Combining Gaia and JADE for Multi-Agent Systems Development

Pavlos Moraitis

And

Nikolaos I. Spanoudakis

Outline

- Main objectives
- Roadmap for MAS development using Gaia and JADE description
- Image project case study
- Conclusions
- Future work

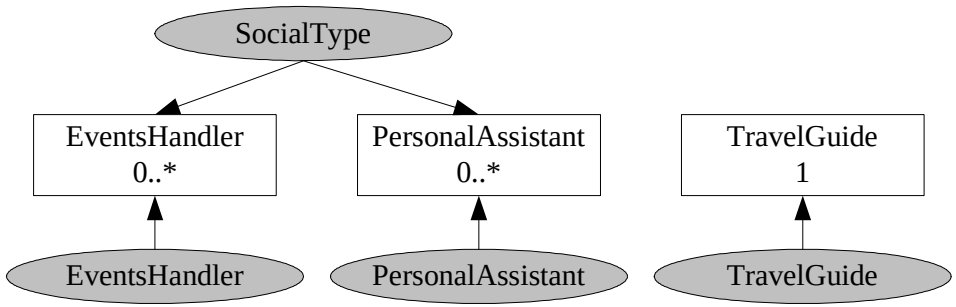
Main Objectives

- Our Work builds upon
  - The Gaia Methodology for Agent-Oriented Analysis and Design
  - Java Agent DEvelopment Framework (JADE)
- Achieving
  - a roadmap for implementing Gaia models using JADE that allows for
  - a complete methodology for MAS development using Gaia for analysis and design and JADE for implementation
- Target audience:
  - AOSE researchers
  - MAS developers

Roadmap Presentation

- First create the Gaia models

Role: SocialType (ST)
Description: It requests agents that perform specific services from the DF. It also gets acquainted with specific agents.
Protocols and Activities: RegisterDF, QueryDF, SaveNewAcquaintance, IntroduceNewAgent,
Permissions: create, read, update acquaintances data structure.
Responsibilities:
Liveness:
SOCIALTYPE = GetAcquainted. (MeetSomeone)$^{\omega}$
GETACQUAINTED = RegisterDF. QueryDF. [IntroduceNewAgent]
MEETSOMEONE = IntroduceNewAgent. SaveNewAcquaintance
Safety: true



IntroduceNewAgent	
ST	ST
A new agent is intantiated	

White and yellow page information of the initiator

Roadmap Step 1

- Define all the ACL messages
  - use the Gaia protocols and interactions model
  - use an ontology editor like the bean generator for the Protege tool in order to graphically design and implement the messages content
  - complete ACL messages definition in a document (performatives, protocols, etc)

Roadmap Step 2

- Define the needed data structures and software modules
  - use the Gaia roles and agent models
  - create the activities refinement table
    - Define algorithms and data structures used by all activities
  - use UML class diagrams in order to define the data structures

Roadmap Step 3

- Decide on the implementation of the safety conditions of each role
  - It might influence all activities of the role
  - Handling a safety condition failure includes two actions:
    - determine the impact on possible running activities and the handling strategy (i.e. reply with a FAILURE message)
    - act in order to restore the safety condition (i.e. message to the administrator)

Roadmap Step 4

- Define the JADE behaviours
  - use the Gaia roles model along with the activities refinement tables and the ACL definitions
  - bottom-up implementation strategy
  - use state diagrams in order to model FSM-like complex behaviours
  - initialize common data structures of more than one behaviours at an upper level behaviour

Roadmap Step 5

- Reuse your code:
  - Use behaviours as components for implementing roles
  - Use roles as components for building agents
- Use agents and the services that they offer as components within larger systems
  - Later you can use the Gaia services model in order to test/verify system functionality

Roadmap Step 6

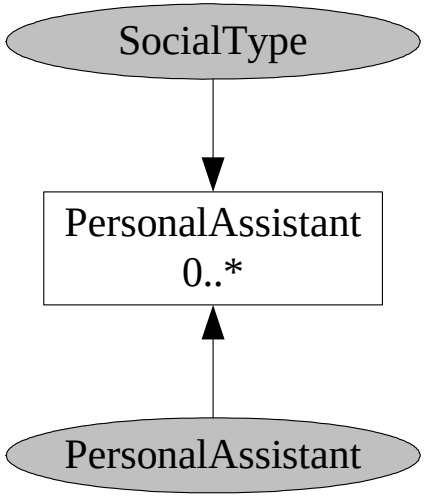
- Finalize agents implementation
  - Initialize all agent level data structures
  - At the setup method of the Agent class add all behaviours, which implement the roles that the agent aggregates, to the agent scheduler

Demonstration

- Bottom-up, modular development approach

Role: SocialType (ST) **Description**: ... **Protocols and Activities**: ... **Permissions**: create, read, update acquaintances data structure. **Responsibilities**: **Liveness**: ... **Safety**: ...	**Role**: PersonalAssistant (PA) **Description**: ... **Protocols and Activities**: ... **Permissions**: ..., read acquaintances data structure. **Responsibilities**: **Liveness**: ... **Safety**: ...

```
public class PersonalAssistantAgent extends Agent {
  //declare agent level data structures
  protected Acquaintances contacts = null;
  protected void setup(){
    ...
    //initialize agent data structures
    contacts = new Acquaintances();
    //activate SocialType and PersonalAssistant behaviours
    addBehaviour(new SocialTypeBehaviour(contacts, ...)
    addBehaviour(new PersonalAssistantBehaviour(contacts, ...));
  }
}
```



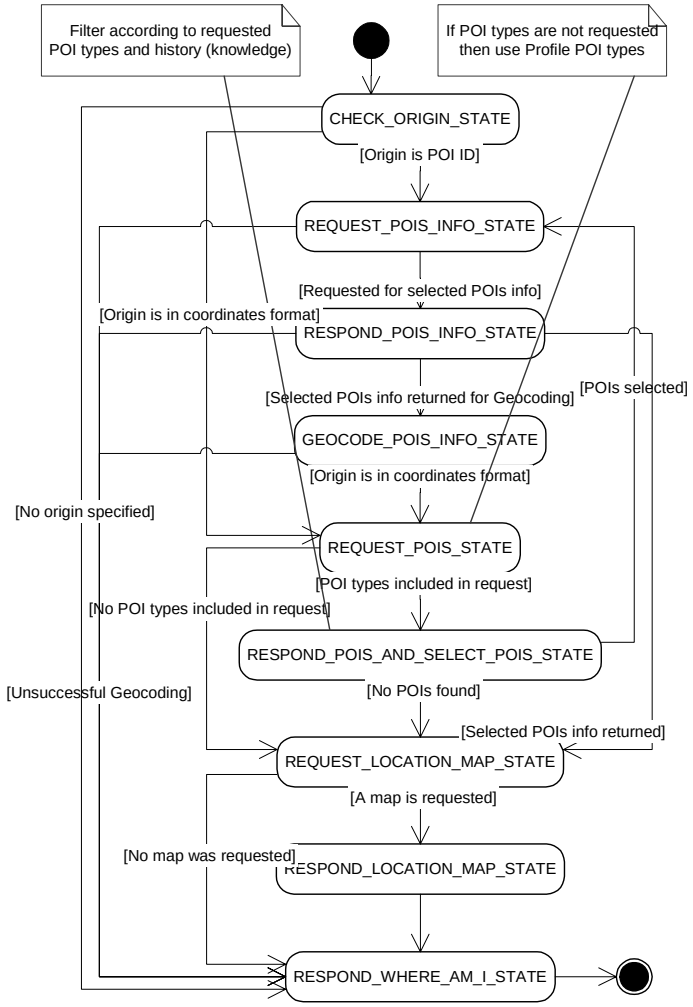
Demonstration (cont)

- We developed complex behaviours using state diagrams that mapped easily to JADE FSMBehaviours

Role: PersonalAssistant (PA)
Description: …
Protocols and Activities: …
Permissions: …
Responsibilities:
  Liveness:
  …
  WHEREAMI = DecideOrigin.
    [GetPOIsInfo] [DecidePOITypes.
    [ProximitySearch. DecidePOIs.
    [GetPOIsInfo. GeocodeRequest]]
    CreateMap] RespondToUser
  …
  Safety: …

Demonstration (cont)

- We developed complex behaviours using state diagrams that mapped easily to JADE FSMBehaviours



Demonstration (cont)

- We developed complex behaviours using state diagrams that mapped easily to JADE FSMBehaviours

Role	Activities	Data Structures		Description
		Read	Update	
PA	DecidePOI Types	user profile user request	-	**if** UserRequest.POITypes.length>0 **then** RequestNearbyPOIs (UserRequest.POITypes) **else if** UserProfile.POITypes.length>0 **then** RequestNearbyPOIs (UserProfile.POITypes) **Else** CreateMap

```
package image.agents;
import jade.core.behaviours.SimpleBehaviour;
import jade.core.Agent;
public class FSMChildBehaviour extends SimpleBehaviour {
  protected boolean finished = false;
  protected int onEndReturnValue;
  public FSMChildBehaviour(Agent a) {
    super(a);
  }
  public void action() {};
  public boolean done() {
    return finished;
  }
  public int onEnd(){
    return onEndReturnValue;
  }
}
```

Demonstration (cont)

- We developed complex behaviours using state diagrams that mapped easily to JADE FSMBehaviours

```
package image.agents.PersonalizedAssistant;

public class PARequestPOIsBehaviour extends FSMChildBehaviour {

  public PARequestPOIsBehaviour(Agent a, Vector selectedPOITypes,
   ACLMessage userRequest, UserProfile userProfile, int fail,
   int pois_selected,  int no_pois_selected) {
    super(a);
    …
  }

  public void action() {
    //if UserRequest.POITypes.length>0
    //Then RequestNearbyPOIs(UserRequest.POITypes)
    //else if UserProfile.POITypes.length>0
    //then RequestNearbyPOIs(UserProfile.POITypes)
    //else CreateMap
  }
}
```

Image project case study

- We successfully implemented a complex system with
  - 7 agent types
  - ~80 behaviours
  - ~900 Kbytes of source code
- in one year
- within a larger project that presumed interaction with 3 other software systems

Image Use Case

Conclusions

- The roadmap
  - can be considered as an agile development process
  - allows for top-down design followed by bottom-up implementation
  - can model and use components in the form of agents and roles/behaviours
- The roadmap was used for MAS development in a successful case study

Future Work

- Use the roadmap for developing MASs in projects
  - IM@GINE IT (IST 508008)
  - NEOGNOS (PRF 39/2002)
- Develop a CASE tool for MAS analysis, design and implementation
- Expand the roadmap to include the full software development life cycle (add the requirements phase)