arrow at the top of Figure 2). The path planning agent responds with the trajectory and the energy needed to reach the destination point (see *InformPathEnergy* arrow at the top of Figure 2).

Moreover, the task planning agent sends this information to the battery charger agent (see *RequestViability* arrow). In turn, this latter agent asks the path planning agent for a trajectory from the desired location to closest battery charger point (*RequestPath* arrow from battery charger agent to path planning agent). When the battery charger agent receives the answer from the path planning agent (*InformPathEnergy* arrow), it knows if there is enough energy for achieving the target position and returning back to the charger point (see *InformViability* arrow). In case there is not enough energy, the battery charger agent informs about the corresponding risk to the task planning agent. Thus, the task planning agent could change the task or not; however, in the latter case, it is assuming the possibility that the battery charger interrupts the task execution, as shown in the next scenario. Other alternatives, as merging trajectories

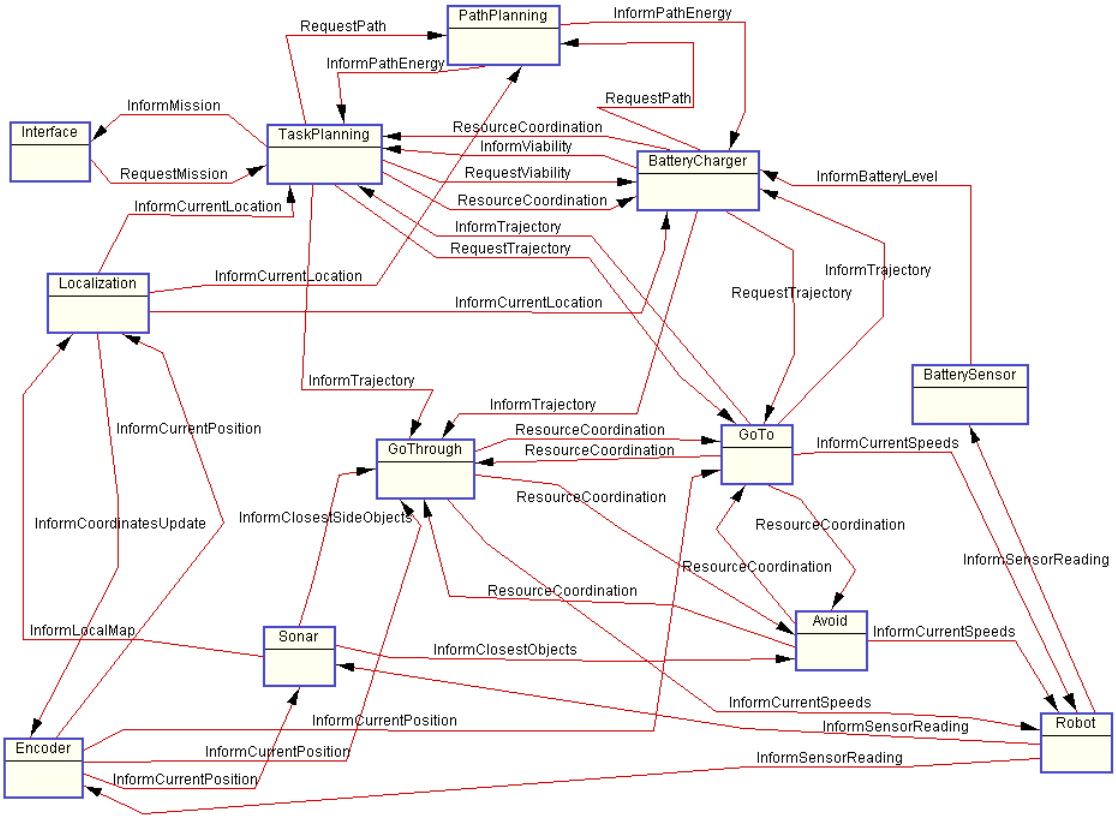


Figure 2: Agent Class Diagram. Boxes are agents and arrows indicates message flow (from an agent and to an agent).

that search a charger point while going to the destination point, are also considered.

As a summary, the task planning, battery charger and path planning are the agents involved in this scenario. Particularly, the path planning agent can be considered as a shared resource between the task and battery charge agents, and it provides the trajectory and energy required to move the robot to a destination point.

2.1.2 *Case 2. Moving the robot.*

Once the task planning agent knows the required trajectory T, to reach a destination position, it sends this trajectory to the goto and gothrough agents (see *InformTrajectory* and *RequestTrajectory* outgoing arrows of the task planning agent in Figure 2). This trajectory T is a sequence of points $p_1, \ldots, p_n$ to reach.

The goto agent, who knows the current position and heading (*InformCurrentPosition* arrow outgoing from the encoder agent), calculates the linear and angular velocities necessary to reach the first point of the trajectory, p_1. The gothrough agent does the same calculations but considering obstacles at the side of the robot (in the case that the robot is in a hallway or narrow place). On the other hand, the avoid agent is constantly looking for obstacles in front of the robot in order to dodge them. Then, after coordination (see *ResourceCoordination* arrow among these three agents), one of them, the winner, sends the computed commands to the robot agent (*InformCurrentSpeeds* arrow).

Once the robot has reached the first position of the trajectory p_1, the goto and the gothrough agents continue with the next one, p_2, and so on until the last point of the trajectory p_n is reached.

It could be the case, however, that when the goto agent is sending a request to the robot agent to move towards the p_i point, the battery charge agent realizes about the battery level is under the minimum security level required. This could happen, for example, when the robot has performed a larger trajectory than the planned one due to the presence of obstacles. Then, the battery charger requests a new trajectory T' to the goto agent in order to reach a charger position as soon as possible.

In these scenario, then, three agents (goto, avoid, gothrough) are sharing the same resource: the robot agent to which they send conflicting commands. In turn, the goto agent is a resource of the task planning and battery charger agents, since it can receive conflicting trajectory requests from them.

2.2 Distributed coordination

Coordination among agents is necessary when there are several agents trying to use the same robot resource at a given time. For example, a conflict can arise between the task planning and charge battery agents when trying to request different destination positions to the path planning agent (case 1) or when requesting different positions to the goto agent (case 2). Also, among the avoid, the goto and the gothrough agents when trying to request conflicting actions to the robot agent (case 2). Note then, that the path planning agent, the goto agent and the robot agent can be

considered shared resources, depending on the situation.

One solution to the problem is that the conflicting resources, being agents, solve by themselves the conflicting situation in a centralized way (with an auction, for example). However, we have adopted a decentralized approach in which the agents involved in the conflict decide which of them take the resource control. This decision avoids having a single agent arbitrating the overall architecture that could become a bottleneck when the number of agents is high.

So, in ARMADiCo, when an agent enters in the system, it provides to the DF agent information about the resources it uses. Then, the DF sends back to the agent the identification of the other agents using the same resources. Thus, any new agent in the system knows which are the set of agents to which it needs to coordinate in case of conflict.

To decide what agent wins the shared resource in case of conflict, each agent computes a normalized utility value (between [0,1]) regarding its actuation. The procedure to compute the utility is only known by the agent itself. The outcome of the computation, that is, the utility value, is the one used for coordination. The utility computation has been described in [12], and other approaches as for example [18] can also be followed.

In a given moment, there is only one agent that uses the resource in conflict. This agent sends its utility to the other agents who share the resource. When another agent in the architecture has a higher utility, it takes the control of the resource, and starts sending to the remaining agents the value of its utility. Thus, the agent who has a higher value of utility wins the resource.

For example, suppose the use case 2, in which the robot agent is the shared resource, that is currently controlled by the goto agent. This control agent is informing to the gothrough and the avoid agents about its utility. In a given moment, the goto agent has a utility value of 0.5, and the avoid agent of 0.7; being 0.7 the higher value. Then the avoid agent takes the control of the situation, and it is the only one that request some actuation to the robot agent.

After a coordination agreement is achieved (a new winner is decided), additional computation is also required to adequate the robot physical actuation that encompass the resource exchange between the two agents.

3. COORDINATION DEPLOYMENT METHODOLOGY FOR PHYSICAL AGENTS

When an agent, after coordination, obtains the opportunity to use a robot shared resource, the agent should proceed on the resource usage taking into account its impact on the physical world in a similar manner than control fusion [9, 12, 22]. For doing so, we propose a method based on the information used in the coordination process.

Let a_w be the agent that wins the resource, and let a_l be the agent that loses the resource (it has been using the resource until now). For example, if the shared resource is the robot agent, a_w could be the goto agent, and a_l could be the avoid agent. Let u_w and u_l be the utilities of a_w and a_l correspondingly (so $u_w > u_l$). In [12] details of the utility computation are given. Also in this previous work arises the problem of smoothing the resource exchange, which we are tackling in this paper.

Let a_r be the shared resource (agent that owns a shared robot resource). The resource is characterized by n parameters that configure the possible actions requested to the resource (for example, robot commands). Then let $a_{w_1}, \ldots, a_{w_n}$ be the actions requested by the winner agent and $a_{l_1}, \ldots, a_{l_n}$ the ones of the loser agent.

First of all, we need to determine the time window frame t_f available in order to perform the resource action exchange, that is, how much time we have to change from the current action of the loser agent to the required action of the winner agent. This time depends on the criticality of the robot state:

- The robot is changing from a critical to a non critical situation
- The robot is changing from a non critical situation to a critical one

How t_f is computed in both cases is shown below.

Then, a progressively change from each loser action a_{l_i} to each winner action a_{w_i} is performed according to the following expression (that extends the work presented by [9]):

$$\frac{\delta_w(t_i, u_w) * a_{w_i} + \delta_l(t_i) * a_{l_i}}{\delta_w(t_i, u_w) + \delta_l(t_i)} \quad (1)$$

where $\delta_w(t_i, u_w)$ is the contribution of a_{w_i} in time t_i, and $\delta_l(t_i)$ is the contribution of a_{l_i}. Time t_i is measured in robot cycles. So after the first cycle since the winner agent obtains the resource, t_1, the contribution of δ_l should be higher than the one of the δ_w. As cycles go on, the value of δ_w wins importance; in a given moment, t_f, the action value should be the one of the winner agent, so δ_w should have the highest value (1) and δ_l the lowest one, (0).

According to this desired behavior, δ_w and δ_l have been defined in [0, 1] as follows:

$$\delta_w(t_i, u_w) = \frac{u_w}{t_f} * t_i \quad (2)$$

$$\delta_l(t_i) = \frac{u_l}{t_f} * (t_f - t_i) \quad (3)$$

Note that the resource exchange is performed by the winner agent that knows all the information regarding the utilities of the loser agent, as well as the actions. However, the information about the loser agent is a "snapshot" of the loser agent values in the moment that the resource exchange is performed, while the values of the winner agent can be updated in each robot cycle. This is the reason that $\delta_w(t_i, u_w)$ changes also according to u_w, while in Equation 3, u_l is static.

The summary of the coordination deployment algorithm is shown in Figure 3. The 2.2 step is briefly described, but involves the agent utility computation, coordination information exchange and others.

In the remaining of this section, we give details of how t_f is computed, and we illustrate the overall methodology with an example.

3.1 Time window frame from critical to non-critical situations

The coordination exchange from critical to non-critical situations can happens, for example, when the loser agent is the avoid agent, the winner the goto agent, and the robot agent is the resource agent. In this situation, the avoid agent

1. Find the time window frame (t_f)
2. While u_w is the resource winner and $t_i < t_f$ do
   {
2.1 Compute current action parameters $a_c = \frac{\delta_w(t_i,u_w)*a_{w_i}+\delta_l(t_i)*a_{l_i}}{\delta_w(t_i,u_w)+\delta_l(t_i)}$
2.2 Execute a_c
2.3 Next i
   }

Figure 3: Coordination deployment algorithm.

has dodged an obstacle, and now, the goto agent wins the resource in order to continue to the next robot goal position.

Then, the time window frame t_f depends on how much different the actions are between the loser and the winner ones. For measuring this difference, we use the Tchebychev distance [24] applied to the most critical action parameters, as follows:

$$max_{criticalParameters(1,\ldots,n)}(|a_{w_i} - a_{l_i}|) \quad (4)$$

The $criticalParameters(1,\ldots,n)$ function is defined for each shared resource, and returns the set of critical parameters of the resource. For example, in the case of the robot agent, the critical parameter is the linear velocity.

So, if this distance is large, the exchange period should be long, while if the distance is short, the period of change should be close to 0. The concepts of large and short can be easily modelled following fuzzy variables, and then computing the time window frame t_f according to a fuzzy system.

Particularly, we have followed a Sugeno fuzzy system. In such a system, we distinguish the input variables, the output variables and the rules. There is a single input variable "diff" that represents the action parameters difference according to the distance function defined in Equation 4.

The $diff$ variable is defined in [0,100], since the maximum velocity of the robot is 100cm/s. Four different fuzzy values are defined for the fuzzy variable: *equal*, *small*, *medium*, and *big*. A gaussian membership function is associated to each value. Figure 4 shows the functions used for each fuzzy value.

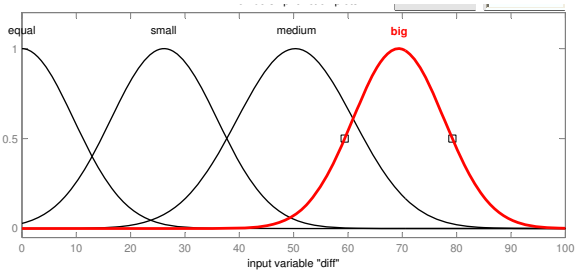


Figure 4: Input fuzzy values of the diff variable.

There is a single output variable too, that represents the length of the time window frame, t_f. Each value of t_f in [0,10] represents the number of cycles required to change from the current action parameters to the new ones. Four different values have been defined for t_f: *null*, *short*, *medium*, *long*. In a Sugeno system, each value of a output variable is a linear combination of the input variables. That is:

$$t_{fx} = c_1 * diff + c_2 \quad (5)$$

In our case, we have taken $c_2 = 0$ (otherwise there will be always a delay in sharing the resource even though when the required actions are the same), and the following c_1 coefficients for each t_f values have been defined:

$$t_{fnul} = 0 * diff$$
$$t_{fshort} = 3 * diff$$
$$t_{fmedium} = 5 * diff$$
$$t_{flong} = 10 * diff$$

Regarding the rules, four possible rules have been considered, according to the input and output possible values. These are the following ones:

R1. If $diff$ is equal then t_f is null
R2. If $diff$ is small then t_f is short
R3. If $diff$ is medium then t_f is medium
R4. If $diff$ is big then t_f is long

Finally, the output value of t_f is defuzzified by using the weighted mean. As the cycle time of the robot is $100ms$, it is possible to neglect the required time to calculate t_f, since all the calculations can be done before a cycle elapses.

3.2 Time window frame from non-critical to critical situations

The coordination exchange from a non critical situation to a critical one happens, for example, when the loser agent is the goto agent, the winner the avoid agent, and the robot agent is the shared resource. In this case, the avoid agent has detected a dangerous situation and the time required to react to it, t_c, is critical. Then, $t_f = t_c$. Note that t_c is known by the winner agent who applies the coordination deployment method. In the next section we show how to calculate t_c for a particular situation.

3.3 Example

The robot agent is shared by the avoid and the goto agent. The conflicting actions requested to the robot agent are the velocities (linear and angular) to which the robot should move. In order to illustrate the coordination deployment mechanism, let us suppose two different scenarios: when the avoid agent obtains the resource and when the goto agent obtains the resource.

In the first scenario, we have the following configuration:

- a_r is the robot agent, with 2 parameters: the linear and the angular velocities.
- a_w is the avoid agent, with $a_{w_1} = 20cm/s$ (linear velocity), and $a_{w_2} = -11degrees/s$ (angular velocity). The avoid agent has a utility value of $u_w = 0.7$.
- a_l is the goto agent, with $a_{l_1} = 76cm/s$, $a_{l_2} = -14degrees/s$, and $u_l = 0.6$.

Since the situation is characterized as a critical one, t_f is set to the critical time computed by the avoid agent. The critical time is computed as the time to collide with the obstacle, that is, the distance to the object divided by the current linear speed. Finally, the second step of the algorithm

of Figure 3 is applied, obtained an smoothing behavior in the global robot control.

In the second scenario, we have the following configuration:

- a_r is the robot agent, as in the previous scenario.
- a_w is the goto agent, with $a_{w_1} = 46cm/s$, $a_{w_2} = 19.4degrees/s$, and $u_w = 0.77$.
- a_l is the avoid agent, with $a_{l_1} = 20cm/s$, $a_{l_2} = 0degrees/s$ and $u_l = 0.28$.

This scenario corresponds to a transition from a critical to non critical situation. The $criticalParameters(1, 2)$ of the robot agent is the first one, that is, the linear velocity. Then, the time window frame t_f is set according to the Sugeno fuzzy system. The difference between the two linear velocities is the following:

$$diff = |a_{w_1} - a_{l_1}| = |46cm/s - 20cm/s| = 26cm/s$$

Such value partially fulfills the small and equal fuzzy values of the $diff$ input variable of our Sugeno fuzzy system. As a consequence, the resulting output variable t_f is set to 3 cycles.

4. EXPERIMENTAL RESULTS

In order to test ARMADiCo we have implemented an ad hoc multiagent platform, programmed in C++ on Linux. This implementation decision is based on the fact that the majority of the commercial platforms have an agent that centralizes the functioning of the entire platform and they are not capable of dealing with systems that need to respond in real time. The robot used for experimentation is a Pioneer 2DX of ActivMedia Robotics. This robot has a minimal sensor configuration: two encoders and eight ultrasound sensors. A complete description of the robot, as well as its dynamics can be found in [13].

In order to test the coordination deployment methodology proposed in this paper, we have implemented the following agents: interface agent, (very simple) task planning, path planning, goto, avoid, sonar, encoder, and robot agents. In this configuration, the robot agent is the shared resource between the goto and avoid agents.

4.1 Experimental setup

Two different ways of taking the resource control have been tested. The first one occurs when the agents change abruptly the control over the shared resource and the second one, when the proposed method is used. Thus, we have two main configurations:

- Abrupt: changing the control abruptly
- Smoothing: using our proposed method. That is, using Equation 1 to have a smoothing behavior when exchanging the resources after coordination.

In order to test the whole behavior of the architecture, we propose the scenario shown in Figure 5, in which the robot must go from room A to room B. For doing so, the *task planning* agent provides to the *goto* agent the corresponding trajectory necessary to pass through the doors.

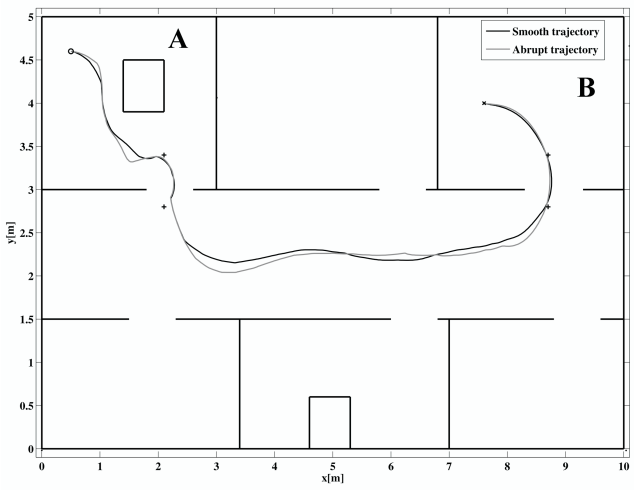


Figure 5: Example of a trajectory with our methodology.

4.2 Results

Figure 5 shows an example of the trajectory described by the robot, when using the abrupt exchange of resources (grey line) and using the proposed methodology (black line). As can be seen in this figure, at the points $(x, y) = (1.5, 3.3)m$ and $(x, y) = (6.3, 2.4)m$ approximately, there are abrupt changes in the trajectory described by the robot using the abrupt resource exchange methodology (grey line), but they are avoid using the smoothing algorithm (black line).

In order to compare the results, the following measures have been considered:

- **Travelled Distance (TD):** the distance travelled by the robot to reach the goal.
- **Final Orientation (FO):** the heading of the robot at the goal position.
- **Total Time (TT):** the total amount of time the robot needs to achieve the goal.
- **Precision (P):** how closed to the goal position is the center of mass of the robot.
- **Time Goto (TG):** the total time the *goto* agent has the robot control.

The experiments consisted of five executions in the scenario for each configuration. Table 1 shows the average and standard deviation of each evaluation measure. Even though mean values are not so different among the two situations, deviations are enhanced.

Table 1: Comparison of the results of the three tested situations.

Parameters	Abrupt	Smoothing
TD	$11.96 \pm 0.103m$	$11.99 \pm 0.17m$
FO	$2.14 \pm 0.47°$	$2.16 \pm 0.85°$
TT	$69.37 \pm 2.43s$	$64.39 \pm 1.24s$
P	$99.648 \pm 0.08\%$	$99.82 \pm 0.05\%$
TG	$73.86 \pm 3.23\%$	$51.25 \pm 5.41\%$

Comparing the abrupt change of control with the smoothing one, the most significant improvement is the total time (TT) needed to reach the goal. In the *smoothing* configuration, TT has decreased meaning that the cruising speed can be maintained for more time and changes between agents with a short duration are almost eliminated, achieving softer trajectories. At the same time, the time the goto agent has the control (TG) has been decreased, while the avoid agent has the control in more consecutive cycles. So, when an agent takes the resource control, it does it for a larger time using our methodology than without using it, thus, the agent can effectively follow their own goals for a larger period of time; and consistently the robot global behavior is more coherent.

5. RELATED WORK

As stated in the introduction, there are some works related to the use of multi-agent system for a single robot architecture [20, 1, 21]. In [16], several agents are organized in a spreading activation network, so that each one is defined with a set of pre- and post-conditions. When an agent accumulates enough activation (i.e., the pre-conditions are satisfied over a given threshold), it becomes active. Conflicting interactions among agents (for example, when parallel actuation requires a three hands robot) are solved according to different selection parameters. These parameters establish the urgency to fulfil the different goals. This approach proposed by Maes's work is more related to coordination (even though at a lower level than our approach and other recent ones [1], [20]) than to the physical resource exchange issue we are dealing here. From our understanding, none of the previous authors address the problem of filling the gap between coordination agreement and coordination execution in multi-agent systems when dealing with physically-grounded shared resources.

In the literature of control, however, there are several proposals to perform this resource exchange. For example, in [10], several controllers are used to control each wheel of the robot. The controllers are modelled as finite automata whose inputs are the location of the robot and the output, the values -1,1 and 0. After each controller produces its output, a central module performs the average of the total vote of the outputs, determining the desired increase in the movement.

Gerkey in [9] presents a similar idea but instead of having an explicit module that fusions command controls, all the controllers actuate directly on the motors, being these elements the ones that overlap the control signals and achieve the resulting movement. He proved that the presence of "malicious" controllers degrade the overall performance or produce a catastrophic failure when the proportion of them is higher enough.

Another example is presented in [22]. In particular, the authors use fuzzy logic to model the control actions coming from the heterogeneous controllers and to decide, in accordance with the dynamic model of the robot, what combination of control actions should be carried out at any given moment. We are also using a fuzzy system but with a different purpose. Instead of using it to compute the final command value, we use a fuzzy system to compute the time interval required to perform the resource exchange.

The change of commands in a robot has also been studied at the task level in behavioral-based architectures. For example, in [18], an adaptation method to deal with conflicting behavior interactions is proposed. After several trials, the robot learns the appropriate sequence of behaviors. Even though Mataric's proposal is closer to coordination mechanism than our resource exchange problem, some similarities can be found with our approach. That is, we are also proposing a sequence of action combination (transition phase) that facilitates the resource exchange between two agents. Note, however, that in [18] and also in [17], only one of the behaviors is active at a time.

From the agent community, physical resources has been studied by the holonic manufacturing community [8]. However, most of these approaches (see for example [11]) are mapping a robot to an agent (or a machine to an agent), conversely to our approach, in which we are building a multi-agent system for a single robot.

6. CONCLUSIONS AND FUTURE WORK

There is an increasing interest in the use of agent technology to develop robots as physical agents, and there starts to be some examples of the use of multi-agent systems for building a single robot architecture. In this paper we describes a multi-agent architecture with this purpose, ARMADiCo and we focus on the importance of the deployment of the coordination outcomes since they affect the robot global behavior. That is, when defining such architecture, several robot resources are shared by different agents. So, a coordination mechanism is required in order to avoid conflicting resource uses. This is something that most of the theoretical studies considers. However, when trying to deploy coordination mechanism to the physical world, as in robots, there is gap related to how the resource exchange is performed; and that matters. In this paper we describe a coordination deployment methodology to cover such a gap. The methodology comprises a Sugeno fuzzy system to determine the time window interval required to perform an smoothing resource exchange.

The methodology presented has been tested using a Pioneer 2DX of ActivMedia Robotics. The results show that the robot maintains a cruising speed more constant when using our coordination method than without it, as a consequence of reducing the number of short resource exchanges between agents, and achieves in the end a smoothing global behavior when moving through a space with obstacles.

As a future work, we are considering new experimental configurations with the incorporation of additional agents and resources. Consequently we are analyzing other possible resource allocation methods of the multi-agent literature (as for example [2]). We are also studying the definition of more complex scenarios that involve higher cognitive capabilities. Finally, we are also analyzing the use of new and still open paradigms as information flows [4] that helps in the understanding of the global emergence multi-agent system behavior.

7. REFERENCES

[1] D. Busquets, C. Sierra, and R. López de Màntaras. A multiagent approach to qualitative landmark-based navigation. *Autonomous Robots*, 15:129 – 154, 2003.

[2] Y. Chevaleyre, P. Dunne, U. Endris, J. Lang, M. Lemaître, N. Maudet, J. PAdget, S. Phelps, J. Rodríguez-Aguilar, and P. Sousa. Issues in

multiagent resource allocation. *Informatica*, 30:3 – 31, 2006.

[3] T. De Wolf. Panel discussion on engineering self-organising emergence. http://www.cs.kuleuven.be/ tomdw/presentations/ presentationSASOpanel2007.ppt, 2007. SASO 2007 10-07-2007, MIT, Boston/Cambridge, MA, USA.

[4] T. De Wolf and T. Holvoet. Using UML 2 activity diagrams to design information flows and feedback-loops in self-organising emergent systems. *Proceedings of the Second International Workshop on Engineering Emergence in Decentralised Autonomic Systems (EEDAS 2007)*, pages 52 – 61, 2007.

[5] S. A. DeLoach. Analysis and design using MaSE and agentTool. *Proceedings of the 12th Midwest Artificial Intelligence and Cognitive Science Conference (MAICS 2001)*, 2001.

[6] J. Fromm. On engineering and emergence. *SAKS/06, Workshop on Adaptation and Self-Organizing Systems (nlin.AO)*, arXiv:nlin/0601002v1 [nlin.AO], 2006.

[7] M. Georgeff and A. Lansky. Procedural knowledge. *Proceedings of the IEEE*, 74(10):1383 – 1398, 1986.

[8] C. Gerber, J. Siekmann, and G. Vierke. Holonic multiagent systems. Research RR-99-03, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1999. www: http://www.dfki.de.

[9] B. Gerkey, M. Mataric, and G. Sukhatme. Exploiting physical dynamics for concurrent control of a mobile robot. *Proceedings ICRA '02. IEEE International Conference on Robotics and Automation*, 4:3467 – 3472, 2002.

[10] K. Goldberg and B. Chen. Collaborative control of robot motion: robustness to error. In *Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 655–660, 2001.

[11] H. Hsu and A. Liu. A flexible architecture for navigation control of a mobile robot. *IEEE Transactions on Systems, Man, and Cybernetics: Part A: Systems and Humans*, 37(3):310–318, 2007.

[12] B. Innocenti, B. López, and J. Salvi. A multi-agent architecture with cooperative fuzzy control for a mobile robot. *Robotics and Autonomous Systems*, (55):881 – 891, 2007.

[13] B. Innocenti, P. Ridao, N. Gascons, A. El-Fakdi, B. López, and J. Salvi. Dynamical model parameters identification of a wheleed mobile robot. *5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles (preprints)*, 2004.

[14] G. A. Kaminka. Robots are agents, too! In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, 2007. Invited talk.

[15] G. A. Kaminka. Robots are agents, too! *AgentLink News*, pages 16 – 17, December 2004.

[16] P. Maes. The dynamics of action selection. *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 991 – 997, 1989.

[17] M. Matarić. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents*, 9 (2-3):323 – 336, 1997.

[18] F. Michaud and M. Matarić. A history-based approach for adaptive robot behavior in dynamic environments. *Proceedings of the second international conference on Autonomous Agents.*, pages 422 – 429, 1998. ISBN:0-89791-983-1.

[19] R. Murray, K. Åström, S. Boyd, R. Brockett, and G. Stein. Future directions in control in an information-rich world. *IEEE Control Systems Magazine*, 23, issue 2:20 – 33, 2003.

[20] M. C. Neves and E. Oliveira. A multi-agent approach for a mobile robot control system. *Proceedings of Workshop on "Multi-Agent Systems:Theory and Applications" (MASTA'97 - EPPIA'97) - Coimbra -Portugal*, pages 1 – 14, 1997.

[21] Y. Ono, H. Uchiyama, and W. Potter. A mobile robot for corridor navigation: A multi-agent approach. *ACMSE'04: ACM Southeast Regional Conference. ACM Press.*, pages 379 – 384, 2004.

[22] A. Saffiotti. The uses of fuzzy logic in autonomous robot navigation. *Soft Computing*, 1(4):180 – 197, 1997.

[23] O. Sauer and G. Sutschet. Agent-based control. *IET Computing & Control Engineering*, pages 32 – 37, 2006.

[24] R. Wilson and T. R. Martínez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1 – 34, 1997.

[25] M. F. Wood and S. A. DeLoach. An overview of the multiagent systems engineering methodology. *Lecture Notes in Computer Science. Vol. 1957/2001, Springer Verlag.*, pages 207 – 221, 2000.

Reactive agent mechanisms for scheduling manufacturing processes

Ask Just Jensen
University of Southern Denmark
Maersk McKinney Moller Institute
5230 Odense M, Denmark
just@mmmi.sdu.dk

Kasper Hallenborg
University of Southern Denmark
Maersk McKinney Moller Institute
5230 Odense M, Denmark
hallenborg@mmmi.sdu.dk

Yves Demazeau
CNRS
LIG Laboratory
38000 Grenoble, France
Yves.Demazeau@imag.fr

ABSTRACT

This paper will present an agent based solution to control chemical processes in a manufacturing environment; items should undergo chemical reactions in different chemical baths to be processed by the system. Recipes for the processes only specify minimum and maximum times for each bath, and recipes depend strongly on characteristics of the item.

We have applied the PACO paradigm for reactive agents to develop control software for the system, and we have designed simple physical force-like interaction mechanisms, so agents manage to fulfill their individual and global goals. We outline the problem and argue why the PACO approach suits this problem, before explaining in detail how the agents are designed and the interactions established for the planning to be successful.

We conclude the paper by presenting results of the agent-based approach, based on data and scenario recipes from the real system.

1. INTRODUCTION

All industrial companies have experienced radical changes in market demands in recent years. Mass series and standardized products have been replaced by order based production and a high degree of user customization. It falls back on the manufacturing system as strong requirements for high flexibility. In some setups flexibility could be meet by low switching times, but ideally the system should be able to handle a large variety of items concurrently in an efficient way. This require new control algorithms. We no longer have dedicated hardware optimized for a specific production. Global optimization is typically NP-complete or inappropriate due to a dynamic production environment, where constant changes will lead to continuous replanning. Thus flexible manufacturing systems have been a focused research area for decades [3, 4, 13, 18], and multi-agent technologies are a natural approach for the control software of such systems.

The paper is organized as follows. First we discuss the general problem of the researched case. Next we relate the problem to previous work, and shortly describe the PACO paradigm used for the agents in our case. In detail we will describe and discuss how PACO agents are organized, modeled, and implemented to solve the problem at hand together with results from real world scenarios. Finally we will bring some concluding remarks and present ideas for future work of this ongoing research.

1.1 The problem

The researched case is conducted in collaboration with Denmark's most well-known manufacturer of high-end audio and video products. The process is known as an anodization process that increase the corrosion resistance of aluminum. In a generalized and simplified form, the problem could be described as a number of chemical baths, which the items have to visit according to a prescribed recipe. Besides containing information about which baths to visit and in what order, the recipe also give an allowed time-span for the item to stay in each bath. Items are grouped on bars with the same recipe, but a mix of different bars (= different recipes) could processed at the same time.

The system consist of about 50 baths, and a typical recipe would have roughly 15-25 baths to visit, even though all recipes do not have to visit all kind of baths, there is still room for additional baths of the same type to overcome bottlenecks as the processing times in the bath types vary a lot. Three slightly overlapping cranes move the bars from one location to another within the array of bars. Apart from the baths and cranes, an important part of the system is the input buffer, where typically around 30 bars are waiting to be processed. Figure 1 gives an overview of the system.

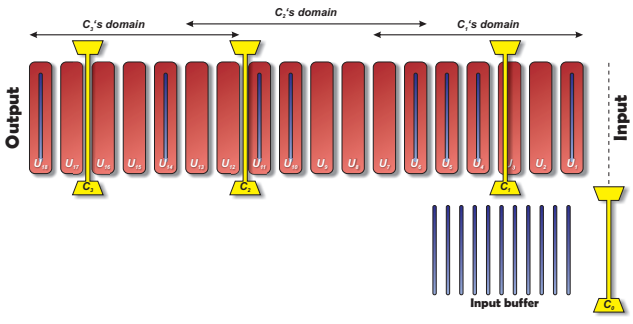


Figure 1: Generalized overview of the system

Another issue is the dynamical production environment,

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

which has great impact on the system's ability to recover and finish the current bars, but also run as best as possible under partly breakdowns. Examples of unpredictable error conditions could be that the temperature of a bath is too low and must be heated before continuing, crane breakdowns, liquid level of a bath too low, rapid orders, etc.

1.2 Related work

The problem is classic - we want to optimize the throughput of a system, by optimizing the flow between subprocesses and handle the inflow correctly to utilize the system as much as possible. In abstract terms we have a number of tasks, q_i for $i = 1, 2, \ldots, n$, with k_i subtasks[1], $q_{i,1}, q_{i,2}, \ldots, q_{i,k_i}$. The subtasks are interconnected and their order cannot be changed, and should be handled as visits to processing stations - determined by the recipe. The agents, which could be the baths and cranes, must coordinate their local activities for the global plan to be optimal. From a modeling point of view the TAEMS[2] Framework [8, 16] would be an obvious choice as a modeling tool to describe the structure of the plan.

GPGP is a set of generic coordination mechanisms for cooperative agents in a task environment proposed by Keith Decker [9]. GPGP is a generalized extension of the PGP (Partial Global Planning) algorithm, which is capable of handling deadlines for the tasks of agents. GPGP could be applied in our case as well, but the homogeneity of the task structures and flow-oriented approach of our case, appeal for other approaches as well. GPGP would be an appropriate candidate if the recipes were more flexible, e.g. the order of bath to visit could be changed dynamically.

Clement and Barrett also introduce the Shared Activity Coordination (SHAC) [6], which provides a general approach for interleaving planning and exchange of plan information among agents based on shared activities.

Both Parunak et al. and Valckenaers et al. have conducted extensive research on organizing and structuring agents in a manufacturing environment as well e.g. [19, 22]. Furthermore Valckenaers et. al have inspired by the principles of stigmergy of swarm intelligence developed some novel approaches to scheduling in holonic and flexible manufacturing [15, 21]. The simple reactive principles of coordination that characterize ant-based approaches appeal very much to our case except that ants are usually very independent, and in our case we more have a string of tasks that cannot be reordered.

Other approaches exists as well, but in general they are focused on the agent performing different tasks in the environment, where the tasks can be regarded as atomic actions. That would make the baths and the cranes to our agents, but in our case the tasks are not atomic in the sense that duration of a task in not fixed.

In principle planning approaches search a solution space to find an appropriate valid plan to a constraint satisfaction problem (CSP). There is no guarantee that the plan is optimal, as the problem often are NP-complete. In our case the task durations vary between the minimum and maximum times specified in the recipes, which give us inequality constraints that complicates the problem even more. In traditional planning we try to find an optimal plan among valid plans, but using the PACO approach we strive for an valid plan using simple reactive mechanisms on a configuration that might be optimal, but not necessarily valid.

[1]note that the number of subtasks might be different for each taskgroup

[2]Task Analysis, Environment Modeling and Simulation

1.3 PACO approach

PACO is a contraction of *coordinated patterns* [12] and takes a simple approach of the agents. PACO focus on reactive agents situated in an environment, where all agents are considered as partial solutions of a global problem [14].

The PACO paradigm states that agents are purely reactive, thus they do not hold an updated internal representation of themselves, other agents, or the environment, so they have to respond all changes of the environment. Whenever a new bar is introduced, or unforeseen or expected events happen within the system, such as a crane breakdown or a bath needs cleaning, it is a new event to the agents, and they will search for a new equilibrium state through their interactions.

Each agent under PACO is defined by three fields, which divide the agent model in coherent components

Perception field determine what the agent can perceive about its environment.

Communication field determine which agents an agent can interact with.

Action field determine the space in which an agent can perform its action.

From a system point of view the PACO paradigm also splits the system into conceptual parts, which follows the VOWELS formalism [7]. Basically the VOWELS formalism decomposes the problem into four components of the MAS domain of a system, described by the vowels-initiated concepts

Agents are the classic entities to consider, when developing a MAS system, as agents are determined to be the local actors carrying out the tasks. *Autonomy* is in focus here to emphasize that agents have goals and are self-determined [23].

Environment is the space in which the agents exist, move, and interact. The space could be both informational and conceptual, but typically the environment is represented by a model of the physical space of the MAS community.

Interactions could be formed as abstract as speech interactions, but the PACO paradigm usually apply interactions as forces, spring-like or electrostatic forces.

Organization: Similar to humans, agents can benefit from being organized, either explicitly defined in classic organizational structures, or the organization could emerge from simple interactions among the agents. Organizations often serve the purpose of grouping agents with similar or related actions or behaviors as usually exploited by PACO.

2. AGENT DESIGN

In this section we will describe and discuss how the PACO approach has been applied to the scheduling problem in the researched anodization system. The following subsections cover each part of the VOWELS formalism.

2.1 Environment

In this case the environment is the baths and cranes. The environment is modeled as passive resources, which the agents can ask about their status and book for specified time slots. Baths are accessed through a bath controller, which makes baths of the same type look as only one bath, capable of containing more than one bar at a time. These baths can be asked about free space in a given direction from a given agent or about whether or not a free time slot for a required timeframe exists in specific period. If the space is occupied, then the bath can tell, which agent is blocking.

2.2 Agents

The recipe for each bar of items is split into a number of agents. One agent is created for each step of the recipe, and all agents of one bar forms a group. An agent is born with some knowledge, as it knows which kind of bath it must go into, it holds the allowed minimum and maximum time to stay in the bath, and it knows its predecessor and successor agent of the group. It does not know the rest of the agents in the group and it does not have a possibility to communicate with them. To succeed, an agent must visit a bath of the right type, but not necessarily at the right time. The agent has a size equal to the time slot it occupies in a bath, because time is the only interesting axis of all decisions, as illustrated in figure 2. Therefore by its representation, agents can be seen as physical manifestations of the problem in focus. Two bars q_i and q_j, split up into two groups of agents, $[q_{i,1}, q_{i,2}, \ldots q_{i,n}]$ and $[q_{j,1}, q_{j,2}, \ldots q_{i,m}]$ added to the virtual model in random places could look like figure 2.

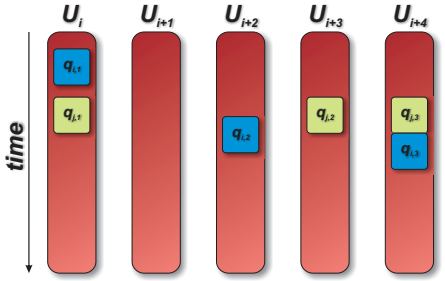


Figure 2: Three agents from each agent group occupying timeslots in the baths

2.2.1 Perception, communication, and action

As stated earlier the PACO paradigm defines three delimited fields.

The perception field consists of the predecessor of the bar, as the movement of that agent affects the forces (described below) applied to an agent. Furthermore the agents above and below (in the time domain) that want to visit the same bath are also observed, to avoid overlap of agents in the same bath.

The communication field solely consists of the predecessor, as it should be notified if the agent could meet its goals.

The action field consist of the baths of the requested type, and organization secures that an agent only sees one particular bath, even if the bath type is duplicated in the system.

2.2.2 Agent goals

An agent has two main goals:

- Go in the right bath
- Stay close to the predecessor agent of its group

When both goals are satisfied, for all agents of a group, the bar represented by the group has a valid way to be processed by the system. Furthermore an agent has some constraints:

- Keep distance to both the min and max time
- Help the successor to stay close
- Help other agents in same bath type to fulfil goals

Constraints are added to make agents cooperate with others in fulfilling their goals too. When agents from two groups share interests to the same time slot for a given bath they have to be able to negotiate about, who will win the timeslot.

2.3 Interactions

Agents move around in the virtual world in discrete steps. They calculate a force vector $\boldsymbol{v}$ as responses to input/output from the three fields. Each discrete time step has two substeps, first all agents calculate which way to go and at what speed. When all agents have new direction and velocity vectors the moves are taken, so force calculations are done according to the current situation.

2.3.1 Basic forces

The most basic behaviors of the agents come from their primary goals and are modelled with two forces; a spring force and a gravity force.
The spring force represents the attraction to the predecessor, if any, and attracts the agent towards the point where the predecessor of the previous bath ends, so the bar can move from one bath to another, which is a criteria for a plan to be valid.
A spring force is denoted: $\boldsymbol{F} = -k\boldsymbol{x}$. $\boldsymbol{F}$ being the force, k the spring constant and $\boldsymbol{x}$ the distance. This gives us: $\boldsymbol{F}_s = -k_{parent}(\boldsymbol{x} - \boldsymbol{x}_p)$. k_{parent} being a static constant, $\boldsymbol{x}$ the position of the agent and $\boldsymbol{x}_p$ the ending point of the predecessor.

To make the system stay in motion, until a valid equilibrium is reached, an extra parameter, a predecessor multiplier (p_m), is added. Each time a spring force is calculated the agent will check if it has come closer to its predecessor. If it has not, it will increase p_m, to enhance the attraction force

$$\boldsymbol{F}_s = -k_{parent}(\boldsymbol{x} - \boldsymbol{x}_p) \cdot p_m \quad (1)$$

p_m is bounded. It can newer go beneath 1 or above some fixed value, in our case fitted through experiments to be 3. It is increased with the same percentage each time the agent has not come closer to its goal and is decreased again with the same percentage, when the agent starts moving.

The second force, the gravity force, tries to pull the agent up. Up in the virtual model represents beginning of time in the real world. The gravity force is given by: $\boldsymbol{F}_g = m\boldsymbol{g}$. m being the mass of the agent and $\boldsymbol{g}$ the gravity acceleration. The mass of all agents is the same, and therefore we have

$\boldsymbol{F}_g = \boldsymbol{k}_g$, $\boldsymbol{k}_g$ being a static constant force vector. This gravity force is only applied to an agent when it is floating freely. If the agent is in contact with another, in the direction pulled by the force, the counterforce from the contact will cancel out the gravity force. The total force is denoted $\boldsymbol{F}_t$

$$\boldsymbol{F}_t = \boldsymbol{F}_s + \boldsymbol{F}_g \tag{2}$$

With only these two simple forces, a set of agents can be added and align them self. See figure 3.

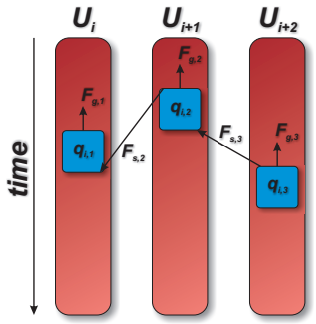


Figure 3: Illustration of the basic forces for the agents

Spring forces serve to compact the plan of a agent group for a bar, to minimize the total processing time for a bar, whereas the gravity force works to compact the entire plan for all bars in order to maximize utilization.

2.4 Organizations

To make the interaction between the agents more flexible, a number of social laws are introduced:

Law 1: If there is a certain amount of free space around the agent, increase size to

$$T_{current} = T_{min} + X(T_{max} - T_{min}) \tag{3}$$

X being a static constant and $T_{current}$, T_{min}, and T_{max} being respectively the current, minimum and maximum time slots of the agent.

Law 2: If another agent, using the bath, comes closer than a given distance, then shrink the current size until T_{min} plus a given margin is reached.

Law 3: If the successor is unable to reach its second goal, then increase $T_{current}$ until T_{max} is reached.

2.4.1 Options for problem solving

If an agent needs to go in a direction blocked by other agents, it should be able jump over, push or switch place with one of the blocking agents, as illustrated in figure 4. For this purpose three laws are introduced. They are respected when agent A wants to go in a direction blocked by agent B.

Law 4: If $\boldsymbol{F}_t$ for A is greater than the current size of B, and a time slot of at least A_{min} is available between the end of B and the length of $\boldsymbol{F}_t$, then A jumps to the other side of B, without notice.

Law 5: If there are no room for A on the other side of B, but B is trying to move in the opposite direction, and

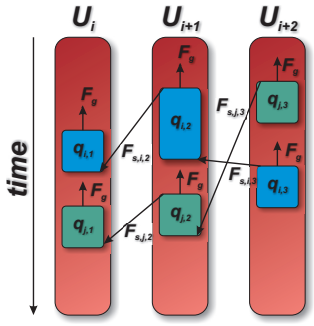


Figure 4: Illustration of a conflict between two agent groups

if the size of $\boldsymbol{F}_t$ for A is greater than half of B_{min}, then they switch place.

Law 6: If none of two previous laws have applied, but A still wants some or all of the time slot assigned to B. A starts a negotiation based on the general satisfaction of group A and B[3]. If A wins, B is pushed away, otherwise they will have to stay.

3. SYSTEM SETUP

The agent-based control that is described in the previous section, is just one part of the entire setup. The control software is coupled to a realistic simulation model, which is created in the AutoMod software - a defacto standard for creating simulation tools to manufacturing applications. We interact with the model using the same protocol that would be used on the real system, so we are as close as possible to the real world, but it allows us to test the control software much easier. In addition it give a perfect setup to test and compare different strategies and fit the coefficients of the forces on the desktop.

Regarding unforeseen events the reactive approach described above has the strength that we can make some time slots or bath unavailable, so agents are pushed and then have to renegotiate for a equilibrium. It requires no change of the agents decision logic.

4. DISCUSSIONS

The real challenges of applying the PACO paradigm to an agent-based control system like the problem presented above is the design and fitting of forces used for interactions. In this section we discuss some of the experienced issues for the case.

4.1 Avoid in-group competing agents

The agent group, which spawns from the creation of agents for a single bar of items, is not modeled or implemented as a sole entity in the system. Thus no overall goal or intensions of the group can be directly implemented, but must be realized through the aggregation of sub-goals met by the agents within the group. The tension appearing inside a group due to the spring forces of the agents, can to some degree lead to competitions among agents within a group, but the flexibility laws presented in section 2.4 make it easier for the

[3]Information about the satisfaction is withdrawn from the attached observer agent

system to reach an equilibrium and dampen the inter-agent tensions. Particular laws 1 and 3 are added to cope with these side effects of the basic forces. Law 1 simplifies the process of attraction and stabilize the movements of a successor agent to its predecessor, due to the expansion of the current time slot for an agent in a bath, if it is too hard to pack the schedule for a bar tighter. Whereas law 3 more directly connects the plan of a group and increase robustness in the coordination process.

4.2 Handling oscillating task shifts and stick to suitable schedule slots

Without doubt the most challenging part of optimizing the overall plan for the system is to decide when and how conflicts between agents should be solved. No method or measurements exist to validate if a current configuration is optimal or jumps between the agents should be handled. Clearly from laws 4 and 5 of section 2.4.1, trivial conflicts are handled without contracting classic local optimization principles. Especially to avoid oscillating shifts between agents from different groups that have interest in the same bath, law 6 serves the purpose of dampen inter-group tensions.

Other approaches could be applied as fallback methods, if the other laws fails, such as

- Only allow shifts that do not overrule other shifts within a certain time period.
- Transaction principle - make the shift, and roll back if the system performance degrade after a predefined number of planning steps.

5. RESULTS

In order to validate the PACO paradigm for the agents, we have to test if the control system can create valid plans for the bars. We measure that as a satisfactory rate of an agent group, where a full valid plan would have 100% satisfactory rate, which means that for a given bar all visits to baths in the recipes comply with the minimum and maximum timeframes, and that moves between two consecutive visits are connected with no stops.

Naturally an indicator of the strength in the approach is the dynamic reactiveness, because the core problem is a classical optimization problem, but as it is NP-complete, the system do not have the time to find an optimal solution in a real world scenario. Thus the number of time steps required for the system to find an acceptable solution is of primary importance.

In figure 5 the average computation time required per agent to calculate the forces and move the agents are shown. Naturally the computation time increases significantly for scenarios with more than one bar, but in all cases it stay below 0.5 ms per agent on an average PC. It is reasonable efficient, as we can see from figure 6 that the agents relative fast move to a rather stable level of their satisfaction rate.

Figure 7, which is a zoom of figure 6, also shows that valid plans (satisfaction rate = 100%) are only created in the 1 group scenario. From the graph of the 5 group scenario it is easy to see that we still have some fluctuations around a level of 90% satisfaction. For 15 and 25 groups the picture is a little more blurred, and more mechanisms should be applied to meet the 100% level.

Finally in figure 8 we see that the end-time (when it have visited all requested baths) of the last bar is decreasing,

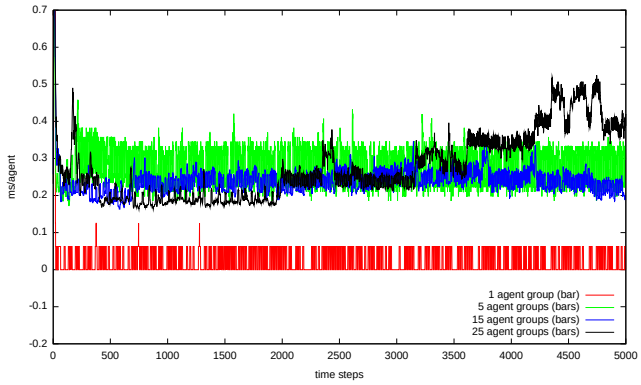


Figure 5: Average computation time per agent per time sted for scenarios with 1, 5, 15, and 25 agent groups

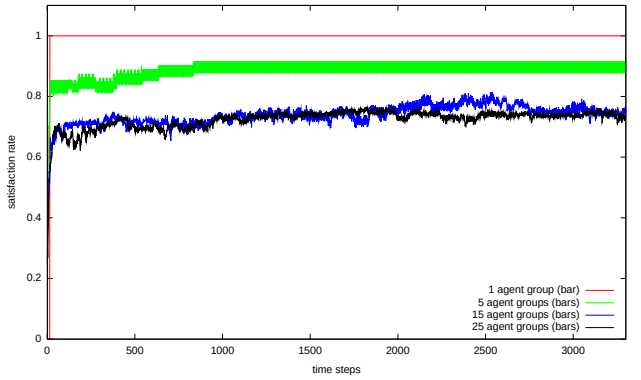


Figure 6: Satisfaction rate for test with 1, 5, 15, and 25 agent groups

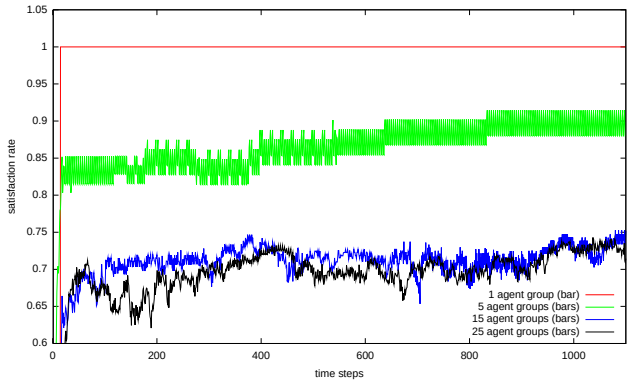


Figure 7: Satisfaction rate for test with 1, 5, 15, and 25 agent groups

so the algorithm is creating more efficient plans. Jumps between agents during the planning process will naturally lead to increased production time before the plans starts to settle again.

6. CONCLUSIONS

We have applied the PACO paradigm that suits the low-

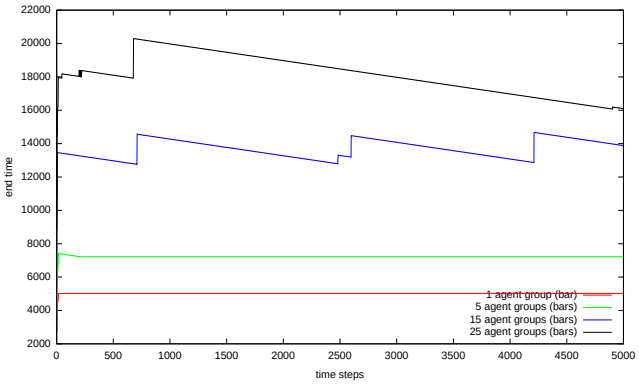


Figure 8: End time for the last bar of scenarios with 1, 5, 15, and 25 agent groups

flexible setup of a production environment. Besides designing the interaction mechanisms for the case, we have extended the model with a number of laws to cope with general problems among the agents.

The approach is radical different to many other paradigms of MAS in production environments, such as GPGP, as agents are not modeled as the resources of the environment, such as robots, baths, etc. Here agents originate from the real issues at hand.

The presented model and design are generalized from the case, so the results would easily map to similar problems that can be modeled with reactive agents.

From the results we see that not all problems related to in-group tension and conflicts among agents are fully solved, but the approach has shown promising results, and we strongly belive that the future work will bring the expected results.

7. FUTURE WORK

We will continue working on the case, and several strategies and modifications have already shown promising results

- We will introduce the concept of active, sleeping and locked agents, so an agent goes from being active to sleeping, when it no longer has intentions to move, and others should not expect to be able to negotiate with it. The agent is not finally locked, so if the forces are growing too large, the agent is awaked again. Only when it no longer accepts impact from the environment, it is considered locked.
- Snapshots. Like with transactions we can make a snapshot of a current plan and allow the system to continue a number of steps in the current direction. If no improvements can be measured we roll back to the snapshot and with modified parameters we continue in another direction.
- Predecessor validation could be introduced to dampen oscillations and improve the settling time. By predecessor validation we mean that an agent is more likely to go in the direction of its predecessor, if the predecessor seems to be settled.
- We will investigate the impact of extending the scope of the agent interactions, so the field do not only consists of successors and predecessors. Also interactions between non-consecutive interests to a bath could influence agent intensions of sticking to a bath or forming multiple jumps.

Acknowledgement

We would like to thank all the participants in the DECIDE project, which is supported by the The Ministry of Science, Technology and Innovation in Denmark. A special thank to Bang & Olufsen [1] and Simcon [2] for their support, feedback, and creation of the simulation model of the system.

8. REFERENCES

[1] Bang & Olufsen. `http://www.bang-olufsen.dk/`, 2007.

[2] Simcon. `http://www.simcon.dk/`, 2007.

[3] S. Bussmann, N. R. Jennings, and M. Wooldridge. *Multiagent Systems for Manufacturing Control*. Springer, 2004.

[4] S. Bussmann and K. Schild. An agent-based approach to the control of flexible production systems. In *8th IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA 2001)*, pages 169–174, Antibes Juan-les-pins, France, 2001.

[5] F.-T. Cheng, C.-F. Chang, and S.-L. Wu. Development of holonic manufacturing execution systems. In *Manufacturing the Future - Concepts, Technologies & Visions*, pages 77–100. Advanced Robotic Systems International, 2006.

[6] B. J. Clement and A. C. Barrett. Continual coordination through shared activities. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 57–64, New York, NY, USA, 2003. ACM.

[7] J. L. T. da Silva and Y. Demazeau. Vowels co-ordination model. In *AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 1129–1136, New York, NY, USA, 2002. ACM Press.

[8] K. Decker. *TAEMS: A Framework for Environment Centered Analysis & Design of Coordination Mechanisms*, chapter 16, pages 429–448. Wiley Inter-Science, 1 1996.

[9] K. Decker and J. Li. Coordinated hospital patient scheduling. In *In Proceedings of the Third International Conference on Multi-Agent Systems (ICMAS98)*, pages 104–111, 1998.

[10] K. S. Decker. *Environment Centered Analysis and Design og Coordination Mechanisms*. PhD thesis, Department of Computer Science, University of Massachusetts, May 1995.

[11] S. M. Deen, editor. *Agent-Based Manufacturing - Advances in the Holonic Approach*. Springer, April 2003.

[12] Y. Demazeau. Coordination patterns in multi-agent worlds: Application to robotics and computer vision. In *IEEE Colloqium on Intelligent Agents*. IEEE, 2 1991.

[13] A. Giret and V. Botti. Analysis and design of holonic manufacturing systems. In *18th International Conference on Production Research (ICPR2005)*, 2005.

[14] Y. Gufflet and Y. Demazeau. Applying the paco paradigm to a three-dimensional artistic creation. In *5th International Workshop on Agent-Based Simulation, ABS'04*, pages 121–126, Lisbon, Portugal, 5 2004. SCS.

[15] Hadeli, P. Valckenaers, M. Kollingbaum, and H. V. Brussel. Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53(1):75–96, 2004.

[16] B. Horling, V. Lesser, R. Vincent, T. Wagner, A. Raja, S. Zhang, K. Decker, and A. Garvey. The TAEMS White Paper, January 1999.

[17] I. IGN. Generalisation modelling using an agent paradigm. Technical Report AG-98-07, AGENT, 1998.

[18] G. Maionea and D. Naso. A soft computing approach for task contracting in multi-agent manufacturing control. *Computers in Industry*, 52(3):199–219, 1996.

[19] H. V. D. Parunak, A. D. Baker, and S. J. Clark. The aaria agent architecture: From manufacturing requirements to agent-based system design. *Integrated Computer-Aided Engineering*, 8(1):45–58, 2001.

[20] W. Shen, D. H. Norrie, and J.-P. A. Barthés. *Multi-Agent Systems for Concurrent Intelligent Design and Manufacturing*. Taylor & Francis, 2001.

[21] P. Valckenaers, P. V. Hadeli, M. Kollingbaum, H. van Brussel, and O. Bochmann. Stigmergy in holonic manufacturing systems. *Integrated Computer-Aided Engineering*, 9(3):281–289, 2002.

[22] P. Verstraete, B. Germain, P. Valckenaers, Hadeli, and H. V. Brussel. On applying the prosa reference architecture in multiagent manufacturing control applications. In *Multiagent Systems and Software Architecture, Erfurt*, pages 31–47, September 2006.

[23] M. Wooldridge and N. R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2):115–152, 1995.

An Agent Model for Collaborative Ubiquitous-Computing Environments

Marco P. Locatelli, Marco Loregian, Giuseppe Vizzari
Department of Informatics, Systems and Communication
University of Milano-Bicocca,
viale Sarca 336, 20126 Milano (Italy).
{locatelli,loregian,vizzari}@disco.unimib.it

ABSTRACT

Besides technological challenges, there is a growing interest on how pervasive computing can deeply affect the way in which people interact and accomplish collaborative tasks. Context awareness plays a crucial role for communities, where the degree of participation of a single user dynamically changes in relation to the distance from the place where practices occur and in relation with the ability of gathering the right information at the right time, and in a proper way. In this paper, we adopt an interpretation of pervasive computing as an environment, populated by interconnected smart devices, where users can interact with each other, individually or in groups, to accomplish collaborative tasks. In other words, we aim at modeling, designing and realizing Collaborative Ubiquitous Environments (CUEs). We present an agent based model, balancing the management of such a complex scenario between agents and a structured multifaceted environment.

1. INTRODUCTION

Since Weiser's seminal work [23], the challenges to pervasive computing have changed thanks to the rapid advances in electronics, distributed systems engineering, and mobile computing in general [20]. In particular, several middleware technologies have been proposed as the glue to keep devices connected in a seamless way, and to allow people to take advantage from *cooperating devices*. In spite of a multiplication of technological approaches [14] — to optimize network performance, reliability, and so on — less attention has been paid on how pervasive computing platforms can adapt to the way in which people act, so to reach an optimization in terms of targeted practices and behaviors.

In this work, we adopt an interpretation of a pervasive computing scenario as an environment, populated by interconnected active devices, where users can interact with each other, individually or in groups, to accomplish *collaborative tasks* [4]. In other words, we aim at modeling, designing and realizing Collaborative Ubiquitous Environments (CUEs) [11].

Among the many challenges that these scenarios pose to several research communities in the Computer Science and Engineering areas [25], we focus on the fact that the relevant entities in an pervasive computing system must be able to opportunistically take advantage of mutual interaction to obtain information or to exploit specific services, in order to ultimately satisfy users' needs. These entities should thus be considered as highly autonomous components driven by specific individual motivations, and capable of establishing high-level interactions with each other. These considerations have lead to consider the agent-based and multi-agent systems (MAS) paradigm as a suitable and effective approach to the modeling, design and engineering of CUEs.

In particular, the approach described in this paper is focused on the definition of proper structures and mechanisms for *environments* [24] that can provide agents with information, through their perceptions, and with means of action and interaction. An environment of this kind can be used to structure and organize a system in order to achieve the desired overall resulting behaviour by means of *simple* agent specifications and architectures.

In our perspective, a proper CUE can be enacted only if the constitutive entities are:

- *cooperative*: i.e., able to operate in strict collaboration by coordinating their actions according to specific patterns of interaction. These patterns define *must policies*, supplying coordination patterns to decide who has to be involved in a task and with which role. Members have to respect them under any condition;
- *context-aware*: i.e., able to perceive and share information about the context so to adapt themselves and to collaborate in a loosely coupled manner – without requiring direct contact or communication. This side of interaction defines opportunities rather than strict rules, *may policies*, supplying information to the involved entities in a less prescriptive way, as their execution is decided by evaluating awareness information and members' state.

These two aspects of a CUE must be properly integrated in order to allow the composing parts of the system in choosing and enacting the most suitable actions to be carried out in order to exploit their contextual situation to fulfill users' needs. To support cooperation among entities we provide them both with an environment to *coordinate* their behaviors and with a separate environment to share and perceive *awareness* information. In this work, we consider the agents' environment not only as a necessary element of the MAS model, but also as an exploitable abstraction supporting the

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

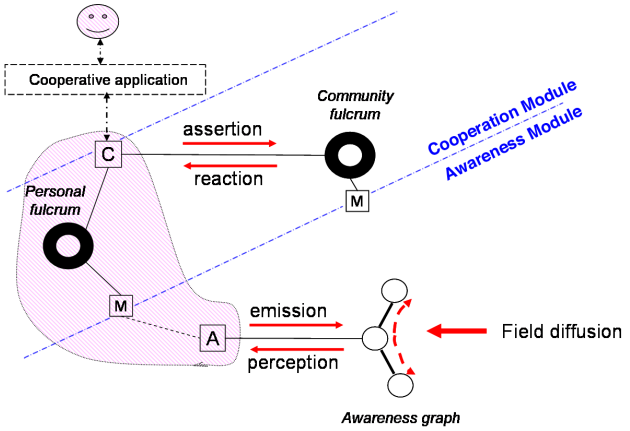


Figure 1: Elements of the reference model.

definition of suitable solutions to face the requirements of CUEs.

The main aim of this paper is to illustrate how a MAS model provide support for the *design* of CUE; the reference model (CASMAS, Community-Aware Multi-Agents Situated Systems [3]) is presented in the following Section. It must be stressed the fact that the model is aimed at the realization of systems that are focused on a given community, encapsulating thus its coordination patterns and the context-awareness policies. These systems are thus aimed at supporting a relatively small number of users (e.g. the persons working and studying in a Department of a University, the different workers of a firm) and technological appliances supporting their activities. The details of the language defined to specify the behaviours of agents in CASMAS is introduced in Section 3. Section 4 presents related work, and concluding remarks end the paper.

2. THE CASMAS MODEL

Ubiquitous-computing systems can shape up as constellations of dynamically defined and interacting *communities* of human and technological *entities*. In the model we are presenting, each entity is represented by two sets of agents belonging to two separate classes — grouped in two logical *modules*, — defined according to roles: agents can either enact *cooperation* mechanisms (within the communities to which entities belong), or supply the entities with information related to context *awareness* [1].

2.1 Cooperation module

Even if CASMAS (Community-Aware Multi-Agents Situated Systems) is designed to provide support to communities of persons, the communities of agents that compose an instance of CASMAS do not try to mimic human behaviors and social organizations. Agents are organized in communities with the purpose of autonomously coordinating their activities and acting to satisfy users' needs. In other words, even if a device is personal and somehow used to qualify a person, its behavior does not directly reflect the behavior of any person: devices interact with each other by means of agents gathering in autonomous communities and enacting (agent-processable community) policies that describe users' needs.

Communities of entities are established in the cooperation module by aggregation points called *community fulcra*. Community fulcra are designed to contain characteristic information: entities gather around them to exploit such information and contribute to the definition of a community. Entities connect to community fulcra by means of cooperation agents (**C-agents**) (Fig. 1).

Entities own a distinct C-agent for each of the communities in which they participate: when created, all C-agents are provided with generic inferential capabilities and with a set of entity-specific statements and rules. By connecting to a community fulcrum, C-agents standing for different entities can share information and acquire community-specific behaviors that are either defined at design time or injected in the fulcrum by other entities.

A *statement* is the unit of information on which an agent reasons and it is characterized by an owner, a list of receivers, and an access control to allow modification only by the owner.

Information (asserted as statements by agents) are shared, and behaviors (rules, also in the form of statements, that express how to infer from other statements) can be asserted in the fulcra and acquired by other agents (Sect. 3). Intelligent behaviors are thus deployed as a result of distributed inferential capabilities of the interconnected computational sites. The blackboard approach makes the environment very flexible with respect to dynamic situations — e.g., entities joining and leaving communities — since variations of interaction patterns among entities can be dynamically managed through mechanisms allowing for the (de)activation of behaviors.

In addition, each entity owns a *personal fulcrum* for its inner coordination (each C-agent is also connected to this fulcrum) and, in case of human entities, for the coordination of the various technological entities that refer to the same person.

The behaviors of C-agents are influenced by the degree of participation of entities in communities, according to additional context information that is supplied by the awareness module. In other words, when a specific application domain is modeled, it is necessary to define which factors concur in the characterization of an entity with respect to a community and the context. The degree of participation is then evaluated by weighting such information according to a domain-specific function.

2.2 Awareness module

The context awareness facet of an ubiquitous computing environment can encompass several different aspects, such as the physical position of an entity as well as its logical (sometimes social) role, such as the responsibility in an organization or a project.

Aim of the **awareness module** is to manage this kind of information, and entities are therein represented by specific awareness agents (**A-agents**).

The (spatial, social, organizational) environments in which entities are situated are modeled as awareness graphs, also called *topological spaces*, and A-agents are connected to their nodes, also called *sites*, accordingly.

Each community can refer to one or more graphs (one for each aspect), therefore, each entity needs an A-agent for each of them. Similarly, the same graph could support more than one community. However, different communities may

have different perceptions of the same aspect (e.g., different notions of distance in a spatial arrangement) thus requiring a more precise definition of interaction on awareness graphs.

The awareness module has been designed according to the perception-reaction paradigm [21]: the behaviors of A-agents are driven by the perception of signals, called *fields*, emitted by other agents and propagated across the awareness graph (e.g., to notify presence in a specific site). The propagation of fields is mediated in intensity according to a graph-specific and community-specific distance function evaluated in and between sites. When a field reaches a site its intensity is computed according to the weight of the arc connecting the current site to the one from which the field is coming, and then the field (if sufficiently intense) can be propagated to adjacent sites. If the field reaches a site where an A-agent is connected, and if it is relevant to the A-agent, the agent evaluates its local intensity according to a characteristic (*sensitivity*) function: possibly, the A-agent can then emit new fields as the result of inner computation, and reach other A-agents.

For example, the closeness of some entities can be considered a fundamental awareness information: A-agents can have a sensitivity function based on "intensity of the perceived presence field greater than x" where a high threshold can be set to perceive entities in the close neighborhood while a lower threshold can be set to perceive also entities further apart.

2.3 Integrating the two aspects

Additionally, A-agents can provide information for the cooperation module, to possibly trigger some reaction in the C-agents. To this aim, special agents bridging information between the two modules are defined: manager agents (**M-agents**) are specialized C-agents that can translate information expressed in terms of fields (exported by the A-agents) into statements that C-agents can understand (by publishing them into the personal fulcrum), and, in the other direction, import fields in A-agents when requests for propagation come from C-agents (e.g., when a device localizes an entity). Finally, each community fulcrum has an M-agent enforcing inter-entity coordination by causing the propagation of general community awareness information on awareness graphs (e.g., causing devices to join the community when they are needed), and by applying access policies and preventing unauthorized entities to join the community.

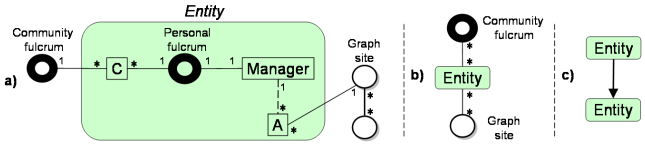


Figure 2: Notation used to represent the entities.

To better represent the instances of CASMAS (e.g., with respect to scenarios), the graphical notation presented in Fig. 2 was defined. Each entity — composed of the elements in a) — is drawn as a rounded rectangle, and can be linked to multiple fulcra and topological spaces. A C-agent is instantiated for each connection to a fulcrum, and an A-agent is instantiated for each connection to a topological space.

An entity such as a person's PDA can be connected to a fulcrum of an entity that represents a person, instead of being connected to a community fulcrum. The notation used to depict such situation is showed in c); in this case, a C-agent is created when an entity connects to another one.

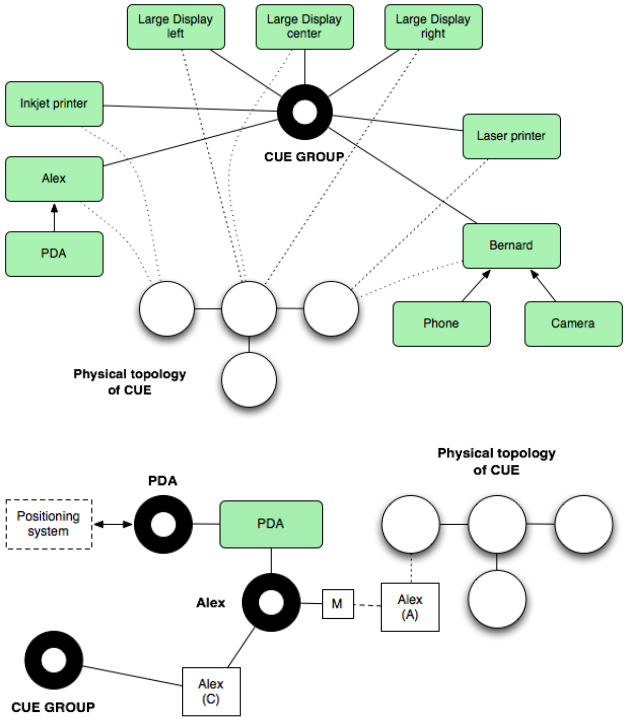


Figure 3: Overall and detailed views of the instantiation of CASMAS for the scenario.

Fig. 3 presents the details of a CASMAS instance supporting a CUE scenario modeled by means of the CASMAS approach. In particular, on the left there is an overall view of a set of CUE (domain) entities that are reified and associated to a CASMAS entity. It must be noted that different technological appliances are represented as entities (e.g. large displays, printers), as well as human actors. They are generally connected both to the workgroup community fulcrum and with the topological space of the building in which the CUE is situated. A *detailed view* of the same instantiation of CASMAS specific for Alex is presented on the right side of the figure: Alex's personal fulcrum, with the associated M-agent, a C-agent that links her to the Atelier community and an A-agent that localizes her on the topological space. Alex's PDA is modeled by its fulcrum and the C-agent linking the PDA to Alex's personal fulcrum.

This structure of the environment allows both the distribution of awareness information, on the topological representation of CUE's physical aspect and the sharing of statements guiding the coordination of activities of entities belonging to the community based in the CUE. The entities act as a bridge between the two aspects of the CUE, being able to perceive and generate awareness information, on one hand, and to exploit it for supporting the coordination of their activities on the other. In the following, the basic linguistic structures to define the behaviours and interactions of the different agents of the CASMAS model will be briefly presented.

3. CASMAS LANGUAGE

A simple language has been defined for CASMAS agents. Basic elements are called *primitives* and allow for simple operations. Primitives can be combined in more complex structures: *mechanisms* or *actions*. The former are employed by C-agents, the latter by A-agents.

A **primitive** implements an elementary function useful for the agents. The primitives used by the C-agents operate on statements and are: *assert*, *modify*, *retract* (erases a statement), *loadBehavior*, *translate*, and *propagate*. The primitives used by the A-agents are: *emit*, *export*, *transport* (moves the agent to another site), and *trigger* (changes the agent's state). These primitives are contextualized in the following where mechanisms and actions that apply them are presented.

3.1 Mechanisms

A **mechanism** is a set of (at least two) rules that implements a mode of interaction that can be adopted by the C-agents. In particular, a mechanism is composed by a rule to *execute* the operation(s) and a rule to *notify* when the mechanism can not be executed — no feedback is otherwise necessary.

A mechanism has the following (basic) form:

```
1  (defrule <execute-rule-name>
2    (<fulcrum-name>::allow (what <wish-statement-name>)
3      (state TRUE) (to ?entityId))
4    (<fulcrum-name>::wishTo (what <wish-statement-name>)
5      <slot>+)
6    <consistency-statement>+
7    =>
8    <operation>+)
9
10 (defrule <notify-rule-name>
11   (<fulcrum-name >::allow (what <wish-statement-name>)
12     (state FALSE) (to ?entityId))
13   (<fulcrum-name>::wishTo (what <wish-statement-name>)
14     <slot>+)
15   =>
16   (assert (_PERSONAL_::notifyFailure (what <wish-statement-name>)
17     [(on <statement-id>)])
```

The activation of the execution rule (line 1) and of the notification rule (10) is guarded by an `allow` statement (2 and 11) with the state set respectively to `TRUE` or `FALSE`, meaning that the entity can (or is forbidden to) execute the mechanism.

If the check is referred to the entity that is executing the mechanism, then the `?entityId` (3) should be changed to `?myId`; if it is referred to the requester, it should be changed to `?ownerId` (matched on the `owner` slot of the `wishTo` statement, that is a basic property of each statement). Moreover, both rules match a `wishTo` statement that expresses the request for activation of the mechanism by an entity. The execution rule can check other consistency conditions for the specific mechanism, for example, a device with a small screen can ensure a proper display only of short messages:

```
1  (defrule _COMMUNITY_::showMessage
2    (_COMMUNITY_::wishTo (owner ?applicantId)
3      (what "showMessage") (message ?message))
4    (_COMMUNITY_::allow (what "showMessage")
5      (state TRUE) (to ?applicantId))
6
7    (test (< ?message ?MAX_LENGTH))
8
9    ?display<-(display)
10   =>
11   (modify ?display (currentMessage ?message))
12 )
```

Mechanisms can be generally classified according to the purpose for which they are designed, i.e., according to the kind of interaction between agents they supply (Fig. 4):

Translation mechanisms are used by C-agents to move information from a source fulcrum to a target one. The information to be moved and the possible elaboration of such information (before it is entered in the target fulcrum) are expressed as statements, and the translation is enacted as an assertion in the target fulcrum.

Membership mechanisms are used by M-agents of personal fulcra to induce the acquisition of behaviors by the C-agents of the entity from the community to which they belong. In practice, M-agents assert a `memberOf` statement in the personal fulcrum; the statement quantifies the degree of participation of the entity to the community. Degree computation is based on awareness information perceived by the entity in relation to the specific community. Then, C-agents can activate the rule of the membership mechanism to load the behavior according to the degree of participation through the `loadBehavior` primitive. Membership mechanisms do not contain a `wishTo` statement since they only describe mandatory C-agent reactions to the presence of the `memberOf` statement.

Propagation mechanisms are used by M-agents — of any fulcrum — to respond to the willingness of C-agents to propagate awareness information within an awareness space by using the `propagate` primitive. Propagation mechanisms are in charge of the conversion between a statement and the field's format representing information perceived and to be propagated by the A-agents on the topological space.

Awareness mechanisms are used by M-agents to insert into their fulcrum the awareness information contained in the above mentioned fields; insertion is accomplished by means of the `assert` primitive. Awareness mechanisms are in charge of the conversion between the field's format and the `awareOf` statement representing awareness information in the target fulcrum.

3.2 Actions

The A-agents have a small set of **actions** that specify whether and how agents change their state or position, and propagate information on the topological space. *Trigger* and *transport* are the actions to change state and position, respectively.

Trigger defines how the perception of a field causes a change of state in the receiving agent, while transport defines how the perception of a field causes a change of position in the receiving agent. *Emit* is the action to propagate information on the topological space; this action can also be viewed as the activating part of a sort of asynchronous communication among agents. In fact, asynchronous interaction among agents takes place through a field emission–propagation–perception mechanism. An agent emits a field when its state is such that it can be source for it. Field values propagate throughout the space according to the diffusion function of the field. Field diffusion along the space allows the other agents to perceive it. Perception function, characterizing each agent type, defines the second side of an asynchronous interaction among agents: that is, the possible reception of broadcast messages conveyed through a field, if the sensitivity of the agent to the field is such that it can perceive it.

In CASMAS these actions are implemented as rules, and

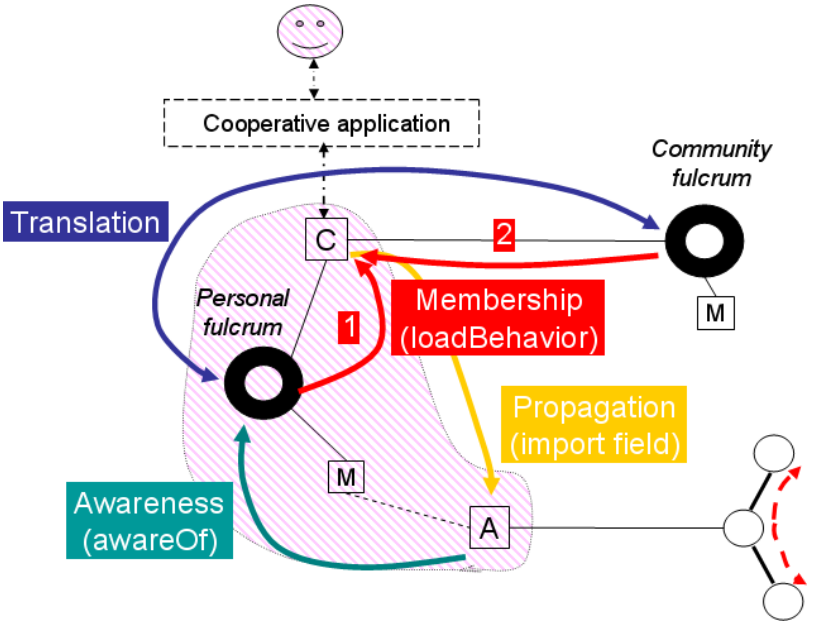


Figure 4: Mechanisms provided by CASMAS.

they can fire both on an `importedField` and on a field perceived from the site where the agent is situated. In the following, `_AW_` is where imported fields are asserted by the M-agent, `_SPACE_` is the topological space where the A-agent is situated.

The `situated` fact reifies on which site an agent is situated. The match between the `siteId` of `field` and the one of `situated` ensures that the field is perceived from the site where the agent is situated.

```
(defrule transport
  (or
    (_AW_::importedField (type ?type) (intensity ?intensity)
      (data ?data) (to ?myId))
    (_SPACE_::field (site ?siteId) (type ?type)
      (intensity ?intensity) (data ?data))
  )
  [<conditions-on-field>]
  (_SPACE_::situated (agent ?myId) (onSite ?siteId))
  (test (neq ?siteId ?NEWSITE_ID))
  (state ?state)
  [<conditions-on-state>]
  =>0
  (transport ?NEWSITE_ID)
)
```

The `NEWSITE_ID` must be the site where the agent has to move in reaction to the perceived field. If necessary, the destination site can be computed instead of defined explicitly. The `transport` is executed only if the destination site differs from the current site where the agent is situated.

```
(defrule emit
  (or
    (_AW_::importedField (type ?type) (intensity ?intensity)
      (data ?data) (to ?myId))
    (_SPACE_::field (site ?siteId) (type ?type)
      (intensity ?intensity) (data ?data))
  )
  <conditions-on-field>
  (_SPACE_::situated (agent ?myId) (onSite ?siteId))
  (state ?state)
  [<conditions-on-state>]
  =>
  (emit "FIELD_TYPE" ?intensity [?data]))
```

In accordance with the model, the emit is possible only on the site where the agent is situated, so the site is not a parameter for `emit`. Moreover, the `data` parameter is used to associate an information to the field and it is optional.

```
(defrule trigger
  (or
    (_AW_::importedField (type ?type) (intensity ?intensity)
      (data ?data) (to ?myId))
    (_SPACE_::field (site ?siteId) (type ?type)
      (intensity ?intensity) (data ?data))
  )
  <conditions-on-field>
  (_SPACE_::situated (agent ?myId) (onSite ?siteId))
  (state ?state)
  [<conditions-on-state>]
  =>
  (trigger ?NEWSTATE)
)
```

If conditions are satisfied, a change of state is triggered.

4. RELATED WORK

The focus of this paper is on the design of pervasive computing environments, particularly with respect to communities of persons: such communities are first-class concepts both in CASMAS and in the realized supporting infrastructure and, therefore, they are present both at the model and the implementation level. From this point of view this work differs from other existing proposals that employ agents and agent–based infrastructures simply as a middleware for the design and implementation of pervasive computing systems (see, e.g., [8, 18]).

Moreover, CASMAS is a model to let software applications coordinate and share awareness information; it is not mainly intended to (re)implement software applications, although it is possible. Instead, approaches such as BEACH [22] or iROS [6] provide support to the development of software applications from data management up to user interfaces. Differently from these proposed approaches, CASMAS defines the concepts of community fulcrum, topological space,

and entity, and collaborative ubiquitous-computing environments are built on them.

CASMAS agents are programmed declaratively (by rules), and they can be classified as reactive agents [5]. Differently from the FIPA approach[1], that is based on the *direct* exchange of messages, and in general from approaches based on Agent Communication Languages (ACLs) [7], interactions among CASMAS agents are mediated by a space both in the coordination module, by exchanging information in fulcra, and in the awareness module, by propagating information on topological spaces.

The **coordination module** is naturally implemented through DJess because it allows to directly map the agents and the fulcra to the inferential systems and to the shared working memory, respectively. Moreover, it provides some useful features such as the sharing of rules that are fully exploited by CASMAS. Among other platforms for pervasive computing environments, the following experiences seem interesting for comparison:

The *Gaia* project is a leading effort in the field of pervasive computing systems. Gaia [19] is an operating systems for pervasive computing environments that are called Active Spaces. In an Active Space everything (users, actions, contexts) is managed by agents with different specifications, belonging to different classes — like CASMAS agents. A generic distinction defines Context Providers, Context Synthetizers and Context Consumers. In the last class are essentially all the application agents. As for CASMAS, this separation requires that each agent embeds some logic capability but information on context is replicated across several agents, and not shared to gather agents, also the characterization (i.e., the configuration) of similar entities is totally independent from an agent to another.

Finally, with respect to the **awareness module**, other approaches have been analyzed during our research, before basing the current implementation on DJess:

- a *framework for MMASS based applications* [2], supplies computational support to the abstractions and mechanisms defined by the model, and if properly integrated with a suitable middleware (the MAIS reflective architecture [17]) supporting network communication it can provide a reasonable support to the realization of the awareness model. However, this framework lacks the possibility of giving declarative descriptions of behaviors, a feature that is currently exploited to facilitate the task of deploying CASMAS modules;

- an *artifact–based approach* [15], supporting the design and engineering of MAS environment, through the adoption of TuCSoN tuple centres [16], could be adopted to represent the internal spatial structure of the topological layer as well as the various related mechanisms (e.g. field diffusion), realized in the form of reaction rules considering the TuCSoN [16] approach to artifact implementation. This approach, however, is mainly focused on the realization of infrastructures supporting agent interaction and coordination, while DJess also supports the realization of simple reactive agents and to describe their reactive behaviors in term of rules, and to distribute the execution of agents and topological spaces in a transparent way for the system designer and of course for the agents too;

- the *TOTA middleware* [13] offers a rich and sophisticated support to the design and engineering of Pervasive Computing applications exploiting the abstractions of agents and MAS environment. However, one of TOTA's most distinguishing features is the possibility to diffuse and keep updated context–awareness information in a dynamic environment, and in particular it offers the possibility to maintain the structure of Co-Fields over a changing network. In this case, however, the structure of the network is not very important in determining the context of a given node. Instead the CUE scenario generally requires the capability to integrate several topological spaces, related to different aspects of the CUE.

[1]Foundation for Intelligent Physical Agents (FIPA) standard specifications available at `http://www.fipa.org/repository/standardspecs.html`

5. CONCLUDING REMARKS

This paper has briefly presented an approach to the modeling and design of pervasive computing systems supporting the coordinated activities of members of communities. In particular, the CASMAS model was introduced as a support for the representation and management of both awareness and coordination facets of a CUE. The CASMAS conceptual model allows for the design of systems focusing more on collaborative tasks — application objectives — rather than on technological issues. The model was applied to represent and manage different types of CUES, in the healthcare context [10] and for the definition of smart environments in a educational facilities [9, 12]. Future works are aimed on one hand at a concrete experimentation of the approach and the supporting infrastructure in the above introduced contexts.

In addition to this form of evaluation of the adequacy of the modeling approach and the realized supporting infrastructure, we also working on instruments supporting a simple configuration of the CUE, its structure, rules and laws for the management of awareness and coordination policies. This aspect is particularly important due to the specific nature of this kind of application: in fact, even if a thorough analysis of methodological aspects of CUE definition must still be carried out, we envisage a specific role of CUE designer to manage the phases leading from an analysis of the specific application scenario (including the involved actors and devices, as well as the relevant facets of the specific notion of context), to the definition of a set of spatial and logical layers of representation, each one endowed with an internal structure, active entities and patterns of interaction among them.

6. REFERENCES

[1] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards a better understanding of context and context-awareness. In *HUC '99*, pages 304–307, London, UK, 1999. Springer-Verlag.

[2] S. Bandini, S. Manzoni, and G. Vizzari. Towards a platform for multilayered multi agent situated system based simulations: focusing on field diffusion. *Applied Artificial Intelligence*, 20(4–5):327–351, 2006.

[3] F. Cabitza, M. P. Locatelli, M. Sarini, and C. Simone. CASMAS: Supporting collaboration in pervasive

environments. In *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, pages 286–295. IEEE, 2006.

[4] R. H. Campbell. Beyond global communications: the active world. In *PerCom*, page 211. IEEE Computer Society, 2005.

[5] A. Hector. A new classification scheme for software agents. In *ICITA '05: Proceedings of the Third International Conference on Information Technology and Applications (ICITA'05) Volume 2*, pages 191–196, Washington, DC, USA, 2005. IEEE Computer Society.

[6] B. Johanson, A. Fox, and T. Winograd. The interactive workspaces project: experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2):67–74, 2002.

[7] Y. Labrou, T. W. Finin, and Y. Peng. Agent communication languages: the current landscape. *IEEE Intelligent Systems*, 14(2):45–52, 1999.

[8] T. C. Lech and L. W. M. Wienhofen. AmbieAgents: a scalable infrastructure for mobile and context-aware information services. In *4rd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005), July 25-29, 2005, Utrecht, The Netherlands*, pages 625–631. ACM Press, 2005.

[9] M. P. Locatelli and M. Loregian. Active coordination artifacts in collaborative ubiquitous-computing environments. In B. Schiele, A. K. Dey, H. Gellersen, B. E. R. de Ruyter, M. Tscheligi, R. Wichert, E. H. L. Aarts, and A. P. Buchmann, editors, *AmI*, volume 4794 of *Lecture Notes in Computer Science*, pages 177–194. Springer, 2007.

[10] M. P. Locatelli and C. Simone. Supporting care networks through an ubiquitous collaborative environment. In C. Nugent and J. Augusto, editors, *Smart Homes and Beyond*, volume 19 of *Assistive Technology Research*. IOS Press, 2006.

[11] M. P. Locatelli and G. Vizzari. Awareness in collaborative ubiquitous environments: the multilayered multi-agent system approach. *ACM Transactions on Autonomous and Adaptive Systems*, 2(4), 2007.

[12] M. P. Locatelli and G. Vizzari. Environment support to the management of context awareness information. In D. Weyns, S. Brueckner, and Y. Demazeau, editors, *Proceedings of Engineering Environment-Mediated Multiagent Systems 2007*, pages 162–169, 2007.

[13] M. Mamei and F. Zambonelli. Programming pervasive and mobile computing applications with the TOTA middleware. In *2nd IEEE International Conference on Pervasive Computing and Communication (Percom2004)*, pages 263–273. IEEE Computer Society, 2004.

[14] C. Mascolo, L. Capra, and W. Emmerich. Middleware for mobile computing (a survey). In E. Gregori, G. Anastasi, and S. Basagni, editors, *Neworking 2002 Tutorial Papers*, volume 2497 of *lncs*, pages 20–58, 2002.

[15] A. Omicini, A. Ricci, M. Viroli, C. Castelfranchi, and L. Tummolini. Coordination artifacts: environment-based coordination for intelligent agents. In N. R. Jennings, C. Sierra, L. Sonenberg, and M. Tambe, editors, *3rd international Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2004)*, pages 286–293. ACM Press, 2004.

[16] A. Omicini and F. Zambonelli. Coordination for Internet application development. *Autonomous Agents and Multi-Agent Systems*, 2(3):251–269, Sept. 1999. Special Issue: Coordination Mechanisms for Web Agents.

[17] B. Pernici, editor. *Mobile Information Systems: Infrastructure and Design for Adaptivity and Flexibility*. Springer, 2006.

[18] M. D. Rodríguez, J. Favela, A. Preciado, and A. Vizcaíno. Agent-based ambient intelligence for healthcare. *AI Communications*, 18(3):201–216, 2005.

[19] M. Román, C. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(4):65–67, 2002.

[20] M. Satyanarayanan. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, pages 10–17, Aug. 2001.

[21] C. Simone and S. Bandini. Integrating awareness in cooperative applications through the reaction-diffusion metaphor. *Computer Supported Cooperative Work*, 11(3-4):495–530, 2002.

[22] P. Tandler. The beach application model and software framework for synchronous collaboration in ubiquitous computing environments. *Journal of Systems and Software*, 69(3):267–296, 2004.

[23] M. Weiser. The computer for the 21st century. In *Scientific American*, volume 265 of *3*, pages 94–104, 1991.

[24] D. Weyns, A. Omicini, and J. Odell. Environment as a first class abstraction in multiagent systems. *Autonomous Agents Multi-Agent Systems*, 14(1):5–30, 2007.

[25] F. Zambonelli and H. V. D. Parunak. Signs of a revolution in computer science and software engineering. In *Proceedings of Engineering Societies in the Agents World III (ESAW2002)*, volume 2577 of *Lecture Notes in Computer Science*, pages 13–28. Springer–Verlag, 2002.

Enhancing Multi-Agent Systems with Peer-to-Peer and Service-Oriented Technologies

Marco Mari, Agostino Poggi, Michele Tomaiuolo and Paola Turci
Dipartimento di Ingegneria dell'Informazione,
Università degli Studi di Parma,
Viale G.P. Usberti, 181/A, 43100, Parma, Italy
Tel. +39 0521 905708, Fax +39 0521 905723
{mari, poggi, tomamic, turci}@ce.unipr.it

ABSTRACT
Peer-to-peer and service-oriented technologies have emerged as the dominant means for realizing scalable and interoperable distributed applications. This incontrovertible fact seems to nullify the expectation of multi-agent system researchers that agents could play a fundamental role in realizing such applications. However, from a deeper analysis, it is plain that neither peer-to-peer nor service-oriented technologies can provide by themselves the autonomy and social and proactive capabilities of agents. Motivated by such evidence, several research works have been undertaken with the aim of tackling the problem of integrating peer-to-peer and service-oriented technologies with multi-agent systems. This paper deals with this issue as well. In particular, it shows how the JADE software framework can take advantage of two such technologies both for realizing the infrastructure of distributed multi-agent systems and for supporting the interaction with non-agentized systems.

Categories and Subject Descriptors
I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence – *multiagent systems*.

General Terms
Management, Performance, Experimentation.

Keywords
peer-to-peer technologies. srvice-oriented technologies, JADE.

1. INTRODUCTION
One of the main challenges of multi-agent systems is to make the realization of scalable distributed systems easy [9][6] and as a consequence to become the main means to support legacy systems interoperability. In the last years, however, two technologies, peer-to-peer and service-oriented, have made an impressive progress and seem to have good chances of competing with multi-agent systems as the main means for the realization of scalable and interoperable systems. Conversely, neither of these two technologies is able to provide by themselves the autonomy and social and proactive capabilities of agents and thus the realization of flexible adaptive distributed systems may be difficult.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,*
May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

Confirming what has been said above, most people involved in multi-agent systems research are firmly convinced that an integration of multi-agent systems with the aforementioned technologies seems to be the most suitable solution for the realization of scalable and interoperable distributed applications. As a matter of fact, in the last years a lot of work has been presented for the integration of multi agent systems with one or both of the two technologies [24][12][13][4].

This paper is situated in this context. In particular, it shows how JADE, one of the best known and most used software framework for the development of multi-agent systems [2][14], has been extended with these technologies both to support the realization of multi-agent systems and to facilitate the interoperability with peer-to-peer and service-oriented systems. The next two sections respectively describe the peer-to-peer and the service-oriented extensions of the JADE software framework. Section 4 presents a JADE multi-agent system that has been realized exploiting the extensions of JADE and that supports collaboration via an information and expert finder service. Finally, section 6 summarizes the contributions of our work and points out future lines of research.

2. EXTENDING JADE WITH PEER-TO-PEER TECHNOLOGIES
The traditional, client-server model describes systems where computational resources and data are centralized in few servers, which respond to requests of clients. On the other hand, clients are supposed to have little capabilities and rely on the resources of servers for most of their tasks. The multi-agent model reverses this paradigm and describes systems organized in a peer-to-peer fashion, where each participant potentially has some resources to share and some services to offer to the community of agents. Thus, according to the context, each agent is able to play either the role of client or server.

JADE implements FIPA specifications for multi-agent systems, and so enables the realization of peer-to-peer distributed systems, constituted by smart and loosely coupled agents communicating by means of asynchronous ACL messages.

Nevertheless, JADE does not exploit some important features of modern peer-to-peer networks, in particular:

1. The possibility to build an "overlay network", hiding differences in lower level technologies and their related communication problems;

2. The possibility to build a completely distributed, global index of resources and services, without relying on any centralized entity.

Some multi-agent systems, like Agentscape, approached the same issues by developing a dedicated peer-to-peer network layer [17]. For JADE, we choose to integrate agent platforms into an already existing and used peer-to-peer environment like JXTA, thus, benefiting from a well tested system and exposing services to other entities participating in the networkr.

2.1 JXTA-MTP

In the course of some large projects based on agent technologies like Agentcities and @lis TechNet [18][1], some recurring problems emerged at the level of connection among remote platforms. The importance of these problems invariably grows with the cardinality and geographical extension of the interconnected infrastructure, and has been acknowledged in other similar large scale environments.

Most peer-to-peer networks specifically address this kind of problems allowing the connection of peers located behind firewalls, Network Address Translators (NATs) and Dynamic Host Configuration Protocol (DHCP) servers, or requiring different and particular protocols like HTTP or WAP. To this end, peer-to-peer networks create an overlay infrastructure above underlying diverse and problematic links in order to realize a more abstract and homogeneous ground and simplify the communications among peers.

One of the most used technologies for this purpose is JXTA [15]. JXTA technology is a set of open, general-purpose protocols that allows any connected device on the network (from cell phones to laptops and servers) to communicate and collaborate in a peer-to-peer fashion. The project was originally started by Sun Microsystems, but its development was kept open from the very beginning. JXTA comprises six protocols allowing the discovery, organization, monitoring and communication between peers. These protocols are all implemented on the basis of an underlying messaging layer, which binds the JXTA protocols to different network transports.

JXTA peers can form peer groups, which are virtual networks where any peer can seamlessly interact with other peers and resources, whether they are connected directly or through intermediate proxies. JXTA defines a communication language which is much more abstract than any other peer-to-peer protocol, allowing to use the network for a great variety of services and devices. A great advantage of JXTA derives from the use of XML language to represent, through structured documents, named *advertisements*, the resources available in the network. XML adapts without particular problems to any transport mean and it is already an affirmed standard, with good support in very different environments, to structure generic data in a form easily analyzable by both humans and machines.

With respect to connectivity, JXTA does not suppose a direct connection is available between all couple of peers. Peers can use the *Peer Endpoint Protocol* to discover available routes for sending a message to a destination peer. Particular peers, called *routers*, are in charge of responding to such queries providing route information, i.e. a list of *gateways* connecting the sender to the intended receiver. A gateway acts as a communication relay, where messages can be stored and later collected by their intended recipient, overcoming problems related to limited connectivity.

JADE, on the other hand, offers an extensible mechanism for the transport of messages among platforms, in the form of pluggable Message Transport Protocols (MTPs). The default implementations are based on IIOP and HTTP, which are both limited by the requirement of a direct connection between sender and receiver.

Exploiting the extensibility of JADE platforms, a JXTA-MTP has been developed at the University of Parma, which overcomes these limitations. To transport messages between two platforms, the new MTP uses JXTA pipes which are bound to specific endpoints (typically an IP address and a TCP port) only dynamically. JXTA pipes are advertised on the network in the same way as other services offered by peers, and provide a global scope to peer connectivity.

The JXTA-MTP implementation allows using not only plain JXTA pipes, but also secure ones with encryption and signature mechanisms guaranteeing privacy, integrity and authenticity of exchanged messages.

2.2 JXTA-ADS

What usually happens in a multi-agent platform is the cohabitation of multiple agents interacting in a common and cohesive environment, making use of a formal communication language defined by its own syntax and semantics, in order to complete tasks demanded by users. For the communication to be constructive, it is necessary to provide agents with a system allowing them to reciprocally individuate offered services. This happens thanks to the presence of a yellow pages service, provided by the platform, which can be consulted by agents when needed. However this often limits the search inside a single platform. Solutions are possible, which allow the consultation of other yellow pages services, but they necessitate the a priori knowledge of the address of the remote platform where services are hosted or listed.

An alternative solution is represented by a yellow pages service leaning on a peer-to-peer network like JXTA, thanks to which each network device is able to individuate in a dynamic way services and resources of other network devices.

Technologies inherent to web services are imposing WSDL as a standard language to publicize all different available resources. In FIPA, a simpler formalism is defined to describe services and resources exposed by agents and linked to their own domain ontology. JXTA does not establish any constraint on the way to describe and invoke services. JXTA protocols simply provide a generic framework, allowing the use of any mechanism, also WSDL or FIPA service descriptions, to exchange information needed to invoke a service.

Particular peers, called *rendezvous* peers, are in charge of indexing resources made available in the network and find them when requested by other peers. Rendezvous peers can also communicate queries to each other, if they do not possess the right information, thus enabling the discovery of advertisements beyond the local network.

In fact, in JXTA, resources are described by advertisements, which are essentially XML documents collecting metadata of available resources. Advertisements are not stored on some single machine, such as a server, or on a hierarchical infrastructure. They are distributed among rendezvous peers, which implement a distributed algorithm, called *shared resource distributed index* (SRDI), for the creation and management of the index of resources available in the network. On the basis of some indexed attributes, the mechanism can solve queries made anywhere in the rendezvous network. Basically, the global index is a loosely consistent distributed hash table, where the hash of an indexed attribute is mapped to some peer responsible for storing the actual advertisement.

FIPA has acknowledged the growing importance of the JXTA protocols, and it has released some specifications for the interoperability of FIPA platforms connected to peer-to-peer networks. In particular, in [7] a set of new components and protocols are described, to allow the implementation of a DF-like service on a JXTA network. These include:

Generic Discovery Service – a local directory facilitator, taking part in the peer-to-peer network and implementing the Agent Discovery Service specifications to discover agents and services deployed on remote FIPA platforms working together in a peer-to-peer network.

Agent Peer Group – a child of the JXTA Net Peer Group that must be joined by each distributed discovery service.

Generic Discovery Advertisements – to handle agent or service descriptions, for example FIPA df-agent-descriptions.

Generic Discovery Protocol – to enable the interaction of discovery services on different agent platforms. It's a request/response protocol to discover advertisements, based on two simple messages, one for queries and one for responses.

The JADE development environment does not provide any support for the deployment of real peer-to-peer systems because it only provides the possibility of federating different agent platforms through a hierarchical organization of the platform directory facilitators on the basis of a priori knowledge of the agent platforms addresses. Therefore, at University of Parma the JADE directory facilitator has been extended to realize a peer-to-peer network of agent platforms thanks to the JXTA technology [15] and thanks to two preliminary FIPA specifications for the Agent Discovery Service [7] and for the JXTA Discovery Middleware [8].

This way, JADE integrates a JXTA-based Agent Discovery Service (ADS), which has been developed in the respect of relevant FIPA specifications to implement a GDS. Each JADE platform connects to the Agent Peer Group, as well as to other system-specific peer groups. The Generic Discovery Protocol is finally used to advertise and discover agent descriptions, wrapped in Generic Discovery Advertisements, in order to implement a DF service, which in the background is spanned over a whole peer group.

3. EXTENDING JADE WITH SERVICE-ORIENTED TECHNOLOGIES

Industry is increasingly interested in executing business functions that span multiple applications, thus requiring high-levels of interoperability and a more flexible and adaptive business process management. The most appropriate response to this need seems to be having systems assembled from a loosely coupled collection of Web services. This technical area appears to be an interesting environment in which the agent technology can be exploited with significant advantages. As a matter of fact, several researches belonging to the agent community have dealt with the issues concerning the interconnection of agent systems with W3C compliant Web services, with the aim of allowing each technology to discover and invoke instances of the other. One evident benefit of this is the central role that agents could play in a service-oriented scenario, by efficiently supporting distributed computing [3] and allowing the dynamic composition of Web services.

The proposed integration approaches [11][16][19] denote different shades of meaning of the same idea, i.e. a wrapper or an adapter module playing the role of mediator between the two technologies. Most of them have adopted the gateway approach, providing a translation of WSDL descriptions and UDDI entries to and from FIPA specifications, thereby limiting the communication to simple request-response interactions. One approach [21], which differentiates quite substantially from the others, realizes a FIPA compliant JADE Message Transport System for Web Services enabling agents to interact through the Web with Web services preserving the FIPA compliant communication framework. But, as stated by the author himself, it only provides a solution for an integration at a low level, leaving a number of issues at higher levels unresolved.

These efforts towards an integration of the two technologies, which imply a mapping between two different ways of thinking about communication patterns, are in our opinion appreciable but at the same time quite arguable or at least unnecessarily complex.

In the following, we try to explain our point of view, analyzing and comparing the two technologies.

As far as FIPA is concerned, we can assert that the inter-agent communication is dealt with in several documents and definitely represents an important part of the overall specifications. In our attempt to be concise and rigorous, we can state that FIPA specifications target autonomous agents expected to communicate at a high level of discourse, whose contents are meaningful statements about agents' knowledge and environment. The FIPA Agent Communication Language is based on the speech act theory; messages are communicative acts that, by virtue of being sent, have effects on the knowledge and environment of the receiver as well as the sender agent. Furthermore, the language is described using formal semantics based on the modal logic. From this it clearly emerges the communication complexity which characterizes multi-agent systems compared to the very simple conversation patterns of the Web services.

It is only fair to say that Web services are also suitable for high-level communication patterns and that in the last years several efforts have been carried out in order to provide description languages enabling service orchestration and choreography and

moreover to make Web services semantically described. Nevertheless, the main goal is still to improve interoperability between entities that are not necessarily characterized by sophisticated reasoning capabilities. Finally, it is important to highlight that, in spite of the efforts dedicated to maintaining the service session, the nature of Web services is almost stateless in contrast with agents that are in essence stateful.

To sum up, the two entities, Web services on the one hand and agents on the other hand, and the corresponding communication patterns are very different from a semantic point of view. That raises doubts about a possible mapping between the two worlds, for two main reasons: (i) the essential differences between the two communication patterns, which imply a loss of descriptive power in the mapping and (ii) the unnecessary overhead that this mapping inevitably causes.

3.1 The Proposed Solution

Bearing in mind what said above, we have implemented a framework which allows a single agent to directly communicate with the world of Web services without passing through the FIPA ACL messages. When the agent needs to invoke a Web service it creates, by using our framework, the SOAP message and sends it to the provider; likewise the agent can provide its services as Web services. For the implementation of our framework we exploited AXIS2 as it is one of the most updated implementations of the Web Services standards. We have followed a similar approach also in the case of the publishing and discovery of services.

At this point a question arises: are these JADE agents still FIPA compliant? In our opinion, they are still FIPA agents since the interaction with other agents is carried out according to FIPA specifications whilst when the agent wants to communicate with the Web service world it conforms itself to the Web services standard.

Our research work does not just provide a “syntactic” support suitable for interactions with Web services compliant with the basic profile, i.e. WS-I Basic Profile 2.0, but we have also tried to cope with the issues related to a semantic support. Even though we were aware that Web services supplied by different providers usually have individual and unique semantics, described by independently developed ontologies, in our attempt towards a semantic support, we consider the simplified, but still significant, case of a shared ontology that gives a common knowledge background to the entities in the system. In order to facilitate the resolution of structural and semantic heterogeneities, semantic Web services have their interfaces semantically described by ontological concepts belonging from this shared ontology.

In the case of semantic Web services, it is appropriate to give the agent support in:

Looking for a service on the basis of the requirements to be met by the service itself;

Service invocation requiring simply the input data, irrespective of the data format and the way of interacting with the real service.

One of the major issues, we dealt with, concerned the mapping between the semantic description of the service and its real invocation. That is, how to deduce, starting from the abstract service description, which refers to ontological concepts, the concrete data required to invoke the service. The way in which this mapping is done heavily depends on the specifications chosen for the incorporation of machine understandable semantics.

Several proposals have been submitted to W3C in order to make Web services semantically described. From an analysis of such submissions two possible alternatives emerge:

The definition of a service ontology (domain-independent), to which one has to refer for a semantic description of the service. Such a description is in correlation with the WSDL document of the real service;

The definition of specifications to semantically annotate the service WSDL document. These annotations associate elements belonging to the WSDL document with concepts belonging to domain ontologies.

Both alternatives are based on the reference to one or more domain ontologies, in which the concepts, referred in the semantic part of the service description, are defined.

Among the second group, one became a recommendation last year, i.e. SAWSDL. From the first group, one is quite interesting, even if not a recommendation yet. It is a proposal submitted by a community of researchers and it is about an ontology of service, called OWL-S, characterized by a more comprehensive approach to the semantic orientation of the Web service description.

When we started our research work, SAWSDL was not a recommendation yet and OWL-S was the most visible of the several proposals. The semantic expressivity of OWL-S is rich and quite flexible. It defines a new way to describe Web service profile and so it slightly overlaps the content of the WSDL document. If one adopts OWL-S as a language to semantically describe Web services, it is necessary to handle two documents: WSDL, mainly for binding information; OWL-S, for semantic references.

Our initial choice fell down on OWL-S. Now we are extending the framework in order to include a support for the SAWSDL specification.

In order to allow agents to be able to produce and consume semantically annotated information and services, it is necessary to provide them with an ontology management support.

Ontologies were considered by the FIPA community too. In fact, ontologies enable agents to communicate in a semantic way, exchanging messages which convey information according to explicit domain ontologies. FIPA specifications, however, do not state anything about neither how to utilize ontology in the message content nor the ontology language to use.

As far as JADE is concerned, the idea which mostly inspired the design of the JADE content language and ontological support was to define an ontology independent abstract model of the content language that could be subsequently bound to any domain ontology representation expressed using an object-oriented data model. This ontological support has been conceived when the Semantic Web was on its very early stage of research and development and OWL was not already established as a standard. Consequently its expressive power is clearly limited with respect to OWL and basically allows expressing taxonomy of concepts, predicate and actions and therefore it is not able to represent completely the different application domains where JADE agent

may be used. In order to provide a JADE agent with an adequate expressive power, it is necessary either to replace or to integrate the JADE ontological support.

In the attempt to find a suitable solution to this problem in a previous work we realized a tool called OWLBeans [23]. OWLBeans, conceived with the goal of providing simple artefacts to access structured information, represents a light support allowing agents to import OWL ontologies as an object-oriented hierarchy of classes. It clearly shows a limited expressive power but still sufficient in the first phase of our development in which we focused only on a hierarchical representation of concepts for the description of input and output service parameters. In this specific context, as a matter of fact, agents do not need to face the computational complexity of performing inferences on large, distributed information sources, but an object-oriented view of the application domain is enough to allow them to complete the tasks of publishing, discovery and invocation of semantic Web services.

Well aware that the implemented framework represents a first step towards the interoperability between agents and semantic Web services, we have already starting working on the realization of a full OWL DL support supplying ontology management and reasoning functionalities, with the main purpose of providing a more expressive and powerful support and in the meantime of reducing the amount of computational resources and time required (compared to the Jena engine).

4. AN INFORMATION AND EXPERT FINDER MULTI-AGENT SYSTEM

RAP (Remote Assistant for Programmers) is a system to support communities of students and programmers during shared and personal projects based on the use of the Java programming language. RAP associates a Personal Agent with each user which helps her/him to solve problems, proposing information and answers extracted from some information repositories and forwarding answers received by "experts" on the topic selected on the basis of their profile.

4.1 System Agents

The system is mainly based on three kinds of agents: Personal Agents, Data Miner Agents and Rate Evaluator Agents.

Personal Agents (PAs) allow the interaction between a user and the different parts of the system and, in particular, between the users themselves. Moreover, these agents are responsible for building the user profile and maintaining it when the user is "on-line". User-agent interaction can be performed in two different ways: when the user is active in the system, through a Web based interface; when it is "off-line" through emails. Usually, there is a Personal Agent for each on-line user, but, when needed, Personal Agents are created to interact with "off-line" users via emails.

Data Miner Agents (DMAs) are responsible for maintaining system documentation and finding the appropriate "pieces of information" to answer the queries submitted by the users. The documentation is composed by three elements: i) code (e.g. Java classes), ii) a repository of all answers provided during system life and iii) a repository of relevant documents submitted by the users (e.g.: tutorials, manuals, presentations).

Rate Evaluator Agents (REAs) perform the calculation of users expertise score and of documents relevance with regard to the answers posed by system users. REAs are closely linked to DMAs, from which they take data for the calculations. The role of REAs is fundamental for achieving the distributed, peer-to-peer nature of the system, as discussed in section 4.4.

Each RAP platform obviously hosts a Directory Facilitator agent. Such agent not only provides the yellow pages service for its platform agents, but integrates the JXTA-based Agent Discovery Service presented in section 2.2. In this way, several RAP communities can be connected as elements of a peer-to-peer network.

4.2 System Behavior at a Glance

The system architecture is quite complex and a complete description of RAP features is beyond the scope of this paper. However, it is possible to outline the RAP behavior showing the scenario of a user asking information to her/his Personal Agent to solve a problem in her/his code. The description of this scenario can be divided in the following steps:

1. Select answer types
2. Submit a query
3. Find answers
4. Rate answer

Select answer types: the user can choose the source for the information to receive among code repositories, system documentation, old answers repositories and new answers sent by other system users.

Submit a query: the user provides the query to her/his Personal Agent. In particular, the user can query either about a class or an aggregation of classes for implementing a particular task or about a problem related to her/his current implementation.

Find answers: the Personal Agent interacts with Rate Evaluator Agents to collect the required answers. If the user requested answers from other system users, the activity is more complex and its description can be divided in three further steps:

3.1) **Receive experts rating**: a rating to answer the query is calculated for each user on the basis of her/his profile. The identity and a sketched profile of each user with a positive rating is forwarded to the requesting Personal Agent.

3.2) **Select experts**: the Personal Agent divides on-line and off-line users, orders them on the basis of their rating and, finally, presents these two lists to its user. The user can select one or more recipients for her/his query. On-line users will receive the query immediately (through their PAs), off-line users via e-mail and as they will connect to the system.

3.3) **Receive answers**: selected "expert" users can answer the query in the system Web interface or through an e-mail. Provided answers are presented to the querying user as soon as they arrive.

Rate answers: after the reception of all the answers (from every requested source), or when the deadline for sending them expired, or, finally, when the user already found an answer satisfying her/his request, the Personal Agent presents the list of read answers to its user asking her/him to rate them. Each rating is

forwarded to a Data Miner Agent that updates the involved profiles.

4.3 User and Document Profile Management

Profiles are represented by vectors of weighted terms whose values are related to the frequency of the term in the document or to the term frequency in the code wrote by the user. Document and user profiles are computed by using term frequency inverse document frequency (TF-IDF) [20] and profiles weighted terms correspond to the TF-IDF weight. Some problems have risen applying this approach in a multi-platform and distributed system: we present these problems and discuss the solutions in the next section. Each user profile is built by user's Personal Agent through the analysis of the Java code she/he has written. The profile built by Personal Agents is only the initial user's profile, and it will be updated when the user submits new software she/he has written and when the user helps other users answering their queries.

4.4 Open and Distributed Communities

An important requirement that guided the design of RAP was the support for open and distributed users communities. In fact, the retrieving of experts and information can be improved if the communities beneath the system have the capability to grow including new users or new communities.

RAP structure is open, because new users can register and access the system, but also because a registered user can acquire new skills or write new code and therefore update her/his profile. Obviously, it's also possible to delete a user.

The community beneath RAP is distributed because the whole system can consist of a dynamic group of local communities. Each community can exist and operate isolated, but can also decide to join a group of communities, sharing the experts and the document repositories. The joining and the leaving of a community are dynamic operations, the single communities of a group are fully independent, just like the components of a peer-to-peer network. A community has the capability to discover and federate with other communities thanks to the enhanced Directory Facilitator presented in section 2.2, therefore without a previous knowledge of other communities platform address.

The open and distributed nature of the system provides the best conditions for sharing and retrieving information, but also entails some significant problems in the evaluation of such information. In fact, the evaluation of both experts and documents is strongly dependent on the actual composition of the community group. For example, if a user is rated the maximum expert to answer a query, he is rated considering only the users registered in the system at the moment of the rating. If a new local community joins the group, it is possible that a user with more experience has become available. In this case, an information like "user A has called n times a method of the class X" is still valid, but an information such as "user A is the maximum expert in class X" may change according to the composition of the community.

The problem rises from the fact that, while the user personal information is still valid, the rating and all other information related to the community must be recalculated. As a matter of fact, TF-IDF algorithm can be easily used in a centralized system where all the profiles and the data to build them are managed. Our context is more complex: the system is distributed, only the Personal Agent can access the software of its user, for privacy and security reasons, and the profiles are maintained by the interaction of Personal Agents and Data Miner Agents.

For these reasons, each profile component of the RAP system is associated with two elements: an absolute element and a TF-IDF weighted element. The absolute one is dependent only on the user (or document) profile. The TF-IDF weighted element is dependent on both the user profile and the whole community profiles. While the absolute element is stored in a database, the weighted one is maintained in memory and it is recalculated when necessary. Obviously, every rating is determined on the basis of the weighted element.

The situations in which could be necessary to recompute the weighted element of one or more profile components can be slightly frequent:

- A new community joins or leaves the community group;
- A new user registers or is deleted from the system;
- Some components of a user profile change: for example the user submits new software or receives a rating for an answer.

For performance reasons, particularly if the community beneath the system is large, the process of recalculation could be too expensive in terms of system resources. In this case, each RAP platform administrator can force the system to perform the necessary recalculations only at a scheduled time.

4.5 External Sources of Information

While RAP information repositories represent a complete source of information when the query is about a class of the code repository, or about a problem already faced by other system users, clearly they lack information about a wide range of programming problems. In this sense, we are planning to open the system to external, possibly heterogeneous, sources of information. In a service-oriented scenario, the natural choice for such sources to provide their content is through a Web Service. For example, Google provides a complete set of SOAP APIs [10] to query its search engine, or Systinet [22] provides a W3C Search Service to search over a repository of tutorials and references covering most XML languages.

Exploiting the framework presented in section 3, we developed and integrated in RAP a Web Service Agent. This agent has the task to receive a query from a Personal Agent, create SOAP messages, send them to a group of registered Web Services and forward the results to the requesting Personal Agent. At the moment, the results are not rated by a Rate Evaluator Agent, but if the resource provided by a Web Service is rated useful by a user, the resource link is registered in the answer repository of the system.

5. CONCLUSION

This paper has dealt with the issue of enhancing the multi-agent systems role in the realization of scalable and interoperable systems by exploiting peer-to-peer and service-oriented technologies as key components for their realization. In particular, the paper has shown how JADE, one of the best known and most used software framework for the development of multi-agent

systems, has been extended with these two technologies both to support the realization of multi-agent systems and to facilitate the interoperability with peer-to-peer and service-oriented systems.

The resulting system shows interesting advantages. On the one hand, the exploitation of the peer-to-peer technology gives a great impulse towards the scalability and interoperability of different agent platforms and agent-based applications. On the other hand the openness towards the Web service standards gives agents the possibility to interoperate with a consolidated industrial reality and one of the most accepted mechanisms used for integration of distributed systems.

Additional advantages can be gained by coupling multi-agent systems with Semantic Web techniques [5]. In fact, agents could play a central role in a service-oriented scenario, by efficiently supporting distributed computing and allowing an automatic and intelligent composition of Web services. Furthermore they may also be the means for harmonising and making straightforward the interoperability between the services provided by peer-to-peer, service-oriented and multi-agent systems. Our future work will be oriented towards the enhancement of the JADE software framework by extending the current ontology support with Semantic Web techniques (i.e., use of OWL and related reasoning techniques) and definition and experimentation of a shared format for the publication of peer-to-peer, service-oriented and multi-agent systems services.

6. REFERENCES

[1] @lis TechNet Web Site (2008). Available from: http://www.alis-technet.org/.

[2] Bellifemine, F., Poggi, A., Rimassa, G. Developing multi agent systems with a FIPA-compliant agent framework. in Software - Practice & Experience, 31:103-128 (2001).

[3] Bergenti, F., Rimassa, G., Somacher, M., Botelho, L.M. A FIPA Compliant Goal Delegation Protocol. Communication in Multiagent Systems, Vol. 2650, pp. 223-238. 2003. Springer

[4] Buford, J. and Burg, B. Using FIPA Agents with Service-Oriented Peer-to-Peer Middleware. In Proc. of the 7th Int. Conf. on Mobile Data Management, Nara, Japan (2006).

[5] Burstein, M.H., Bussler, C., Zaremba, M., Finin, T.W., Huhns, M.N., Paolucci, M., Sheth, A.P., Williams, S.K. A Semantic Web Services Architecture. IEEE Internet Computing 9(5):72-81 (2005).

[6] FIPA Specifications Web Site . Available from http://www.fipa.org.

[7] FIPA Agent Discovery Service Specification. 2003. Available from http://www.fipa.org/specs/fipa00095/PC00095.pdf.

[8] FIPA JXTA Discovery Middleware Specification. 2004. Available from http://www.fipa.org/specs/fipa00096/PC00096A.pdf.

[9] Genesereth, M.R. An agent-based framework for interoperability. In Software Agents, J. M. Bradshaw, pp. 317-345, Ed. MIT Press, Cambridge, MA (1997).

[10] Google SOAP API home page. Available at: http://code.google.com/apis/soapsearch/index.html

[11] Greenwood D. and Calisti M. Engineering Web Service-Agent Integration. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, pages 1918–1925, The Hague, Netherlands (2004)

[12] Greenwood, D., Nagy, J., Calisti, M. Semantic Enhancement of a Web Service Integration Gateway, Proc. of the AAMAS 2005 workshop on Service Oriented Computing and Agent Based Engineering (SOCABE), Utrecht, Netherlands (2005).

[13] Huhns, M.N., Singh, M.P., Mark H. M.H., Decker, K.S., Durfee, E.H., Finin, T.W., Gasser, l., Goradia, H.J., Jennings, N.R., Lakkaraju, K., Nakashima, H., Parunak, K., Rosenschein, J.S., Ruvinsky, A., Sukthankar, G., Swarup, S., Sycara, K.P., Tambe, M., Wagner, T., Zavala Gutierrez, R.L. Research Directions for Service-Oriented Multiagent Systems. IEEE Internet Computing 9(6):65-70 (2005).

[14] JADE Web Site (2008). Available from: http://jade.tilab.com/.

[15] JXTA Web Site (2008). Available from: http://www.jxta.org/.

[16] Nguyen X. T. Demonstration of WS2JADE. In Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, pages 135–136, Utrecht, The Netherlands, (2005).

[17] Overeinder, B.J., Posthumus, E., Brazier, F.M.T. Integrating Peer-to-Peer Networking and Computing in the AgentScape Framework. In Proc. Second International Conference on Peer-to-Peer Computing, P2P02, Linköping, Sweden (2002).

[18] Poggi, A., Tomaiuolo, M., Turci, P. Using agent platforms for service composition. In. Proc. 6th International Conference on Enterprise Information Systems (ICEIS-2004), pp. 98-105, Porto, Portugal, 2004.

[19] Shafiq M. O., Ali A., Ahmad H. F., Suguri H. AgentWeb Gateway - a Middleware for Dynamic Integration of Multi Agent System and Web Services Framework. In Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, pages 267–270, Washington, DC, (2005)

[20] Salton, G.: Automatic Text Processing. (1989), Addison-Wesley

[21] Soto E. L. Fipa agent messaging grounded on web services. In Proceedings of the 3rd International Conference on Grid Service Engineering and Management, (2006)

[22] Systinet Home Page. Available at: http://www.systinet.com

[23] Tomaiuolo, M., Turci, P., Bergenti, F., Poggi, A., An Ontology Support for Semantic Aware Agents. In Proc. Seventh International Bi-Conference Workshop on Agent-Oriented Information Systems (AOIS-2005 @ AAMAS), Utrecht, The Netherlands, (2005)

[24] Willmott, S., Pujol, J.P., Cortés, U. On Exploiting Agent Technology in the Design of Peer-to-Peer Applications. Proc. of the 3rd International Workshop on Agents and Peer-to-Peer Computing, pp. 98-107, New York, NY (2004)

Interaction among agents that plan

Felipe Rech Meneguzzi
King's College London
Department of Computer Science
London, United Kingdom
felipe.meneguzzi@kcl.ac.uk

Michael Luck
King's College London
Department of Computer Science
London, United Kingdom
michael.luck@kcl.ac.uk

ABSTRACT

The development of practical agent languages has progressed significantly over recent years, but this has largely been independent of distinct developments in aspects of multiagent cooperation and planning. For example, while the popular AgentSpeak(L) has had various extensions and improvements proposed, it still essentially a single-agent language. In response, in this paper, we describe a simple, yet effective, technique for multiagent planning that enables an agent to take advantage of cooperating agents in a society. In particular, we build on a technique that enables new plans to be added to a plan library through the invocation of an external planning component, and extend it to include the construction of plans involving the chaining of subplans of others. Our mechanism makes use of *plan patterns* that insulate the planning process from the resulting distributed aspects of plan execution through local *proxy plans* that encode information about the preconditions and effects of the *external plans* provided by agents willing to cooperate. In this way, we allow an agent to discover new ways of achieving its goals through local planning and the delegation of tasks for execution by others, allowing it to overcome individual limitations.

Categories and Subject Descriptors

D.2.5 [**Artificial Intelligence**]: Programming Languages and Software; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*multi-agent systems*

General Terms

Planning, Interaction

Keywords

Agent Languages, BDI, Cooperation, Planning

1. INTRODUCTION

Agent-based software has been advocated as an ideal technique for the development of large, distributed applications, viewing them as a number of independently controlled parts that interact and cooperate to achieve their design objectives. The agent model perhaps most commonly used in the development of agent-oriented programming languages is based on the mental states of beliefs, desires and intentions, or BDI. In real-world scenarios, BDI-type agents often have a large plan-library to cope with a complex world, and need to include plans to deal with all contingencies foreseen by the designer. In consequence, much research dealing with agent languages has focused on the description of plans used by an *individual* agent to interact with the world. Although in multiagent systems, agents are assumed to be able to use interaction to achieve goals, agent languages seldom provide mechanisms to do so, and cooperation is generally developed in an *ad hoc* fashion. Even when cooperation is involved, it tends to use a highly specialised version of any of a number of existing cooperation techniques, assuming a distributed but ultimately *predefined* set of abilities in the society.

Previous work has shown the applicability of planning algorithms in the generation of new individual plans through composition of existing plans in a plan library [12]. When a planning-capable agent needs to achieve a new goal, it searches its plan library for applicable plans. If no suitable plan is found in the plan library, the planner is invoked in an attempt to generate a new plan to satisfy the desired goal. Plans generated by the planner in this way are added to the plan library and become available to the agent to solve future instances of the particular goal. However, if the planner fails to generate a new plan, this (normally) means that the proposed goal is impossible to achieve, given the world state and capabilities known to the agent at the time of planning. Such an approach to planning has been studied for individual agents, planning over their own capabilities, under the premise that these capabilities are static throughout its life cycle.

However, in a multi-agent environment the limitations of an individual agent may be overcome with the help of others, either by delegating tasks to other agents, or by learning new ways of achieving goals. This means that the assumption that capabilities are static no longer holds. In this case, a failure to generate a plan from an individual's capabilities does not necessarily mean the goal is impossible, since other agents in the society might have complementary capabilities. If an agent is capable of generating new plans at runtime by taking into consideration the capabilities of others, new *multiagent* plans can be used to overcome individual limitations, and can also be added to the plan library for future use. In a simplistic approach, a planning capable agent interpreter can be used to achieve this effect.

Cooperative action involves communication and coordination, as well as an increased degree of risk for the success of a plan, given that the agents relied upon may break their commitments to achieve their own goals. But from the planner's perspective, the composition of new plans based on preconditions and effects upon the environment can be performed independently of these factors. It is important to point out that, though there are a number of issues that

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

must be addressed in order to perform planning in distributed systems [4], communication and coordination can be abstracted away from the planning process and inserted at a later point in time [9]. In this paper we develop a new technique for agents with plan generation capabilities to cooperate in a multiagent society. In particular, our technique relies on plan patterns (described in Section 3.2) that encapsulate communication and coordination in such a way that the planning algorithm can ignore them when chaining operators in a plan. Though many existing approaches to cooperative action handle communication and coordination together with plan composition [11], we choose to separate these two tasks in order to allow the use of *off-the-shelf* planning algorithms. The rationale behind this choice of approach is as follows:

- there is a wide range of local planning algorithms;
- active research on planning algorithms yields new potentially useful algorithms;
- some planning algorithms are better suited for certain specific domains;
- integrating communication and cooperation in the planning algorithm is not always easy; and
- our approach delegates actions to the agent architecture, allowing new planners to be used seamlessly.

Even if the issues arising from interaction can be ignored by the planner, they must be addressed in order to ensure the long-term effectiveness of the plan library. In particular, relying on third parties to accomplish one's goals can be a problem due to unreliability and broken commitments. Moreover, it is possible that an agent whose cooperation is necessary for some plan in the plan library may leave the society. This renders such a plan not only useless, but also damaging to an agent's efficiency if the plan is eventually selected to achieve some goal, since this plan will always require the agent to drop it after wasting the effort of starting to execute it. Therefore, we provide a mechanism to *clean up* cooperative plans when they become obsolete. By extracting key information from other agents' plans, particularly in relation to the declarative consequences of local plans, an agent can be informed of the problem-solving capabilities of others, allowing it to delegate the achievement of specific world-states, and using this information in its own planning process.

Our contribution in this paper is twofold: a generic technique to reduce multiagent planning into a traditional planning problem, and the practical integration of such a technique in a BDI-like agent language. In our approach, external plans are encapsulated into patterns of local plans in order to abstract the communication and coordination aspects away from the planner.

The paper is organised as follows: in Section 2 we summarise the background work upon which our own contribution is based, reviewing AgentSpeak(L) and AgentSpeak(PL); we then proceed to explain in detail our multiagent planning technique in Section 3; followed by a discussion of related work in Section 4; finally, in Section 5, we summarise and conclude.

2. BACKGROUND

2.1 Cooperation

Cooperation is often cited as one of the main characteristic properties of multiagent systems [7, 6]. Yet there are several different modes of cooperation that can be identified: *i)* multiple agents acting towards a common joint goal; *ii)* one agent acting to achieve goals for another agent; and *iii)* agents synchronising their actions so as to avoid negative interference.

The first, and most common mode of cooperation in agents consists of a group of agents sharing a possibly implicit joint goal and acting to achieve this goal in a coordinated way. This goal might be negotiated at runtime or exist in all agents by design. The second possible mode of cooperation consists of one or more agents performing actions that are not directly related to their own goals, but rather support the achievement of the goals of another agent. The third and final mode of cooperation commonly considered consists of agents agreeing on some coordination of their individual actions towards their individual goals in such a way that no agent jeopardises the operation of another. Because our starting point is one individual agent, seeking to achieve its goals through the assistance of another, our focus in this paper is on the second mode of cooperation.

2.2 AgentSpeak(L)

AgentSpeak(L) [16] is an agent language, as well as an abstract interpreter for the language, and follows the *beliefs, desires and intentions* (BDI) model of practical reasoning. In simple terms, a BDI agent tries to realise the *desires* it *believes* are possible by committing to carrying out certain courses of action through *intentions*. The language of AgentSpeak(L) allows the definition of *reactive procedural plans*, so that plans are defined in terms of events to which an agent should react to by executing a sequence of steps (*i.e.* a procedure). Plan execution is further constrained by the context in which these plans are relevant. Here, a plan is executed under the assumption that some implicit goal is being accomplished by the plan at the particular moment.

The control cycle of an AgentSpeak(L) interpreter adopts plans in reaction to events in the environment and executes their steps. If the step is an action it is executed, while if the step is a goal, a new plan for the goal is added into the intention structure. Failures may take place either in the execution of actions, or during the processing of subplans. When such a failure takes place, the plan that is currently being processed also fails. Thus, if a plan selected for the achievement of a given goal fails, the default behaviour of an AgentSpeak(L) agent is to conclude that the goal that caused the plan to be adopted is not achievable. This control cycle[1] strongly couples plan execution to goal achievement.

In order to better understand the relationship between the control cycle and the plan library, it is necessary to introduce the notation of AgentSpeak(L) plans. Events on an agent's data structures that can trigger the adoption of plans consist of additions and deletions of goals and beliefs, and are represented by the plus ($+$) and minus ($-$) sign respectively. Goals are distinguished into *test goals* and *achievement goals*, denoted by a preceding question mark (?), or an exclamation mark (!), respectively. For example, the addition of a goal to achieve g is represented by $+!g$. Belief additions and deletions arise as the agent perceives the environment, and are therefore outside its control, while goal additions and deletions only arise as part of the execution of an agent's plans. Plans in AgentSpeak(L) are represented by a header comprising a triggering condition and a context, as well as a body describing the steps the agent takes when a plan is selected for execution as is illustrated in Figure 1. If e is a triggering event, $b_1, \ldots, b_m$ are belief literals, and $h_1, \ldots, h_n$ are goals or actions, then $e : b_1 \& \ldots \& b_m \leftarrow h_1; \ldots; h_n.$ is a plan. As an example, consider a plan associated with the triggering event $!move(O, A, B)$ corresponding to the goal of moving an object O from A to B, where:

[1]For a full description of AgentSpeak(L), see d'Inverno *et al.* [5]

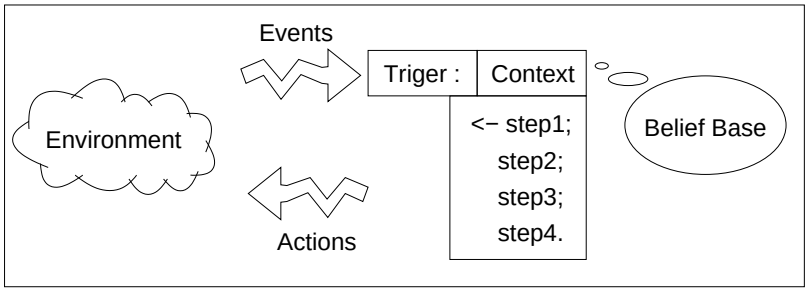


Figure 1: AgentSpeak(L) plan and dynamics.

- e is `!move(O,A,B);`
- `at(O,A)` and **`not`** `at(O,B)` are belief literals; and
- `-at(O,A)` and `+at(O,B)` are two steps in the plan body, consisting of information about belief additions and deletions.

The plan is then as follows:

```
+!move(O,A,B) : at(O,A) & not at(O,B)
    <- -at(O,A);
       +at(O,B).
```

When this plan is executed, it should result in the agent believing that O is no longer in position A, and then believing it is in position B. For an agent to rationally want to move O from A to B, it must believe O is at position A and not already at position B.

2.3 Planning in AgentSpeak(PL)

AgentSpeak(PL) [12] is an extended AgentSpeak(L) interpreter that integrates a planning module capable of generating new high-level plans by chaining lower-level plans in an agent's plan library. Planning in AgentSpeak(PL) relies on a process that extracts information about the declarative *consequences* of simple AgentSpeak(L) plans and uses this information, together with these plans' *context conditions*, to generate equivalent STRIPS-like planning operators. Here, we review the relevant aspects of AgentSpeak(PL), which we later use in the description of our multiagent technique.

The design of a traditional AgentSpeak(L) plan library follows a similar approach to programming in procedural languages, where a designer typically defines fine-grained actions to be the building blocks of more complex operations. These building blocks are then assembled into higher-level procedures to accomplish the main goals of a system. Analogously, an AgentSpeak(L) designer traditionally creates fine-grained *plans* to be the building blocks of more complex operations, typically defining more than one plan to satisfy the same goal (*i.e.* sharing the same trigger condition), while specifying the situations in which it is applicable through the *context* of each plan. Here, STRIPS actions are likened to low-level AgentSpeak(L) *plans*, since the effects of primitive AgentSpeak(L) actions are not explicitly defined in an agent description.

Once the building-block procedures are defined, higher-level operations must be defined to fulfil the broader goals of a system by combining these building blocks. In a traditional AgentSpeak(L) plan library, higher-level plans to achieve broader goals contain a series of goals to be achieved by the lower-level operations. This construction of higher-level plans that make use of lower-level ones is analogous to the planning performed by a propositional planning system. By doing the *planning themselves*, *designers* must cope with every foreseeable situation the agent might find itself in, and generate higher-level plans combining lower-level tasks accordingly. Moreover, the designer must make sure that the subplans being used do not lead to conflicting situations. In AgentSpeak(PL), by contract, this responsibility is delegated to a STRIPS planner.

Plans resulting from propositional planning can then be converted into sequences of AgentSpeak achievement goals to comprise the body of new plans available within an agent's plan library. Here, an agent can still have high-level plans pre-defined by the designer, so that routine tasks can be handled exactly as intended. At the same time, if an unforseen situation presents itself to the agent, it has the flexibility of finding novel ways to solve problems, while augmenting the agent's plan library in the process.

3. PLANNING AND COOPERATION

As we have seen, when an agent has exhausted its individual options to achieve a goal, it may be able to accomplish this goal through others. In order to generate new plans that rely on cooperation with others, we define a practical strategy for multi-agent planning and cooperation that allows an agent to share the knowledge of the consequences of its plans so that others can delegate parts of their high-level plans and achieve new goals. We introduce *external plans*, which are plans *owned* by one agent, whose declarative consequences are known by others and can, therefore, be requested by others to help achieve their aims. Newly constructed external plans can be integrated into multi-agent plans generated through classical planning problems by considering their preconditions and consequences and equating them to STRIPS/PDDL operators, as described in Section 2.3. These newly created plans are then integrated into an agent's plan library for future use and efficiency gains. Furthermore, our strategy takes into consideration the unreliability of cooperation in the context of self-interested and unreliable agents by associating failure handling plans to manage multi-agent plans, and eventually to remove plans that include such unreliable partners.

In more detail, given an agent with plans it is willing to execute on behalf of others (*i.e.* its *shared plans*), our technique consists of automatically generating plans in both the sharer's plan library and the requester's plan library using reusable *plan patterns*. These new plans encode all the communication necessary for a requesting agent to delegate the achievement of the external plan, and encapsulate information about the declarative effects of such an external plan, allowing a planning module on the requester's side to use these plans in newly created plans. Moreover, we use plan patterns to generate failure handling plans to cope with the potential unreliability of the sharer.

In this section we define three plan patterns that generate new plans based on an existing plan an agent is willing to execute on

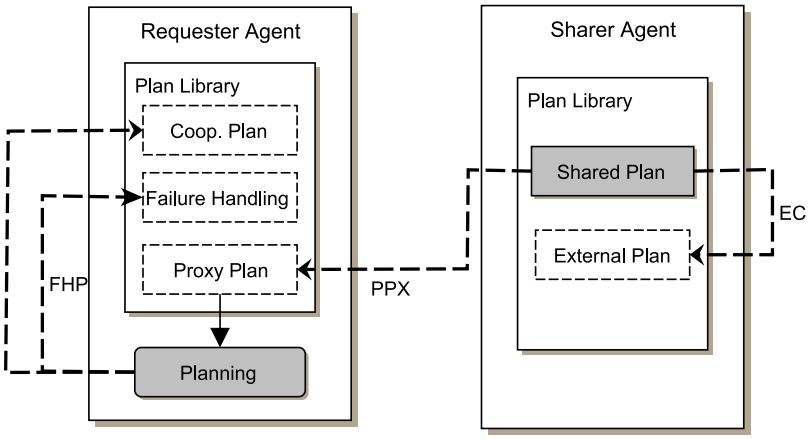


Figure 2: Plan patterns involved in the sharing and use of a plan.

behalf of others, generating the necessary framework for a form of cooperation based on delegation without the need for the designer to predefine cooperative plans. More specifically, given a *shared plan*, we define an *external plan* (EC) pattern that includes the steps necessary for another agent to request the execution of the shared plan. On the requester's side, we define a *proxy plan* (PPX) pattern that encodes the declarative information of the shared plan's preconditions and consequences, and contains the steps necessary for the requester to request the remote execution of the shared plan. Ultimately, proxy plans can be used by the planning process of AgentSpeak(PL) as if they were local plans, to provide new *cooperative plans*, but given the uncertain nature of agent cooperation, there is also a need to provide *failure handling plans* (FHP) to cope with unreliable partners. These patterns and their resulting plans (*i.e.* the plans that are generated from the plan pattern) are summarised in the diagram of Figure 2, where dashed arrows represent the creation of new plans through a plan pattern.

3.1 Communication for Cooperation in AgentSpeak

Our technique assumes a BDI-style language [16] with a construct for declarative goals, and speech-act based communication. We also assume two other language features: the ability to *annotate plans* with additional information; and the notion of *internal actions*. Examples of agent languages suitable for implementing this strategy are CANPLAN2 [17] and Jason [3]. Descriptions of agent plans throughout this section use Jason, but these plan definitions can be easily converted to any BDI-like agent language. Jason is a recently developed AgentSpeak(L) interpreter with a number of extensions necessary for our technique to function in practice. Here, we summarise the language features we use in the descriptions throughout this section, giving notation details when relevant.

3.1.1 Internal actions

The common understanding of agent actions is that they are environment transformation operators, so that when an agent invokes an action, some consequence in the environment is expected. However, when some custom computation needs to take place within a single reasoning cycle, Bordini *et al.* use the concept of an *internal action* in AgentSpeak(XL) [2]. This allows an agent to access extensible libraries of custom procedures that can be executed instantaneously by an agent. Unlike traditional actions, internal actions do not cause changes in the environment, and since they are executed instantly, they can be included in either the body or the context of a plan, to refine the process of selecting applicable plans. Syntactically, internal actions are denoted in the language by a preceding "." character, so the invocation of a *check* internal action with two parameters is represented as $.check(a, b)$.

3.1.2 Speech-act based communication

Effective cooperation between autonomous agents requires some form of communication, typically using an agent communication language, such as FIPA or KQML [19]. From an agent language perspective, Moreira *et al.* [13] have introduced an operational semantics of speech-act based communication for AgentSpeak(L), defining plan rules for handling several of the performatives defined by Searle [18]. These plan rules are given from both a sender and a receiver point of view, allowing them to be implemented in practical AgentSpeak(L) interpreters. In this paper we are concerned with three performatives:

- *ask*, used by an agent to request information from others;
- *tell*, used by an agent to supply information to others; and
- *achieve*, used by an agent to request another agent to achieve a procedural goal.

From an operational perspective, we consider an implementation of agent message passing using the concept of *internal actions* described above, because messages between agents are not expected otherwise to cause the environment to change. Messages are therefore sent using the $.send$ internal action, which takes three parameters: the identification of the receiver, the performative, and the message content [2]. In terms of representation of beliefs, annotations [2] have been used to provide additional information regarding the source of events from external communication rather than the environment. Thus, if the addition of the belief $time(hockey, 1020)$ by $randall$ is a result of communication from $dante$ rather than a simple perception, the event posted to $randall$ is represented as $time(hockey, 1020)[source(dante)]$, denoting this belief's origin.

[2]In this paper we provide a simplified overview of how these performatives are operationalised in AgentSpeak(L), overlooking a number of details regarding cooperation policies, and more complex handling of communication-related event processing by AgentSpeak(L). For additional information on these details, consult [13].

When an agent sends a message with an *ask* performative, it wants to ascertain that some expression unifies with another's belief base. For example, suppose agent *randall* wants to know the time of the hockey game, stored in the belief base of agent *dante* as the belief $time(hockey, T)$. To discover this information, it executes an internal action $.send(dante, ask, time(hockey, T))$, which causes an event $+?time(hockey, T)$ to be posted to *dante*. If *dante* accepts the message, and has $time(hockey, 1020)$ in its belief base, the $.send$ action in *randall* is executed successfully, resulting in T being unified with 1020. Notice that since the effect of this send is an event in the receiving agent, it might be handled by a plan with a triggering event matching the query being made to it, rather than a direct query to its belief base.

Similarly, when an agent sends a message with a *tell* performative, it wants to make another agent aware of some belief expression. Now suppose *dante* wants to make *randall* aware that the hockey game is at 1020 by executing the action $.send(randall, tell, time(hockey, 1020))$. If *randall* accepts this message, it causes the event $+time(hockey, 1020)$ to be posted to *randall*.

Finally, when an agent wants another agent to adopt a particular achievement goal, it sends a message with an *achieve* performative. So if *dante* wants *randall* to come to the hockey game now, and it knows that *randall* has a plan to come to the game associated with the triggering event $+!comeToHockey$, it executes $.send(randall, achieve, comeToHockey)$. Again, if *randall* accepts this message, $+!comeToHockey$ is posted to *randall*, and the plan it executed.

3.2 A multiagent planning mechanism

When an agent has failed to achieve a goal through its individual capabilities and its previously known cooperative strategies, it engages in multi-agent planning to try and solve the problem with a new cooperative plan. Our technique is divided into three main parts: the discovery of potential cooperation partners, the creation of cooperative plans while abstracting cooperation, and the execution of multiagent plans.

3.2.1 Plan patterns

While many researchers have chosen to create new languages to add notions such as declarative goals [21, 3] and failure handling mechanisms [20, 17], it is possible to represent these, and many other notions using simpler, existing agent languages. For example, in AgentSpeak(L), all of these notions can be represented by multiple related plans, as shown by Hübner *et al.* [10], who introduce the notion of *plan patterns* to facilitate the designer's task of creating multiple, related, plans that serve a particular purpose.

Here, we consider a plan pattern to be an agent program rewriting rule with a numerator describing the original plan (or plans) description, and a denominator describing the resulting agent program. So, for example, if we wish to define a plan pattern that adds a printed message before and after a certain plan body b is executed, called **PDB** (Plan Debug), the rule is defined as:

$$\frac{+e : c \leftarrow b.}{+e : c \leftarrow .print(\text{“}Start\text{”}); b\,; .print(\text{“}End\text{”}).}\textbf{PDB}$$

3.2.2 Primitives

In order for an agent to find external plans in a society, it must seek partners willing to carry out plans on behalf of the requesting agent. These willing partners then send declarative information about their plans, that is their preconditions and effects. Here, we are not concerned with the actual mechanism used in the discovery of partners, and plan patterns are meant to be an abstraction of any of a number of existing partner selection mechanisms, such as that described in [14]. In this description of our method, we assume that cooperation partners have already been selected somehow, and our capability discovery method consists of broadcasting a request for external plans, which is answered by all available agents in the society. However, if a partner selection mechanism is in place, the requests for external plans will only be sent by selected cooperation partners.

Partners wishing to inform others of their external plans need to gather the plan invocation parameters, preconditions and declarative effects and send this information to their peers. This information can be retrieved using the same process as in AgentSpeak(PL), but instead of using this information to generate a STRIPS-like operator description, an agent sends this as a reply to another agent requesting external plans, along with the identification of the agent supplying the external plan. This is represented in the tuple $\langle g, a, P, E\rangle$, where:

- g is the achievement goal (including parameters) in the sharing agent's plan library that will be adopted on behalf of the requesting agent;
- a is the identifier of the sharing agent that owns the external plan;
- P is a set $\{p_0, \ldots, p_n\}$ of preconditions of g; and
- E is a set $\{r_0, \ldots, r_m\}$ of declarative effects expected to hold after the external plan is executed.

This information is used in the creation of plan patterns that serve as local placeholders for the invocation of externally executed plans, which we call *proxy plans*. The creation of proxy plans is detailed next.

3.2.3 Creating proxy plans

Once an agent is aware of the external plans of others in the same environment, it can try to use these capabilities in its own problem-solving. In this approach, we make the *external* aspect of *shared plans* transparent to an agent's local planner through *proxy plans*. These proxy plans describe the expected outcome of a successful invocation of a third party capability and encapsulate the communication and coordination necessary for effective cooperation. A proxy plan pattern **PPX** for an external plan $\langle g, a, P, E\rangle$, where $P = \{p_0, \ldots, p_n\}$, and $E = \{b_0, \ldots, b_m\}$ (and b_i are belief additions or deletions) is:

$$\frac{\begin{array}{rl} +!g : p_0 \& \ldots \& p_n \leftarrow & b_0; \ldots; b_m \end{array}}{\begin{array}{rll} +!remoteG : & p_0 \& \ldots \& p_n \& ready(a, g) & \\ \leftarrow & .send(a, achieve, requestG); & \\ & .wait(done(g)); & \\ & b_0; \ldots; b_m. & \\ +!check(a, g) : & true & \\ \leftarrow & +ready(a, g). & \end{array}}\textbf{PPX}_{g,a,P,E}$$

This plan pattern creates two plans, one of which replicates all the logical constraints required for a to be successful in executing this plan locally. The plan body includes a communication action ($.send$) that uses the *achieve* performative to request the sharing agent to carry out the specified plan, followed by an action to wait for confirmation that the plan was executed. Finally, the plan pattern replicates the belief additions expressed in the sharing agent's external plan, so that the planning process of AgentSpeak(PL) [12]

can process this plan in the same way as it would process local plans.

In addition to the action-related part of the proxy plan to invoke the external plan, one may also want to check that the owner of the external plan is ready and willing to adopt the external plan. This is represented in the **PPX** plan pattern by the precondition $ready(a, g)$, which is added to those preconditions already present in the original external plan, and is the result of an extra plan to ensure that the sharing agent will actually carry out that action when the requesting agent needs it to do so. In the **PPX** plan pattern, this plan is simply a placeholder for any mechanism used to ascertain the reliability of a cooperation partner, which can be replaced by any mechanism preferred by the designer. Such a mechanism can be introduced using a new **CA** (check agent) plan pattern, which rewrites the $check$ plan so that it calls a plan in the plan library associated with this mechanism. For example, if there is a trust verification mechanism associated with a *verifyTrust* achievement goal (which we will not specify, but assume to be specified by the designer), a plan pattern **CA** for the readiness of an agent to execute external plan $\langle g, a, P, E\rangle$ through an achievement goal *verifyTrust* is:

$$\frac{+!check(a,g) : true \leftarrow +ready(a,g).}{\begin{array}{rl} +!check(a,g) : & true \\ \leftarrow & !verifyTrust(a,g); \\ & +ready(a,g). \end{array}} \mathbf{CA}_{g,a}$$

3.2.4 Creating external plans

An important property of our proxy plans is that they succeed when the sharer agent succeeds, and fail if either the sharer agent fails in its execution or it refuses to carry out its commitment. Hence, from the requester agent's point of view, the execution of a local plan and an external plan is the same.

Naturally, an agent sharing an external plan needs to have in its plan library the achievement goal that corresponds to the *achieve* performative sent by the requesting agent. We refer to this achievement goal as a *plan endpoint* to the **PPX** plan pattern, which is associated with an actual plan in the sharing agent's plan library. The external plan, therefore, is generated from a local plan in the sharer's plan library using the **EP** (external plan) pattern, which is as follows:

$$\frac{+!g : e \leftarrow b.}{\begin{array}{rl} +!g : e & \leftarrow b. \\ +!requestG[source(S)] & : true \\ \leftarrow & !g; \\ & .send(S, tell, done(g)). \end{array}} \mathbf{EP}_g$$

3.2.5 Creating cooperative plans

Given the properties of the proxy plans described above, it is easy to use the planning approach of AgentSpeak(PL) to generate new multi-agent plans, since the AgentSpeak(PL) planning module is insulated from the communication and cooperation aspects of planning. However, although the generation of a sequence of actions (from a cause and effect perspective) does not depend directly on whether it includes external and internal capabilities, high-level plans that depend on the compliance of third parties must contain guards to prevent initiating the plan when it has become infeasible. These guards are derived by propagating the preconditions of external proxy plans to the precondition of the high-level plan generated by the planning module. Propagating these preconditions ensures that a plan will not be initiated until all parties are ready to comply with requests for cooperation, while making sure that the cooperating agent is queried for availability just before its cooperation is needed.

```
+!goal_conj([closed(store)]) : at(randall, store)
     & ready(randall)
   <- !remoteClose(store).
```

Listing 1: A cooperative plan.

As an example, suppose that $dante$ is aware that $randall$ can achieve a goal to close the store on its behalf. If $dante$ needs $randall$ to close the store on its behalf, it requires $randall$ to be at the store, and results in the store being closed; a cooperative plan to achieve these goals generated in our system is shown in Table 1.

When a cooperative plan is adopted by an agent, it eventually reaches the step corresponding to the adoption of the proxy plan ($remoteG$). The proxy plan causes this agent to send a message to the sharer requesting it to execute its external plan ($requestG$), which corresponds to delegating the adoption of a plan to achieve goal g in the sharer's plan library. If the plan to achieve g is executed successfully, the sharer sends confirmation of having achieved g. This sequence of events is illustrated in Figure 3.

3.2.6 Failure handling for new plans

Although the ability to create new plans taking advantage of the external plans of other agents allows the creation of plans that achieve goals otherwise impossible to an agent, the dependence on other self-interested agents poses another challenge, coping with possibly unreliable partners. Plans created at design time tend to be very efficient by making assumptions about aspects of the environment that do not change at runtime, whereas the *generation* of plans at runtime involves a great deal of computational effort. However, plans created in a dynamic society in which autonomous agents may join and leave at any point in time cannot make many assumptions regarding the availability of capabilities shared by third parties. The likelihood of failure for plans that depend on others can, therefore, be considered greater than for plans that rely on an individual's own capabilities. Thus, it is necessary for dynamically generated plans, especially those that depend on unreliable capabilities, to have associated failure handling plans. Here, handling plan failures is important to ensure that an agent can cope with faults due to failed cooperation. It also allows an agent to manage its plan library in the long term, removing plans that are no longer relevant due to the absence of, or consistent lack of reliability of, necessary parties. For example, if an agent creates a plan that involves cooperation with an agent a, we introduce a *failure handling plan* **FHP** pattern that removes the failed plan when a fails to cooperate for some reason, as follows:

$$\frac{+!goal\_conj([g_1, \ldots, g_n]) : e \leftarrow b.}{\begin{array}{rl} +!goal\_conj([g_1, \ldots, g_n]) & : e \leftarrow b. \\ -!goal\_conj([g_1, \ldots, g_n]) & : notready(a) \\ \leftarrow & .remove\_plan(goal\_conj([g_1, \ldots, g_n])). \end{array}} \mathbf{FHP}_a$$

4. RELATED WORK

Previous work by Ancona *et al.* [1] provides a cooperation technique that allows agents to expand their problem solving capabilities by exchanging plans at runtime. Although this technique relies on a very similar basic agent framework (aside from the planning component), it has a distinct approach to addressing the shortcomings of an agent, as it relies on an agent receiving entire plans from

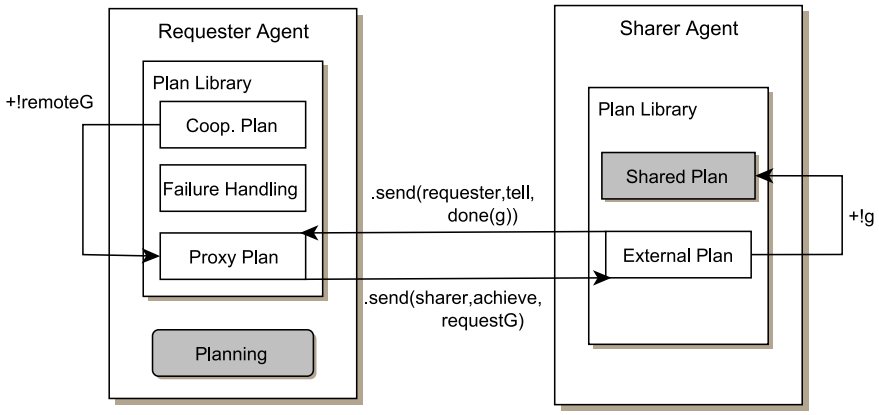


Figure 3: Proxy plan communication.

others. In particular, it assumes that all agents in an environment are able to execute the same set of basic actions, which may not be the case in many real world scenarios. For example, agents might require different levels of authorisation to perform specific actions in the environment: an agent running in a user-level account, doing maintenance in a Unix filesystem may need to change a file that is owned by the root user, and clearly the plans that the root can execute cannot simply be sent to this agent. Ancona's approach is complementary to ours in the sense that it can, for example, replace the planning module we use to generate new plans from scratch and allow an agent to get new plans from others.

It may be argued that creating cooperative plans using preconditions and effects information in AgentSpeak(L) is akin to *Service Oriented Architectures* (SOA), through web services being shared by a directory service accessed through some protocol like the *Universal Description, Discovery and Integration* (UDDI) protocol [15]. Indeed, web services are a possible technology for the instantiation of an agent system using an SOA to provide web-protocols for the communication layer of such a system. Unlike web services on their own, however, agents have intentionality, and do not necessarily carry out the requests of a client. Directory services could also be used in a web service-based implementation, but they add a centralising characteristic that is not entirely necessary for our technique, since the directory service does not take into account the dynamic nature of an agent's willingness to cooperate; that is, an agent *A* may agree to execute an action on behalf of agent *B* at one point in time, but not at another, whereas a service is expected always to respond in the same way.

5. CONCLUSIONS

By taking advantage of recent developments in practical agent languages, we have described a practical, yet flexible, technique for multiagent planning. This technique extends previous work on agent planning [12] to take advantage of the availability of cooperating agents in a society, allowing agents to overcome individual limitations by delegating *parts* of locally generated plans for execution by others. In this paper we have shown how this technique can be implemented using recent extensions to the AgentSpeak(L) language, without affecting the generality of our approach, since any other BDI-like language with declarative goals and communication capabilities can be extended with the planning we propose.

The focus of the paper is on the structural and functional aspects of the plan library, and as a consequence we have sidestepped any detailed account of how to address two major issues with cooperation in agents: the distribution of the planning effort, and the evaluation of reliability of cooperation partners. However, by modularising our technique, a designer can choose from the existing body of work in both these areas. Moreover, we acknowledge that issues of trust and reliability of cooperation partners are of paramount importance in any deployment of a system composed of agents that use our technique, but this is a separate issue, and is isolated from the rest of our planning process. Regarding the issue of distribution, although the classical planning module leveraged from AgentSpeak(PL) [12] is simple and centralised, we see no hurdles in using our technique with distributed plan formation algorithms, such as that proposed by Zhang *et al.* [22]. In this respect, our method is flexible in that it allows any planning algorithm with a PDDL [8] compatible planner to be used in the planning module.

Acknowledgments

The first author is supported by Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) of the Brazilian Ministry of Education.

6. REFERENCES

[1] D. Ancona, V. Mascardi, J. F. Hübner, and R. H. Bordini. Coo-agentspeak: Cooperation in agentspeak through plan exchange. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 696–705, 2004.

[2] R. H. Bordini, A. L. C. Bazzan, R. de O. Jannone, D. M. Basso, R. M. Vicari, and V. R. Lesser. AgentSpeak(XL): efficient intention selection in BDI agents via decision-theoretic task scheduling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1294–1302, 2002.

[3] R. H. Bordini, M. Dastani, J. Dix, and A. E. Fallah-Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, 2005.

[4] M. E. desJardins, E. H. Durfee, C. L. O. Jr., and M. J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 20(4):13–22, 1999.

[5] M. d'Inverno and M. Luck. Engineering AgentSpeak(L): A formal computational model. *Journal of Logic and Computation*, 8(3):233–260, 1998.

[6] M. d'Inverno, M. Luck, and M. Wooldridge. Cooperation structures. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence*, pages 600–605, 1997.

[7] J. E. Doran, S. Franklin, N. R. Jennings, and T. J. Norman. On cooperation in multi-agent systems. *Knowledge Engineering Review*, 12(3):309–314, 1997.

[8] M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[9] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Elsevier, 2004.

[10] J. F. Hübner, R. H. Bordini, and M. Wooldridge. Programming declarative goals using plan patterns. In M. Baldoni and U. Endriss, editors, *Proceedings of the Fourth Workshop on Declarative Agent Languages and Technologies*, volume 4327 of *LNCS*, pages 123–140. Springer, 2006.

[11] D. Kalofonos and T. J. Norman. An investigation into team-based planning. In *2004 IEEE International Conference on Systems, Man and Cybernetics*, pages 5590–5595, 2004.

[12] F. Meneguzzi and M. Luck. Composing high-level plans for declarative agent programming. In *Proceedings of the Fifth Workshop on Declarative Agent Languages*, pages 115–130, 2007.

[13] Á. F. Moreira, R. Vieira, and R. H. Bordini. Extending the operational semantics of a BDI agent-oriented programming language for introducing speech-act based communication. In J. A. Leite, A. Omicini, L. Sterling, and P. Torroni, editors, *Proceedings of the First Workshop on Declarative Agent Languages and Technologies*, volume 2990 of *LNCS*, pages 135–154. Springer, 2003.

[14] S. Munroe, M. Luck, and M. d'Inverno. Motivation-based selection of negotiation partners. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 1520–1521, 2004.

[15] Organization for the Advancement of Structured Information Standards. Introduction to UDDI:Important Features and Functional Concepts. Online, 2004. http://uddi.xml.org/files/uddi-tech-wp.pdf.

[16] A. S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In W. V. de Velde and J. W. Perram, editors, *Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, volume 1038 of *LNCS*, pages 42–55. Springer, 1996.

[17] S. Sardina and L. Padgham. Goals in the context of BDI plan failure and planning. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 16–23, 2007.

[18] J. R. Searle. *Speech Acts : An Essay in the Philosophy of Language*. Cambridge University Press, 1969.

[19] M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, 31(12):40–47, 1998.

[20] J. Thangarajah, J. Harland, D. Morley, and N. Yorke-Smith. Aborting tasks in BDI agents. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 8–15, 2007.

[21] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative & Procedural Goals in Intelligent Agent Systems. In D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, editors, *Proceedings of the Eighth International Conference on Principles and Knowledge Representation and Reasoning*, pages 470–481. Morgan Kaufmann, 2002.

[22] J. F. Zhang, X. T. Nguyen, and R. Kowalczyk. Graph-based multi-agent replanning algorithm. In *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 793–800, 2007.

Implementing a Cognitive Model in Soar and ACT-R: A Comparison

Tijmen Joppe Muller
tijmen.muller@tno.nl
Department of Training and Instruction
TNO Defence, Security and Safety
P.O. Box 23, 3769 ZG
Soesterberg, The Netherlands

Annerieke Heuvelink
annerieke.heuvelink@tno.nl
Department of Training and Instruction
TNO Defence, Security and Safety
P.O. Box 23, 3769 ZG
Soesterberg, The Netherlands

Fiemke Both
f.both@few.vu.nl
Department of Artificial Intelligence
Vrije Universiteit Amsterdam
De Boelelaan 1081a, 1081 HV
Amsterdam, The Netherlands

ABSTRACT

This paper presents an implementation of a cognitive model of a complex real-world task in the cognitive architecture Soar. During the implementation process there were lessons learned on various aspects, such as the retrieval of working memory elements with relative values, alternative approaches to reasoning, and reasoning control. Additionally, the implementation is compared to an earlier implementation of the model in the ACT-R architecture and both implementations are discussed in terms of cognitive theories.

1. INTRODUCTION

People performing tasks in uncertain and dynamic environments require much training in order to gain the necessary expertise. However, the nature of these tasks makes it hard to set up real world training. An appropriate alternative for training decision making in complex environments is scenario-based simulation training [17]. To create a useful training, a simulation needs to represent the aspects of the real world that are vital for achieving the learning objectives. One of these aspects is human interaction; therefore, simulated entities that respond naturally and validly are needed. These entities, known as *agents*, can be used to simulate team members, opponents or bystanders. There is growing conviction and evidence that *cognitive agents* can be developed by capturing human cognitive processes in a cognitive model and implementing it in a cognitive architecture [19, 20, 8].

An architecture poses constraints on the implementation of a model and therefore influences design choices. This paper reports the experiences of implementing the same formal cognitive model in two different cognitive architectures. First, the implementation of the model in the cognitive architecture Soar [13] is presented. This agent performs a real-world task in a complex environment. Implementing the cognitive model provides insights into the use of Soar for agent applications and it may be used to validate the model's behavior in future research. Next, the Soar implementation is compared to an earlier implementation of the same model in the cognitive architecture ACT-R [2]. This allows for the second goal of this paper: the comparison of Soar and ACT-R.

The next section presents the cognitive task and the formal model. Section 3 presents the ACT-R architecture and the implementation, BOA. Section 4 elaborates on the implementation in Soar, which resulted in the agent named Boar. The paper concludes with a comparison of both implementations on various aspects and their connection to the cognitive theories.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

2. COGNITIVE TASK AND MODEL

The real-world task that has been modeled is the *tactical picture compilation task* (TPCT) from the naval warfare domain. In this task, a navy operator sees a large number of radar *contacts* on his display. Each contact indicates a detected vessel in the vicinity of the own ship. The identities and classifications of these vessels are unknown. The operator can obtain information on these tracks by monitoring the radar screen, such as speed, course, distance to own ship and adherence to shipping lanes. The task of the operator is to use this information to determine both the identity (e.g. hostile, friendly) and the classification (e.g. frigate, fishing boat) of each contact.

A complete cognitive model of the TPCT is constructed using the principles described in this section. The model is based on an extended Belief, Desire and Intention (BDI) framework [7]. BDI facilitates the translation of domain knowledge into a model since domain experts tend to talk about their knowledge in terms similar to beliefs, desires and intentions. The domain knowledge needed to model the TPCT has been elicited from naval experts [6].

In order to develop cognitive agents for training purposes, cognitive behavior that can vary in level of rationality needs to be modeled: agents that can perform a task on different levels of expertise are needed. To make this possible, a belief framework was developed [9]. Three arguments are added to beliefs: a timestamp, the source of the belief and a certainty level. BDI models usually throw away beliefs as soon as a new belief is created that causes an inconsistency. However, to enable reasoning over time, every belief is kept and labeled with the time of creation. The source and cer-

tainty labels make it possible to reason about information from multiple sources and with uncertainty, and reasoning can be done in both a rational and a biased way. A belief `belief(P(A,V),T,S,C)` has a predicate `P` with attribute `A` and value `V`, a timestamp `T`, source `S` and certainty `C`. Below is an example belief – there was an `Identification` of `contact1` as `friendly` with certainty `0.7`, done by reasoning rule `determine-id` on timestamp `12`:

```
belief(Identification(contact1, friendly),
          12, determine-id, 0.7)
```

A cognitive model typically consists of declarative knowledge, denoting facts, as well as procedural knowledge, denoting reasoning rules. Besides modeling how to reason, it is also necessary to model the control on when to reason about what. The next subsection presents the format of the declarative and procedural knowledge and subsection 2.2 explains the control structure of the model.

2.1 Reasoning over Beliefs

The goal in the tactical picture compilation task is to correctly classify and identify all contacts. In order to fulfill this goal, the agent needs information about the contacts' behavior. There are two ways to gather such information. The first is from the external world, e.g. the agent can watch the screen that displays information from sensors such as the radar system. Additionally, the agent can decide to perform actions that lead to more knowledge about the situation, such as activating its radar or sending a helicopter to investigate a contact. The second method to gain information is through the internal process of reasoning about beliefs to deduce new beliefs. In the reasoning process, often multiple beliefs form the evidence for the formation of a new belief. Any uncertainty in the source beliefs will be transferred to the new belief.

An example of this type of deduction is reasoning about formations. If a number of vessels have the same course and are close to one another (source beliefs), they might move in formation (new belief). Moving in formation is an indication that these vessels are frigates. Figure 1 contains an example rule that is part of reasoning about formations. The position of the contact that the agent is currently reasoning about is compared to the position of every other contact that is detected by the radar system. A new belief is created for every pair that indicates how certain the agent is that they are within a distance that can indicate a formation.

The function **Position-Difference** calculates the distance between two positions, **Certainty-Handling-Positions-Difference** calculates the certainty of the distance given the position certainties, **Possible-Distances** returns all possible distances given the calculated distance and certainty, and **Reason-Belief-Parameter** adds the timestamp and stores the belief in long-term memory. In this example, the latest beliefs about the positions are used. In other rules, beliefs are used that ever had a specific value, or those beliefs the agent is most certain about.

2.2 Control of Reasoning

Control is an important aspect of a cognitive agent; it determines when the agent does what. In the TPCT there is one main goal, which is considered the navy operator's desire in the BDI model: to identify all contacts correctly.

Determine-Within-Formation-Distance-Contact(X)
FOR ALL Y
  IF (
    belief(`PositionContact`$(X, P_1), T_1, S, C_1)$
    belief(`PositionContact`$(Y, P_2), R_1, S, C_2)$
    Position-Difference(P_1, P_2, D)
    Certainty-Handling-Positions-Difference
      (C_1, C_2, D, C_3)
    Possible-Distances$(D, C_3, [R])$
    M = *maximum-distance-relevant-for-formation*
    C_4 = (number-of-$[R \leq M]$)/(number-of-$[R]$)
) THEN (
    Reason-Belief-Parameter
      (`WithinFormationDistanceContact`(X, Y),
       `DetermineWithinFormationDistanceContact`$, C_4)$
)

Figure 1: Rule for determining if two contacts are within formation distance

The three subtasks that the agent can perform in order to fulfill this desire are:

1. processing information about contacts on the screen;
2. changing the activity of the radar system; and
3. sending the helicopter on observation missions to gain more information about a specific contact.

The subtasks above are the intentions of the BDI model that the agent can commit to. A valid manner in cognition to determine when which intention becomes a commitment is to have events in the world trigger an intention. For example, when a contact suddenly changes its behavior, the attention of the agent should be drawn to this contact, regardless of the current intention. However, this type of control requires a parallel processing of all events in the world and a parallel checking of relevancy for all subtasks, which is hard to implement. This is why currently a simpler, linear control system is modeled. The agent alternately commits to one of the three intentions to simulate parallel processing. Within the subtasks, the control is also kept simple, e.g. in the first subtask all contacts on the screen are monitored consecutively.

The rule in Fig. 2 illustrates a part of the simple control structure. It determines when the agent starts committing to a new intention. The input parameter I is the current intention, the rule **Start-New-Intention-Selection** determines which intention is selected next depending on beliefs about contacts, and the rule **Select-Next-Contact-To-Monitor** selects the next contact from the list.

3. BOA

In order to execute the model that was presented in the previous section, a cognitive agent needs to be implemented; cognitive architectures are a suitable platform for this purpose. Such an architecture specifies a fixed set of processes, memories and control structures [15] that define the underlying theory about human cognition. The architecture limits implemented cognitive models by this set and consequently

Determine-New-Intention(I)
IF (
  I = Monitor-Contacts
  belief(`NumberOfContactsMonitored`(X), T_1, S, C)
  X = *maximum-number-of-contacts-to-monitor*
) THEN (
  Start-New-Intention-Selection(I)
) ELSE IF (
  I = Monitor-Contacts
  Number-of-Contacts(X)
  $X <$ *maximum-number-of-contacts-to-monitor*
) THEN (
  Select-Next-Contact-To-Monitor()
  Reason-Belief-Parameter
    (`NumberOfContactsMonitored`($X + 1$),
    `DetermineNewIntention`, 1)
)

Figure 2: Rule for determining a new intention

imposes its cognitive theory on these models: it should make correct models easier and incorrect models harder to build. Moreover, the actual behavior of the agent is influenced by the architecture [12].

The presented model has already been implemented in the cognitive architecture ACT-R [4] – this implementation was named BOA. Since this research has been done earlier, several new developments in ACT-R are not taken into account [1]. However, the insights reported here are nevertheless of interest from an agent-application perspective: several of the issues mentioned in this paper have been changed in the latest version of ACT-R. These developments in ACT-R seem to support our experiences that the architecture was too restrictive on some aspects.

3.1 ACT-R

The theory of ACT-R incorporates two types of memory modules: declarative memory and procedural memory. Declarative memory is the part of human memory that can store items; procedural memory is the long-term memory of skills and procedures. ACT-R consists of a central processing system, where *production rules*, representing procedural memory, are stored and executed. The central processing system can communicate with several modules through buffers. One of those modules is the *declarative memory module* where memory items are stored. These memory items, called *chunks*, are of a specific chunk-type, which can be defined by the modeler. In a chunk-type definition, the modeler defines a number of *slot*s that chunks of this type can assign values to. Chunks from the declarative memory module can be placed in the *retrieval buffer* if they match a retrieval request made by a production rule. A retrieval request must contain the requested chunk-type, and may contain one or more slot-value pairs that the chunk must match. The matching chunk is then placed in the retrieval buffer, so it can be read by a production rule. All buffers in ACT-R, including the retrieval buffer, can only store one chunk at a time, even when more chunks match the conditions of the request. If more chunks are available, an activation function defining the accessibility of chunks is used to select a single candidate.

3.2 Implementation Issues

The implementation of the cognitive model in ACT-R resulted in three main observations. The first focuses on the limit of one chunk in the retrieval buffer. The model prescribes access to multiple beliefs in the working memory at the same time in order to reason over them. For example, different positions in time are compared in order to determine a contact's speed. The ACT-R implementation supported this by using the goal buffer for temporary storage and LISP functions to retrieve beliefs.

The second observation was the fact that retrieving a belief with specific features (for example, the belief created last, i.e. with the highest value for the *time* slot) is not guaranteed by using ACT-R's activation function. For example, the agent often uses the latest position of a contact, so he needs the latest belief with predicate `position-contact` for a specific contact. It may however be that an older chunk has been retrieved more often than the latest chunk, resulting in a higher activation score and subsequently the older chunk being retrieved. As a solution, LISP functions were created as substitute to the activation function.

The third issue is about the many calculations the cognitive model requires: these can only be modeled in a low-level manner, making it inefficient to implement them in the architecture. For example, calculating the speed of a contact from its positions over time would require many production rules, while it would not represent the actual cognitive processes of a warfare officer. Here too LISP functions were used for these type of calculations. As a result of this problem and the previous problem, about half of the programming code consists of ACT-R production rules and the other half of supporting LISP functions.

3.3 Control

Control in the context of BDI agents aims at specifying the commitment of the agent at a certain time. The intentions to which the agent can commit and the type of control in the cognitive model were described in section 2.2. The BOA agent implements a simple, linear control system. The agent commits alternately to each intention and within the intention of processing screen information, the contacts are monitored sequentially. This is illustrated by the rule in Fig. 3.

```
(p select-next-contact-goal1
   =goal>
      ISA         commitment1
      goal        monitor-contacts
      state       next-contact
      contact     =contact1
==>
   !bind! =contact2 (determine-next-contact)
   !eval! (determine-rate-other-desires)
   !bind! =eop (= (mod *counter* *rate-other*) 0)
   =goal>
      plan        read-basal-info
      state       start-step
      contact     =contact2
      eop-marker  =eop
)
```

Figure 3: ACT-R code for intention selection

The rule requires the agent to be committed (`commitment1`) to monitoring contacts and be ready to select a new contact to monitor. This new contact is determined by the user-defined LISP function `determine-next-contact`, which loops through the list of contacts. The `*rate-other*` variable defines the number of contacts after which the agent switches to another intention: if this number is reached, the end-of-process marker (`eop`) is set to true. The agent will then consider committing to sending the helicopter, followed by considering to commit to changing the radar. After these considerations and, possibly, reasoning and actions, the agent continues monitoring contacts. Reacting to events in the environment is limited to altering the order of the list of contacts in the 'monitor contacts' intention: if a contact has been identified by the helicopter, that contact is moved to the top of the list to force the agent to monitor it next.

4. BOAR

This section presents the Boar agent, which is the implementation of the model from section 2 in Soar. The next subsection will explain this architecture in more detail and subsection 4.2 describes several implementation issues.

4.1 Soar

Soar, like ACT-R, is a well-known cognitive architecture. Soar defines the world as a large problem space with states and goals. It considers behavior as movement in the problem state by performing actions, either internal (mental activity) or external (observable in the environment). In Soar, this is done by *operators*; in a single cycle, more operators can be proposed, one of these is selected and eventually applied, changing the state of the environment. Goal-directed behavior states that the agent will choose those operators that lead to a goal state. [14]

The memory structure of Soar is somewhat similar to that of ACT-R. It specifies two types of memory: the long-term memory, consisting of procedural, semantic and episodic knowledge, and the *working memory*, corresponding to ACT-R's declarative memory module. The working memory consists entirely of working memory elements (WMEs), which are attribute-value pairs. The attributes of a WME need not be defined beforehand, as is the case with the slots of a chunk. Additionally, the number of WMEs that can be accessed at one moment is not limited – there is no such thing as a retrieval buffer in Soar.

In long-term memory, the procedural knowledge is primarily responsible for the behavior of an implemented model and is defined in terms of *productions*. When conditions apply, a production either proposes the execution of an operator or it executes some reasoning independent from an operator – both may result in changes to the working memory. The difference lies within the persistence of the changes: a WME that was created by an operator will stay in working memory until an explicit change is made. A production without operator reference, also called *elaboration*, creates WMEs that only exist as long as the conditional part of the elaboration matches. The first is said to have operator support or *o-support*, while the latter has instantiation support or *i-support*.

Soar's productions fire in parallel: all productions that have one or more matches for their conditional part in the current state will execute. Consequently, many operators may be proposed at a single moment. Which operator is selected is resolved by means of preferences, which can be added to an operator.

The fact that Soar allows more production rules to fire simultaneously is in contrast to ACT-R's procedure: here, only one production rule can fire at a single moment. If more chunks are available for retrieval by this production, the activation function determines beforehand which chunk is picked.

If a task is too complex to solve by simply adding some beliefs to the working memory, it can be decomposed in subtasks. An example is reasoning about the usage of the helicopter. In order to decide which contact the helicopter is sent to, all contacts are scored. The rule for proposing the operator that scores a single contact is shown in Fig. 4. If this operator is chosen, there is no immediate score available to be added as belief: it needs to be calculated. As a result, there is an impasse and a new substate is created which has the goal to calculate this score. Various operators are available to calculate a part of the score; after each operator calculated its part, the score is available and the attribute `heli-score` will have a value. Consequently, the operator shown in Fig. 4 will be retracted, having achieved its goal.

```
# Propose to score a contact
sp {consider-heli*propose*score-contact
   (state <s> ^name consider-heli
              ^contacts.contact <contact>
              ^top-state.constants <const>)
   (<constants ^max-distance <max>)
              # No score, no visual id
      # and in range of self
   (<contact> -^heli-score
              -^visual-id-belief
               ^distance-to-self <= <max>)
-->
   (<s> ^operator <op> + =)
   (<op> ^name score-contact
         ^current-contact <contact>)
}
```

Figure 4: Soar code for proposing to score a contact

4.2 Implementation

4.2.1 Retrieving Beliefs

In section 2 we argued that the model needs to be able to reason over time. For example, in Fig. 5 the latest position of the own ship (`self`) is retrieved, i.e. the belief with the maximum value for the `time` attribute. This is an example where a belief with a *relative* value is needed for a certain attribute and the absolute value is of no importance. However, it is not easy to match such a belief if no absolute value is available. We tackled this problem by ordering the beliefs with greater-than and smaller-than relations. In order to retrieve the last-but-one belief a new production needs to be added, another for the last-but-two, and so on.

4.2.2 Reasoning Efficiency

Two alternatives arise when generating new knowledge by means of reasoning (i.e. internal action). As explained in subsection 4.1, there are two ways of adding new elements to the working memory: either with o-support or with i-support. The advantages of using i-supported WMEs are:

```
# Calculate distance of contact to self.
sp {boar*elaborate*contacts*distance-to-self
   (state <s> ^name boar
      ^beliefs <beliefs>
      ^contacts.contact <contact>)
   # Retrieve latest position of self
   (<beliefs> ^belief (^predicate position-contact
                       ^attribute self
                       ^time <time>
                       ^value <self-pos>))
  -(<beliefs> ^belief (^predicate position-contact
                       ^attribute self
                       ^time > <time>))
   # Retrieve position of contact
   (<contact> ^position-belief.value <pos>)
-->
   (<contact> ^distance-to-self (float (exec
                      calcPositionDifference
                      <self-pos> |;| <pos>)))
}
```

Figure 5: Soar code for retrieving the distance between contact and own ship

1. they are created automatically if the conditions or the creating production apply in the current state;

2. they are removed if this not the case anymore and thus are not valid in the current state; and

3. they are updated automatically if new information is available.

The advantage of o-supported WMEs is that they are only created when the operators are proposed explicitly, so only at these times some reasoning is done.

If the beliefs that are used for reasoning stay the same for some time, it is more efficient to use elaborations and thus create i-supported WMEs, because if operators are used for this reasoning, they may perform the same reasoning steps multiple times. If beliefs change continuously, the use of elaborations may become computationally expensive, because they perform the reasoning at every change, even if the results are not used. In this case using operators and o-supported WMEs is more efficient. There is no clear procedure for choosing i-support or o-support: one should think about the trade-offs for every situation in order to pick the most efficient option.

To draw the differences between creating i-supported and o-supported WMEs more clearly, an implemented example of both types is given. First consider the production in Fig. 5. It continuously creates an i-supported WME with the distance of contact `<contact>` to the own ship. Every time a new position is observed, either from the own ship (`<self-pos>`) or the contact (`<pos>`), the conditional part of the production for the old WME does not match the current state anymore and the WME is discarded. At the same time, the newly observed information is used to create a new WME for the contact with the `distance-to-self` attribute, and thus the knowledge has been automatically updated. For this reasoning step the i-supported option has been chosen, since the distance is needed continuously for other reasoning. Using an operator would mean that this operator needs to calculate this distance every time the information is needed.

Now consider the production in Fig. 4. This production sets in the creation of knowledge with o-support: it proposes an operator that, when applied, will assign a certain score to a contact. This score is used for considering to send a helicopter to the contact for identification. This scoring is only done incidentally, which makes the use of an operator a better choice. An elaboration would update this score continuously, even while it is not needed most of the time.

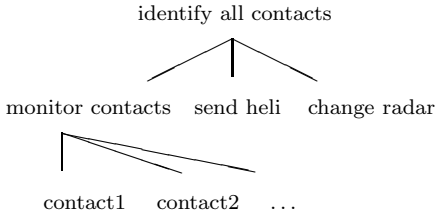


Figure 6: Overview of intentions

4.2.3 Control

The Soar architecture does not provide the means to easily keep a list of contacts, which makes it hard to implement a sequential control for committing to the three intentions described in subsection 2.2. Alternatively, it easily allows the creation of subgoals, as explained in subsection 4.1. By defining the monitoring of every contact as a subgoal of the 'monitoring contacts' intention, the structure of goals and subgoals becomes as shown in Fig. 6.

The commitment to one of the three intentions is decided as follows: if an event triggers the intention to send a helicopter or change radar activity, the agent commits to this intention. Otherwise it will first monitor all contacts, then consider sending the helicopter and finally consider changing radar activity. The linearity of this cycle is forced by explicitly remembering the control status in WMEs. Figure 7 shows the commitment to monitoring contacts: when a new cycle starts, the time is saved in the `start-process-passive` attribute. Then every contact not checked after this time (`-^checked > <starttime>`) is monitored and gets tagged with a new time, until all contacts have been checked this way. After completing the helicopter and radar intentions, a new cycle is started. The order of checking contacts is random and may be different each cycle.

```
sp {boar*propose*process-passive-information
   (state <s> ^name boar
              ^start-process-passive <starttime>
              ^contacts.contact <contact>)
  (<contact> -^checked > <starttime>
              ^id <contact-id>)
-->
   (<s> ^operator <op> + =)
   (<op> ^name process-passive-information
         ^current-contact <contact>)
}
```

Figure 7: Soar code for proposing to monitor a contact

An example of an event triggering the 'send heli' intention

is shown in Fig. 8. It simply states that if the helicopter is airborne and does not have a mission, for example just after identifying a specific contact, the agent should commit to reasoning about what it should do. This commitment can break into the aforementioned cycle at any time the conditions apply.

```
# A heli is considered when it has
# no mission and is airborne.
sp {boar*propose*consider-heli*airborne
   (state <s> ^name boar
              ^helis.heli <heli>)
   (<heli> ^id <heli-id>
           # Heli has no mission and is airborne
           ^mission free
           ^status airborne)
-->
   (<s> ^operator <op> + =)
   (<op> ^name consider-heli
         ^heli <heli>)
}
```

Figure 8: Soar code for the event-driven selection of the 'send heli' intention

4.2.4 External Functions

The simulation environment for performing the tactical picture compilation task is created in Game Maker [18]. It simulates a radar screen with information about the contacts in the surroundings. The simulation environment reacts on certain actions, e.g. clicking on a contact will provide detailed information about it.

For letting Soar communicate with this Game Maker environment an interface is needed, which is implemented in Java. The actions of the agent are written to a text file by the Java interface, read by Game Maker and consequently performed in the environment. An example of an agent action is the request for detailed information, which simulates a mouse click by a human. Any input from the environment, such as a new contact or the position of the own ship, is written to a text file by Game Maker, read by the Java interface and presented as input for the agent. This form of communication slows the execution of the agent down, since it continuously waits for reactions from the environment.

To perform complex calculations, user-defined functions in Java are needed, similar to one of the issues when implementing the formal model in ACT-R (see subsection 3.2). These functions are called from inside the agent, but can only be used in the actions of a production. Consequently, if a calculation needs to be performed as part of the condition of a production, it has to be executed by another production and the result needs to be made available through the working memory.

5. CONCLUSION AND DISCUSSION

This paper presents an agent built in the cognitive architecture Soar, capable of performing a complex real-world task. The implementation is based on a formal model of the task and has previously been implemented in ACT-R. The remainder of this paper will present the lessons learnt on several aspects of agent practice and links them to cognitive theories.

Two implementations of a single cognitive model give only one point of view: a different model may have different demands, especially when a different framework is used. Additionally, the implementations have not been validated – further work in this direction may consist of experiments with subject-matter experts comparing the performance of BOA, Boar and humans performing the tactical picture compilation task.

Nevertheless, this paper shows that one should consider the functionalities requested by the model and the possibilities an architecture offers to implement those demands.

Working Memory Access.

Most of the cognitive theories about human working memory agree on a storage capacity of multiple but limited amount of items [16, 3, 11, 5]. This assumption was used in designing the cognitive model: several rules in the cognitive model need multiple beliefs at the same time for reasoning (an example of such a rule is in Fig. 1).

The ACT-R theory used for implementing BOA proved to be too restrictive: access to only one chunk at a time is allowed. In order to access more beliefs, a work-around was used in the ACT-R implementation. On the other hand, Soar does not limit the number of accessible working memory elements, so this did not pose any problems implementing Boar. Different approaches to the working memory theory result in different types of behavior: if only a limited number of elements is accessible, reasoning will be restricted to these elements, which can cause a different way of acting than when all elements are available.

Retrieving Beliefs.

In order to reason over time the retrieval of specific beliefs with a relative value is needed, such as the 'last' or 'one-but-last' belief of some kind – a capability humans apply unconsciously. Unfortunately, this type of retrieval operator is not yet available in either architecture.

In ACT-R the working memory items are retrieved by means of an activation function. However, this function does not guarantee the retrieval of a memory item based on such a relative value. The solution was to create LISP functions for retrieving beliefs. Soar allows ordering the beliefs in the conditional statement of a production rule, making it possible to retrieve beliefs with a relative value. However, operators need to be created for each relative value, making the translation from model to Boar somewhat inefficient.

Control.

A linear control was modeled in favour of event-driven parallel control. This choice was made in order to simplify the process of committing to an intention, even though human decision-making will be more reactive to cues from the environment. ACT-R's sequential execution of production rules fits this simplified model, but as a result the BOA agent reacts slowly on important changes in the environment, because the agent's behavior cannot be interrupted by these external events [10]. In Soar the sequential execution of plans is forced by letting production rules fire in an explicit order (as shown earlier in Fig. 4), but this architecture more easily allows event-driven control.

Calculations.

The tactical picture compilation task contains many sit-

uations in which the human expert makes estimations, for example on how close a ship is to the the own ship or whether ships are moving in formation. Currently there is no method available to model the process of these estimations. Instead, the estimations are replaced by exact calculations and made into an 'estimation' by adding the notion of uncertainty to the resulting belief. Modeling these calculations as cognitive tasks in an architecture would require an infeasible amount of productions, without actually copying human behavior. Therefore, the execution of complex calculations is done externally by LISP functions or Java methods.

Speed.

Humans are able to use multiple beliefs that were gathered over time for reasoning. This is represented in the belief framework by adding a time tag to every belief and storing all beliefs in memory. As a result, the agent can access multiple beliefs over time for reasoning. For example, it can access several beliefs about the position of a contact to reason about the contact's speed and movement behavior.

Unfortunately, this creates a practical problem: there is an exponentially growing amount of beliefs, which means no system will eventually be able to cope with the resulting CPU-expensive searches for the necessary beliefs during real-time simulation. It is necessary to deal with this more efficiently. Even though certain facts in the past need to be remembered by the agent, it is not necessary to remember every specific detail, which is the case in this implementation. Humans do not remember every detail exactly, but compress their memories by conceptualizing or clustering them. Future agents that incorporate the belief framework used in this research will need some form of compression or smart discarding of beliefs to copy this behavior. We are currently developing a method to cluster and abstract beliefs over time, sources and certainties, in order to form a more realistic model of episodic memory.

Clearly, this problem has its effect on the implementations. The ACT-R agent becomes slow over time, even though some functionality to remove unimportant beliefs had been implemented. This slowness makes the observed behavior of the agent not very human-like, especially in reacting to changes in the environment [10]. On the other hand, Boar has been used in a demonstration of about twenty minutes, in which the agent showed no reduction in speed. To draw general conclusions about the performance of both architectures, further research is needed.

6. ACKNOWLEDGEMENTS

This research has been supported by the research program "Cognitive Modelling" (V524), funded by the Netherlands Defence Organisation.

7. REFERENCES

[1] J. R. Anderson. *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press, 2007.

[2] J. R. Anderson and C. Lebiere. *The Atomic Components of Thought*. Lawrence Erlbaum Associates, 1998.

[3] A. D. Baddeley and G. J. Hitch. Working memory. *Recent Advances in Learning and Motivation*, 8:647–667, 1974.

[4] F. Both and A. Heuvelink. From a formal cognitive task model to an implemented ACT-R model. In *Proceedings of the 8th International Conference on Cognitive Modeling*, 2007.

[5] N. Cowan. *Working memory capacity*. Psychology Press, New York, NY, 2005.

[6] B. J. v. Dam and H. F. R. Arciszewski. Studie commandovoering do-2: Beeldvorming. Technical Report FEL-02-A242, TNO-FEL, 2002.

[7] M. P. Georgeff and A. L. Lansky. Reactive reasoning and planning. In *Proceedings of the 6th National Conference on Artificial Intelligence*, pages 677–682, Menlo Park, California, 1987.

[8] K. A. Gluck and R. W. Pew, editors. *Modeling Human Behavior With Integrated Cognitive Architectures: Comparison, Evaluation, and Validation*. Lawrence Erlbaum Associates Inc, 2005.

[9] A. Heuvelink. Modeling cognition as querying a database of labeled beliefs. In *Proceedings of the 7th International Conference on Cognitive Modeling*, 2006.

[10] A. Heuvelink and F. Both. BOA: A cognitive tactical picture compilation agent. In *Proceedings of the 2007 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IAT 2007*, Silicon Valley, California, Nov. 2007. IEEE Computer Society Press.

[11] C. Hulme, S. Roodenrys, G. Brown, and R. Mercer. The role of long-term memory mechanisms in memory span. *British Journal of Psychology*, 86:527–536, 1995.

[12] R. M. Jones, C. Lebiere, and J. A. Crossman. Comparing modeling idioms in ACT-R and Soar. In *Proceedings of the 8th International Conference on Cognitive Modeling*, 2007.

[13] J. E. Laird, A. Newell, and P. S. Rosenbloom. SOAR: an architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[14] J. F. Lehman, J. Laird, and P. Rosenbloom. A gentle introduction to Soar, an architecture for human cognition: 2006 update, 2006.

[15] R. L. Lewis. Cognitive theory, Soar, Oct. 1999.

[16] G. A. Miller. The magical number seven, plus or minus two: Some limtis on our capacity for processing information. *Psycological Review*, 63:81–97, 1956.

[17] R. L. Oser. A structured approach for scenario-based training. In *Proceedings of the 43rd Annual Meeting of HFES*, volume 43, pages 1138–1142, Oct. 1999.

[18] M. Overmars. Game maker: http://www.yoyogames.com/gamemaker/.

[19] R. W. Pew and A. S. Mavor, editors. *Modeling Human and Organizational Behavior*. National Acadamy Press, 1998.

[20] F. E. Ritter, N. R. Shadbolt, D. Elliman, R. M. Young, F. Gobet, and G. D. Baxter. Techniques for modeling human and organizational behaviour in synthetic environments: A supplementary review. Technical report, Human Systems Information Analysis Center, June 2003.

A Semantic Description For Agent Design Patterns

Luca Sabatucci
Dipartimento di Ingegneria Informatica,
University of Palermo,
Italy
sabatucci@csai.unipa.it

Massimo Cossentino
ICAR-CNR,
Consiglio Nazionale delle Ricerche,
Palermo, Italy
cossentino@pa.icar.cnr.it

Salvatore Gaglio
Dipartimento di Ingegneria Informatica,
University of Palermo,
Italy
gaglio@unipa.it

ABSTRACT

In last years, multi-agent systems (MAS) have achieved a remarkable success and diffusion in employment for distributed and complex applications. A fundamental contribution has come by the adoption of reuse techniques and tools providing a strong support during the design phase. Even though design patterns have been widely accepted by industrial and academic organizations as a proper technique for reuse, their definition still imposes deep concerns on contemporary software engineers. Design patterns are largely sensitive to different contexts where they are employed, especially on how they are blended with each other. This work introduces a design language for describing fine-grained pattern formalizations and compositions based on structural semantics. This formalization as been used in order to describe a couple of design patterns for agents and their composition.

Keywords

Design Patterns, Multi-Agent Systems, Agent Oriented Software Engineering

1. INTRODUCTION

In last decade design patterns have been widely accepted by industrial and academic organizations, even if their definition and reuse still impose deep concerns on contemporary software engineers.

The pivotal difficulty stems from the fact that reuse of design patterns in realistic software systems is often a result of blending multiple patterns together rather than instantiating them in an isolated manner. The composition of design patterns can results in an intricate twine of pattern participants and the target application [14]. Pattern blending may entail significant morphs of the original pattern solutions through the merge of structural and behavioral elements. Also, the lack of explicit documentation for recurring compound patterns leads to design rationale being irrecoverable [4]. Patterns should be systematically documented so that they can be unambiguously instantiated, traced and reused within and across software projects [11, 5].

In our research we deal with design process of agent societies; this activity involves a set of implications such as capturing the ontology of the domain, representing agent interactions (social aspects), and modelling intelligent behaviours. In the following, we are going to pursuit a specific goal: lowering the time and costs of developing a MAS application without forgetting the necessary attention for quality of the resulting software and documentation. We have always considered design patterns for agents as a fundamental contribution to the agent-oriented software engineering. In past works we have defined some reuse techniques and tools based on design patterns [9, 18]; this approach has been integrated with the PASSI design process [7], a step-by-step requirements-to-code methodology for developing multi-agent software. Design patterns have been conceived as a crosscutting design activity occurring during almost all the phases of PASSI.

Now we concentrate on another aspect of pattern reuse. This paper deals with formalization of design patterns for agents, by using a semantic approach. Patterns are represented with semantic networks that can be modeled and transformed by using a set of operators. This formalization is perfectly suitable to be integrated in a tool that support the designer during the development process of a multi-agent system.

Section 2 presents motivations for pattern formalization and composition. A design approach based on the semantic description of patterns is proposed in Section 3 and a graphic notation, the Pattern Semantic Description is discussed. Section 4 introduces an expressive yet simple set of operators for *unifying*, *conjoining*, *concealing* and *externalizing* pattern elements, illustrating a composition example. Section 5 reports an analysis on the reusability and expressiveness of our language. Finally, some concluding remarks are reported in Section 6.

2. MOTIVATION

This section presents an analysis of heterogeneous forms of pattern blending, which are commonly found in real multi-agent system development. This analysis provides some motivations for the introduction of our language.

The Digital Business Ecosystem [10] consists of a research project supported by the European Commission's 6th Framework Programme IST Thematic Priority. In particular, the Sicilian DBE project supports Sicilian micro and middle size companies in order they can access to advanced information and communication technologies to grow their business. We have realized a multi-agent system, the *Sicilian Digital Business Ecosystem Simulator* (SDBE Sim), that simulates the business evolution of companies, in which several design pat-

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

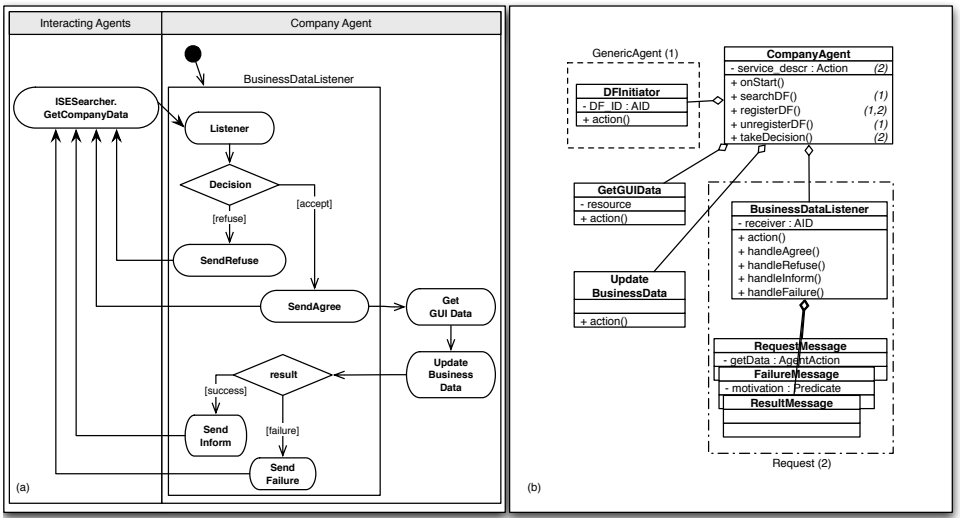


Figure 1: Two PASSI design slices of the SDBE case study. (a) Task Specification for the Company agent. A number of agent's tasks and interacting agents were omitted for simplification reasons. (b) A Single Agent Structure Description diagram that illustrates the implementing class structure of the same agent. This diagram has been decorated with references to patterns that have been employed.

terns for agents have been used and combined to achieve system requirements of scalability and adaptation. Figure 1 shows two design slices in which two patterns have been used and combined. In Figure 1.a the internal architecture of the Company agent is shown. This agent simulates the core business of a company. The agent provides the user a GUI where simulation data can be edited. Other agents, for example the ISESearcher agent, may request company's business data by using a FIPA Request communication protocol [12]. According to agent autonomy feature the Company agent may decide whether to give back its data or to refuse. If the agent accepts the request, it must read user's data from the GUI and eventually update its core business. After this update, the agent may reply to the request message. Figure 1.b reports the Jade [2] implementing structure for the Company agent. This class diagram contains a class for the agent, and another class for each agent's task. In addition three message classes, are shown, that are used during conversations. This class diagram is useful to illustrate the employment of design patterns for implementing this agent. The *GenericAgent* pattern as been used to give the Company agent the abilities to interact with the system Directory Facilitator. This is fundamental for agents that want provide services to the community. This pattern has introduced the searchDF, registerDF, unregisterDF methods in the agent class, and a new DFInitiator task class. The *Request* pattern has been used, in order to give the agent the ability to participate to communications by using the FIPA Request protocol [12]. This pattern has introduced the service_description attribute in the agent class that contains a reference to the action to execute to provode the service. The registerDF is modified in order to register the agent's service and a new method takeDecision is introduced to encapsulate the decision process in order to evaluate whether accept or refuse incoming requests. Finally, the BusinessDataListener task class and three message classes are introduced in the structure in order to handle incoming communications.

The attachment of a number in the CompanyAgent class implies that the respective method or attribute is part of the implementation of the corresponding pattern. Each number represents a specific pattern and the aim of this representation is to illustrate how various pattern realizations affect internal members of a single class. The result of the application of these two patterns, is a typical pattern composition that occurs when implementing an agent that offers services to a community. In the composition each pattern introduces specific agent abilities and features, and some of these are influenced by more than one pattern at the same time. This creates a synergy of pattern functionalities and responsibilities that are merged together in a new pattern structure. The remaining part of this pattern illustrates how pattern compositions may be formalized and employs in the agent paradigm.

3. SEMANTIC DESCRIPTION FOR PATTERN STRUCTURES

Last years revealed the importance of semantic description of data, especially in certain context, such as the web. The semantic descriptions are generally based on the use of ontologies in order to structure informal description in hierarchical relationships of concepts on which it is possible to operate logical reasoning [21].
The remaining sections describe the POLaR language [17, 19], a fine-grained design approach to support the pattern solution definition that is based on a set of constituents that can be combined in order to define the structure and the behavior of the solution. In order to discuss the language we give some definitions:

Pattern Description Element (PDE). An atomic constituent of a pattern that describes the structure or the behavior of the solution. They are: (i) participants, (ii) collaborators, (iii) events and (iv) actions.

Language Element (LE) Element of the target programming language used for implementing the pattern. Lan-

guage Elements are expressed by using elements of the meta-model. In our repository for agents, examples of LEs are: agent, organization, communication, role and task.

Affected System Element (ASE) Element of the system that is influenced by the pattern application. A typical example of ASE is a business class that is assigned to a participant of the pattern. Its structure is modified because it must be compliant with PDE constraints.

PSEs are the constituents of a pattern solution. The definition of pattern solutions encompasses alternant levels of stability: some PDEs (Pattern Description Elements) are precisely described and do not require further details through the pattern instantiation, whereas some others are only sketched and their concrete definition is delayed to the pattern instantiation phase. The structure and behaviour of those PDEs depends on the application context and on the other patterns to which they are going to be composed. This kind of PDE supports the generalization and reuse of patterns in very distinct contexts where the nature of the problem may be different.

Participant. Participant are placeholders for assigning responsibilities to ASEs. This pattern constituent is similar to the classic "role" element, introduced in [13] and detailed in several pattern formalization approaches [16][15]. Participant is a more general concept because roles can be played by classes only, whereas every element of the MAS meta-model (for instance an Agent or a Task, or even a Communication) may be a participant of the pattern. Responsibilities assigned to a participant will be taken by LEs that are assigned to this participant.

Collaborator. A collaborator is a concrete element of the pattern, totally defined in every its feature. It is used in order to introduce in the system an element that generally mediates other pattern's elements with a standard behavior. Its behavior is pre-defined, and except for special situations, it does not require a further specialization. A collaborator owns a type, which refers to a LE (Language Element). Therefore, a collaborator may be an instance of any element of the MAS Meta-Model.

Event. An event encapsulates an abstract circumstance that is the cause of triggering a specific behavior, involving one or more pattern elements. Typically the context that generates an event is external to the pattern. It is an non deterministic condition (from the point of view of the pattern), that is generated by the specific needs of a participant. The event execution may be considered as a service request operated by the participant in order to produce the desired behavior.

Action. Together with events, actions have a fundamental role in the definition of the behavior of a pattern. An action encapsulates what happens when an event occurs. Actions must not be considered as merely methods. In the agent paradigm, actions may correspond to agent's abilities or tasks depending by implementing issues.

For problems of space, in this paper we mainly focus on the static structure of the pattern description. Even if the complete formalization approach also supports the dynamic description and composition [19] this is only shortly discussed in this paper.

3.1 The Pattern Semantic Description

Semantic networks are often used as a form of knowledge representation. They consist in declarative graphic representations that are expressed with models of interconnected nodes and arcs. Semantic networks generate machine-readable dictionary that can be used either to represent knowledge or to support automated systems for reasoning about knowledge [20].

Several graphical notations exist for representing a semantic network. The UML class diagram is often used in order to represent concepts and their relationships. This diagram is perfectly suitable for our aims, but we have introduced the two stereotypes, *participant* and *collaborator*, in order to immediately distinguish participants and collaborators. Relationships are used to connect participants and collaborators, thereby creating a semantic network.

The class diagram that uses these stereotypes to describe the semantic structure of a pattern is named Pattern Semantic Description (PSD) diagram. An example of this diagram is shown in Figure 2, that illustrates the *Request* pattern. In this figure we have introduced a graphic notation in order to reduce the space: participants are shown by using ovals, whereas collaborators are shown by using boxes. This description is designed to assert propositions about the structure of pattern solutions (assertional networks [20]). Participants and collaborators are the *concepts* of this network. In particular, collaborators are concepts whereas participants are classes of concepts. Relationships express semantic connections among these concepts. Relationships can not freely connect every kind of concepts in a PSD; they are precisely ruled by the MAS meta-model [1, 3, 8].

The information in a PSD is a set of conditions that should be contingently true in order to apply the pattern in the system where the problem occurs. The aim of PSD diagrams is twofold:

- they are human readable, so that designers can easily understand and apply the rationale of each pattern solution;
- the use of a limited set of concepts and relationships allows the realization of a parser for automatic interpretation of patterns, thus resolving syntactic ambiguities.

3.2 An Example of Pattern for Agent

The pattern, we consider here, is the *Request* pattern, from our repository, briefly introduced in Section 2. This pattern has been conceived for giving agents the ability to initiate and participate to communications that are compliant to the FIPA Request protocol.

This communication is useful in several circumstances. The delegation of a task is a typical scenario useful to illustrate the aim of this pattern. It occurs when an agent has to perform an action (for example, an interaction with a physical resource) but it is not able or has not sufficient permissions to do. So the agent can ask to another agent of performing that action (because of agents autonomy, the

involved agent can refuse or accept the request according to its personal goals).

The analysis of the description of the protocol, reported below, has been the core for the identification of the participants of this pattern (Figure 2). The elements that are included in the description of the protocol but are outside the definition of the pattern are the two agent roles: initiator and participant. Any couple of agents may participate in this protocol by simply playing these two roles. This is the reason for which we defined these two roles as participants of the pattern: (i) *Agent Role Init* is responsible to begin the communication and (ii) *Agent Role Part* is responsible to maintain a listener for this communication. Designer must specify the couple of agents to assign to these participants. Other elements that can not be encapsulated in the pattern definition are: (i) the *Action*, that is requested by the initiator by using the communication and (ii) the *Decision* task used to encapsulate the decision process to activate when the agent receives the request.

Figure 2 also shows the collaborators of this patterns. The *Request Communication* is the description of the kind of communication among initiator and participant. This must be compliant to the FIPA Request protocol [12], whose description is given in the *FIPARequest AIP* collaborator. Several *Messages* are included for information exchange. In addition, two tasks are defined as collaborators: (i) the *Request Initiator Task* is responsible to send the request message and wait for a reply, and (ii) the *Request Participant Task* is responsible to wait a request message and to reply with a result.

4. OPERATORS FOR PATTERN BLENDS

This section presents the operators for pattern composition based on the fine-grained pattern elements. These operators can be used in order to modify the structure of the pattern solution that is represented by a semantic graph in PSD diagrams. The following list briefly describes all the static composition operators. Their concrete usage will be later discusses in a detailed example.

Unification The unification is used to express overlapping compositions. The rationale behind this operator is to operate fusions of couples of static elements with a consequent merging of responsibilities. The result is to overlap the structure of two patterns using the two elements as pivots for the operation. This produces strong changes in the resulting pattern structure.

Conjunction The conjunction operates a conservative pattern blending. The rationale behind this operator is to create a synergy among the responsibilities of two patterns, by maintaining them separated. The two elements are linked by a new element, introduced in the structure. Only marginal changes are visible in the resulting structure of the involved patterns, promoting the traceability of the involved elements.

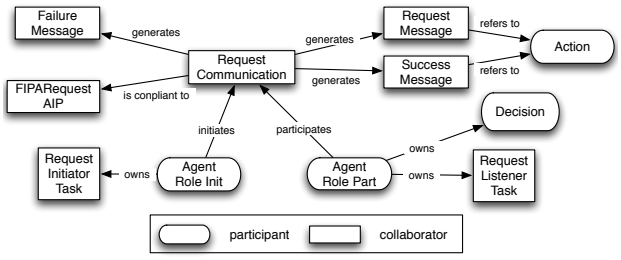


Figure 2: Pattern Semantic Description diagram for the *Request* pattern

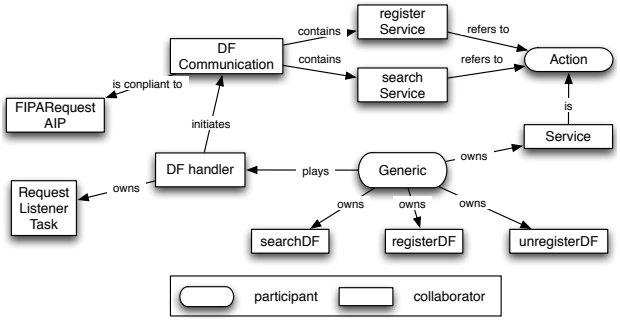


Figure 3: The pattern semantic description for the *GenericAgent* pattern.

Concealing This unary operator has been conceived to modify the nature of a participant into a collaborator. The responsibilities assigned to a participant are imposed to the elements of the system that participates to the pattern. Concealing a participant means that all its responsibilities are delegated to a collaborator. They are no more visible outside the pattern. The visible effects of this operation are i) to allow mixed composition (unification and conjunction) among participants and collaborators ii) to internally set some responsibilities in order to assign a standard behavior and iii) to reduce the complexity of the pattern.

Externalization This unary operator has been conceived to modify the nature of a collaborator into a participant. The rational behind this operator is to delay the assignment of these responsibilities till the instantiation phase, exactly like for participants. The visible effects of this operation are i) to allow mixed composition (unification and conjunction) among participants and collaborators and ii) to change the standard behavior of a pattern, by delegating some aspects of its structure to elements of the system.

In order to discuss the usage of these operators in the static context, a composition of the *GenericAgent* pattern with the *Request* pattern is illustrated. The result of the

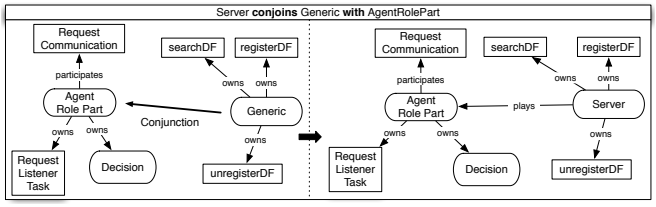


Figure 4: An example of conjunction among participants for the *SequentialShareResource* pattern.

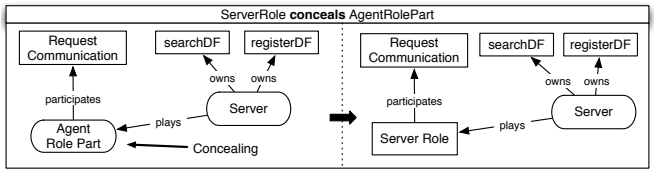


Figure 5: An example of concealing of participant for the *SequentialShareResource* pattern.

composition is a new pattern, the *SequentialShareResource*, that solves the classical problem of providing a service to the remaining part of the society. A service is an action performed by an agent when requested by another agent. Service provisioning is subject to certain conditions (pre-condition, post-condition, grounding) to be verified. In particular this pattern provides the other agents with an access to a physical resource that the agent may manipulate; the specific characteristic of this access is that resource parameters are read/affected each time an access request occurs. A typical example of service that depends on a resource has been given in Section 2. The Company agent provides information on its business, but this depends on the user's data introduced by a GUI. The solution proposed by this pattern is to sequentially execute three activities: (i) the verification of the conditions under which the service may be provided, (ii) the update of the status of the physical resource and (iii) the execution of the service-action.

The first component of this blend is the *GenericAgent* pattern (shown in Figure 3), that has been conceived as a root for giving an agent the ability of interacting with the yellow pages service of the platform. The agent resulting from this pattern is able to register/unregister services to/from the yellow pages and to search them in.

The second component of the blend is the *Request* pattern, already discussed in Section 3.2. This pattern is used to implement a FIPA Request communication. By using this pattern an agent can request to another agent to perform some actions.

The composition process details (for the static part) are described in the following list:

- The first operation in this composition is a conjunction among the *Client* participant of the *GenericAgent* pattern and the *AgentPartRole* participant from the *Request* pattern. The rationale of this operation is to assign an agent, the *Server*, to play the *AgentPartRole*. Figure 4 shows the result of this operation: after the conjunction the *Server* participant plays the *Agent Role Part*.

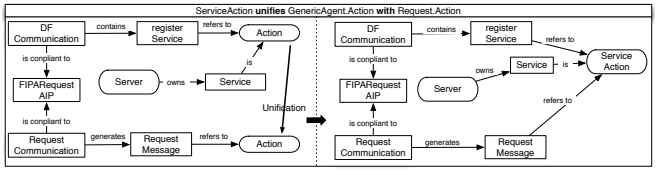


Figure 6: An example of unification among participants for the *SequentialShareResource* pattern.

- After the conjunction the designer can delegates a concrete agent of the system to be the *Server* of this new pattern (since it is a participant). This agent will play the *AgentPartRole* in the request communication. Therefore this role can be converted to a concrete element of this pattern: designer is not required to specialize this element. For this reason the *AgentPartRole* is concealed by the *Server Role*, a new collaborator of the pattern. This operation is shown in Figure 5.
- The third operation is an unification between the *Action* from *GenericAgent* with the *Action* from the *Request* pattern. The meaning of this operation is to specify that the action registered to the yellow pages by the *Server* agent is the same action that the agent provide to the community. This unification is shown in Figure 6.
- Finally, we also introduced a new participant in the structure, the *UpdateResource* pattern, that is a task responsible for accessing the resource and update its state. Since the access to the resource is variable, depending by the nature of the resource, this element is defined as participant. Figure 7 shows the resulting structure of the new pattern after all these operations.

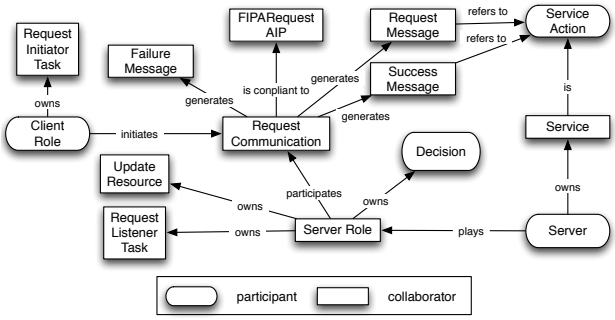


Figure 7: The pattern semantic description for the *SequentialShareResource* pattern.

5. DISCUSSION

This section discusses the pattern reuse process, and some results obtained by the application of the pattern composition technique to three different case studies are reported. The reuse process encompasses four phases for introducing patterns in a system:

1. Meta-model definition or importing. This phase defines the domain where patterns can be employed. The Meta-Model defines a set of rules to be considered during next phases. Concepts and relationships in a PSD must be compliant to the Meta-Model. The definition of a meta-model is a complex activity but several reusable meta-models already exist.
2. Pattern structure and behaviour definition. In this phase the pattern is modelled by using a fine-grained description based on PDEs. Pattern modellers define the core semantics of patterns, in order to formalize their descriptions. This phase of the reuse process is supported by the Pattern Semantic Description diagram for representing the structure of the proposed solution.

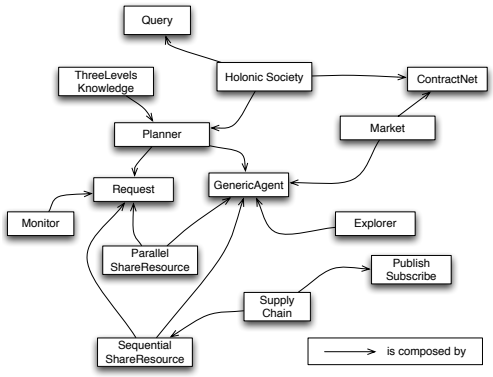


Figure 8: Composition relationships among patterns in our repository.

3. Pattern composition. Patterns can be also defined by using pattern blending that allows for creating pattern synergies to solve more specific problems. This phase is supported by four operators: unification, conjunction, externalization and concealing.

4. Pattern instantiation. This is the final phase of the pattern lifecycle, where designers are involved in applying patterns to under development systems.

Discussion in this section is focussed on the pattern instantiation phase. The semantic approach for describing patterns, as introduced in Section 3, can be manually employed to introduce solutions in systems, or can be automatically interpreted by a tool that may generate the desired solution. Subsection 5.1 illustrates some statistics obtained by manual reuse of patterns from our repository. Finally, Subsection 5.2 introduces a tool, we are developing, that obtains benefits from the semantic approach.

5.1 Reusability of Pattern Blends

Our repository is by now composed of 22 patterns for agents. We have already formalized 14 of these patterns in order to evaluate our language. Patterns we have represented and composed with our approach are shown in Figure 8. Only 4 of these are atomic patterns: (i) *GenericAgent*, (ii) *Request*, (iii) *Query* and (iv) *ContractNet*. The remaining 10 patterns in Figure 8 are obtained by composition. This situation is represented by relationships "is composed by".

All patterns in this repository have been manually reused in three different case studies reported in Table 1. We have chosen these applications because they are from heterogeneous application domains: (i) *SBE Sim* is a software that simulates business evolution of Sicilian micro and mid-size companies, (ii) *CiceRobot*[6] is a robotic application able to give guided tours of the Agrigento's Regional Archaeological Museum and (iii) Iron Manufacturer is an application for supporting a B2B scenario involving an iron manufacturing company. The number of different application contexts, proves the feasibility of our patterns and their compositions.

Table 1 reports that 8 patterns have been reused in at least two case studies. The most used patterns are the*GenericAgent*, the *Request*, and the *Query* that are generally easily to combine with other ones.

Table 1: Statistics of pattern usage in our three case studies

	SDBE Sim	CiceRobot	Iron Manufacturer	TOT
ContractNet	1		1	**2**
Explorer			1	**1**
GenericAgent	4	5	5	**14**
Holonic Organization	1		1	**2**
Market	1			**1**
Monitor	1		1	**2**
Parallel ShareResource			1	**1**
Planner		3		**3**
Publish-Subscribe	1	1		**2**
Query	9	4	7	**20**
Request	9	7	21	**37**
Sequential ShareResource	3	1	2	**6**
Supply Chain		1		**1**
ThreeLevels Knowledge		1		**1**
TOT	**30**	**23**	**40**	

5.2 A Tool for Pattern Reuse

In previous section we proved that

- a framework is provided for representing declarative knowledge on design pattern, that is based on a semantic network,
- the syntax and semantics of the network are clearly defined

Under these conditions, pragmatics of the network are defined by several rules which are problem-independent. This allows us to formulate control algorithms to handle the described structures. In particular we are developing a tool with the following requirements: (i) complete control of the entire pattern reuse process, from meta-model definition to pattern definition, composition and instantiation; (ii) verification of pattern syntax and semantics; (iii) management of pattern applicability pre and post conditions; finally (iv) automatic generation of code and documentation for the multi-agent system generated by using pattern composition.

The latter requirement of this tool has been already completed, and discussed in [18]. This component of the tool uses an aspect oriented approach for weaving together different contribution to the final workproduct. These contributions are generated separately by *aspect weavers* who are specialized to realize code for a specific aspect of the system. Typical aspect weavers are: (i) the *architecture weaver* who is responsible to define the basic structure of an agent, its capabilities and the its basic life-cycle activities; (ii) the *role weaver* is responsible to manage complex agent activities, both internal processes than social behavior; (iii) the *communication weaver* is responsible to give agents the ability to interact by using communications; (iv) the *protocol weaver* generates the structure for managing agent interaction protocols; and finally (v) the *ontology weaver* is responsible to generate agent knowledge structure. All these aspect weavers must collaborate in order to generate a unique source code.

The other requirements, we are facing for the development of the tool, are related to the pattern formalization problem. A semantic definition of patterns is easily representable in a formal language. We already defined this language, that we named POLaR (Pattern Ontology Language for Reuse) but it was not discussed in this paper for limit of space. Anyway the production of the parser for the POLaR language is under construction. A portion of the BNF code used to

represent the syntax of this language is shown in the following:

```
<Pattern_Descr> ::= <Pattern_Header> "{" <Pattern_Definition> "}"
<Pattern_Definition> ::= { < Element_Clause > }
<Element_Clause> ::=  <Participant>
        | <Collaborator>
        | <Event>
        | <Action>
<Collaborator> ::= collaborator <Identifier> is <Element_Descr> ";"
<Element_Descr> ::= <Element>
        | <Operator>
<Operator> ::= <Unification>
        | <Conjunction>
        | <Promotion>
        | <Externalization>
```

6. CONCLUSIONS

This paper presented an innovative formalization technique for describing and composing design patterns for agents. The technique is based on a semantic analysis of the pattern solution and introduces a graphic notation to represent pattern's concepts and their relationships. The proposed formalization has been conceived for dealing with composition, presenting a set of operators to manage different pattern blending styles. The peculiarity of the approach is the fine grained level chosen for fronting with the composition, which makes it possible to combine pattern elements in the resulting composite pattern. We have applied our approach in three agent oriented applications, thus obtaining encouraging results in terms of reusability and expressiveness. In addition we introduce a tool we are developing in order to automatically compose and instantiate design patterns and to automatically generate implementing code and documentation.

7. REFERENCES

[1] C. Atkinson and T. Kiihne. The essence of multilevel metamodeling. *Uml 2001: The Unified Modeling Language: Modeling Languages, Concepts, and Tools: 4th International Conference, Toronto, Canada, October 1-5, 2001: Proceedings*, 2001.

[2] F. Bellifemine, A. Poggi, and G. Rimassa. Jade - a fipa2000 compliant agent development environment. In *Agents Fifth International Conference on Autonomous Agents (Agents 2001)*, Montreal, Canada, 2001.

[3] C. Bernon, M. Cossentino, M. Gleizes, and P. Turci. A study of some multi-agent meta-models. *Agent-Oriented Software Engineering V: 5th International . . .*, Jan 2005.

[4] J. Bosch. Specifying frameworks and design patterns as architectural fragments. In *Proceedings of TOOLS '98*, page 268, Washington, DC, USA, 1998. IEEE Computer Society.

[5] F. J. Budinsky, M. A. Finnie, J. M. Vlissides, and P. S. Yu. Automatic code generation from design patterns. *IBM Syst. J.*, 35(2):151–171, 1996.

[6] A. Chella, M. Liotta, and I. Macaluso. CiceRobot: a cognitive robot for interactive museum tours. *Industrial Robot: An International Journal*, 34(6):503–511, 2007.

[7] M. Cossentino. From requirements to code with the PASSI methodology. In *Agent Oriented Methodologies*, chapter IV, pages 79–106. Idea Group Publishing, Hershey, PA, USA, June 2005.

[8] M. Cossentino, S. Gaglio, L. Sabatucci, and V. Seidita. The passi and agile passi mas meta-models compared with a unifying proposal. In *In proc. of the CEEMAS'05 Conference*, pages 183–192, Budapest, Hungary, Sept. 2005.

[9] M. Cossentino, L. Sabatucci, and A. Chella. Patterns reuse in the PASSI methodology. In *ESAW*, pages 294–310, 2003.

[10] Digital Business Ecosystem. http://www.digital-ecosystem.org. onsite.

[11] A. H. Eden, A. Yehudai, and J. Gil. Precise specification and automatic application of design patterns. In *Proceedings of ASE '97*, page 143, Washington, DC, USA, 1997. IEEE Computer Society.

[12] Foundation for Intelligent Physical Agents. *FIPA Interaction Protocol Library Specification*, 2000.

[13] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements od Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY, 1995.

[14] I. Hammouda and K. Koskimies. An approach for structural pattern composition. In *Proceedings of SC 2007*, Braga, Portugal, March 2007.

[15] E. Kendall. Role modeling for agent system analysis, design, and implementation. In *IEEE Parallel and Distributed Technology*, volume 8, pages 34 – 41, IEEE, Apr-Jun 2000. IEEE Computer Society.

[16] D. Riehle. Describing and composing patterns using role diagrams. In K.-U. Mätzel and H.-P. Frei, editors, *1996 Ubilab Conference*, pages 137–152, Zürich, Germany, June 1996.

[17] L. Sabatucci. *A Framework for Rapid Development of Multi-Agent System*. PhD thesis, Dipartimento di Ingegneria Informatica, University of Palermo, Italy, 2008.

[18] L. Sabatucci and S. Gaglio. Separation of concerns and role implementation in the passi design process. In *5th International Conference on Industrial Informatics (INDIN 07)*, 2007.

[19] L. Sabatucci, A. Garcia, N. Cacho, M. Cossentino, and S. Gaglio. Conquering fine-grained blends of design patterns. In *10th International Conference on Software Reuse (ICSR 08)*, Lecture Notes in Computer Science. Springer, (in printing) 2008.

[20] S. Shapiro. *Encyclopedia of Artificial Intelligence*. John Wiley & Sons, Inc. New York, NY, USA, 1992.

[21] E. Sirin and J. Hendler. Semi-automatic Composition of Web Services using Semantic Descriptions. *WSMAI 2003*, 2003.

Institutional Environments

Porfírio Silva Rodrigo Ventura Pedro U. Lima
Institute for Systems and Robotics,
Instituto Superior Técnico
Lisbon, Portugal
porfiriosilva@isr.ist.utl.pt yoda@isr.ist.utl.pt pal@isr.ist.utl.pt

ABSTRACT
The concept of environment is of paramount relevance for new strategies to model systems of multiple artificial agents. This paper introduces a set of definitions designed to guide the modelling of institutional environments. This is part of ongoing research on a new strategy to conceptualize multi-robot systems, which takes a network of institutions as the control system for a collective of artificial embodied agents with bounded rationality and bounded autonomy. The definitions, given as structured tuples, attempt to capture a hypothesis on the main constitutive elements of the social order dynamics. That hypothesis is part of the institutional approach, which aims at responding to some difficulties of current perspectives on environment.

Categories and Subject Descriptors
J.4 [**Computer Applications**]: Social and Behavioral Sciences – *Economics, Sociology.*

I.2.9 [**Computing Methodologies**]: Artificial Intelligence – Robotics.

General Terms
Design, Economics, Theory

Keywords
Institutional Environments, Institutional Robotics

1. INTRODUCTION
This paper is part of an ongoing research on a new strategy to conceptualize multi-robot systems, which takes a network of institutions as the control system for a collective of artificial embodied agents with bounded rationality and bounded autonomy [15]. We conceive institutional environments as networked institutions embedded in wider environments. Our aim here is to suggest a set of definitions designed to guide the modelling of institutional environments.

The definitions, given as a tuples structure, try to capture a hypothesis on the main constitutive elements of the social order dynamics. The suggested definitions for "node of the institutional network", "institutional agent", and "institutional network", framed by an explicit presentation of our hypothesis on dynamics of social order, are given in Section 4.

Motivations for our perspective on institutional environments are presented in Sections 2 and 3. In Section 2 we mention some contributions to the emergence of the concept of environment as a tool of paramount relevance for new strategies to model systems of multiple artificial agents. Some difficulties associated with such concepts are also considered. In Section 3 we refer to the Institutional Robotics approach as the framework for our current research on institutional environments.

2. MIND AND ENVIRONMENT: FROM MENTALISM TO INTERACTION
The concept of environment emerges as a tool of paramount relevance for new strategies to model systems of multiple artificial agents. In this Section we mention some contributions to that process, mainly related to a shift from mentalist to interactionist underlying metaphors.

According to [1:49–54], a metaphor has been prevailing over cognitive science research programme on mind. It is the metaphor of an abstract inner space opposed to the outside world, whether the outside world includes the body or not. That same metaphor conceives a boundary between inner and outer spaces being traversed by perceptive stimuli (headed inward) and behavioural responses (headed outward). The unsuitability of this metaphor reveals itself where this dominant approach to mind is driven by its own difficulties to blur the difference between inside and outside, as a consequence of the endeavour to reproduce the entire world inside the head. This diagnosis of what Agre calls "mentalism" helps to understand the three great neglects at the heart of Good Old-Fashioned Artificial Intelligence: the neglect of the body, of the world, and of other agents.

Philip Agre is one of the proponents of interactionism as an alternative to mentalism, both to analyse living agents and to design artificial ones. To the interactionist alternative the central phenomenon is the interaction of agents with their environment [1:57–58]: "I propose thinking about computation in terms of machinery and dynamics. A machine (. . .) is an object in the physical world that obeys the laws of physics. [The dynamics] concerns the interactions between an individual (robot, ant, cat, or person) and its surrounding environment."

Andy Clark [3] also explains why there is a plastic frontier between mind, body, and world. On the one hand, it comes from natural evolution. Clark stresses that most of our daily behaviour is niche dependent. This means that we are not "general machines" prepared for every possible contingency, but instead sensitized to those particular aspects of the world that have special significance because of our way of life. On the other hand, there is also the impact of our culture on the world. We adapt our surroundings to our needs and life style. We perform "epistemic actions" [9], we organise things on space to unload computation

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

to the environment. One of the many examples given by David Kirsh, and mentioned by Clark, is: ”To repair an alternator, take it apart but place the pieces in a linear or grouped array, so that the task of selecting pieces for reassembly is made easier.” Language and arithmetic are widespread cognitive scaffolding tools for human beings.

Paul Dourish [5], while sharing interactionist views, strives for “embodied interaction”. Embodiment, the central element of the perspective Dourish puts forward, focus the study of cognition on the agent’s practical action on his world. Embodiment, far from being restricted to an agent situation in a physical environment, also counts on the social and organizational environments, and stresses the participative status the agent enjoys [5:19]: “Physical environments are arranged so as to make certain kinds of activities easier (or more difficult), and in turn, those activities are tailored to the details of the environment in which they take place. The same thing happens at an organizational level; the nature of the organization in which the work takes place will affect the work itself and the ways it is done.”

According to Henry Petroski [12], a deep aspect of our worldly condition is that we are surrounded by objects that are shaping and are being shaped by the slightest aspects of our daily life. He mentions Donald Norman’s suggestion that “there are perhaps twenty thousand everyday things that we might encounter in our lives”. However, he argues against rationalist conceptions of artefacts: form does not follow function. Necessity and utility does not determine technological diversity. Already in 1867 Karl Marx was surprised to learn that five hundred different kinds of hammers were produced in Birmingham. Practical use always goes beyond rational anticipation, and the variety of entrants in any design competition shows at what extent the specification of a design problem in no way dictates its solution. Petroski’s reflections could help to generalise the notion of artefact, which, according to [13:130], has been introduced recently in Multiagent Systems (MAS) as a first-class abstraction representing devices that agents can either individually or collectively use to support their activities.

Researchers within the MAS framework are calling for an explicit recognition of the responsibilities of the environment, irrespective of the agents. Since there are lots of things in the world that are not inside the minds of the agents, there is a need to surpass the subjective view of MAS, where the environment is somehow just the sum of some data structures within agents, and to embrace an objective stance towards environment, enabling modellers to deal with systems from an external point of view of the agents [18:128].

The point is that the active character of the environment must be taken seriously: some of its processes change its state independently of the activity of any agent (a rolling ball that moves on); multiple agents acting in parallel can have effects any agent will find difficult to monitor (a river can be poisoned by a thousand people depositing a small portion of a toxic substance in the water, even if each individual portion is itself innocuous) [17:36]. Moreover, dynamic environmental processes independent of agents’ purposes and almost unpredictable aggregate effects of multiple simultaneous actions are not phenomena restricted to physical environments. Similar phenomena can occur in organizational environments: if nine out of ten of the clients of a bank decide to draw all their money out at the same date, bankruptcy of that institution could be the unintended effect. Furthermore, taking into account environment opens new means to deal with indirect or mediated interaction, which [17:14] considers characterized by properties such as name uncoupling (interacting entities do not have to know one another explicitly), space uncoupling and time uncoupling (they do not have neither to be at the same place nor to coexist at the same time).

Trying to understand environment mediated interaction, stigmergy is another worth mentioning point. The term “stigmergy” captures the notion that, if multiple agents leave signs in a shared environment and their subsequent actions are determined by they sensing those signs, coordination within large populations is achievable by simple means, namely without any direct communication. Most common examples coming from insects and ant societies, stigmergy is usually associated with simple agents with severely bounded computational resources. Yet, Parunak, along with researchers talking of self-organisation emerging just from mere local interaction as a widespread phenomenon, even for more sophisticated agents, claims that stigmergy is pervasive also in human societies. ”It would be more difficult to show a functioning human institution that is not stigmergic, than it is to find examples of human stigmergy” [11:163].

Contributing the “cognitive stigmergy” notion, [13] converges on this view. The point is that, since the agents we work with have not just reactive, but also cognitive activities and can adapt and learn, there is a need to generalise from stigmergy to cognitive stigmergy. Now, cognitive stigmergy asks for more sophisticated environments, being “in general more articulated than a mere pheromone container”, where “the effects of agent actions on the environment are understood as signs”, and “hold a symbolic value” [13:127,132].

We have just mentioned a few examples of recent interesting developments on the role of environments for systems of multiple agents. But some difficulties associated with these developments are worth mentioning.

A difficulty that must be a concern for all systems with just software environments is raised, e.g., by [17]. Contrary, for example, to real robots systems evolving on physical environments, all aspects of a purely virtual environment (and of a purely virtual agent) must be modelled explicitly. This raises conceptual concerns related to the role of the modeller, and asks for a clarification of the very concept of environment. Because a computational environment that is part of a software system should not be confused with the environment with which the system interacts, the different levels and dynamics at stake must be made explicit.

That point is mentioned by [19]. Discussing the Human-Computer Interaction issue, the authors say: “the role of humans in multiagent systems can be very diverse. In some applications, humans can play the role of agents and interact (. . .) with the application environment” [19:21].

Another promising issue is raised by the same researchers, talking of a “reflective level”. Writing that “Such reflective interface enables cognitive agents to modify the functional behaviour of the environment”, and that the reflective level can be seen as “a means for self-organizing MAS” [19:11], they are opening new

frontiers for artificial collective systems, promising more careful attention to the real meaning of the ”autonomy” of the agents.

Our institutional environment approach could give some ways to deal with these difficulties. And, additionally, incorporate a factor with easily recognisable importance to human societies but usually forgotten in systems of multiple artificial agents. It is all about history and accumulation. Throughout the centuries, humans have been accumulating small modifications to myriads aspects of our physical and social world, not necessarily being aware of all them. In a wholly different attitude, designers of artificial systems pretend to be able to play gods and genesis anew each time they start modelling another version of their systems. Our institutional approach also intends to respond to that situation, giving place to history and accumulation within systems of multiple artificial agents.

In the next Section we present some global aspects of this institutional approach, so paving the way to their concrete application in Section 4.

3. INSTITUTIONAL ENVIRONMENTS

We have proposed Institutional Robotics [15] as a new approach to the design of multi-robot systems, mainly inspired by concepts from Institutional Economics, an alternative to mainstream neoclassical economic theory [7]. The Institutional Robotics approach intends to sophisticate the design of collectives of artificial agents by adding, to the currently popular emergentist view, the concepts of physically and socially bounded autonomy of cognitive agents, and deliberately set up coordination devices.

On the one hand, full autonomy is not attainable. Autonomous agents are not necessarily self-sufficient. Most of the time agents depend on resources and on other agents to achieve some of their goals. Dependences imply interests: world states that objectively favour the achievement of an agent’s goals are interests of that agent. Limited autonomy of agents comes from these dependences and interests relations [4].

On the other hand, collective order does not always emerge from individual decisions alone. A set of experiences within MAS, reported in [2], proved that, at least in some situations, merely emergent processes may lead to inefficient solutions to collective problems. Due to the absence of any opportunity for individuals to agree on a joint strategy, this is true even in some situations where the best for each individual is also the best for the collective. Thus, coordination devices deliberately set up by agents could be useful and must be considered. Still, this approach does not preclude emergence. Bounded rationality combines with bounded autonomy to give place to emergent phenomena: there are deliberate planned actions but they may produce unintended effects beyond reach of the agents’ understanding.

The Institutional Robotics approach endeavours to reflect these aspects taking institutions as decisive elements of the environment of multi-agent systems. Within this approach, the control system for a collective of artificial agents is a network of institutions. However, in this context, we adopt a broad concept of institution [15:600]: “Institutions are coordination artefacts and come in many forms: organizations, teams, hierarchies, conventions, norms, roles played by some robots, behavioural routines, stereotyped ways of sensing and interpret certain situations, material artefacts, some material organization of the world. A particular institution can be a composite of several institutional forms.” In the next section we further refine some concepts that are crucial to future implementation of this approach.

4. A NETWORK OF INSTITUTIONS AS THE CONTROL SYSTEM FOR A COLLECTIVE OF ARTIFICIAL AGENTS

4.1 A hypothesis on the main constitutive elements of the social order dynamics

The classic problem of the social sciences, the problem of social order or the micro-macro problem, is the question that introduces [6]: “How does the heterogeneous micro-world of individual behaviours generate the global macroscopic regularities of the society?”. Our institutional approach aims to contribute to a better understanding of that problem within systems of multiple artificial agents interacting with natural ones. Our strategy consists of putting together the main constitutive elements of the complex dynamics of institutional order, let them interact and let us interact with them, draw some lessons from the experiment, and test these lessons on new generations of experiments. Our tentative hypothesis is that the main constitutive elements of the social order dynamics to experiment with are as follows.

4.1.1 The powerful engine of the interactive workings of inner life and outer life mechanisms of the agent

Agents have built-in reactive behaviours, routines, and deliberative competences. Agents have partial models of themselves (they know some, but not all, of their internal mechanisms). Some of the internal mechanisms known by the robots can be accessed and modified by themselves. These elements are constitutive of the inner life of the agent.

The continuing functioning of any agent depends on some material conditions. Basic needs drive the activity of agents and lead to modifications of both physical and social world. How an agent interprets its world and the possibilities it affords depends on the physical and social world models the agent bears upon. An agent’s links to some, and not others, available institutions on its environment influence the world models it puts to use, thus biasing its behaviour. Beyond being influenced by its links to a subset of the existing institutions, the agent also is, at some extent, able to exert some influence on institutional mechanisms. However, autonomous agents do not transcribe institutional models without (slightly or not) modifying them. So, basic needs, fundamentally disposed by nature, have strong, even if indirect, interaction with social mechanisms like institutions. These elements are at the root of the dynamics we call “outer life of agents”.

The inner life of the agent has multifaceted effects at behavioural level, and thus on its participation in social interaction. The agent’s activities on its social and material environments interact intensively with its internal mechanisms. The joint workings of inner and outer life are of paramount importance for the emergence of complex collective phenomena. The diffuse frontier between nature and nurture is also captured by our notion of interaction between inner life and outer life of an agent.

4.1.2 Agents with reactive and deliberative mechanisms in a world with mental and material aspects

Let us, following a number of researchers (e.g., [10][16]), call coordination artefacts to those artefacts shaped for coordinating the agents' actions. Now, some interesting coordination artefacts are associated not only with physical but also with cognitive opportunities and constraints (deontic mediators, such as permissions and obligations). Recognizing all of those enables a single agent to act in a coordinated way: a driver approaching a roundabout is obliged, only by physical properties of the artefact, to slow down and go right or left to proceed; traffic regulations add something more indicating which direction all drivers have to choose not to crash with others. In another example, some rules (or other kinds of mental constructs) can be associated to a material object to implement some aspect of the collective order (a wall separating two countries is taken as a border; there are some doors in the wall to let robots cross the border; some regulations apply to crossing the border).

We can say that material objects are devices for institutions when they implement some aspect of the collective order. Notwithstanding, the boundaries between institutional and purely physical aspects of the world are not sharp. Consider a wall separating two buildings: it effectively implements a prohibition of visiting neighbours if the robots are not able to climb. However, if the wall is seen as just an element of the physical world, some robots gaining access to opposite building with newly acquired tools or physical capabilities will not be minded as a breach of a prohibition. Still, modifications of the material world creating new possibilities of interaction can become institutional issues. If the collective prefers to preserve the previous situation of separated buildings, the new capability of the robots to climb the wall could give place to new regulations.

This kind of artefacts, along with the coordination purposes they serve, illustrates how much could it be difficult to separate, either in conceptual or in practical terms, material from mental aspects of our world. That difficulty is closely related to our condition as complex agents combining reactive and deliberative ties, both to the physical and the social world.

4.1.3 Nobody is born alone in the wild. Not even artificial agents. And, at times, humans act as ancestors for artificial agents.

When a natural human agent comes into world, generations of ancestors have been shaping physical and social environments for centuries. Yet, the human agent can contribute with some modifications, some of which will last; some others will vanish sooner or later. The same happens with institutions for artificial collectives. When an artificial agent comes into existence, designers have already settled most contingencies that can determine its life. But, if it enjoys some kind of autonomy, it will also contribute to the continuing evolution of its world. The institutional environment at any point in the history of a collective is always a mix of inherited and newly adopted forms. So, the designers of an artificial collective must shape the first version of an institutional network. Thus, they play the role of predecessors for the artificial agents and (at least some aspects of) their environment. And, if we want to develop a better understanding of the interaction between human and artificial agents, designers must stay involved; say "as participative gods".

4.2 Definitions

Now, we will try to capture the tentative hypothesis stated above with a set of definitions designed to guide the modelling of institutional environments: node of the institutional network, institutional agent, and institutional network.

Departing from prevalent approaches (e.g., [14],[8]), we bring forward the following tentative informal definition: «Institutions are cumulative sets of persistent artificial modifications made to the environment or to the internal mechanisms of a subset of agents, thought to be functional to the collective order.» Building upon this, the main constituents of institutional environments will be defined by structured tuples.

Starting with the definition of "node of the institutional network", instead of with the definition of "institutional network", deserves an explanation. Since we are not usually able to reach an external viewpoint on complex societies, especially where we enjoy a participative status, a top down approach could prove unrealistic. From an epistemological standpoint, starting with some particular institutions, and then trying to broaden our understanding of the network, looks like a more modest but reliable strategy. Additionally, this approach better accommodates the existence of genuine emergent dynamics.

Moreover, we talk of "node of the institutional network", and not of "institution", because we don't know a principled way to get general clear-cut distinctions between an institution and a network of institutions. For example: the judicial system of a country must be seen as an institution or as a net of institutions (a net of courts of justice)?

Definition 1. A Node of the Institutional Network is a tuple < *ID, Rationale, Modifiers, Network, Institutional Building, History* > where:

ID = < *Label, Form* >

Label: Unique ID for this node of the institutional network.

Form: Generic form of this node (formal organisation, informal group, role, rule (law, norm, convention, right), behavioural routine, stereotyped way of sensing and interpret certain situations, material artefact, some material organisation of the world, a composite of several basic institutional forms). To each form corresponds a specific way of communicating to agents the expectations embedded on a specific node of the Institutional Network.

Rationale = < *Goals, Activities* >

Goals: Collective goal this institution is thought to be functional to.

Activities: Specific activities of the agents this node of the institutional network is supposed to serve to.

Modifiers = < Cognitive Modifiers, Praxic Modifiers >

Cognitive Modifiers = < Ideologies-P, Ideologies-S, Material Infrastructure for Cognitive Modifiers, Mental Infrastructure for Cognitive Modifiers >

Ideologies-P: ideologies about the physical world.

Ideologies-S: ideologies about the social world.

(Ideologies are partial world models provided by institutions, and so in principle shared by the subset of all agents with links to specific institutions. One and the same institution can provide several ideologies to agents. There is no consistency requirement associated to the set of ideologies provided by one and the same institution. Ideologies include partial ontological assumptions about some regions of the multi-agent system's world: entities, their properties, relations possibly holding among them.)

Material Infrastructure for Cognitive Modifiers: Material aspects of the institution that impact the cognitive mechanisms of the agents (for example, tools for augmented computational power - like calculator or computers, or tools for modified perception, like microscopes, telescopes, sensors for sound or light waves outside the range of natural equipment of the agents - where the access to those tools is not granted to every agent and depends on institutional appurtenance or institutional position).

Mental Infrastructure for Cognitive Modifiers: Mental aspects of the institution that impact the cognitive mechanisms of the agents (for example, concepts that apply some specific distinctions to organize some region of the perceptive space - where the access to those concepts is not granted to every agent and depends on institutional appurtenance or institutional position).

Praxic Modifiers = < Material Infrastructure for Praxic Modifiers, Mental Infrastructure for Praxic Modifiers, Enforcement >

Material Infrastructure for Praxic Modifiers: Material aspects of the institution that impact the action mechanisms of the agents (for example, physical objects functioning exclusively by means of its physical characteristics given the physical characteristics of the agents: a wall separating two buildings implements the prohibition of visiting neighbours if the robots are not able to climb it).

Mental Infrastructure for Praxic Modifiers: Mental aspects of the institution that impact the action mechanisms of the agents (e.g., a program to control a sequence of operations). Some infrastructures combine material and mental aspects (for example, a traffic sign is a physical object which functioning is due to a specific link to a mental construct: a traffic rule).

Enforcement: Mechanisms associated with this node of the institutional network specifically designed to prevent or to redress negative effects of violation of expected behaviour (examples are fines and reputation) and to reward observance (examples are prizes and advancement in rank or status). Enforcement mechanisms affect future acting possibilities of agents.

Network: Links to other nodes of the institutional network (the existence of a link, its nature).

Institutional Building = <Institutional Imagination, Co-operative Decision-making >

Institutional Imagination: Mechanisms designed to facilitate "thought experiments" about possible modifications to actual institutions, or even alternative institutions (agents could test alternatives without actually implement them). Results of Institutional Imagination (possibly fuelled by access to the *Institutional Memory* of the *Institutional Network*, and to the *Lineage & Accumulation* element of *History* of a *Node of the Institutional Network*) would eventually be put forward to Co-operative Decision-making mechanisms specific to this node of the institutional network.

Co-operative Decision-making: Mechanisms designed to implement collective deliberation about possible modifications to actual institutions, or about alternative institutions.

History = < Material Leftovers, Mental Leftovers, Lineage & Accumulation >

Material Leftovers: Material objects that once served some aspect of the institutional dynamics but have gotten disconnected from it. (Because the continuing existence of a material object can be uncoupled from the continuing existence of the institutional device it implements – e.g., the wall could be demolished without eliminating the border; the border can be eliminated without demolishing the wall – a material leftover of a discarded institution can last as an obstacle in the world.)

Mental Leftovers: Mental constructs that once served some aspect of the institutional dynamics but have gotten disconnected from it (for example: norms that once served a collective goal and persist notwithstanding the goal having been relinquished).

Lineage & Accumulation: Old versions of this node of the institutional network, saved as a list of cumulative modifications to the oldest known version.

Definition 2. An Institutional Agent is a tuple < *ID, Nature, Individual Links, Institutional Links, Ideas, Praxis* > where:

ID = < Name, Natural Group Name >

Name: Specific individual identification.

Natural Group Name: (for example) Family name, for humans.

Nature =<Relatives, Species, Basic Needs, Built-in Mechanisms>

Relatives: Names of the other members of the Natural Group.

Species: Human, Non-Human Animal, Robot, ...

Basic Needs: Material conditions for continuing functioning of the agent.

Built-in Mechanisms: Built-in perceptive and motor apparatus, reactive behaviours, routines and deliberative competences.

Individual Links: Names of other agents this agent can identify by their names.

Institutional Links: Nodes of the institutional network the agent is currently linked to.

Ideas = < Current Ideologies-P, Current Ideologies-S, Current Opinions, Models of the Self, Institutional Knowledge >

Current Ideologies-P: Ideologies-P the agent adheres to at present.

Current Ideologies-S: Ideologies-S the agent adheres to at present.

(Notwithstanding the fact that Institutional Links determine in principle which ideologies the agent adheres to, actually not all agents are fully aware or fully adhere to all ideologies proposed by the institutions they are linked to.)

Current Opinions: Opinions the agent currently holds. An "opinion" is an individual deviation from world models provided by institutions. By virtue of bearing an "opinion", as well as bearing an "ideology", the behaviour of an agent can be modified.

Models of the Self: Every agent know some, but not all, of their internal mechanisms (agents have partial models of themselves).

Institutional Knowledge: Knowledge the agent has about the Institutional Network.

Praxis = < Physical World Tools, Social World Tools, Self-Improvement Tools >

Physical World Tools: Tools enabling the agent to modify the material organisation of the physical world, and thus, the material infrastructure of the institutions (including, but not restricted to, physical tools: influencing other agents is a possible delegate way of modifying the physical world).

Social World Tools: Tools enabling the agent to modify the organisation of the social world.

Self-Improvement Tools: Some of the internal mechanisms known by the agents can be accessed and modified by themselves.

Definition 3. An Institutional Network is a tuple < *Nodes, Connections, Institutional Memory, Emergency Observatory, Participative Gods* > where:

Nodes: Currently active institutional nodes.

Connections: Known/explicit links between active nodes.

Institutional Memory: Incomplete repository of old/inactive institutions which can be used to feed Institutional Building mechanisms. Each old/inactive institution is saved as a list of cumulative modifications to the oldest known version.

Emergency Observatory: Available information about emergent collective phenomena within the multi-agent system which is under control of this Institutional Network.

Participative Gods =<Customer, Designer, Rationale, Ontology>

Customer: Who ordered this control system for a collective of artificial agents.

Designer: Who designed this control system for a collective of artificial agents.

Rationale = < Goals, Activities >

Goals: Goals Customer and Designer want this multi-agent system to be functional to.

Activities: Activities Customer and Designer want this multi-agent system to serve to.

Ontology: Ontological assumptions of the Customer and the Designer about the multi-agent system's world (entities, their properties, relations possibly holding among them), given the goals they (the Customer and the Designer) place on it (the system).

4.3 How basic dynamics are represented within the tuples structure

We have tried to capture our tentative hypothesis on the main constitutive elements of the social order dynamics (see 4.1. above) with definitions 1 to 3. The tuples structure expresses the complex interaction of some basic dynamics of the social life of artificial agents in interaction with human beings. We will now underline the main components of these dynamics within the tuples structure.

The agent modifies itself as it modifies its world in ways that are not always fully intentional and that cannot be completely anticipated. The dynamics of interaction between inner life and outer life of an agent (see 4.1.1. above) is mainly represented by interactions of the following elements:

Agent → Nature → Built-In Mechanisms.

Agent → Ideas → Models of the Self.

Agent → Praxis → Self-Improvement Tools.

Agent → Nature → Basic Needs.

Agent → Praxis → Physical World Tools, Social World Tools.

Node of the I. Network → Modifiers → Cognitive Modifiers → Ideologies-P, Ideologies-S.

Agent → Ideas → Current Ideologies-P, Current Ideologies-S, Current Opinions.

Node of the I. Network → Network, Institutional Building.

Physical and cognitive opportunities and constraints represented by artefacts in the environment, and sometimes recognized by the agents, combine with internal mechanisms of the agents to give rise to complex behavioural patterns. Thus, behaviour can be modulated by way of environmental or internal mechanisms which are partly modifiable by the agents themselves. The dynamics of intertwined reactive and deliberative mechanisms of agents in a world with mental and material aspects (see 4.1.2. above) is mainly represented by interactions of the following elements:

Node of the I. Network → Modifiers → Cognitive Modifiers → Material Infrastructure for Cognitive Modifiers, Mental Infrastructure for Cognitive Modifiers.

Node of the I. Network → Modifiers → Praxic Modifiers → Material Infrastructure for Praxic Modifiers, Mental Infrastructure for Praxic Modifiers.

Node of the I. Network → Institutional Building.

Node of the I. Network → History → Material Leftovers, Mental Leftovers.

Agent → Nature → Built-in Mechanisms.

Agent → Ideas → Models of the Self.

Autonomous agents, coming into existence in a world shaped by generations of predecessors or designers, can also contribute to the continuing evolution of their environment. The dynamics of inherited vs. newly adopted institutions (see 4.1.3. above) is mainly represented by interactions of the following elements:

Node of the I. Network → Institutional Building.

Node of the I. Network → History → Lineage & Accumulation.

Agent → Ideas → Institutional Knowledge.

Institutional Network → Institutional Memory.

Where human beings are designers and users of collectives of artificial agents, the understanding of interaction between human and artificial agents becomes part of the understanding of the artificial system. Modelling crucial aspects of the interaction between human and artificial agents within the control system of the collective can improve that understanding. The dynamics of human/artificial agents' relationships (see 4.1.3. above) is mainly represented by interactions of the following elements:

Node of the I. Network → Rationale.

Institutional Network → Participative Gods → Rationale, Ontology.

Node of the I. Network → Modifiers → Cognitive Modifiers → Ideologies-P, Ideologies-S.

Node of the I. Network → ID → Form.

(The latter element will ease comparisons between artificial institutions and characteristic institutions of the Customer and Designer environments, thus fuelling understanding of constraints imposed by goals/activities the multi-agent system is supposed to serve.)

Our notion of interaction between inherited and newly adopted institutional forms leaves room both for deliberately set up institutional mechanisms and for emergent aspects of institutional evolution, as represented by these elements of the tuples structure:

Node of the I. Network → Institutional Building.

Node of the I. Network → History → Material Leftovers, Mental Leftovers, Lineage & Accumulation.

5. CONCLUSIONS AND FUTURE WORK

We introduced a set of definitions designed to guide the modelling of institutional environments, as part of a strategy to control collectives of artificial embodied agents (e.g., multi-robot systems), with bounded rationality and bounded autonomy, by a network of institutions. Building upon an informal definition, the main constituents of institutional environments (nodes of the institutional network, institutional agents, and institutional networks) were defined by structured tuples. The social order dynamics results of interactions among the elements of the defined tuples.

It is clear for us that deeper work must be done to gain further insight on the relevance of the constituent elements and their interactions. This will be the subject of the next steps in our research. We are working on two scenarios of different levels of complexity in order to experiment with partial aspects of our concept. The simpler scenario consists of a set of roundabouts designed to regulate urban traffic, directly associated with traffic signs and framed in a more general way by a road code. The more complex scenario consists of a "search and rescue" operation, where heterogeneous cognitive robots must cooperate, both with other species of robots and with humans, on an unstructured landscape, aiming to help victims of some kind of disaster or emergency situation.

Once the required clarifications are achieved, the tuple definitions will act as prescriptions for an ontology to be used in the software programs we plan to design and implement, so as to control a collective of real robots and their environments, including the interaction among humans and robots. Such a demonstration will act as a proof of concept of the Institutional Robotics framework.

6. ACKNOWLEDGMENTS

The research of the first author is supported by Fundação para a Ciência e a Tecnologia (grant SFRH/BPD/35862/2007). This work was partially supported by Fundação para a Ciência e a Tecnologia (ISR/IST pluri-annual funding) through the POS Conhecimento Program that includes FEDER funds. We would like to thank Fausto Ferreira, Gonçalo Neto, and Matthijs Spaan for the fruitful discussions on various aspects of Institutional

Robotics. Moreover, we thank anonymous reviewers for constructive comments and suggestions.

7. REFERENCES

[1] Agre, P. 1997. Computation and Human Experience. Cambridge University Press, Cambridge.

[2] Castro Caldas, J., Coelho, H. 1999. The Origin of Institutions: socio-economic processes, choice, norms and conventions. In Journal of Artificial Societies and Social Simulation, 2:2 (http://jasss.soc.surrey.ac.uk/2/2/1.html).

[3] Clark, A. 1997. Being There: Putting Brain, Body, and the World Together Again. The MIT Press, Cambridge, MA.

[4] Conte, R., Castelfranchi, C. 1995. Cognitive and Social Action. The University College London Press, London.

[5] Dourish, P. 2001. Where the Action Is: The Foundations of Embodied Interaction. The MIT Press, Cambridge, MA.

[6] Epstein, J.M., Axtell, R. 1996. Growing Artificial Societies: Social Science from the Bottom Up. The Brookings Institution and the MIT Press, Washington, DC.

[7] Hodgson, G.M. 1988. Economics and Institutions: A Manifesto for a Modern Institutional Economics. Polity Press, Cambridge.

[8] Hodgson, G.M. 2006. What Are Institutions? In Journal of Economic Issues, 40:1, 1-25.

[9] Kirsh, D., Maglio, P. 1994. On distinguishing epistemic from pragmatic action. In Cognitive Science, 18, 513–549.

[10] Omicini, A., Ricci, A., Viroli, M., Castelfranchi, C., Tummolini, L. 2004. Coordination Artifacts: Environment-Based Coordination for Intelligent Agents. In Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (New York, NY, July 19 - 23, 2004). IEEE Computer Society, Washington, DC, 286-293.

[11] Parunak, H.v.D. 2006. A Survey of Environments and Mechanisms for Human-Human Stigmergy. In Environments for Multi-Agent Systems II, E4MAS 2005, Selected Revised and Invited Papers, Springer-Verlag, Berlin Heidelberg, 163–186.

[12] Petroski, H. 1992. The Evolution of Useful Things. Vintage, New York.

[13] Ricci, A., Ominici, A., Viroli, M., Gardelli, L., Oliva, E. 2007. Cognitive Stigmergy: Towards a Framework Based on Agents and Artifacts. In Environments for Multi-Agent Systems III, E4MAS 2006, Selected Revised and Invited Papers, Springer-Verlag, Berlin Heidelberg, 124–140.

[14] Searle, J.R. 2005. What is an institution? In Journal of Institutional Economics, 1:1, 1-22.

[15] Silva, P., Lima, P.U. 2007. Institutional Robotics. In F. Almeida e Costa et al. (Eds.): ECAL 2007, LNAI 4648, Springer-Verlag, Berlin Heidelberg, 595–604.

[16] Tummolini, L., Castelfranchi, C. 2006. The cognitive and behavioral mediation of institutions: Towards an account of institutional actions. In Cognitive Systems Research, 7:2-3, 307–323.

[17] Weyns, D., Parunak, H.v.D., Michel, F., Holvoet, T., Ferber, J. 2005. Environments for Multiagent Systems, State-of-the-art and Research Challenges. In Weyns, D., Parunak, H.v.D., Michel, F. (eds.): Proceedings of the 1st International Workshop on Environments for Multi-Agent Systems. Springer-Verlag, Berlin Heidelberg, 1–47.

[18] Weyns, D., Schumacher, M., Ricci, A., Viroli, M., Holvoet, T. 2005. Environments in Multiagent Systems. In The Knowledge Engineer Review, 20:2, 127-141.

[19] Weyns, D., Omicini, A., Odell, J. 2007. Environment as a first class abstraction in multiagent systems. In International Journal on Autonomous Agents and Multi-Agent Systems, 14:1, 5-30.

Enhance Collaboration in Diabetic Healthcare for Children using Multi-agent Systems

Peng Zhang
Blekinge Institute of Technology
PO Box 520, 37225
Ronneby, Sweden
+46 457 385855
peng.zhang@bth.se

Bengt Carlsson
Blekinge Institute of Technology
PO Box 520, 37225
Ronneby, Sweden
+46 457 385813
bengt.carlsson@bth.se

Stefan J. Johansson
Blekinge Institute of Technology
PO Box 520, 37225
Ronneby, Sweden
+46 457 385831
stefan.johansson@bth.se

ABSTRACT
We developed a multi-agent platform as a complement to the existing healthcare system in a children's diabetic healthcare setting. It resolves problems related to the difficulty of collaboration between the stakeholders of the problem domain. In addition, it gives us an opportunity to support the decision making of the stakeholders using Multi-agent Systems. The collaboration situation is believed to be improved by the agent-based services, such as, diabetes monitoring and alarm, scheduling, and task delegation.

Categories and Subject Descriptors
I.2.11 [Computing Methodologies]: Artificial Intelligence: Distributed Artificial Intelligence - *Multiagent systems*, J.3 [Computer Applications]: Life and Medical Sciences - *Medical information systems*

General Terms
Design, Reliability, Security, Human Factors.

Keywords
Multi-agent System, children's diabetic healthcare, collaboration, MAS coordination, diabetes monitoring and alarm, agent based scheduling.

1. INTRODUCTION
The need to make work more effective and an ongoing technical progress illustrate the necessity of coordinating and collaborating activities within health care systems. Agent technology is believed to be able to alleviate this necessity, as Nealon and Moreno stated in [1]: "the basic properties of intelligent agents (autonomy, proactivity, social ability) and the features of Multi-agent Systems (MAS) (management of distributed information, communication and coordination between separate autonomous entities) suggest that they offer a good option to consider when trying to solve problems in health care domains."

MAS techniques have been applied in the various health care fields of telemonitoring [2], medical monitoring and diagnosis [3-5], remote service and scheduling [6], elderly care and home care [7, 8] and healthcare coordination [9, 10].

Especially there are some applications in diabetic health care:

- *Diabetic monitoring and alarm*: The SuperAssist project introduces personal assistants in the care of diabetes patients, assisting the patients themselves, the medical specialists looking after the patients healthcare, and the technical specialists responsible for maintaining the health of the devices involved [11]. The SuperAssist framework aims to reduce the costs by improving the local, self-care capacity of people by efficient employment of remote, distributed expertise. The M2DM telemedicine service can monitor and receive blood glucose data, pass it to an intelligent agent that interprets the data and if needed trigger an alarm [12, 13]. The main contribution of the M2DM project is combining statistics, rule-based techniques and model-based techniques in its Knowledge Management agents in order to process the patient data and generate alarms automatically.
- *Biomedical control and management in diabetes*: Amigoni et al. introduced MAS to diabetic information management with the *anthropic agency architecture* [14]. The T-IDDM project provides telemedicine services to diabetic patients especially the management of insulin dependent diabetic patients [15]. The DIABTel telemedicine system complements the daily care and intensive management of diabetic patients through telemonitoring and telecare services [16]. Li and Istepanian proposed a model for diabetes management [17] and Tian and Tianfield developed a telemedicine system for diabetes based on agent technology [18].

Most of the applications above are focused on decision support systems, diagnosis and monitoring, coordination and management etc., in the hospitalized settings. In the case of chronic diseases, e.g. diabetes, most of the health care is given in homes or in other un-hospitalized settings. This paper describes how software agents can be used in children's diabetic healthcare in un-hospitalized settings. We argue that the agents we have implemented are able to help us overcome weaknesses in children diabetic healthcare of today, such as, communication problems, poor decision support, etc.

The paper is organized as follows. First, the diabetic healthcare management is described, with a focus on children diabetes. Then we describe the design and implementation of our MAS – IMAS.

Jung, Michel, Ricci & Petta (eds.): *AT2AI-6 Working Notes, From Agent Theory to Agent Implementation, 6th Int. Workshop,* May 13, 2008, AAMAS 2008, Estoril, Portugal, EU.

Short descriptions of the functionalities are given and last we discuss the system, draw conclusions and line out future work.

2. DIABETIC HEALTHCARE MANAGEMENT: CHILDREN DIABETES IN FOCUS

The diabetic healthcare management has been described in previous work [19]. Two main characteristics are the distribution of patients and the level of multi-care involved. Diabetic patients are geographically distributed and the care-providers (doctors, nurses, parents) are only occasionally working together. Diabetes regularly leads to many kinds of complications, e.g. for the eyes, feet, skin, heart, kidneys, nervous system, celiac diseases, etc. The diabetic healthcare normally takes place in un-hospitalized settings, e.g., at homes, at work, or during travels. Diabetes cannot be cured permanently, so the patients must be able to take care of themselves in the daily life. Corresponding care-providers must be involved, e.g., doctors and nurses from corresponding fields, dietitians, personal assistants, parents and school nurses in the case of diabetic children.

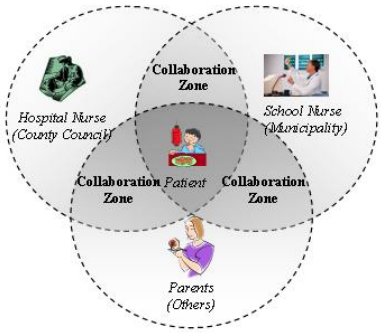


Fig. 1 The 'Collaboration Zones' between hospital nurses, school nurses and parents in the children diabetic healthcare setting. The patient is the least common denominator of the interactions.

It has been required by the nurses that this collaboration situation must improve. Based on interviews with hospital nurses and school nurses, the following problems/issues within the children diabetic healthcare were identified.

- *Communication problems*. The computer systems of the County Council and the municipalities are not compatible. In acute situations in school, the parents and the patients have to manually contact the hospital doctors or nurses by phones.
- *Intelligent Decision Support*. There is software attached with most of medical devices in the market to display the measured data in charts. This kind of software is installed on the patients' private computers where the measured data is also stored without easy access for nurses. Even if a graphical data presentation is available to the nurses, it may still be difficult to explain the situation to the patients via telephone or mail. A more intelligent computerized system is preferable.

Privacy. According to the law in a number of countries, medical journals of the patients cannot be transferred to anybody without the permissions from the patients or their guardians. Thus, neither the municipality nor the County Council can freely share a journal. Any new system is forbidden to break this privacy rule. This issue is left out of the scope of this paper, although we should be aware of it.

2.1 Scenario

To better explain the collaboration issues, we provide the following scenario.

Linus is a student of the fourth grade in a primary school. Unfortunately he is suffering of diabetes of type one and needs regular insulin injections every day. One day Linus felt unwell. So he went to the school nurse for the glucose measurement. The result from the glucometer did not provide enough information to the nurse since the nurse did not have the history record of Linus. Therefore, the nurse made phone calls to the parents and the hospital nurses for more information about Linus. After several phone calls, it was found that Linus had some high carbohydrate food at lunch, and did not take the insulin injection on time.

The above scenario illustrates a typical collaboration situation where the following problems can be found:

1. The glucose records are stored at separate places, e.g., hospital, home, schools etc. So it is difficult to access all of them from one location, when needed.
2. In acute situation, nurses have to communicate with the other care providers who are related to the patient, in order to get a systemic overview of the patient. Some of the communications can be automated, if the healthcare provider can remotely access part of the patient record, e.g., glucose record.

Insufficient information is provided to the care providers to make decisions. Some relevant information, e.g., patient activities, insulin injection records, is necessary for nurses in order to make decisions. However, that information is not presented.

3. DESIGN OF IMAS

In order to solve the above problems, we developed a MAS based system, IMAS, to support the daily healthcare activities of patients and healthcare staff. The IMAS system provides them with real time glucose monitoring and management, intelligent decision supports, user task delegation.

Various agent development methodologies and platforms exist. We adopted the Gaia methodology [20] for the system analysis and design of IMAS. Gaia was designed to deal with coarse-grained computational systems, to maximize some global quality measure, and to handle heterogeneous agents independent of programming languages and agent architectures [21]. The IMAS agents were implemented with JADE [22] making IMAS FIPA compliant [23]. In the IMAS settings, we assume that each agent is associated with a human user in the real world, e.g., a patient, a nurse, a parent, etc. Human users are considered necessary in this

case, because we cannot rely on computer agents alone in acute situations for healthcare treatments. However, in a daily diabetic healthcare setting, computer agents can work by themselves without human control.

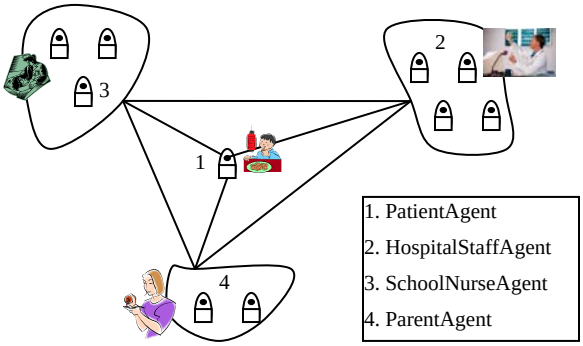


Fig. 2 The required communication paths of the IMAS architecture

IMAS agents work either on stationary PCs or on mobile devices like Pocket PC and Smart Phones. Each agent works for a user and provides some predefined services. The coordination among agents is realized though predefined IMAS communication protocols, i.e., the AlarmProtocol, the MeetingProcotol, the TaskProtocol, etc. Fig. 2 is an overview of the IMAS agent system. More details of each agent will be given later

3.1 Needs Specifications

There are many miscellaneous tasks in the daily diabetic healthcare. Some of them are critical and must be completed in a timely and precise fashion. For instance, the diabetic care providers (especially the nurses in this case) work on similar tasks by routine. They need to check the records of the patients before they visit them, and report the new information as they finish. The same task is performed several times every day, since they need to visit many patients. The task of retrieving and reporting the information of a patient is simple but must be done in an efficient and correct way, because delayed or wrong information may lead to serious consequences.

Through interviews, surveys and questionnaires with school nurses, hospital nurses and parents with diabetic children, we worked out the specification [24], based on which we defined the functionalities of the IMAS system. The system:

- receives data from medical devices and sends alarms to the corresponding agents/users when necessary,
- automatically organizes meetings for users, and
- decomposes predefined tasks and delegate sub-tasks to other agents/users.

3.2 Agent Services

Since children are not able to take full care of themselves, their parents are considered as important actors in the daily diabetic care. Various services are defined for the agents to handle. An agent may offer more than one services at the same time.

- Patient Manager (PM). The PM receives data from a sensor physically attached to the patient. It is also responsible for updating the patient record in the database.
- Patient Alarm (PA). The PA sends alarms to other relevant agents in an acute situation, e.g., if the glucose value is outside the allowed interval.
- Alarm Receiver (AR). The AR receives alarms and informs the human users.
- Meeting Manager (MM). The MM proposes a meeting to other agents, awaits their replies and informs them about the result.
- Meeting Responder (MR). The MR receives meeting proposals and reply to the MM. When a meeting is set, the MR also updates the calendar of its user.
- Task Manager (TM). The TM proposes tasks to other agents, awaits their reply and informs them with the final results (success or fail).
- Task Handler (TH). The TH receives task proposals from TM, accomplishes the tasks and reports to the TM.

3.3 Agent Roles

As illustrated in Fig. 2, we defined four kinds of agent roles, namely PatientAgent, ParentAgent, HospitalStaffAgent and SchoolNurseAgent.

- The PatientAgent works as a personal assistant to a patient. It provides its owner with real time glucose monitoring and management. When the glucose level is considered too high or too low, an alarm will be automatically generated and sent to corresponding agents, which continue to forward the alarms to their human users. Patients can also use their PatientAgents to record their daily activities, e.g., food intake, jogging, etc. This information will be provided to the medical staff for diagnosis.
- The ParentAgent works for the parents who have children with diabetes. It provides real time glucose monitoring, making it possible for the parents to access the current glucose status of their children.
- The HospitalStaffAgents work for the medical staff at hospitals. They manage the diabetic healthcare activities, e.g. meeting arrangement, task delegation.
- The SchoolNurseAgent works for the medical staff at school, e.g. a school nurse. The HospitalStaffAgent and the SchoolStaffAgent are functionally very similar.

3.4 Individual Agent Architecture

The design of individual IMAS agent is based on Müller's vertical layered architecture [25]. IMAS agent architecture consists of three components, namely Interface Layer, Agent Control Unit and Knowledge Base. See Fig. 3.

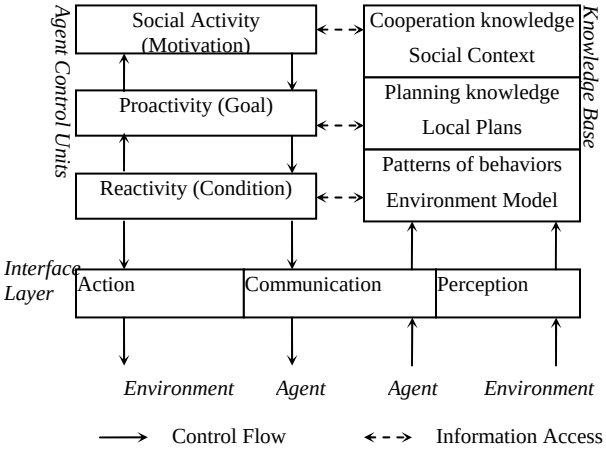


Fig. 3 IMAS Agent Architecture, based on Müller et al [25], consists of three main components, Interface Layer (IL), Control Units (CU), and Knowledge Base (KB).

The Interface Layer (IL) consists of *Perception*, *Action*, and *Communication*. *Perception* percepts the changes from the environment or receives orders from users, and updates the *Environment Model*. *Action* performs the agents' behaviors and *Communication* is the interface to other agents.

The Control Unit (CU) is designed on three levels, *Reactivity*, *Proactivity*, and *Social Activity*, which are connected to the levels *Environment Model* (EM), *Local Plans* (LP), and *Social Context* (SC) in the Knowledge Base (KB). The CU and the KB are interpreted and re-conceptualized with Engström's Activity Model [26]. Therefore, the new architecture, to some extent, manifests human activity aspects in the activities of software agents

The Reactivity component is driven by conditions. That is, when a specified signal is perceived by the Perception that requires action, the Reactivity component will be activated at once. The Proactivity component is responsible for local deliberation process. When perceiving changes in the environment and the EM is updated, the agent will deliberate its goals (desires) and generate plans from the KB to achieve the goals. The Social Activity component is responsible for the interaction among agents. To perform social activities, agents need to learn about the social context from KB.

3.5 Knowledge Base Design

One important aspect of our research is the re-interpretation and re-conceptualization of Müller's vertical layered agent architecture [25]. The vertical layer agent architecture requests that the knowledge base should be structured in order to satisfy two criteria: 1) the knowledge base is better to be structured in similar layers as the control unit, 2) the knowledge base should possess a structure that may cover not only the holistic aspects of collective agent activities, but also detailed plans that may accomplish individual agent tasks. We keep the three level knowledge base structure from Müller's model [25], and interpret it with the Activity Theory triangle model (see Fig. 4).

According to Engström's Activity Model, a human activity is composed of six components, subject (activity transformer), object (activity transferee), tools (activity media), rules (norms), community (relevant stakeholders), and division of labor (roles) [26]. We believe that the activities of software agents, who represent the human owners, can also be analyzed with this model. The agent activities, e.g., reactivity, proactivity, and social activity, can also be defined using the activity triangle model [27].

The agent activity consists of six components:

- Subject agent: this is the agent that conducts the activity.
- Object agent: this is the agent that may benefit from the output of the activity
- Tools: Tools is interpreted as Local Plans of the subject agents.
- Rules: The norms that the agents should follow when conducing this activity.
- Community: Community consists of the context of the activity. For example, what are the relevant agents that are involved in this agent activity?
- Division of Labor (DoL): how the subject agents divide their work.

Fig. 4 illustrates the six components of an agent activity. The six components are correspondingly stored in different levels in the KB according to Müller [25].

The Environment Model/world model is at the lowest level in the KB. Three factors are considered necessary in designing the patterns of agent behaviors, namely rules, division of labor, and community. Rules contains constrains and permissions that the subject agent has. Division of labor contains the information about the agent roles and responsibilities of the activity that the object agent benefits from. Community contains information about the agents and human users who are related to a specific activity. The basic analysis unit of such a database is the activity.

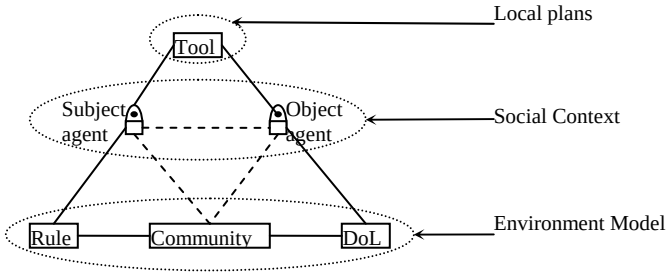


Fig. 4 Activity model used in Knowledge Base design. In a community (MAS or holonic agent groups), the subject agent conducts an agent activity, which the object agent benefits from. Subject agent and object agent are mediated by Tools (local plans); subject agent and community are mediated by Rules (norms); object agent and community are mediated by Division of Labor (agent roles). The mediation is represented by the solid lines. Dashed lines indicate the relations are not direct and need mediations.

The Social Context consists of the contacts and the relationships between the subject agent and the other directly related agents (object agents). That is, how to define the relationship and the coordination between subject agents and object agents.

The Local Plans/Tools consist of planning knowledge that reflects the mental context of the agent. Local Plans store agent plans that are basically series of agent actions. Each plan is associated with pre-condition and post-condition/goal. Agents may choose plans to satisfy their goals at run time.

The Knowledge Base is designed as a recursive hierarchy of Local Plans, Social Context and Environment Model. The design and implementation of the KB require a thorough description of the diabetic healthcare field, which is not in the scope of this paper and will be described in future work.

4. IMPLEMENTATION OF IMAS

So far, the first prototype of IMAS has been developed with functionalities to meet the objectives in the needs specifications. Issues about usability, user interface design, are omitted since this paper focuses on the agent development.

4.1 Patient Control Panel

The patient is provided with a Patient Control Panel (PCP), which is a part of the PatientAgent. It may help the patient to record diabetes related activities, (food intakes, insulin injections, exercises etc.). The recorded activities will be provided together with historical glucose data to the care-providers when an alarm occurs.

The Patient Control Panel has several kinds of user interfaces implemented depending on the platform of the users' devices. It can be an application running on Windows Mobile, or a Java program running on a Java-supported mobile, or an application running on stationary PC. Fig. 5 is an example of the Patient Control Panel running in Windows Mobile 5.

Fig. 5 Patient Control Panel on mobile

4.2 Real-time Glucose Monitoring

Software agents make continuous glucose monitoring possible. In the testing phase, the glucose level is simulated. In practice the glucose level may be measured automatically by a glucose sensor which, for example, is a chip built into the body of the patient. In this case, the glucose level is monitored in real time. Traditionally, the glucose levels are measured four times every day: breakfast, lunch, dinner and bedtime. Fig. 6 shows a situation where the glucose level is too high during a restricted amount of time. The real-time monitoring function is provided to the patient, the parents and the medical staff.

Fig. 6 Real-time glucose monitoring where 8.0 mmol/l and 5.0 mmol/l represent high and low level of glucose.

With real-time monitoring, it is possible to provide alarm functionality to the patients and their care providers. A preferred interval is pre-defined by the patients or their care providers based on the specific situation of the patients. When the glucose level is detected to be dangerous, i.e. outside the preferred interval, an alarm is sent to the corresponding agents and human actors automatically. The receivers of the alarm are chosen based on an analysis of the KB. Since the KB keeps track of the SC and EM, which in turn is based on real world measures, it is quite straightforward to choose which corresponding care-providers that should receive the alarm.

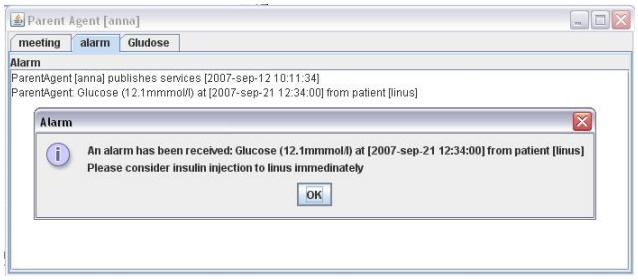


Fig. 7 ParentAgent receives an alarm

4.3 Decision support to diagnosis

When care-providers receive alarms, they are notified that the patient is in an acute health state. Necessary actions must be taken as soon as possible. But first, a detailed check on the health state of the patient is needed. For example, what are the daily average glucose levels during the last week or month, what kind of food has been eaten in the last week or is the insulin injection done properly in time?

In this case, IMAS agents will automatically generate useful diagrams and information for the care-providers, such as daily average glucose levels in the last month, patient activities in the last week etc. Fig. 8 shows the daily average glucose level in the last 30 days.

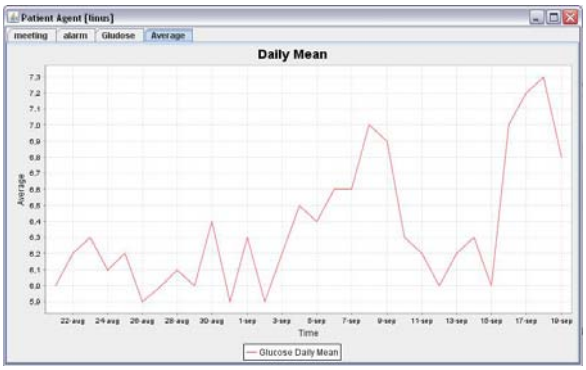


Fig. 8 Daily average glucose level.

4.4 Meeting arrangement

The healthcare actors often need to meet, e.g., routine meeting every three months, or urgent visit when acute situation occur etc. Based on the analysis in the KB, the IMAS agents may propose a meeting to its human actor with details of the meeting participants, length, location, time and so on. The human actor may edit the meeting proposal and ask the agent to send it to the agents that represent the invited meeting participants (see Fig. 9). The responding agents will check the calendars of their human actors and reply with reasonable meeting configurations. We reuse the protocol of Heine et al. [6], to coordinate the meetings.

The meeting proposing functionality is only provided to the medical staff, e.g., school nurses, hospital doctors and nurses, due to the reality that the medical staffs are normally quite occupied during their work. Therefore, it is better for them to start the meeting proposal with their available meeting alternatives. And all agents are afterwards able to respond to the meeting proposals.

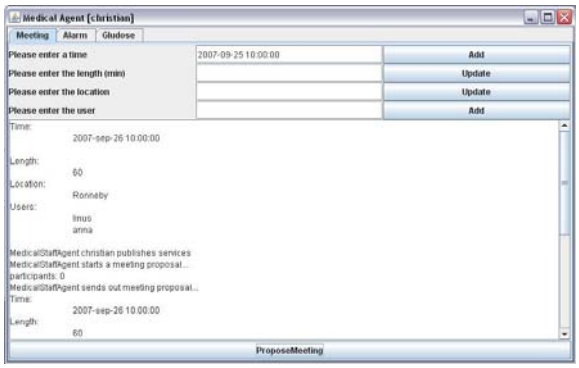


Fig. 9 Meeting proposal

5. DISCUSSIONS

This research has impacts and further improvement on both theoretical MAS development and practical perspectives of children's diabetic healthcare.

Theoretically, the Vertical Layered agent architecture is interpreted and re-conceptualized with the Activity model. This is important when the agents are working in a social and human-involved environment such as the medical healthcare environment is.

The initiative to communicate is not only taken by the human user but also by the software agent, e.g. reporting exceptional measurements of the health state of the patients based on an all day around monitoring. Real-time monitoring combined with user provided activity recording constitute the core of IMAS. By combining new and historical data to appropriate agents, the diabetic health care setting should improve in both efficiency and quality.

Task delegation is another aspect that the IMAS can deal with. Some tasks, e.g., arranging activities like meetings, can be delegated to the agents. It is always a problem when arranging a multi-person meeting; especially when the participants are distributed at different places [6]. With agents provided, such activities can be easily delegated to the agents that will coordinate the meetings using predefined protocols.

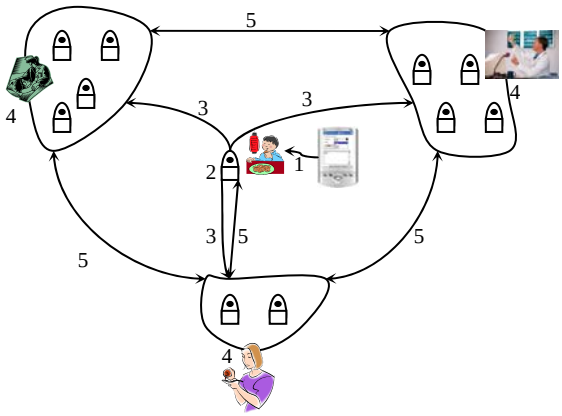


Fig. 10 Information flows in IMAS. (1) Patient Control Panel records activities. (2) Glucose monitoring. (3) Alarm. (4) Decision support to human actors. (5) Meeting arrangement.

Fig. 10 illustrates how the IMAS system helps to practically improve the diabetic healthcare collaboration with its agent based services. The diabetic healthcare collaboration situation that is illustrated by the scenario in Section 2 is believed to be improved by the IMAS system.

1. The glucose records and diabetes relevant information are now stored in a centralized database, which can be accessed by the care providers from any location. The patient now uses a mobile with a Patient Control Panel installed. Whenever there is a new glucose record, the Patient Control Panel will store the new record in a central database. The storing process is done automatically via the connection between the glucometer and the Patient Control Panel. Besides the glucose record, each time the patient eats anything or takes any exercises, he/she will use his/her mobile to report the activity, , which will also be stored in the database. See Fig. 5.

2. When an acute situation happens, PatientAgent will check the glucose record history and the activity record of the patient. Based on the Division of Labor, which is stored in the EM in its KB, PatientAgent chooses corresponding care providers, and sends them alarm messages with appropriate suggestions. (See Fig. 7). Other irrelevant care providers

will not receive this alarm message. Nevertheless, a system log message will be generated and stored, so that all care providers can see what has happened to Linus, if they want to.

When the alarm message is received, the agents of the care providers will search for useful information about the patients and present it with suggestions to the care providers, who need to confirm that the suggestions from the agents are convincible. Nevertheless, the decisions are made by the human users; the agents only provide relevant information to them.

6. CONCLUSION AND FUTURE WORK

The collaboration situation in diabetic healthcare can be improved with IMAS. The IMAS provides a formal communication channel for all human actors. With predefined protocols, much information, such as glucose values, meeting details and task reports, can be automatically transferred to relevant agents. Some unnecessary visits to the medical staff may be avoided, since the software agents can provide intelligent decision support to the patients. This obviously decreases the communication and coordination burden of the diabetic healthcare management and gives the staff the possibility to provide healthcare of higher quality to a lower cost.

In the future, we will continue working with the details of the design of the Control Unit and Knowledge Base. Firstly, the KB, especially the EM, will be further developed to model the important aspects of the reality. Secondly, the stored knowledge should be processed in the Control Unit in a way so that the agent can make more effective decision. Principles from Activity Theory is hoped to be applied in this case.

With the introduction of IMAS, the quality and efficiency of diabetic healthcare can be improved. The IMAS provides diabetic healthcare actors, especially the patients, with glucose monitoring and decision supports in diabetic diagnosis. From care providers' side, part of their jobs can be accomplished by the agents, e.g., arranging meetings, generating, storing, and fetching glucose reports, thus saving time for the staff, at the same time as the quality is improved.

7. REFERENCES

[1] Nealon, J.L. and A. Moreno, Agent-Based Applications in Health Care, in Applications of Software Agent Technology in the Health Care Domain. 2003: Basel, Germany. p. 3-18.

[2] Barro, S., et al., Intelligent telemonitoring of critical-care patients. IEEE Engineering in Medicine and Biology Magazine, 1999. 18(4): p. 80-88.

[3] Ciampolini, A., P. Mello, and S. Storari, Distributed Medical Diagnosis with Abductive Logic Agents. 2002: Bologna, Italy.

[4] Godo, L., et al., A Multi-agent System Approach for Monitoring the Prescription of Restricted Use Antibiotics. Artificial Intelligence in Medicine, 2003. 27: p. 259-282.

[5] López, B., et al. A Multi-agent System to Support Ambulance Coordination in Time-Critical Patient Treatment. in 7th Simposio Argentino de Intelligencia Artificial - ASAI2005. 2005. Rosario.

[6] Heine, C., et al. ADAPT: Adaptive Multi-agent Process Planning & Coordination of Clinical Trials. in AMCIS 2003. 2003. Tampa, Florida.

[7] Camarinha-Matos, L.M. and H. Afsarmanesh, Virtual Communities and Elderly Support. Advances in Automation, Multimedia and Video Systems, and Modern Computer Science, 2001: p. 279-284.

[8] Koutkias, V.G., I. Chouvarda, and N. Maglaveras, A Multiagent System Enhancing Home-Care Health Services for Chronic Disease Management. IEEE Transactions on Information Technology in Biomedicine, 2005. 9(4).

[9] Aldea, A., et al. A Multi-agent System for Organ Transplant Co-ordination. in the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine, LNCS vol 2101. 2001: Springer-Verlag.

[10] Lanzola, G., et al., A Framework for Building Co-operative Software Agents in Medical Applications. Artificial Intelligence in Medicine, 1999. 16(3): p. 223-249.

[11] de Haan, G., O.B. Henkemans, and A. Aluwalia. Personal Assistants for Healthcare Treatment at Home. in the 2005 annual conference on European association of cognitive ergonomics. 2005. Ghania, Greece.

[12] Hernando, M.E., et al. Multi-agent Architecture for the Provision of Intelligent Telemedicine Services in Diabetes Management. in the workshop on Intelligent and Adaptive Systems in Medicine. 2003. Prague, Czech.

[13] Hernando, M.E., et al., Intelligent Alarms Integrated in a Multi-agent Architecture for Diabetes Management. Transactions of the Institute of Measurement and Control, 2004. 26(3): p. 185-200.

[14] Amigoni, F., et al., Anthropic Agency: a Multiagent System for Physiological Processes. Artificial Intelligence in Medicine, 2002. 27: p. 305-334.

[15] Bellazzi, R., et al., A Telemedicine Support for Diabetes Management: the T-IDDM Project. Computer Methods and Programs in Biomedicine, 2002. 69: p. 147-161.

[16] Gómez, E.J., et al., Telemedicine as a Tool for Intensive Management of Diabetes: the DIABTel Experience. Computer Methods and Programs in Biomedicine, 2002. 69: p. 162-177.

[17] Li, M. and R.S.H. Istepanian. 3G Network Oriented Mobile Agents for intelligent Diabetes Management: a conceptual model. in 4th Annual IEEE Conf on Information Technology Applications in Biomedicine. 2003. UK.

[18] Tian, J. and H. Tianfield, A Multi-agent Approach to the Design of an E-medicine Systems. LNAI, 2003. 2831: p. 85-94.

[19] Zhang, P., Multi-agent Systems in Diabetic Health Care, in School of Engineering. 2005, Blekinge Institute of Technology: Ronneby, Sweden.

[20] Wooldridge, M., N.R. Jennings, and D. Kinny, The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems, 2000. 3: p. 285-312.

[21] Luck, M., R. Ashri, and d.I. Mark, Agent-Based Software Development. 2004: Artech House.

[22] Bellifemine, F., G. Caire, and D. Greenwood, Developing Multi-agent Systems with JADE. 2007: Wiley.

[23] FIPA, The Foundation for Intelligent Physical Agents. 2001, see http://www.fipa.org.

[24] IMIS, IMIS User Needs Specification, http://www.ipd.bth.se/imis/IMIS/20060613.IMIS.need.specification.doc. 2006.

[25] Müller, J.P., M. Pischel, and M. Thiel, Modelling Reactive Behavior in Vertically Layered Agent Architectures, in Intelligent Agents: Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence 890, M. Wooldridge and N.R. Jennings, Editors. 1995, Springer: New York.

[26] Engeström, Y., Learning by Expanding. 1987, Helsinki: Orienta-konsultit.

[27] Kuutti, K., Activity Theory as a Potential Framework for Human-computer Interaction Research, in Context and consciousness, Nardi, Editor. 1996, Massachusetts Institute of Technology.

Author Index