Reflective Infrastructure for Autonomous Systems

Christopher Landauer
Aerospace Integration Science Center
e-mail: cal@aero.org

Kirstie Bellman
Aerospace Integration Science Center
e-mail: bellman@aero.org

Abstract

Models of emotion and autonomy impose stringent requirements on computing systems. In this paper, we argue that the present state of emotion models and the highest levels of autonomy require complex infrastructure to manage all the internal processes that we believe must be present. We also describe our proposed approach to such an infrastructure.

Our *Wrapping* approach to integration provides a Knowledge-Based reflective infrastructure for complex computing systems, which supports the integration of the many different kinds of processes that are necessary for highly autonomous systems. We argue that reflection and explicit context management are essential for complex adaptive systems, since explicit treatment of context allows the system to organize different kinds of processing differently in otherwise superficially similar environments, and since reflection allows the system to consider its own place in that environment, and reason about its own capabilities.

1 Introduction

What is the connection between complex software architectures and emotions? Most people seem to think emotions differ from cognition, that our hot emotions are different kinds of processes from our cool rational thought. This paper begins with a hypothesis that emotion, intuition, and cognition are different in style and degree, but NOT in kind. The fact that we experience them very differently is a hint about the organization of our own experience, not about difference in their inherent structure.

Our hypothesis means that ALL activity managed by the brain, from movement to language, from emotion to thought, is all defined by layers of symbol systems [Bellman and Goldberg, 1984], organized into a what we have called a "sloppy recursive hierarchy" (that is, some kind of directed graph that is largely but not entirely partially ordered).

This hypothesis comes from an observation about biological organisms: As soon as living things can separate phenomena in their external world that are significant to them from their time and exact detail of occurrence (and even single cells can do this), they are using representational systems (many of them are not explicit, but of course, they must be explicit in our models, even if they are implicit in our artifacts).

Each symbol system is used as a small sort of machine (we do NOT mean Turing machines here: rather, they are automatic physical processes, largely chemical reactions and electrical phenomena at the lower reaches, and we don't exactly know what at the higher reaches). Such a machine has some inputs (physical events that cause symbols to be produced), some outputs (symbols that cause physical effects to be produced), and some kind of processing (the processes that map between these two activities).

These machines are piled on top of each other in great profusion, in a haphazard historical arrangement driven by natural selection, with overlapping domains, frequently incompatible processes, and no particular commonality in their behavior. Our problem is to make computable models of as much of this hierarchy as is needed to build systems that have the behaviors we are intending to produce.

From a different point of view, it is important, as we try to build *synthetic* agents that are intended to be more realistic in situations appropriate for human participation, that their behavior can be recognized as compatible with our expectations. These agents need to be more emotionally believable than the usual robots [Petta et al., 1999; Petta, 1999; Bellman, 1999b], and frequently will need to have a virtual appearance and behavior that is also realistic [Hayes-Roth and van Gent, 1997; Perlin and Goldberg, 1999], especially in large-scale simulations. In groups, their behavior must be appropriate to the problem at hand, and individually, their demeanor must also be appropriate. It is widely expected that proper models of emotional aspects of thought will make this possible.

This paper is mostly about how to implement such a system to examine such hypotheses.

In our work, we've drawn ideas from both biology and software engineering. We use hints from the

amazing diversity of successful biological systems, and continue to develop criteria, partially based on biological ideas, by which we can evaluate proposed architectures for autonomy, and to propose an architectural style that shows great promise [Landauer and Bellman, 1997c; Bellman, 1999b].

We make use of our Wrapping approach to integration infrastructure for the implementation architecture [Landauer, 1990a; Bellman, 1991b], since we have shown that it is extremely flexible and robust. It is inherently computationally reflective, which we think is one of the most important aspects of autonomy control [Landauer and Bellman, 1997f; 1999a; 1999b]. In particular, we take as a guiding principle that there are no "privileged" resources anywhere in such a system: ALL parts of the system can be potentially monitored and reasoned about (especially including, of course, the monitoring and reasoning facilities).

Lastly, we also show how the Problem Posing Interpretation of programming languages [Landauer and Bellman, 1999b] helps us change our expectations for autonomy in computing systems, from some kind of nebulous "intentionality" to a much more reasonable ability to make some decisions for itself, and to know to ask for help when it needs to.

2 Autonomous Systems

In our opinion, a computing system that is given the freedom to solve problems presented to it in unspecified ways is autonomous in many useful ways, but we want more. We want a highly autonomous system to make up the problems also, though we can expect to constrain its behavior somewhat, with some kind of overall goals, policies, or principles (designing appropriate goals correctly is very difficult, as all of the stories of human interactions with genies or devils show).

Over the years since we started studying what we have called *Constructed Complex Systems* [Landauer, 1990a], we have found that it is important to build an infrastructure that allows the system a great deal of flexibility in the utilization of its own resources. To provide this flexibility in turn requires a number of reflective monitoring and reasoning capabilities. In other words, in order to build adaptive complex systems, and to build complex systems that humans can understand and interact with in sophisticated ways, we have backed into building increasingly autonomous systems [Bradshaw, 1997a; Landauer and Bellman, 1999o]. We claim, based on this experience, that an autonomous system must have a reflective architecture [Landauer and Bellman, 1996c].

2.1 Biology

We have previously described several hard-won lessons from biological research about what it means to carry out intelligent functioning within ANY world (be it abstract or physical) [Bellman and Landauer, 1997a; Landauer and Bellman, 1997c; Dautenhahn, 1999a]. These four lessons include:

1. developing "ecological niches" or portable contexts for computational agents;
2. creating artificial "embodiment" for abstract agents;
3. creating entities with "social" behaviors; and
4. developing capabilities for growth and adaptivity of behaviors.

We can strengthen the agents' performances by creating new and stronger information with which to evaluate our ideas of intelligent functionality by agents. We believe that with proper attention to niches, embodiment, social behaviors, and architectures supporting adaptive and reflective behaviors, we can build agents that are more useful partners in an increasingly complex information environment. In order that the system can respond to a wide dynamic range of possible environmental conditions, a very broad range of potential behaviors must be available to the system.

In order to capitalize on the creation of such processes, the agent needs (1) architectures that allow it to pull in new types of processing resources, and (2) self-reflective capabilities. For computational systems, these properties require a very flexible architecture [Hayes-Roth et al., 1995; Landauer and Bellman, 1996c; 1999s].

2.2 Autonomy Criteria

In our opinion, there are really only two classes of (difficult) requirements for effective autonomy: robustness and timeliness. Robustness means graceful degradation in increasingly hostile environments. Timeliness means that situations are recognized "well enough" and "soon enough", and that "good enough" actions are taken "soon enough". Both of these are forms of adaptive behavior, and neither one of these necessarily implies any kind of optimization.

They are both generally difficult properties to achieve, but they do not magically appear, as is too often assumed. They must be *explicitly* designed into a system from the beginning. Controlling this variability is the responsibility of the system infrastructure.

Our approach to constructing autonomous agents is based on theoretical work on organization of language and movement processes [Bellman and Goldberg, 1984], and the structure of Constructed Complex Systems mediated or integrated by software [Landauer and Bellman, 1997c]. Our agents are computer systems in a world of conceptual structures, with models of internal and external phenomena, and processes that operate on those models [Bellman, 1999b].

2.3 Agent Architecture

Building agents is hard, since we don't know enough about the interaction between the computing engines inside and the real world outside (and there is a school of thought that thinks it is the wrong boundary to consider anyway). It is our opinion that in order to perform the experiments required to identify the boundary more explicitly, to determine whether or not it is useful (we believe that it is essential), and to understand how to construct systems that deal adequately

with the crossover of this "map-territory" conceptual boundary, we need to have much more flexibility in the basic agent architectures and environments we use, and much more focussed attention on what we are trying to do with the agents (not the application problem, but the essential components of autonomy).

As we devise and perform these experiments, we should remember that any fixed part of the architecture and any "privileged" component that is so built-in that it cannot be replaced, cannot be studied, so its effect cannot be evaluated. This is why we only consider architectures with no fixed or privileged resources, and why we have developed our particular approach to infrastructure. The next section describes this approach. It is necessarily technical and fairly cryptic, since it describes a long series of developments in a very small space. Readers are encouraged to consult the references for more information.

3 Integration Infrastructure

We have developed a knowledge-based approach to integration infrastructure called "wrappings" [Landauer, 1990a; Landauer and Bellman, 1996c; 1997f], which is based on two key complementary parts: (1) explicit, machine-interpretable information ("meta-knowledge") about all of the uses all of the computational resources in a *Constructed Complex System* (including the user interfaces and other presentation mechanisms); and (2) a set of active integration processes that use that information to Select, Adapt, Assemble, Integrate, and Explain the application of these resources to posed problems [Bellman, 1991b]. We have shown how the approach can be used to combine models, software components, and other computational resources into Constructed Complex Systems, and that it is a suitable implementation approach for complex software agents [Landauer and Bellman, 1997c]. The process begins with our new interpretation of all programming and modeling languages that we have called the *Problem Posing Interpretation* [Landauer and Bellman, 1996c; 1999b].

3.1 Problem Posing Interpretation

All programming languages have a notion of information service, with information service requests and information service providers. Even such a primitiva machine as a Turing machine has this property, with symbols on tape as service requests and the defined reactions as information providers. Side-effects are also information services that can be requested, including object deletion, file manipulation, and real-time measurements or control pulses. In most programming languages since compilers were invented, we connect the requests to the services by using the same names [Landauer and Bellman, 1999b].

"Problem Posing" is a new declarative programming interpretation that unifies all major classes of programming, including functional, imperative, relational, and object-oriented. Programs interpreted in this style "pose problems"; they do not "call functions", "issue commands", "assert constraints", or "send messages" (these are information service requests). Program fragments are "resources" that can be "applied" to problems; they are not "functions", "modules", "clauses", or "objects" that do things (these are information service providers).

The Problem Posing Interpretation separates the requests from the providers. This is easy: compilers and interpreters always know the difference anyway, and so do we when we write the programs. We call the service requests "posed problems". We call the service providers "resources".

We can connect them through Knowledge Bases or by other means, but we have mainly considered the former, the "Knowledge-Based" Polymorphism that maps problems to resources, from the problem specification in its context to the computational resources that will organize the solution.

Any programming or modeling language can be interpreted in this new way.

The Problem Posing Interpretation changes the semantics of programming languages, not the syntax. It turns a program into an organized collection of posed problems, instead of an organized collection of solutions without problems. That should make programs easier to understand, because the problems at all levels of detail remain in it (we believe that part of the difficulty of debugging programs is that they are written as solutions without explicit statements of the corresponding problems).

Problem Posing also allows us to reuse legacy software with no changes at all, at the cost of writing a new compiler that interprets each function call, for example, not as a direct reference to a function name or address, but as a call to a new "Pose Problem" function, with the original function call as the specified problem and problem data. With this change from function calls to posed problems, the entire Wrapping infrastructure can be used. In particular, as the usage conditions for the legacy software change (which they always do), that information can be placed into the problem context, and used to divert the posed problems to new resources written with the new conditions in mind (only the timing characteristics will change, but those changes are frequently completely subsumed by using faster hardware). The gradual transition away from the legacy code is extremely important. Writing such a compiler is a well-understood process, and it is often worthwhile to do so.

The Problem Posing Interpretation radically changes our notion of autonomy, because it eliminates the notion of users "commanding" a system. It replaces that notion with the inclusion of users among the resources that can be used to address a problem. From this viewpoint, the more autonomous agents are merely the ones that need less help in deciding what to do, whether the decision is about choosing high-level goals or lower-level tasks that are expected to address previously determined goals.

3.2 Wrapping

The Wrapping approach to integration infrastructure is particularly well-suited to the Problem Posing In-

terpretation. It not only emphasizes meta-knowledge about the uses of computational resources, together with brokering and mediation of all component interactions (all critical concepts, as seen increasingly in other approaches), but it also regards as equally important the special resources for organizing and processing this information in a flexible and evolvable fashion. These algorithms are called *Problem Managers*, and they are used as a heartbeat to drive any system organized around posed problems.

The Wrapping approach, because it wraps all of its resources, even the active integration processes, results in systems that are Computationally Reflective. That is, a system organized in this way has a machine-processable model of itself; the Wrapping resources and their interactions allow, in essence, a simulation of the entire system to be contained within the system. This allows sophisticated instrumentation and adaptive processing. It is this ability of the system to analyze and modify its own behavior that provides the power and flexibility of resource use. These ideas have proven to be useful, even when implemented and applied in informal, *ad hoc* ways.

The Wrapping theory has four fundamental properties that we regard as essential:

1. Everything in a system is a *resource* that provides an *information service*.
2. Every activity in a system is *problem study* in a particular *problem context*. All activities occur as a resource is *applied* to a *posed problem* in a particular *problem context*). Problems to be studied are separated from the resources that might study them.
3. *Wrapping Knowledge Bases (WKBs)* contain *wrappings*, which are explicit machine-processable descriptions of all of the resources in a system and how they can be applied to problems. Wrappings contain much more than "how" to use a resource. They also include both qualitative and quantitative information to help decide "when" it is appropriate to use it, "why" it might be used, and "whether" it can be used in this current problem and context.
4. *Problem Managers* (PMs) are the active integration processes that organize the use of resources. They are algorithms that use the wrapping descriptions to collect and select resources to apply to problems, using implicit invocation, both context- and problem-dependent. These wrapping processes are also resources; they are also wrapped (Computational Reflection).

The most important conceptual simplifications that the Wrapping approach brings to integration are the uniformities of the first two features: the uniformity of treating everything in the system as resources, and the uniformity of treating everything that happens in the system as a problem study. The most important algorithmic simplification is the reflection provided by treating the PMs as resources themselves: we explicitly make the entire system reflective by considering these programs that process the Wrappings to be resources also, and wrapping them, so that all of our integration support processes apply to themselves, too. It is this ability of the system to analyze its own behavior that provides some of the power and flexibility of resource use, and that we believe is essential for effective autonomy in computing systems.

The key to all of this flexibility is the computational reflection that allows the system to make choices of computational resources at its very core; every resource, including the ones that make the choices, can be chosen, according to the posed problem at hand and the computational context in which the problem is being addressed.

In summary, an infrastructure needs to put pieces together, so it needs the right pieces (resources and models of their behavior), the right information about the pieces (Wrapping Knowledge Bases), and the right mechanisms to use the information (Problem Managers).

3.3 Reflection

We expect any sufficiently complex computing system to be "Computationally Reflective" [Maes, 1987b; Maes and Nardi, 1988], using explicit models of its own behavior to adjust that behavior. To this end, there have to be active coordination processes that use the models to provide the Intelligent User Support functions. They also provide overview and navigation tools, context maintenance functions, monitors, and other explicit infrastructure activities. It is this ability of the system to analyze its own behavior that provides some of the power and flexibility of resource use. The self-model supports explicit software engineering functions from within the same system [Landauer and Bellman, 1996c; 1997f], and provides a basis for self-monitoring, explanation, and failure diagnosis.

Another way to view this property is that the system can make recursive calls in the "meta-direction", i.e., to switch from studying some application problem to studying the problems of its own internal workings.

At least part of the point of reflection is to allow users to extend the interpreters, by adding functionality to them, but there are a couple of significant difficulties with the usual approach. There is no notion of removing functionality when new functions are provided, or of using several alternatives in their respective appropriate contexts. Our separation of the problems posed from the resources applied allows a system to choose different processes, and even different kinds of processes, in different contexts, and keep them all in the same system.

It has become common in certain kinds of agent research to make them partially reflective [Kennedy, 1999], so that some of their monitoring and control components are subject to monitoring, and the rule-firing or other activity patterns can be used as indicators of activity. This is an interesting first step, because it shows some of the problems with trying to build a reflective agent without having a reflective infrastructure to begin with. The reflection provided by the Wrapping infrastructure makes the management

of reflection much easier, by allowing the system designer to think about the reflective properties desired in the problem domain (the objects of the system processing), without also having to consider the reflective properties desired in the implementation (the subject of the system processing).

In the reflective systems now being created as agents, we have an opportunity to go beyond biology in two very important ways: (1) biological systems are not fully reflective – they have many levels which they cannot esxplicitly examine and reason about; (2) they certainly aren't able to swap out at will old processing capabilities, and replace them with new ones. Imagine, if you will, eventually having agents that could simply add new terabyte databases or new effectors and sensors or new processing capabilities, understanding while they did so some of the limitations of their older capabilites and documenting those limitations.

Certainly we are aiming for a very powerful type of reflective capabilities – ones that will help us begin to do the very difficult task of not only creating capabilities but of learning by experience how to use these capabilities within certain contexts. It is this mapping of an agent's sensors and effectors (its embodiment) and goals and capabilities into specific task environments that remains the next major challenge of agent research.

It is important that context be treated explicitly in a reflective language, because it is context that provides the interpretation mappings for the symbols and symbol structures in any language. If we cannot access that context, we cannot examine its assumptions and possibly change them to handle different situations. It is important also that it *can* be handled implicitly in a program, since it seems that in all of our real communication, context is inherited from history and location, and partially inferred from the content of the communication, and not very much appears explicitly in the body of the communication.

In the case of reflective agents, context information is a critical part of what the agent reasons about, e.g., "Do I see what I need to in this environment?"; "How can I change this environment to fit my needs?"; "How can I change myself to fit into this environment?". These questions are relevant for a broad class of agents, ranging from web crawlers to robots. Eventually, most evaluative and feedback processes in agents will come down to reflection about contexts.

Of course, in the case of simple sensors with minimal rulebases and switch-like behavior, this "reflection" may be merely embodied implicitly in its components. However, for more intelligent agents, we will want the ability for the agent to understand that it has a limited viewpoint and range of sensory-motor capabilities, reason about what it needs for the environment in which it finds itself, and perform activities (like moving) to gain additional information. Reflection here is the beginning of a concept of self.

4 Conclusion and Future Prospects

Most of the work we have done so far is essentially creating the infrastructure for powerful future testbeds for autonomy and emotion models, though we also believe that the "layers of symbol systems" hypothesis requires much more infrastructure than is usually used in these studies.

We have identified some of the aspects of dynamical structure that make biological systems so much more robust and flexible than computational systems. We have described in brief an approach to infrastructure that is computationally reflective, and argued that it supplies the flexibility needed to study these aspects of autonomy adequately. It clearly does not solve the hard individual problems about how autonomous behavior is produced and managed, but it does allow us to combine proposed solutions quickly and easily, and to keep alternatives available for comparative experiments.

It is clear that understanding agents requires a wide diversity of expertise to realize even our modest goals for them. To create powerful new agents that will helpfully work with us requires that we humans learn more about working together to create complex systems. Ironically, maybe working on artificial coworkers will help us learn more about the technology needed to support human co-workers. Understanding more about emotions necessarily plays a part in that work.

References

[Bellman, 1991b] Kirstie L. Bellman, "An Approach to Integrating and Creating Flexible Software Environments Supporting the Design of Complex Systems", pp. 1101-1105 in *Proc. 1991 Winter Simulation Conf.*, 8-11 December 1991, Phoenix, Arizona (1991)

[Bellman, 1999b] Kirstie L. Bellman, "Emotions: Meaningful Mappings between an Individual and its World" (invited paper), *Proc. Workshop on Emotions in Humans and Artifacts*, 13-14 August 1999, Vienna, Austria (1999)

[Bellman and Goldberg, 1984] Kirstie L. Bellman and Lou Goldberg, "Common Origin of Linguistic and Movement Abilities", *American J. of Physiology*, Volume 246, pp. R915-R921 (1984)

[Bellman and Landauer, 1997a] Kirstie L. Bellman, Christopher Landauer, "A Note on Improving the Capabilities of Software Agents", pp. 512-513 in [Johnson, 1997]

[Bradshaw, 1997a] Jeffrey M. Bradshaw (ed.), *Software Agents*, AAAI Press (1997)

[Dautenhahn, 1998] Kerstin Dautenhahn, "The art of designing socially intelligent agents: science, fiction, and the human in the loop", *Applied Artificial Intelligence*, Volume 1, No. 7 (1998)

[Dautenhahn, 1999a] Kerstin Dautenhahn, "Socially Situated Life-Like Agents", pp. 191-196 in *Proc. 1999 Virtual Worlds and Simulation Conf.*, 18-20 January, San Francisco (1999)

[Dautenhahn, 1999b] Kerstin Dautenhahn (ed.), *Human Cognition and Social Agent Technology*, Benjamins (1999, expected)

[Hayes-Roth et al., 1995] Barbara Hayes-Roth, Karl Pfleger, Philippe Lalanda, Philippe Morignot, Marka Balabanovic, "A Domain-Specific Software Architecture for Adaptive Intelligent Systems", *IEEE Trans. Software Engineering*, Volume SE-21, No. 4, pp. 288-301 (April 1995)

[Hayes-Roth and van Gent, 1997] Barbara Hayes-Roth, Robert van Gent, "Story-Making with Improvisational Puppets", pp. 1-7 in [Johnson, 1997]

[Hexmoor, 1999a] Henry Hexmoor (ed.), *Proc. Agents'99 Workshop on Autonomy Control Software*, 1 May 1999, Seattle, Washington (1999)

[Johnson, 1997] W. Lewis Johnson (ed.), *Proc. First Int. Conf. on Autonomous Agents*, 5-8 February 1997, Marina Del Rey, California (1997)

[Kennedy, 1999] Catriona M. Kennedy, "Distributed Reflective Architectures for Adjustable Autonomy", in [Kortenkamp et al., 1999c]

[Kortenkamp et al., 1999c] David Kortenkamp, Gregory Dorais, Karen L. Myers (eds.), *Proc. IJCAI-99 Workshop on Adjustable Autonomy Systems*, 1 August 1999, Stockholm, Sweden (1999); available on the World-Wide Web at URL "http: //tommy. nasa. gov /∼korten /ijcai99/" (availability last checked 21 September 1999)

[Landauer, 1990a] Christopher Landauer, "Wrapping Mathematical Tools", pp. 261-266 in *Proc. the 1990 SCS Eastern MultiConference*, 23-26 April 1990, Nashville, Tennessee, Simulation Series, Volume 22, No. 3, SCS (1990); also pp. 415-419 in *Proc. Interface'90: The 22nd Symp. on the Interface (between Computer Science and Statistics)*, 17-19 May 1990, East Lansing, Michigan (1990)

[Landauer and Bellman, 1996c] Christopher Landauer, Kirstie L. Bellman, "Constructed Complex Systems: Issues, Architectures and Wrappings", pp. 233-238 in *Proc. 13th European Meeting on Cybernetics and Systems Research, Symp. Complex Systems Analysis and Design*, 9-12 April 1996, Vienna (April 1996)

[Landauer and Bellman, 1997c] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Constructing Autonomous Software Systems", pp. 42-54 in Judith A. Lombardi (ed.), *Continuing the Conversation: Dialogues in Cybernetics, Volume I, Proc. 1997 ASC Conf.*, Am. Soc. Cybernetics, 8-12 March 1997, U. Illinois (1997); revised and extended version in Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Constructing Autonomous Software Systems", pp. 131-168 in *Cybernetics and Systems*, Volume 30, No. 2 (1999); poster summary in [Bellman and Landauer, 1997a]

[Landauer and Bellman, 1997f] Christopher Landauer, Kirstie L. Bellman, "Wrappings for Software Development", pp. 420-429 in *Proc. 31st Hawaii Conf. on System Sciences, Volume III: Emerging Technologies*, 6-9 January 1998, Kona, Hawaii (1998)

[Landauer and Bellman, 1999a] Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", *Proc. 32nd Hawaii Conf. on System Sciences, Track III: Emerging Technologies, Software Process Improvement Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999); revised and extended version in Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Chapter 8, pp. 108-154 in Gene Mcguire (ed.), *Software Process Improvement*, Idea Group Publishing (1999)

[Landauer and Bellman, 1999b] Christopher Landauer, Kirstie L. Bellman, "Problem Posing Interpretation of Programming Languages", *Proc. 32nd Hawaii Conf. on System Sciences, Track III: Emerging Technologies, Engineering Complex Computing Systems Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999)

[Landauer and Bellman, 1999c] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Agents as Constructed Complex Systems", Chapter 11 in [Dautenhahn, 1999b]

[Landauer and Bellman, 1999o] Christopher Landauer, Kirstie L. Bellman, "Reflective Infrastructure for Autonomous Systems", in [Hexmoor, 1999a]

[Landauer and Bellman, 1999s] Christopher Landauer, Kirstie L. Bellman, "Architectures for Embodied Intelligence", pp. 215-220 in *Proc. 1999 Conf. Artificial Neural Nets and Industrial Engineering, Special Track on Bizarre Systems*, 7-10 November 1999, St. Louis, Mo. (1999)

[Maes, 1987b] Pattie Maes, "Concepts and Experiments in Computational Reflection", pp. 147-155 in *Proc. OOPSLA '87* (1987)

[Maes and Nardi, 1988] P. Maes, D. Nardi (eds.), *Meta-Level Architectures and Reflection, Proc. Workshop on Meta-Level Architectures and Reflection*, 27-30 October 1986, North-Holland (1988)

[Perlin and Goldberg, 1999] Ken Perlin, Athomoas Goldberg, "Improvisational Animation", pp. 9-14 in *Proc. 1999 Virtual Worlds and Simulation Conf.*, 18-20 January, San Francisco (1999)

[Petta, 1999] Paolo Petta, "The Role of Emotions in a Tractable Architecture for Situated Cognizers", (invited paper), *Proc. Workshop on Emotions in Humans and Artifacts*, 13-14 August 1999, Vienna, Austria (1999)

[Petta et al., 1999] Paolo Petta, Carlos-Pinto Ferreia, and Rodrigo Ventura, "Autonomy Control Software: Lessons from the Emotional", in [Hexmoor, 1999a]