A FRAMEWORK FOR A.I. INTEGRATION

Kristinn R. Thórisson, Thor List
Christopher C. Pennock, John DiPirro

RUTR-CS05002

2005

Reykjavik University – Department of Computer Science

Technical Report

A FRAMEWORK FOR A.I. INTEGRATION

Kristinn R. Thórisson
Center for Analysis & Design of Intelligent Agents
Reykjavik University, Iceland

Thor List
Institute for Perception, Action & Behaviour
Edinburgh University, U.K.

Christopher C. Pennock
Media Research Laboratory
New York University, U.S.A.

John DiPirro
Communicative Machines
Edinburgh, U.K.

INTRODUCTION

A number of present-day problems act to hold back progress in the field of artificial intelligence (A.I.), both theoretical and pragmatic. Among the most serious pragmatic issues has to do with integration and large-scale systems construction, as much recent work on humanoids and interactive robots has shown (cf. Thórisson 2005a, Pagello et al. 2004, Simmons et al. 2003, Maxwell et al. 2001, Bischoff et al. 2000, Martinho et al. 2000, Fink et al. 1996, 1995). Numerous barriers must be faced by any researcher wanting to reuse systems developed by others in the creation of large integrated A.I. systems. The barriers are caused for example by the use of different programming languages, by different operating assumptions, and by lack of computing power. These barriers prevent integration of isolated efforts and thus the creation of larger, more capable A.I. systems.

This paper presents a framework addressing many of the pragmatic issues the field has to deal with and intended to significantly speed up research advances in A.I. The framework addresses three main issues: Building large systems, connecting heterogeneous software components and collaborating effectively. The first prong in our four-prong approach to these issues is a methodology for creating and managing large, modular systems. The second

prong is an API for facilitating integration of code developed by different authors at different times. Our third component is a software platform that uses the API and makes it significantly easier to integrate software written in different programming languages and create single systems running on many different computers. Finally, a fourth component is a forum for code exchange, software integration and collaboration.

The motivations for our four-pronged approach are numerous. One of the main challenges of A.I. is complexity management. The complexity is quite different from that encountered in e.g. computer graphics, where a small set of basic principles applied to a somewhat larger number of object types results in well-understood and predictable behavior, enabling the power of graphics systems to grow at roughly the same rate as the hardware. Not so in artificial intelligence, where the complexity stems from the need to coordinate a large number of functions at multiple levels of detail, to serve a compound set of high- and low-level goals.

To understand intelligence fully a study of every aspect of intelligent behavior needs to be studied. Our interest is therefore in the full spectrum of intelligent processes, from planning to perception, animation to emotion, operating autonomously. Included in the list is realtime operation in real environments, such as cognitive robotics and communicative virtual humanoids.

One of the main assumptions behind this work is that a human mind, or at least significant portions of it, can be modeled through the adequate combination of interacting, functional machines, or modules. The inspiration for our approach comes from several sources, two of which reflect the views of important contributions to the field. First, we agree with Minsky's proposal (1986) that a mind is composed of a multitude of heterogeneous, interacting components. Our methodology and software foundation directly supports the construction of such systems. Second, Brooks (1989) has argued that a focus on isolated cognitive simulators, instead of whole cognitive beings, has been leading the field astray. We agree with this view and would add that not much has changed since the argument was made.

The motivations for our work go further. There is a serious lack of incremental accumulation of knowledge in A.I. and related computer graphics. By supporting re-use of prior work we hope to enable the building of increasingly powerful systems, as core system elements do not need to be built from scratch.

This is not an easy challenge. Much software has been written in the last two decades for handling cognitive, sensory, and motor tasks, but these are typically implemented in isolation, in various programming languages, and with a wide range of background assumptions about the operating environment. There is thus a pragmatic kind of complexity with which the A.I. practitioner must cope: The broad set of skills required to create and/or configure the necessary systems for processing input and output in these areas. The problem is exacerbated in tasks such as that of building e.g. large virtual worlds with simulated inhabitants. Such systems cannot be built – from scratch or from off-the-shelf components – without bringing together a large team of experts in each of the fields that such a system naturally

encompasses. Our methodology aims to help coordinate such efforts.

As mentioned above, we take a four-pronged approach to A.I. integration. The paper is organized as follows: We will first discuss related research, categorized into the four main threads described above. Then we will present, in order, the Constructionist Design Methodology, the routing and message specification, called OpenAIR, the software development platform, called Psyclone. Following this we will describe two systems that were constructed using these elements. Finally we describe the MINDMAKERS.ORG effort.

BACKGROUND & RELATED WORK

Related research could be grouped in several different ways: Efforts to promote collaboration and interaction between researchers, systems to facilitate and manage integration of components, message and routing protocols, and design methodologies. Rather than discussing each of them separately we will discuss them in combination.

One of the major factors separating prior efforts to address communication among modular systems is selection of communication model, which traditionally has either been of the object-oriented type or of the agent-based type.

CORBA[1] is a relatively general technology that allows transparent communication between programs running on multiple computers that are written in different languages. CORBA takes the object-oriented approach: An object makes a request for a service or for information, and this request is brokered by a central server, simulating an extended function call. This general mechanism works well for systems that can assume a larger temporal granularity than the network can provide. In real-time systems, however, this assumption is both simplistic and insufficient. An extension to CORBA, Real-time CORBA[2], is meant to address this shortcoming in the original design. However, because CORBA and other object-oriented approaches (e.g. DCOM (Microsoft 1998)[3] try to make the whole system behave like one big computer program, it becomes cumbersome to deploy and debug, in many cases, and impossible to deploy in some cases, especially in systems where real-time performance is paramount.

The alternative to the object-oriented approach is the agent-based approach, which relies on message-based routing. Several notable projects have been started in recent years focusing on this approach, among them KQML[4], and the more recent efforts related to grid computing[5]. KQML (Knowledge Query and Markup Language) was an initiative that predates most of the current work in this area and provided a framework for talking about message-based machine communication that was modeled after natural language performatives (Searle 1969). While providing a boost for the subsequent

[1] http://www.corba.org/
[2] http://www.cs.wustl.edu/~schmidt/TAO.html
[3] http://www.microsoft.com/com/wpaper/default.asp#DCOMpapers
[4] http://www.cs.umbc.edu/kqml/
[5] See for instance http://www.naradabrokering.org/

semantic Web effort[6], it suffered from a semantic-pragmatic confusion: The "envelope" representation of messages and the surface representation of its content was not sufficiently separated. This has been addressed in some subsequent efforts.

Narada[7] is an example of a system that has solved numerous problems regarding message-based routing, including communication through firewalls. However, Narada has only been implemented in Java, and a practical problem with Java is that many real-time applications may require a native C/C++ implementation. So instead of being pure Java the system loses a lot of its platform independence while at the same time possibly running more slowly than a clean native implementation in C/C++ would. Speed is just one reason why someone might want to use one programming language over another: Any feature of a particular programming language might be a reason for an A.I. practitioner choosing it over another language, which means that the solution must be cross-language. A requirement of the present work is that the message services be light-weight and result in relatively easy-to-use adapters for every relevant programming language.

A system as big as Narada[8] is in many ways unwieldy; with a goal of solving a fairly large set of design problems the footprint becomes prohibitively large for a number of uses, for example, it depends not only on Xerces[9] and Xalan[10], both which are fairly large libraries, but about 10 other external libraries. This makes it difficult to port the system to other programming languages, and to deploy it on platforms with restricted memory sizes. A related problem, which it shares by CORBA, is that it is not simple to set up or use.

The emphasis on realtime in the present work, mixed data types and support for both "push" and "pull" data transfer makes many of the prior systems insufficient. CORBA, for example, only provides a "pull" mechanism. Blackboards (Adler 1989, Dodhiawala 1989, Selfridge 1959) are a general way of addressing some of these issues. Thórisson (1998, 1997) used blackboards with a (non-dynamic) publish-subscribe routing protocol to enable inter-module communication: Modules posted data to a central blackboard server; that data was in turn delivered to subscribed modules. Modules could also retrieve old messages from the blackboards.

In addition to the above mentioned features, however, a platform for interactive A.I. systems must have temporal accountability, i.e. the system must be able to track of, and make available to elements in the architecture, the timing of events and delays in the system. Except for temporal accountability, systems such as Elvin[11] and OAA[12] (Martin et al. 1999) have to a smaller or larger extent addressed the above requirements. Elvin is a content-based router with a central routing station. The system has been used

[6] http://www.w3.org/2001/sw/
[7] http://www.naradabrokering.org/
[8] The full system counts more than 1200 classes and subclasses organized into over 130 packages.
[9] http://xml.apache.org/xerces2-j/
[10] http://xml.apache.org/xalan-j/
[11] http://elvin.dstc.edu.au/
[12] http://www.ai.sri.com/~oaa/

in some systems with good results (Johnson et al. 2004), showing that the publish-subscribe approach is a powerful way to construct modular A.I. systems. The OOA is a hybrid architecture that relies on a special inter-agent communication language (ICL) – a logic-based declarative language which is good for expressing high-level, complex tasks and natural language expressions. While this is precisely what is needed for many A.I. applications, it requires a special-purpose parser, and makes it necessary for all agents in the system to contain this parser, which makes it harder to integrate heterogeneous components to create a single system. This shortcoming is also shared with another similar effort from FIPA[13]: The FIPA message and routing specification uses a special syntax for messages, requiring it to use non-standard parsers. A more general way of representing the outermost "envelope" of a routed message would be to use XML. This achieves a higher level of generality in the outermost layer while allowing the content of such messages to be represented in any applicable language, including ICL.

While all of the above approaches have pros and cons, and many may come close to being usable as a basic foundation for integration in cognitive robotics and interactive applications, the best ones still fall short because time is by and large an ignored problem: Temporal information is only managed within the agents or processing nodes themselves, but not in the transmission infrastructure. This means that a receiver of a message cannot know now long ago the message was posted and thus how long ago its content was collected. Message and stream time stamping, as well as quality of service via prioritized scheduling, are functionalities still missing in CORBA and most of the other message-based and publish/subscribe-based approaches, including EQUIP[14] (Greenhalg 2002), Elvin,[15] OOA (Martin et al. 1999) and the FIPA message and routing specification.

The object-oriented model of distributed computing can be expected to be overtaken by the agent/message model. Examples showing the benefits of the agent/message-based architecture, especially those making use of blackboard technology, are already showing up in the literature (c.f. Maxwell et al. 2001, Thórisson et al. 2004).

Few design methodologies have been developed in A.I. One could perhaps try to classify the work of Brooks (1986) on the subsumption architecture as a kind of design methodology, it seems much closer to being a theory of intelligence, as are other efforts such as for example Soar (Newell 1990) and frames (Minsky 1974). With a design methodology we are referring to engineering methodologies such as pair programming[16] and object-oriented programming. A design methodology is more general than the principles touted in A.I. such as subsumption architectures, fuzzy logic, artificial neural nets and similar.

While many software development methodologies – such as those mentioned above – can be applied directly in the development of A.I. systems, they do

[13] http://www.fipa.org/
[14] http://www.equator.ac.uk/PublicationStore/2002-greenhalgh.pdf
[15] http://elvin.dstc.edu.au/
[16] http://www.pairprogramming.com/

not deal with the special issues arising in this effort that is different from traditional software development. Frequently the development of A.I. software relies on an untested approach to, or analysis of, a problem. In this way A.I. systems development is much more like that of simulation development, where the natural system to be simulated seldom has been simulated before and certainly has not been modeled in the particular way that the scientists are trying it most of the time. When looking at the methodologies applied in simulation science, however, one finds that the methodologies there, such as discrete simulation (cf. Stchedroff & Cheng 2003, Pidd & Castro 1998) rarely address the kind of complexity that an A.I. system naturally must address (cf. Simmons et al. 2003, Laird 2002, Thórisson 1999), especially not those encountered in intelligent systems that must work in realtime in real settings.

Constructionist Design Methodology

The first prong in our attack on the pragmatics of creating large, integrated A.I. systems is the Constructionist Design Methodology (CDM[17]), a design methodology geared towards helping with the construction of large, complex systems. The methodology "glues together" the three first components by offering a formal structure for the system modularization process and enabling the execution of large A.I. systems in a distributed fashion.

Constructionist Design Methodology (CDM) is an emerging set of principles for designing and implementing interactive intelligences and systems. It is a practically-driven approach that can help speed up implementation of relatively complex, multi-functional systems.

Although the methodology is fairly general and can be applied to the building of various systems, including subsystems, we have a particular focus on developing relatively complete humanoids with realtime perception-action loop capabilities. Of special interest to us is human-robot communication, including speech, gesture, facial expressions, and gaze, in both the input and output. The embodiment of our robots can come in various levels of completeness, from just a face to a full body, and includes virtual robots that can perceive and act in virtual as well as real environments. CDM helps lower the complexity of building such systems. Here we give a short overview of the methodology; those interested in more detail are referred to Thórisson et al. (2004).

When beginning the construction of a mind we start with the high-level goals of the system. This is done by writing scenarios with narratives that cover the various parts of the system. Then follows a period of modularization where division of labor is used to come up with functional modules and message types. The role of each module is determined in part by specifying the message types and content that needs to flow between the various functional parts of the system to support the system's operational goals. The boundaries between modules are delineated by, in parallel, specifying their functionality along with messages defining the semantics of their inputs and outputs. Modularization through explicit messages means that system designers can

[17] Some earlier papers refer to the methodology as Constructionist A.I. Methodology (CAIM).

build out several parts of the system in parallel. Messages, and their content, is continuously refined as design progresses.

A total of 9 steps are identified in CDM, most of which are iterative in nature, delineated in Thórisson et al. (2004). The principle of divisible modularity prescribes iterative revision of modules through repeated division of their functionality into a set of ever-smaller interacting modules. CDM provides several heuristics for how to best to modularize. There are primarily two modularization motivators: related to architecture and related to efficiency. Among the former are classifying modules into the functional roles (e.g. perception, decision, and action); among the latter is avoiding duplication of information in various parts of the system and following the natural breaks along low-bandwidth information flow.

CDM proposes a breadth-first approach; every module starts out relatively simple, reading one type of message and posting another; the complete set of modules and their interaction is laid out very early in the design process. This way, a full end-to-end chain of the whole system can be built for a single interaction case. Every element in the path should be tested on paper, along with the routing mechanisms, timing assumptions, etc., very early in the design process, and continuously iterated over the course of development.

OpenAIR

The second prong in our framework is an information exchange and routing specification that provides a language-independent messaging format and network-independent routing protocol. The specification, called OpenAIR,[18] is based on a publish-subscribe architecture that is simple enough to be implementable in any (object-oriented) programming language, yet it is powerful enough to scale to support complex A.I. systems.

OpenAIR is intended to be a highly practical solution, allowing A.I. researchers to share code more effectively. In a nutshell, it is a definition or a blueprint of the "post office and mail delivery system" for distributed, multi-module systems.

OpenAIR is historically related to efforts such as KQML and Open Agent Architecture.[19] Its implementation is based around standard TCP/IP and XML and consists of a simple but solid message semantics and network protocols, and provides a foundation within which other markup languages and semantics can be transported between processes and over networks. For example, a gesture recognition and generation system might represent analyzed time segments with their own gesture-specific XML; this content would then be exchanged between the modules in the system through an OpenAIR message envelope.

OpenAIR has been in development for several years and has reached

[18] The full technical specification can be found at http://www.mindmakers.org/openair

[19] The initial OpenAIR specification was done by Communicative Machines and is now exclusively managed by the members of MINDMAKERS.ORG.

sufficient maturity to be used in courses and research projects at several universities in both Europe and the U.S. It promotes simple yet sufficient semantics to foster increased collaboration, communication and cooperation between software, systems, people, and institutions.

In pub-sub systems a module or agent can register for a message type, and any time a message of that type is posted (by anyone in the system), the message will be delivered to the subscribed module.

PSYCLONE

The third component in our framework is a software platform that incorporates the OpenAIR specification and provides additional functionality related to setting up, monitoring and maintaining heterogeneous systems running on multiple computers. The platform, called Psyclone (CMLabs 2002), supports both streaming data and discrete message passing, implements a querying interface and runtime monitoring and management tools.

Psyclone is best viewed as a system for handling the next level of architectural complexity above the object and the application; it sits at the same level as component-based frameworks such as Enterprise JavaBeans[20] (Sun Microsystems 2002), yet is fairly different in most respects, especially in that it is cross-language and has a powerful built-in load-balancing facility.

In accordance with OpenAIR, Psyclone supports publish-subscribe data routing. In addition to discrete data transfer Psyclone provides streaming data transfer between modules. Through modularity principles inherited from the Constructionist Design Methodology (CDM) it simplifies the design of complex systems and their connection to input and output systems like vision, body tracking, graphics and more.

Built around the concepts of modules and "whiteboards" (Thórisson et al. 2005a), which are a version of blackboards, Psyclone takes full advantage of the benefits of a message-based, publish-subscribe system. Among the benefits of this system is that the messages embody an explicit representation of each module's contextual behavior, and can carry with it their state. The whiteboards provide a (nicely centralized) recording of all system events and system flow.

Supported programming languages, C++ and Java at present, have an OpenAIR "plug", which provides simple API with full messaging and streaming[21] capabilities.

OpenAIR messages effectively implement APIs between the system's modules, specifying their message exchange through a common protocol.

Psyclone is especially relevant for incremental building and debugging of

[20] http://java.sun.com/products/ejb/

[21] Streams are not part of the OpenAIR specification yet, but are scheduled to be proposed as an extension in the future.

interactive systems. Monitoring systems, such as Psyclone's web-based run-time system inspection tool, can display the state of modules and the whole system. It allows the causal chains, embodied in the message flow, to be visualized and inspected directly. Monitoring tools work from any remote location, which can be very useful on systems where the deployment is on physically separated machines.

Interactive systems typically have unpredictable input streaming and output generation. They can be modeled as being asynchronous, real-time and open-loop. Comparing Psyclone to CORBA[22], which certainly shares some of its goals and features, Psyclone is created specifically for use in designing interactive A.I. systems, and thus has many built-in features for supporting the creation of such systems, e.g. temporal accountability, which means that time is kept track of in the system architecture. In contrast to CORBA, where an object makes a request for a service or for information, simulating an extended function call, Psyclone uses non-blocking message passing via whiteboards: Modules post data to a central server, and that data is delivered to subscribed modules. Additionally, modules in Psyclone can retrieve old messages from the blackboards, through a simple query language. At a high level, then, CORBA is “pull” whereas Psyclone is both "pull" and “push”. Psyclone also offers message time stamping and quality of service via prioritized scheduling, functionalities still missing in CORBA.

When a Psyclone module receives a message type to which it has subscribed, it may in turn post zero or more messages. Modules can also post messages at any time, independent of other message flow. To simplify development in Psyclone, the full set of modules and their attributes can be specified in an XML file called a psySpec. Modules that run on machines other than the Psyclone server can also be configured via this file.

At run-time, all data in the system travels in messages, via blackboards.[23] A message is a convenient metadata wrapper around the message's content. The metadata includes the message’s type, a globally unique ID (GUID), the language that the content is represented in, name of sender, and time of posting, along with other timestamps. This metadata can be useful in making queries about the system’s past behavior.

All messages in Psyclone have a type (most often assigned by the system designer and specified in the psySpec). For example, a face detector may post a piece of data of the type “Vision.Face”, containing partially processed or detailed facial data. There may be any number of different face detector modules, each of which posts that same data type, but with different content and emphasis. Message type names are represented as a tree with dot-delimitation, e.g. Vision.Face.Human and Vision.Face.Dog. One-to-one messaging between any two modules is done using unique message types.

In addition to standard ASCII messages, Psyclone provides powerful facilities for publishing and subscribing to binary data streams, using Psyclone’s whiteboards, which natively support streaming media. In Psyclone, all

[22] http://www.omg.org/cgi-bin/doc?formal/97-02-25

[23] An exception to this is the handling of streaming media, a topic beyond the scope of this paper.

modules and whiteboards have unique names; modules subscribe to message types from particular whiteboards as specified in the psySpec. They can also unregister and register dynamically for message types at run-time. Through dynamic subscriptions, interaction between modules can thus be "re-wired" on the fly, with any module able to alter its "connection" to other modules by registering or unregistering for the type(s) of data they produce.

It may be desirable to have the sensory I/O and cognition modules running in different languages and/or on different operating systems or hardware. Some components may only run on certain hardware architectures or configurations, as is often the case with open-source packages for e.g. speech recognition, speech synthesis, and computer graphics. Another reason to run components separately is that one may want particular components to take advantage of specialized hardware. Yet a third reason is cost: Since hardware is now cheaper than ever, access to multiple pieces of older yet perfectly usable hardware is becoming the norm.

Psyclone free the developer from concerning themselves with sockets and other details of networking. Daemons facilitate starting and managing processes on remote machines: On startup, Psyclone tells the daemon running on each of the computers to start up all relevant executables on that machine, automatically sending over any configuration parameters for these executables specified in the psySpec.

A strategy like CDM, that supports modular separation of functionalities during development and allows for rearranging the components and their interaction in a highly dynamic way, is, to our knowledge, the most powerful methodology currently available for creating broad, interactive A.I. systems.

Two Example Systems

We will now briefly describe how Psyclone was used with the Constructionist Design Methodology to produce fairly complex systems. The first system is an augmented-reality room inhabited by an embodied virtual agent, Mirage. The second is a large simulation of a market.

The system was developed by Thórisson and students at Columbia University (Thórisson et al. 2004) to enable interaction with a virtual agent in an augmented reality room: The user puts on a pair of optical see-through glasses[24] that reveal the location of the Mirage humanoid agent in three-space. Wearing the glasses the user sees the real world, but superimposed on it is a stereoscopic, ghost-like apparition of Mirage, whose behavior is generated by the system in real-time.

The mind of Mirage consists of eight main components: Two perception modules, an action/animation scheduling module, a speech recognition module[25], a speech synthesis module[26], a decision module and a news

[24] Sony LDI-D100B

[25] *Sphinx* speech recognition system. http://www.speech.cs.cmu.edu/sphinx/

[26] *FreeTTS* speech synthesis system. http://freetts.sourceforge.net/docs/index.php

summary module (McKeown 2002); a detailed 3-D model of the entire room, including tables, chairs, etc., gives Mirage a certain amount of knowledge about his surrounding. Mirage also has access to a database providing propositional information about the name, type, use, and other attributes of individual objects. All of these are implemented as separate modules (executables) communicating through Psyclone.

To make the agent interactive and conversational a multimodal input system was implemented that uses speech recognition and motion tracking of the user's right hand. The agent itself is capable of multimodal communication with the user via a speech synthesizer, body language, and manual gesture. Users can point at any object in the room and ask Mirage what that object is. When asked about the news, Mirage will recite up-to-date news summaries. The agent is also aware of his own proximity and orientation to the user; when either his name is spoken or the user comes within communicative distance, Mirage will turn to greet.

The total development time for Mirage was only an estimated 2 mind-months ("man"-months), over a period of 9 weeks – well under anyone's prior expectations. This result is comparable or better than that of others using blackboard-like architectures, e.g. Maxwell et al. (2001), who constructed a highly sophisticated robotic system with 10 full-time students, over a period of 8 weeks. Further details about this system can be found in Thórisson et al. (2004).

A second system built using the methodology and Psyclone is a simulation model of knowledge evolution in a market economy. Innovation is a central element of economic development and knowledge is the force behind it. Understanding knowledge – its organization and especially its dynamics in a market – becomes therefore the main challenge when explaining economic development in general, and the competitiveness and growth of firms and industries in particular. The simulation system developed includes a fine-grain, dynamic, morphogenic model of knowledge that is easy to manage, interpret and extend. The knowledge model is embedded a larger market simulation where selected elements of an economy, including employees, companies, banks and consumers, are modeled at multiple levels of abstraction, from agents to monolithic entities, comprising 5 distinct module types. On any run several instances of each type are run, for up to 80 modules in a single run. The results show the model's excellent potential to address questions about emergent phenomena related to knowledge evolution, knowledge transfer and knowledge use in market innovation. The system was built by 10 Master's-level students over a period of 5 weeks. The results support prior conclusions that CDM is a powerful high-level design approach for developing systems with a large number of modules with complex interactions.

MINDMAKERS.ORG

The fourth prong is an open platform for code exchange, documentation, discussion and storage. It is really a project-exchange, but thinking about it at the code level makes its primary focus more obvious: To facilitate increased

testing and use of A.I. code written by others. MINDMAKERS.ORG is the website we have set up for this purpose. This research forum is done incollaboration with a number of other researchers. We invite anyone with interest to join in the MINDMAKERS effort.

Acknowledgments

The authors would like to thank Rögnvaldur Sæmundsson, and students in the Agent-Based Modeling & Simulation course at RU, students in the Humanoids in Virtual & Augmented Realities at Columbia U. Thanks also to the MINDMAKERS Pioneers.

References

Adler, R. (1989). Blackboard Systems. In S. C. Shapiro (ed.), *The Encyclopedia of Artificial Intelligence*, 2nd ed., 110-116. New York, NY: Wiley Interscience.

Bischoff, R. (2000). Towards the Development of 'Plug-and-Play' Personal Robots. *1st IEEE-RAS International Conference on Humanoid Robots*. MIT, Cambridge, September 7-8.

Brooks, R. A. (1989). How To Build Complete Creatures Rather Than Isolated Cognitive Simulators. In K. VanLehn (ed.), *Architectures for Intelligence*, 225–239. Hillsdale, NJ: Erlbaum.

Brooks, R. A. (1986). Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, **2**(1), March, 14–23. Also MIT AI Memo 864, September 1985.

CMLabs (2002). The Psyclone Manual. http://www.cmlabs.com/psyclone/psyclone-manual.html

Dodhiawala, R. T. (1989). Blackboard Systems in Real-Time Problem Solving. In Jagannathan, V., Dodhiawala, R. & Baum, L. S. (eds.), Blackboard Architectures and Applications, 181-191. Boston: Academic Press, Inc.

Fink, G. A., Jungclaus, N., Kummer, F., Ritter, H., Sagerer, G. 1996. A Distributed System for Integrated Speech and Image Understanding. *International Symposium on Artificial Intelligence*, 117-126, Cancun, Mexico.

Fink, G. A., Jungclaus, N., Ritter, H., Saegerer, G. 1995. A Communication Framework for Heterogeneous Distributed Pattern Analysis. *International Conference on Algorithms and Architectures for Parallel Processing*, 881-890, Brisbane, Australia.

Greenhalgh, C. EQUIP: a Software Platform for Distributed Interactive Systems. Technical Report Equator-02-002, University of Nottingham.

Johnson, W. L., S. Marsella, N. Mote, M. Si, H. Vilhjálmsson & S. Wu (2004). Balanced Perception and Action in the Tactical Language Training System. Workshop on Embodied Conversational Agents: Balanced Perception & Action, July 20th, 18-25. *AAMAS 2004: The Third International Joint Conference on Autonomous Agents & Multi Agent Systems*, New York, July 19-23.

Laird, J. (2002). Research in Human-Level A.I. Using Computer Games. Communications of the ACM, january, vol 45(1), 32-35.

Martin, D., A. Cheyer, & D. Moran (1999). The Open Agent Architecture: A Framework for Building Distributed Software Systems. *Applied Artificial Intelligence*, **13**(1-2), 91-128.

Martinho, C., A. Paiva & Gomes, M. R. 2000. Emotions for a Motion: Rapid Development of Believable Pathematic Agents in Intelligent Virtual Environments. *Applied Artificial Intelligence*, **14**(1), 33-68.

Maxwell, B. A., L. A. Meeden, N. S. Addo, P. Dickson, N. Fairfield, N. Johnson, E. G. Jones, S. Kim, P. Malla, M. Murphy, B. Rutter, & E. Silk (2001). REAPER: A Relfexive Architecture for Perceptive Agents. *A.I. Magazine*, spring, 53-66.

McKeown, K.R., R. Barzilay, D. Evans, V. Hatzivassiloglou J. L. Klavans, A. Nenkova, C. Sable, B. Schiffman, & S. Sigelman (2002). Tracking and Summarizing News on a Daily Basis with Columbia's Newsblaster.

Human Language Technology ’02 (*HLT* ’02), San Diego, California.
Microsoft Corporation (1998). Distributed Component Object Model Protocol – DCOM/1.0. January. Redmond, NJ: Microsoft Corporation.
Minsky, M. (1986). *The Society of Mind*. New York: Simon and Schuster.
Minsky, M. (1974). A Framework for Representing Knowledge. MIT-AI Laboratory Memo 306, June. Reprinted in P. Winston (Ed.), *The Psychology of Computer Vision*, McGraw-Hill, 1975. Shorter versions in J. Haugeland, Ed., *Mind Design*, MIT Press, 1981.
Newell, A. (1990). *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.
Pagello, E., E. Mengegatti, A. Bredenfel, P. Costa, T. Christaller, A. Jacoff, D. Polani, M. Riedmiller, A. Saffiotti, E. Sklar, T. Tomoichi, 2004. RoboCup-2003: New Scientific and Technical Advances. *A.I. Magazine* **25**(2), 81-98. AAAI. Menlo Park, CA: AAAI Press.
Pidd, M. & R. B. Castro (1998). Hierarchical Modeling in Discrete Simulation. In D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan (eds.)*, Proceedings of the 1998 Winter Simulation Conference.*
Searle, J. R. 1969. *Speech Acts: An Essay in the Philosophy of Language*. Cambridge: Cambridge University Press.
Selfridge, O. (1959). Pandemonium: A Paradigm for Learning. Proceedings of Symposium on the Mechanization of Thought Processes, 511-529.
Simmons, R., D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A. Schultz, M. Abramson, W. Adams, A. Atrash, M. Bugajska, M. Coblenz, M. MacMahon, D. Perzanowski, I. Horswill, R. Zubek, D. Kortenkamp, B. Wolfe, T. Milam, B. Maxwell (2003). GRACE: An Autonomous Robot for the AAAI Robot Challenge. *A.I. Magazine*, **24**(2), 51 – 72.
Stchedroff, N & R. C. H. Cheng (2003). Modelling a Continuous Process with Discrete Simulation Techniques and its Application to LNG Supply Chain. In S. Chick, P. J. S·nchez, D. Ferrin, and D. J. Morrice (eds.), *Proceedings of the 2003 Winter Simulation Conference.*
Sun Microsystems (2002). *Enterprise JavaBeans Specification V2.1*. Palo Alto: Sun Microsystems.
Thórisson, K. R., T. List, C. Pennock, J. DiPirro (2005a). Whiteboards: Scheduling Blackboards for Semantic Routing of Messages & Streams. In K. R. Thórisson, H. Vilhjalmsson & S. Marsela (Eds.), *AAAI-05 Workshop on Modular Construction of Human-Like Intelligence*, Pittsburgh, PA, July 10. AAAI Technical Report WS-05-08, 8-15.
Thórisson, K. R., H. Benko, A. Arnold, D. Abramov, A. Vaseekaran (2004). A Constructionist Methodology for Interactive Intelligences. Submitted to *A.I. Magazine*.
Thórisson, K. R. (1999). A Mind Model for Multimodal Communicative Creatures and Humanoids. *International Journal of Applied Artificial Intelligence*, **13**(4-5), 449-486.
Thórisson, K. R. (1998). Decision Making in Real-Time Face-to-Face Multimodal Communication. *Second ACM International Conference on Autonomous Agents ‘98,* Minneapolis, Minnesota, May 12-15, 16-23.
Thórisson, K. R. (1997). Layered Modular Action Control for Communicative Humanoids. *Computer Animation '97*, Geneva, Switzerland, June 5-6, 134-143.