Intelligent System Architectures with Wrappings

Christopher Landauer
Aerospace Integration Science Center
The Aerospace Corporation, Mail Stop M6/214
P. O. Box 92957, Los Angeles, California 90009-2957, USA
cal@aero.org, +1 (310) 336-1361

Kirstie L. Bellman
Principal Director, Aerospace Integration Science Center
bellman@aero.org, +1 (310) 336-2191

Abstract

The context of this problem is the System Engineering of Constructed Complex Systems, which are artificially constructed systems that are managed or mediated by computing systems. In particular, we are concerned with autonomous intelligent behavior in such systems, which means that the system takes a major role in selecting its own goals. When Constructed Complex Systems operate autonomously, whether out in the real world or in cyberspace, they need a great deal of flexibility and adaptability in their architecture and implementation. This paper shows how to organize and implement a Constructed Complex System to have the requisite qualities needed for autonomy, while avoiding the most common difficulties found in computing systems: rigidity and brittleness.

Our architecture includes both our Wrapping infrastructure to provide a Computationally Reflective base for all component integration, and our conceptual categories to provide a flexible representation mechanism that separates model structures from the roles they play.

To make things even more interesting, we are currently developing approaches whereby the system also decides for itself when it needs to be re-organized, because its fundamental symbol systems are not expressive enough, and carries out the reorganization automatically, by defining new symbol systems and re-expressing itself in the new terms. This behavior is hard to implement, but we have identified many of the important issues.

There are several fundamental mathematical questions involved in this study: (1) how self-reference can be made not only possible but sensibly computable, (2) how formal mathematical structures can be extended to incorporate more information about context and situation, (3) how to move formal structures into new contexts and assess the resulting validity, (4) how to define mathematical structures before their basic elements are defined, (5) how to capture more of the modeling processes in mathematical structures, (6) how to decide when a notational system is inadequate, and (7) how to fix it.

International Journal of Computing Anticipatory Systems, Volume 12, 2002
Edited by D. M. Dubois, CHAOS, Liège, Belgium, ISSN 1373-5411 ISBN 2-9600262-6-8

Keywords: Anticipatory Systems, Model-Based Adaptation, Computational Reflection, Computational Semiotics, Knowledge-Based Integration Infrastructure.

1 Introduction

It is our opinion that in order to develop computing systems that have any useful properties of intelligence, system designers are going to have to devise much more flexible architectures and much more robust infrastructures than are presently being proposed and developed.

In this paper, we describe two aspects of such systems: the computational infrastructure and the explicit treatment of symbol systems.

We have defined *Constructed Complex Systems* [18] to be large, distributed, heterogeneous, and otherwise complex systems that are mediated or managed by computing systems. They tend to be software and hardware systems that have heterogeneous processing requirements or that have to function in complex environments. We believe that designing, building, and managing such a system requires explicit attention to the infrastructure, including explicit models of the system, its architecture, and the environment in which it is expected to operate [4], and suitably flexible computer-based design support [7] [34].

We have developed a particular approach to coordination among Constructed Complex System components called "Wrapping" [16] [17] [20] [26], which uses explicit meta-knowledge about every computational resource in the system, and processes that use that knowledge to organize and monitor system behavior. In our view, sufficiently complex systems cannot be modeled with just one model [12] [4], so we need to consider model integration as one specific technical area [37].

We use these methods to construct systems that can have a high degree of autonomy, with its need for flexibility in choices of action, and its need for flexible representation systems. We believe that these properties are important for intelligent systems, and are conducting an assortment of experiments in that area.

1.1 Infrastructure

We believe that most of the crippling rigidity in complex software systems is in the interfaces; the messy hierarchies of variable scopes and the use of global variables to circumvent those hierarchies has led to many errors in systems. We note here that because we can separate resource implementations from posed problems, as we describe below, and because most of the language design of problem posing notations is defined by what resources are available in the system, we can avoid many of these problems.

In our approach to what we have called *Integration Science* [9], we have developed some powerful techniques for managing complex software systems [26]. This research in Constructed Complex Systems [5] [7] [34] [19] [20] has shown the importance of

infrastructure, that is, explicit components and activities of the system whose main function is to help organize other parts of the system (either by performing actions on other resources, as script generators, activity monitors, and planners do, or by reasoning about other resources' capabilities and behavior, before, during, and after applying the other resources), to identify and address problems, and to monitor their behavior.

An infrastructure should make tools more helpful about what they do and when it is appropriate to use them. It should also provide what we have called the *Intelligent User Support* (IUS) functions [5]:

- *Selection* (which resources can be applied to a particular problem),
- *Assembly* (how to let them work together),
- *Integration* (when and why they should work together),
- *Adaptation* (how to adjust them to work on certain kinds of problems), and
- *Explanation* (why certain resources were or will be used).

This infrastructure should make communication less a matter of knowing how to combine which resources than of knowing what to ask for (the problem specification), and letting the system work out what resources can be adapted and applied. Our Wrapping approach provides the necessary infrastructure by making the system itself provide help to the user based on explicit information.

2 Wrappings

In this section, we briefly describe our research results about Wrappings. Much more information can be found in the references.

Our Wrapping approach to constructing heterogeneous software and hardware environments is based on two key complementary parts: (1) explicit, machine-processable descriptions of all software, hardware, and other computational resources in a system, and (2) active integration processes that select, adapt, and combine these resources for particular problems.

The Wrapping approach 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 also regards as equally important the special resources for organizing and processing this information in a flexible and evolvable fashion.

The Wrapping approach, because it Wraps all of its resources, even the active integration processes, results in systems that are Computationally Reflective [35] [36] [14] [29]. 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 and ad hoc ways.

2.1 Wrapping Overview

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

1. EVERY part of a system architecture is a *resource* that provides an *information service*, including programs and data, user interfaces, architecture and interconnection models, scripts and analysis tools that refer to other resources, and everything else.

2. EVERY activity in a system is *problem study* in a particular *problem context*, (i.e., all activities occur as *applying* a resource to a *posed problem* in a particular *problem context*), including user interactions, information requests and announcements within the system, service or processing requests, and all other processing behavior. We therefore specifically separate the problem to be studied from the resources that might study it.

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 to support the five IUS functions above. 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" you might want to, and "whether" it can be used in this current problem and context.

4. *Problem Managers (PMs)*, including the *Study Managers(SMs)* and the *Coordination Manager (CM)*, are the active integration processes. They are algorithms that use the Wrapping descriptions to collect and select resources to apply to problems. They use implicit invocation, both context and problem dependent, to choose and organize resources. The PMs are also resources, and they are also Wrapped.

First, every part of the system is a *resource* that provides some kind of *information service*. This includes tools, functions, ordinary files, databases, programs, data, user interfaces, other communication interfaces, interconnection architectures, symbolic formula manipulation systems, scripts that refer to other resources (e.g., plans), and analysis tools that refer to other resources (e.g., parametric study), and everything else (Everything!). We think about *applying resources* instead of "invoking tools" because the resource being applied might not be the active part of that process.

Second, everything that happens in the system is the response to a *posed problem*. Since not all problems can be solved, we think of *studying* problems rather than solving them. Moreover, that allows the system to do more or less undirected explorations as it studies certain kinds of problems, so it can treat some problems as suggestions for study when appropriate, not as strict goals. Our notion of problems deals with context as an explicit part of the problem study process: there must be a problem context before posing a problem even makes sense. Therefore, problem study always occurs after a context is chosen and a problem is posed (we allow these choices to be made either by human users or by other programs as users). Therefore, instead of thinking about "issuing commands" to the system, we think about *posing problems* for the system. Then the Wrapping processes find resources that can deal with the problems by studying them directly or decomposing them into collections of simpler problems. This *Problem Posing* Interpretation of programs and systems allows the Wrapping processes to mediate all problem study using the Wrapping Knowledge Bases. Instead of having direct calls between resources, we have the resources pose problems that correspond to service requests. Other resources announce information services that they provide and the interactions are all mediated through the Wrapping Knowledge Base.

Third, every resource has one or more *Wrappings*, which are explicit machine-processable descriptions of the different ways to use the resource. A Wrapping is not simply an interface "to" a resource; it is an interface to the "use" of a resource. We Wrap "uses" of resources because many important analysis tools in many application domains have grown by accretion over many years, and we gain conceptual simplicity by separating them into different styles of use. There will be separate Wrappings for different common uses of certain complex tools, especially analysis tools that have grown by accretion. Similarly, combinations of resources that often work together may have a single Wrapping for the combination, in addition to separate Wrappings for separate ways to use the resources by themselves.

Fourth, the *Problem Managers* (PMs) are algorithms that use the Wrapping descriptions to collect and select resources to apply to problems. There is a distinguished class of PMs called the *Study Managers* (SMs) that coordinates the basic problem study process, and a specialized PM called the *Coordination Manager* (CM), which is a kind of basic "heartbeat" that drives all of the processing. The SMs mediate between the problem at hand and the Wrappings to select and apply resources to the problem, and the CM cycles between posing problems and using the SM to study them.

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 pro-

grams 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 (Study Manager, Coordination Manager and other Problem Managers).

2.2 Problem Posing Interpretation

In this subsection, we describe a slightly different interpretation of programming languages that greatly facilitates our search for flexibility: the *Problem Posing Interpretation* [23]. In this case, the program does not issue commands, invoke functions, or even send messages; it *poses problems.* This approach extends the application of Wrapping, using the Knowledge-Based way it connects problems (information service requests) with resources (information service providers) that can apply to those problems, all the way down to the expression evaluation level of detail.

We have demonstrated the conceptual utility of "problem posing" in our own descriptions of systems. "Problem posing" is a new declarative programming paradigm that unifies all major classes of programming. Programs written in this style do not "call functions", "issue commands", "assert constraints", or "send messages"; they "pose problems". Program units are not written as "functions", "modules", "clauses", or "objects" that do things; they are written as "resources" that can be applied to problems. The problems are connected to the resources using what we have called *Knowledge-Based Polymorphism*, in which the WKBs are used by the SMs to select resources.

The WKBs that are used by the PMs to connect the problem statements to the resources that can address them also allow the programmers to provide explicit guidance for the PMs as they make those connections. For example, the problem statement need not specify the access or invocation method; that is part of the "assembly" information given with the different resources that might be used to study the problem, and it might be different in different contexts. The resource selection process can use the time or space requirements to help make the selection (as long as that information is provided in the resource use descriptions in the WKBs).

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 High-Autonomy Systems

Our view of autonomy includes not only the simpler version, in which systems can carry out tasks with little or no supervision, but also the more interesting and difficult one, which requires that systems can participate in the creation of their own goals. This kind of open-ended ability requires an enormous amount of knowledge of the current situation, the possibilities for action in that situation, the capabilities of the system itself, the consequences of those actions, and the valuations placed on those actions or their results.

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.

These high-autonomy systems are interesting because they impose many stringent requirements on the architectures that might implement them, flexibility and robustness among the most difficult. These high-autonomy systems are hard to build because it is difficult to make a sufficiently flexible infrastructure.

Our approach to constructing autonomous systems is based on theoretical work on organization of language and movement processes [8], and the structure of Constructed Complex Systems mediated or integrated by software [18].

3.1 Autonomy Requirements

We have described an architecture for autonomous systems that is knowledge-based, computationally reflective [35] [36] [14] [29], and contains many kinds of models of itself and its environment [19] [25] [27]. Several questions about high-autonomy systems arise while studying our approach.

High autonomy systems do not automatically require anticipatory modeling. However, reactive systems are much too slow in a complex environment, so anticipation of environmental effects, and modeling of the potential effects of actions, are necessary to keep up with the pace and variability of the external environment.

High autonomy systems do not automatically require reflective complex architectures However, they do require

- self- and situation-monitoring,
- selection from alternatives that are already in progress,
- heterogeneous-initiative behavior (not just mixed-), and
- continual contemplation for learning.

We think that these are enough reasons to want reflection (if the system knows what it needs to do, knows what it is doing, and knows what it can do, then it can make better choices).

High autonomy systems do not necessarily require knowledge-based architectures. However, biological systems have an enormous and mysterious capability for generative processes to create the variation spaces within which activities are constrained [8] [11], and then from which they are selected ("controlled sources of variation"), and until we have such generative processes in computing systems, we have to replace them with something else. We have chosen to use explicit knowledge bases and interpreters thereof, since we have flexible methods for implementing them, as described in the next section.

High autonomy systems do not necessarily require explicit models. However, they do require models that they can develop, adjust, and interpret, and having an explicit modeling notation to which the system has access is an important part of being able to analyze the models.

3.2 Model Interactions

There are several different kinds of models:

- capability (this is the self-model from the Wrappings),
- empirical,
- inferential,

- exploratory, and
- anticipatory.

We describe these models in this subsection.

There are also models of the space of possible behaviors, that is, the trajectories that the system may traverse, and its actions and the corresponding transitions (some of the actions of the system will cause changes in the environment in response). These can be organized into interactive game strategies, in which the system can consider the alternation of its own actions and those of the environment (though of course, most complex environments are not strictly taking turns). These models can also be organized by time, both actual and potential, and also organized in several other ways.

In order for the system to have a rich enough set of models to consider, we expect the system to build many of its own models. The system builds new models in two ways: empirical and inferential. The empirical models are the data-driven observations, with many measurement processes and induction as the main component, and the inferential models are the causality hypotheses. The system also needs to construct Wrappings for the new models, which in general is very difficult, but is relatively simple in this case: each of these model construction methods is based on a hypothesis about the input, and the model can be used in reasoning about that input. In particular, the system will evaluate the models according to how well they help the system predict its environment.

This model development starts with observations of the environment and of itself. The Computational Reflection provided by Wrappings allows model analysis and improvement to include all of these kinds of models.

Empirical models use induction to construct descriptions of the observations. There are many approaches to inductive inference [1], and which ones to use will depend on the application area, but there will at least be some common sequence recognition algorithms, and some partial parsing algorithms.

Inferential models use abduction to construct explanations of the observations and empirical models. The relationship between the inferential models and the phenomena they explain is exactly the same as the relationship between mechanism specifications and service specifications in communication protocol design: since we cannot expect the system to be able to invent all the implementations for its observed behaviors, we settle for trying a few well-known styles of implementation that are specific to the application domain.

Both the empirical and inferential models are reactive, but they produce models that can be used for prediction. the anticipatory models are simulations (see the next subsection).

All of these models are constructed in model spaces, and the model spaces in our ideal system are often also constructed (though some are provided). The specific inductive and abductive methods use spaces that are constructed in advance,

according to which method is used (each of these methods specifies that its models are built in some well-defined space).

There are also exploratory models, processes that use evolutionary or adaptive programming (using our semantically neutral Wrapping expression notation *wrex* for its meta-programming capabilities [22] [23]) to develop more precise or more accurate models of the observations, the explanations, and the consequences of activity. Some of these more exploratory methods also build a model space, in addition to the model they build within that space.

4 Symbol Systems

In this section, we turn to the symbol systems that underlie all explicit models in the autonomous systems we consider. All behavior in the system is expressed in the programs that define the set of resources available for selection by the CM. In the simplest case, they are all expressed in *wrex*, but there are often models and other resources that are not explicit, and therefore not directly analyzable.

We are looking for criteria that can be used by a Constructed Complex System to decide when its own use of its own fundamental symbol systems has become inadequate, and for methods by which such a system can use the deficiency criteria to help it develop new symbol systems that do not have the detected deficiencies (without getting into an indefinite or cyclic series of changes that all have different deficiencies).

We start by mentioning two theorems that seem to be well-known in the folklore, though we have not seen any proofs of them before [21]. We called them "Get Stuck" theorems, since they assert that any finite system of symbols and symbol grouping constructs would eventually get stuck unable to express more complex relationships) in a certain way. The relevance here is that it directly supports the assertion that the ability to create new units and new symbol grouping methods (that is, new symbol systems for representation), and the corresponding ability to re-express all of its own behavior using the new units and the new constructs, is essential for certain kinds of systems (in particular, the ones that we want to call intelligent). This notion of systems that can change their own symbol systems is something new in both Computing and Mathematics. Since, after all, any formal logical system depends for its proofs on a fixed set of symbols, it follows that when we can change the symbols, we obviate the proofs, and sometimes invalidate the theorems. In particular, Gödel's Theorem only applies to logics with fixed symbol systems (this is not to say that it cannot be proved for certain logical systems with variable symbol systems, only that it has not yet been proved).

Making systems that can change their own symbols seems to be a hard problem, but we think it is possible to solve it. The advantage of our system design for working on this problem is that it already has a system infrastructure that is computationally reflective (our "Wrapping" approach to integration described above),

which means that our Constructed Complex Systems have access to a model of their own capabilities and behavior that they can monitor and analyze.

So there are two hard parts to this problem: how to detect inadequacy in a symbol system, and how to invent new symbols (actually, the invention of new names for the new symbols is easy; it is deciding on the appropriate meanings that is hard).

For the first problem, performance assessment of a symbol system, there must be a definition of what the performance of a symbol system is, and criteria for the adequacy of that performance. The symbol system is used to express posed problems, find appropriate resources to address those problems, and apply the resources. Adequacy criteria therefore fall into several classes: (1) ease of identification (how easy it is to produce the symbol structures that correspond to a particular situation or event), (2) expressive coverage (how much of the situation can be expressed), (3) expressive power (how small the resulting structures are), and (4) ease of interpretation (how easy it is to use the structures for further processing). Note that classes (3) and (4) are more easily handled because they occur entirely within the system, and so can be controlled or at least observed and influenced by the system.

Classes (1) and (2) require a way to describe situations that may be outside the system (all of this evaluation also applies to events and situations that occur inside the system, but as before, those ones are easier to deal with), and that therefore have no explicit representation inside the system, except for the one provided by the existing symbol system. So in order to make the comparison, there must be some basic interactions between the system and its environment that are not covered by the symbol system. For both Virtual Worlds (VW) and Real-Life (RL) environments, they can be taken to be interaction items. For RL, these are usually energy transfers of some kind, and for VW, they are text structures. The RL version of the interaction is called the symbol-grounding problem, and is in general much harder. Even for VW, however, where the discrete nature of the world provides a lower bound on required resolution, there are interesting problems because we will eventually want to share those virtual environments with our computational agents, which we expect will be implemented as this kind of Constructed Complex System [6] [24].

For the second problem, that is, to decide what changes to make, the performance measurements of the symbol system provide hints about what must be changed and how to do so. When situations are not expressible, extensions must be made, which means addition of new symbol definitions (without removing the old ones). When situations are conflated that should not be (similarly described events or situations have different effects or consequences), the identification context must be extended to improve the discrimination between the situations. When the constraints among a set of symbols become sufficiently complex, the set of interconnected symbols needs to be re-expressed. The constraints guide the selection of new symbols, and show how the old ones are partitioned among the new ones.

We expect that there are other important processes here, but that these ones are sufficient for a good start.

5 Conclusions

This paper describes a research program in Integration Science [10], using autonomous and intelligent systems as a driving application [19] [15] [32]. It describes our approach to Constructed Complex Systems, using Wrappings, and our insistence that these systems manage their own representational mechanisms [28] [32].

There are several fundamental mathematical questions involved in this study: (1) how self-reference can be made not only possible but sensibly computable [2] [3], which we have emphasized by using the Computational Reflection of Wrappings, and our engineering notions of depth of self-modeling [26], (2) how formal mathematical structures can be extended to incorporate more information about context and situation, which we describe in detail elsewhere [32] (3) how to move formal structures into new contexts and assess the resulting validity [30], (4) how to define mathematical structures before their basic elements are defined [32], (5) how to capture more of the modeling processes in mathematical structures, (6) how to decide when a notational system is inadequate, and (7) how to fix it.

References

[1] Dana Angluin, Carl H. Smith, "Inductive Inference: Theory and Methods", *Computing Surveys*, Volume 15, Number 3, pp. 237-269 (September 1983)

[2] Jon Barwise, John Etchemendy, *The Liar: An Essay on Truth and Circularity*, Oxford U. (1987)

[3] Jon Barwise, Lawrence Moss, *Vicious Circles*, CSLI Lecture Notes No. 60, Center for the Study of Language and Information, Stanford U. (1996)

[4] Kirstie L. Bellman, "The Modelling Issues Inherent in Testing and Evaluating Knowledge-based Systems", pp. 199-215 in Chris Culbert (ed.), *Special Issue: Verification and Validation of Knowledge Based Systems, Expert Systems With Applications Journal*, Volume 1, No. 3 (1990)

[5] Kirstie L. Bellman, "An Approach to Integrating and Creating Flexible Software Environments Supporting the Design of Complex Systems", pp. 1101-1105 in *Proceedings of WSC '91: The 1991 Winter Simulation Conference*, 8-11 December 1991, Phoenix, Arizona (1991); revised version in Kirstie L. Bellman, Christopher Landauer, "Flexible Software Environments Supporting the Design of Complex Systems", *Proceedings of the Artificial Intelligence in Logistics Meeting*, 8-10 March 1993, Williamsburg, Va., American Defense Preparedness Association (1993)

[6] Kirstie L. Bellman, "Sharing Work, Experience, Interpretation, and maybe even Meanings Between Natural and Artificial Agents" (invited paper), pp. 4127-4132 (Vol. 5) in *Proceedings of SMC'97: the 1997 IEEE International Conference on Systems, Man, and Cybernetics*, 12-15 October 1997, Orlando, Florida (1997)

[7] K. Bellman, A. Gillam, and C. Landauer, "Challenges for Conceptual Design Environments: The VEHICLES Experience", *Revue Intern. de CFAO et d'Infographie*, Hermes, Paris (September 1993)

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

[9] Kirstie L. Bellman, Christopher Landauer, "Integration Science is More Than Putting Pieces Together", in *Proceedings of the 2000 IEEE Aerospace Conference (CD)*, 18-25 March 2000, Big Sky, Montana, IEEE Press (2000)

[10] Kirstie L. Bellman, Christopher Landauer, "Towards an Integration Science: The Influence of Richard Bellman on our Research", *Journal of Mathematical Analysis and Applications*, Volume 249, Number 1, pp. 3-31 (2000)

[11] K. L. Bellman and D. O. Walter, "Biological Processing", *American Journal of Physiology*, Volume 246, pp. R860-R867 (1984)

[12] Richard Bellman, P. Brock, "On the concepts of a problem and problem-solving", *American Mathematical Monthly*, Volume 67, pp. 119-134 (1960)

[13] Les Gasser, Michael N. Huhns (eds.), *Distributed Artificial Intelligence*, Volume II, Morgan Kaufmann (1989)

[14] Gregor Kiczales, Jim des Rivières, Daniel G. Bobrow, *The Art of the Meta-Object Protocol*, MIT Press (1991)

[15] Christopher Landauer, "Some Measurable Characteristics of Intelligence", Paper WP 1.7.5, *Proceedings of SMC'2000: The 2000 IEEE International Conference on Systems, Man, and Cybernetics (CD)*, 8-11 October 2000, Nashville Tennessee (2000)

[16] Christopher Landauer, Kirstie L. Bellman, "The Organization and Active Processing of Meta-Knowledge for Large-Scale Dynamic Integration", pp. 149-160 in *Proceedings 10th IEEE International Symposium on Intelligent Control, Workshop on Architectures for Semiotic Modeling and Situation Analysis in Large Complex Systems*, 27-30 August 1995, Monterey (August 1995)

[17] Christopher Landauer, Kirstie L. Bellman, "Integration Systems and Interaction Spaces", pp. 161-178 in *Proceedings of FroCoS'96: The First International Workshop on Frontiers of Combining Systems*, 26-29 March 1995, Munich, Germany (March 1996)

[18] Christopher Landauer, Kirstie L. Bellman, "Constructed Complex Systems: Issues, Architectures and Wrappings", pp. 233-238 in *Proceedings EMCSR 96: Thirteenth European Meeting on Cybernetics and Systems Research, Symposium on Complex Systems Analysis and Design*, 9-12 April 1996, Vienna (April 1996)

[19] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Constructing Autonomous Software Systems", pp. 131-168 in *Cybernetics and Systems: An International Journal*, Volume 30, Number 2 (1999)

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

[21] Christopher Landauer, Kirstie L. Bellman, "Situation Assessment via Computational Semiotics", pp. 712-717 in *Proceedings ISAS'98: the 1998 International MultiDisciplinary Conference on Intelligent Systems and Semiotics*, 14-17 September 1998, NIST, Gaithersburg, Maryland (1998)

[22] Christopher Landauer, Kirstie L. Bellman, "Generic Programming, Partial Evaluation, and a New Programming Paradigm", Paper etspi02 in *Proceedings of HICSS'99: The 32nd Hawaii Conference on System Sciences (CD), 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)

[23] Christopher Landauer, Kirstie L. Bellman, "Problem Posing Interpretation of Programming Languages", Paper etecc07 in *Proceedings of HICSS'99: the 32nd Hawaii Conference on System Sciences, Track III: Emerging Technologies, Engineering Complex Computing Systems Mini-Track*, 5-8 January 1999, Maui, Hawaii (1999)

[24] Christopher Landauer, Kirstie L. Bellman, "Computational Embodiment: Agents as Constructed Complex Systems", Chapter 11, pp. 301-322 in Kerstin Dautenhahn (ed.), *Human Cognition and Social Agent Technology*, Benjamins (2000)

[25] Christopher Landauer, Kirstie L. Bellman, "New Architectures for Constructed Complex Systems", in *The 7th Bellman Continuum, International Workshop on Computation, Optimization and Control*, 24-25 May 1999, Santa Fe, NM (1999); in *Applied Mathematics and Computation*, Volume 120, pp. 149-163 (May 2001)

[26] Christopher Landauer, Kirstie L. Bellman, "Lessons Learned with Wrapping Systems", pp. 132-142 in *Proceedings of ICECCS'99: The 5th IEEE Interna-*

tional Conference on Engineering Complex Computing Systems, 18-22 October 1999, Las Vegas, Nevada (1999)

[27] Christopher Landauer, Kirstie L. Bellman, "Architectures for Embodied Intelligence", pp. 215-220 in *Proceedings of ANNIE'99: 1999 Artificial Neural Nets and Industrial Engineering, Special Track on Bizarre Systems*, 7-10 November 1999, St. Louis, Mo. (1999)

[28] Christopher Landauer, Kirstie L. Bellman, "Symbol Systems in Constructed Complex Systems", pp. 191-197 in *Proceedings of ISIC/ISAS'99: The 1999 IEEE International Symposium on Intelligent Control*, 15-17 September 1999, Cambridge, Massachusetts (1999)

[29] Christopher Landauer, Kirstie L. Bellman, "Reflective Infrastructure for Autonomous Systems", pp. 671-676, Volume 2 in *Proceedings of EMCSR'2000: The 15th European Meeting on Cybernetics and Systems Research, Symposium on Autonomy Control: Lessons from the Emotional*, 25-28 April 2000, Vienna (April 2000)

[30] Christopher Landauer, Kirstie L. Bellman, "Can Formal Mathematics Model Non-Formal Phenomena?", Abstract 171-3 in *Proceedings IMACS'2000: The 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation (CD), Invited Session on Bioinformatics*, 21-25 August 2000, Lausanne (August 2000)

[31] Christopher Landauer, Kirstie L. Bellman, "Symbol Systems and Meanings in Virtual Worlds", *Proceedings of VWsim'01: The 2001 Virtual Worlds and Simulation Conference, WMC'2001: The 2001 SCS Western MultiConference*, 7-11 January 2001, Phoenix, SCS (2001)

[32] Christopher Landauer, Kirstie L. Bellman, "Conceptual Modeling Systems: Active Knowledge Processes in Conceptual Categories", pp. 131-144 in Guy W. Mineau (Ed.), *Conceptual Structures: Extracting and Representing Semantics, Contributions to ICCS'2001: The 9th International Conference on Conceptual Structures*, 30 July-03 August 2001, Stanford University (August 2001)

[33] Christopher Landauer, Kirstie L. Bellman, "Computational Infrastructure for Experiments in Cognitive Leverage", in *Proceedings of CT'2001: The Fourth International Conference on Cognitive Technology: Instruments of Mind*, 6-9 August 2001, Warwick, U.K. (2001)

[34] Christopher Landauer, Kirstie L. Bellman, April Gillam, "Software Infrastructure for System Engineering Support", *Proceedings AAAI '93 Workshop on AI for Software Engineering*, 12 July 1993, Washington, D.C. (1993)

[35] Pattie Maes, "Computational Reflection", technical report 87-2, Vrije Universiteit Brussel, Artificial Intelligence Laboratory (1987)

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

[37] Donald O. Walter, Kirstie L. Bellman, "Some Issues in Model Integration", pp. 249-254 in *Proceedings of the SCS Eastern MultiConference*, 23-26 April 1990, Nashville, Tennessee, Simulation Series, Volume 22(3), SCS (1990)