National Aerospace and Electronics Conference (NAECON)
Dayton, OH, October 10-12, 2000.

A KNOWLEDGE BASE FOR KNOWLEDGE-BASED MULTIAGENT SYSTEM CONSTRUCTION

MARC J. RAPHAEL[1] and SCOTT A. DELOACH[2]

[1] Space and Missile Center, SMC/XRDM, LAAFB, California 90245, USA, Marc.Raphael@losangeles.af.mil
[2] Air Force Institute of Technology, WPAFB, Ohio 45433, USA, Scott.DeLoach@afit.wpafb.af.mil

Abstract. The goal of the agentTool project at the Air Force Institute of Technology is to specify, design, and semi-automatically generate multiagent systems. The key to this ability is an underlying knowledge base that manages the knowledge used by the system designer in creating intelligent agent-based software applications. This paper describes the Agent knowledge Interchange Mechanism (AIM), the agentTool knowledge base. There are four main components to AIM: the knowledge parser, the Multiagent Markup Language, the AIM domain model, and the Agent-oriented Random-Access Meta-Structure (ARAMS). AIM is implemented using a multiagent system architecture that permits multiple developers to share a single knowledge base thus directly supporting collaborative system design and knowledge reuse.

Key Words: Agent, multiagent system, knowledge, software, artificial intelligence

1. INTRODUCTION

With the advent of the Internet, many researchers have been taking a closer look at distributed software systems. Recently, a large share of this research has focused on intelligent distributed systems, which have come to be known as multiagent systems. As a result, new development methodologies specifically designed for multiagent systems have been introduced [3] and several tools are now available for building multiagent systems [12].

The goal of our research at the Air Force Institute of Technology (AFIT) is to define a complete multiagent system development methodology and an associated toolset to support its use. Our methodology, Multiagent Systems Engineering (MaSE), provides a complete life cycle approach to developing multiagent systems from their initial requirements specification to the generation of code [1]. The MaSE methodology and its implementation within agentTool follow the seven steps shown in Figure 1. At each step, the user uses the graphically based models listed to further elaborate and define the system under development. To evaluate and support MaSE, we developed the agentTool development environment, which provides automated support for developing the graphical models required by MaSE [2]. Unlike many previous efforts, we did not tie MaSE and agentTool to any one communication framework, architecture, or implementation language; we allow the user to develop truly heterogeneous multiagent systems. To support distribution and reuse of the knowledge required to build systems in agentTool, we developed a knowledge management system called the Agent knowledge Interchange Mechanism (AIM). AIM allows the agentTool user to save all or part of the current system analysis or design and reuse the analysis or design from previous systems as well.

The remainder of the paper discusses the agentTool knowledge base, AIM. Section 2 describes the four AIM components, while Section 3 provides an example of its use in agentTool. Finally, Section 4 presents conclusions based on our work to date and future areas of research.

2. KNOWLEDGE INTERCHANGE

The goal of AIM is to allow agentTool (and other tool) users to store and reuse the knowledge contained in the MaSE models in a concise, reusable, and adaptable form and do so in either a local or distributed mode. This allows a user flexibility by providing either standalone access or the ability to work in collaborative network environment. AIM also strives to provide rapid location and retrieval of stored elements while guarding against errors that might arise by having more than one agentTool instance attempting to read and write to a common knowledge entity. AIM consists of four key elements: 1) an overarching multiagent system domain model, 2) a knowledge representation and interchange language called the Multiagent Markup Language (MAML), 3) a tool-specific MAML parser, and 4) a knowledge storage utility named the Agent-

oriented Random-Access Meta-Structure (ARAMS). These elements are integrated via a multiagent system consisting of a Guardian agent and one or more Connection agents. Since agentTool's internal object model was readily available, it was the first tool integrated into AIM. Figure 2 reflects generically how AIM elements and agentTool work together.

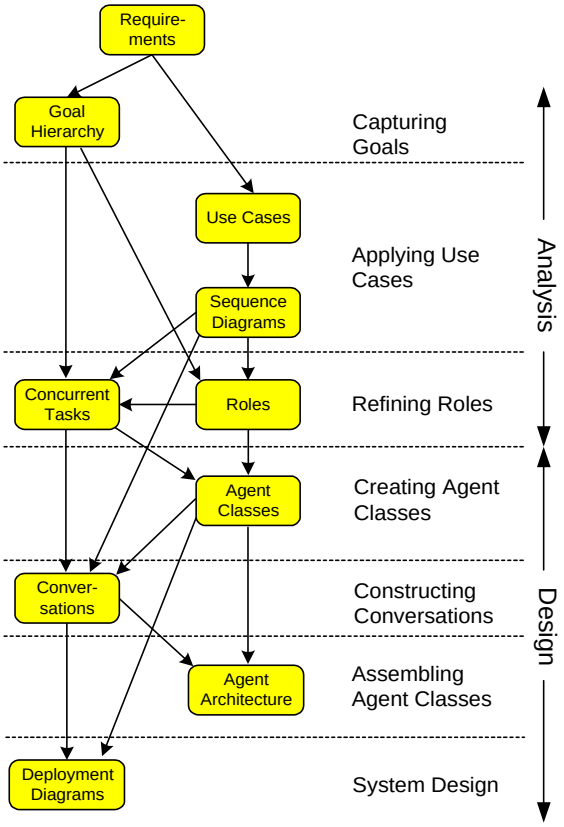

Figure 1. MaSE Overview

2.1. AIM and Related Efforts

AIM is unique in its approach to the storage and management of design knowledge for sharing and reuse. Complimentary work, however, is extensive. For instance, research has looked at the knowledge required to design a multiagent system given a particular methodology [13]. Additionally, efforts are underway to standardize and model core functions of agent development tools to permit interoperability [6,10]. One of these efforts, led by the Foundation for Intelligent Physical Agents (FIPA), differs from AIM by focusing on how to develop applications with cooperating agents, rather than on facilitating cooperation between the development tools. However, the goal of the Object Management Group Agent Working Group (OMGAWG) is closer to ours as its focus is on developing a common agent domain model, a key element to enabling proprietary applications to work together. Both AIM and OMGAWG focus on the interoperability between development tools and not just the agents those applications create. However, while the OMGAWG has yet to produce tangible results, we have actually implemented a system with a means to represent, store, retrieve, and reuse agent domain knowledge.

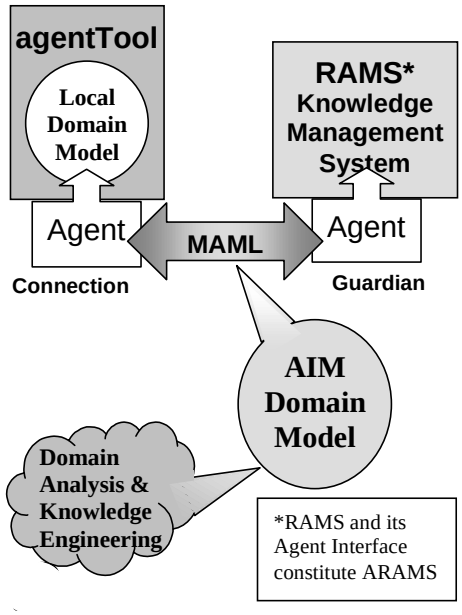

Figure 2. AIM – agentTool Environment

2.2. AIM Domain Model

The heart of AIM is its domain model, which defines the elements used to analyze, design, and implement multiagent systems. This domain model was developed using a top-down domain analysis effort augmented with an investigation into prior multiagent system research. The intent of the domain model is to encompass enough of multiagent system domain space to provide solid common ground for multi-tool collaboration. Our domain model includes both the objects used in designing multiagent systems as well as the relationships between those system objects. Figure 3 shows a simplification of our first iteration of the AIM domain model in the Unified Modeling Language (UML) and how we capture the multiagent system domain objects and relationships.

2.2.1 Domain Objects

Theory: Collects and models conceptual knowledge, often formally. One popular theory

is Dennett's *Intentional Systems Theory*, which was adapted to agents by Cohen and Levesque collects [7]. The resulting intentional theory for agents models agents in terms of belief, desire, and intention concepts.

Goal: An objective for the agent or multiagent system to achieve.

Role: An intended pattern of behavior, responsibilities, and collaborations for an agent within an overall structure or system. Roles contain generalized *tasks* and a set of *resources*.

Communication: A simple model of an interaction between *roles*.

Role Model: A collection of *roles* and patterned *communications* between roles [8].

Task: A method that constrains and defines how a *goal* is to be achieved.

Resource: Something that a *role* accesses to aid or support the *role* in its responsibilities.

Component: The fundamental functional module of an agent. Components have methods and attributes.

Architecture: An abstraction for using a set of *components* together in a particular way. Although an agent always has an architecture, *components* themselves may also have architectures for organizing their sub-components.

Communication Framework: A set of protocols, components, and mechanisms that permit relay of data and knowledge. Java RMI and sockets are two common communication frameworks. Framework specifications have close ties with framework implementations.

Data Construct: Analogous to a resource for a role, but for an agent. Data constructs are not necessarily part of an agent, though an agent may have access to them. Components of an agent may utilize this access in order to function.

Conversation: A state-transition table (or finite-state machine) defining interactions between agents.

2.2.2 Domain Relationships

Played-by: A Role is Played-by an agent. Alternatively, an agent may assume a Role.

Achieves: A Task achieves one or more Goals

Participates-in: An agent may participate in one or more Conversations in a given system design.

Uses: A Role may require the use of some Resource to fulfill its responsibilities.

Interfaces: Components that interoperate must interface one another.

Accesses: An architecture may require Access to some class of Data Construct

Involves: A Role may involve the use of one or more Communications.

Specifies: A Conversation specifies a communication at a higher level of detail.

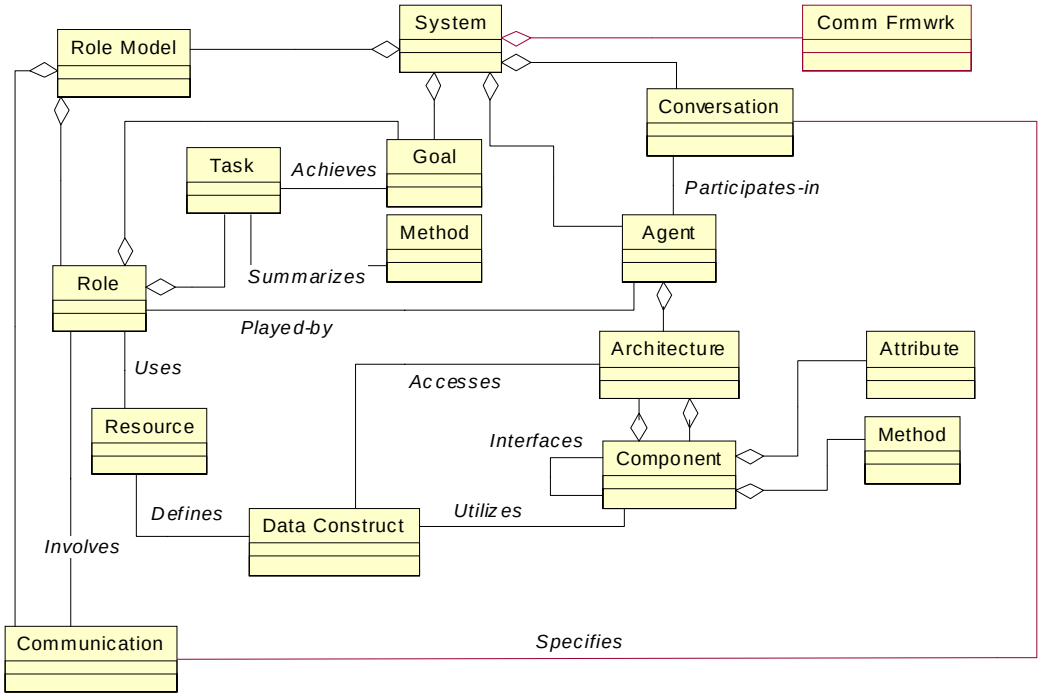

Figure 3. AIM Object Model

Utilizes: A component utilizes the content of a Data Construct to which it belongs or *accesses*.

Defines: A Data Construct defines a resource in the same way a Conversations specifies a communication.

2.3. Multiagent Markup Language

Although UML can be used to model multiagent system knowledge graphically, it is not ideal for knowledge interchange. What we need instead is a concise notation amenable to network transfer and automatic parsing and generation. To achieve these goals, we took advantage of the eXtensible Markup Language's (XML) extensibility and compact nature to define the Multiagent Markup Language (MAML). MAML is the actual representation and interchange language of AIM. MAML document type definitions (DTDs) specify XML syntactic and semantic elements for each of the knowledge entities in the domain model (conversations, components, architectures, etc.). DTDs allows any tool supporting XML 1.0 [9] to parse a correctly constructed MAML document into its own local object model, whether the parsed MAML document was created by hand or a MAML-enabled tool.

Figure 4 shows the MAML description of a relatively simple conversation between two agents. The two sides of the conversation are shown in Figures 5 and 6 as designed and used within agentTool. The agentTool interface uses the simplicity of the graphical form for users developing systems, but relies on the MAML representation for storage and transfer. Syntax and semantics captured in the agent MAML DTDs permit representation in the form shown in Figure 4, which is far more compact that even zipped compression of the actual Java objects.

2.4. Knowledge Parser

Any tool requiring the ability to store, retrieve, or exchange of multiagent system knowledge in AIM needs a MAML compliant parser. The agentTool parser consists of a set of methods in each Java object class in the underlying agentTool object model. These methods are used to either create a MAML text string representation from an agentTool object model or instantiate an agentTool object model from an existing MAML description. Individual MAML elements can be pulled-from or inserted-into any system being developed in agentTool. This allows individual agents, conversations, agent components,

```
<conversation version="1.1" name="SendRawIntell">
  <description> Conversation used for data transfer </description>
  <participant name="initiator">
    <statetable owner="MissionControl">
      <state name="StartState"> </state>
      <state name="EndState"> </state>
      <state name="wait" action=""/> </state>
      <transition rMessage="" tMessage="tell(rawIntell)" guard=""
                  cstate="StartState" nstate="wait">
      <transition rMessage="ack" tMessage="" guard=""
                  cstate="wait" nstate="EndState">
      <transition rMessage="failure()" tMessage="tell(rawIntell)"
                  guard="" cstate="wait" nstate="wait">
    </statetable>
  </participant>
  <participant name="responder">
    <statetable owner="Intelligence">
      <state name="StartState"> </state>
      <state name="EndState"> </state>
      <state name="wait" action=""/> </state>
      <state name="validation" action="valid = validate(rawIntell)"/> </state>
      <transition rMessage="tell(rawIntell)" tMessage=""
                  guard="" cstate="StartState" nstate="validation">
      <transition rMessage="" tMessage="ack" guard="valid"
                  cstate="validation" nstate="EndState"
      <transition rMessage="" tMessage="failure()" guard="NOT valid"
                  cstate="validation" nstate="wait">
      <transition rMessage="tell(rawIntell)" tMessage="" guard=""
                  cstate="wait" nstate="validation">
    </statetable>
  </participant>
</conversation>
```

Figure 4. Sample MAML Document

etc. to be stored and reused independently as the need arises.

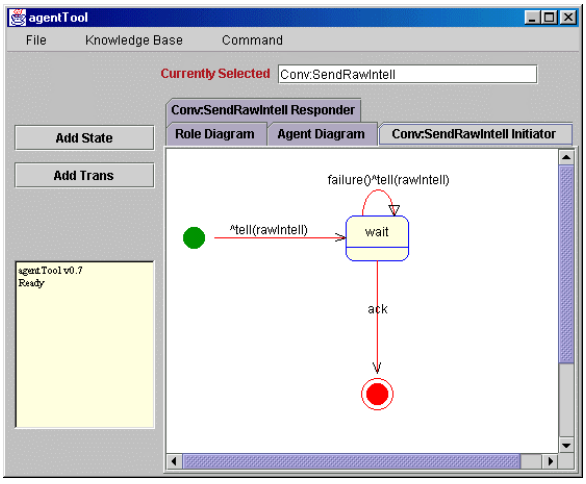

Figure 5. Example Initiator Conversation

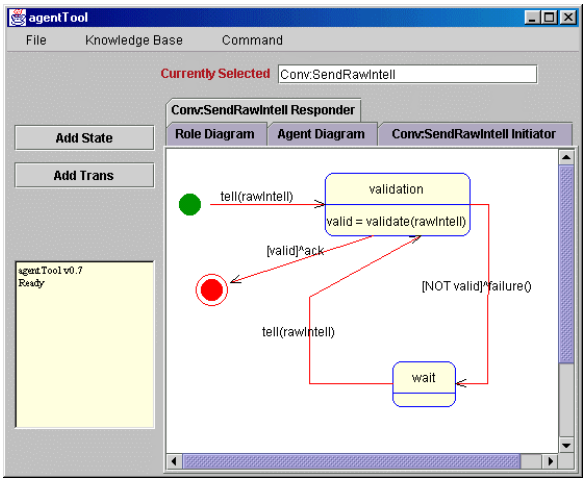

Figure 6. Example Responder Conversation

2.5. RAMS

When agentTool or some other AIM-enabled multiagent system either produces or requires multiagent system knowledge, it can proceed in one of three ways.

1. It may store or retrieve a MAML document locally.
2. It may access a centralized repository.
3. It may communicate directly with another tool.

The second option is the most useful since companion tools are likely to be busy or not running and local storage is not likely to contain collaborative knowledge. In AIM, the central repository is called the Random-Access Meta-Store (RAMS). RAMS allows us to organize knowledge into libraries, is persistent, and is generally quite responsive to library storage and retrieval requests. Currently there are RAMS libraries for Conversations, Agents, Multiagent Systems, Communication Frameworks, Agent Components, Roles, and Agent Architectures. Each library is based on the same file structure, which in turn is based on the Java RandomAccessFile class. The file structure includes modifications suggested by Hamner, which include header and data storage sections [4]. The library header holds a set of key-pointer pairs, where the key is a user-defined name for a stored item and the pointer points to the item's location in data storage section. If the library grows to large, a knowledge-base maintainer can partition it into two libraries or a set of volumes within the original library. An example name for a RAMS *library* would be "Conversations", while a possible RAMS *volume* in that library might be "RegistrationConversations".

2.6. Guardian and Connection Agents

To store MAML descriptions in a central repository, they must be sent to RAMS. This is accomplished using a multiagent system framework. Each tool instance (e.g., agentTool) has its own Connection agent, which it calls to transfer data to or from RAMS. The Connection agent guides the user through the storage process, asking for an object description and an identifying key. The Connection agent then initiates a *StoreObject* conversation with a Guardian agent. The Guardian agent resides local to the RAMS file structure and has exclusive access to it. The Guardian agent has the responsibility to allow a MAML document to be saved only if it has a unique key. If the key is not unique, the agent requests a different key from the Connection agent, which passes the request on to the user. When the user wishes to retrieve an item for reuse, the local Connection agent initiates a *ListObject* conversation with the Guardian. The Guardian returns a complete list of keys for the selected library and the Connection agent presents them to the user and waits for a choice. If the user is unsure which key to use, the user can read the item description that appears upon key selection. When a key (description) is selected, the object is transferred and parsed into the tool's internal data structure. The Guardian is designed to handle multiple storage and retrieval requests, and to prevent errors such as the reading and writing simultaneously to a keyed library item. The inclusion of a Guardian and Collection agents as the interface to RAMS

constitutes what we call Agent-oriented RAMS (ARAMS). Figure 7 shows the ARAMS architecture. Other information on ARAMS libraries and agents is available [5].

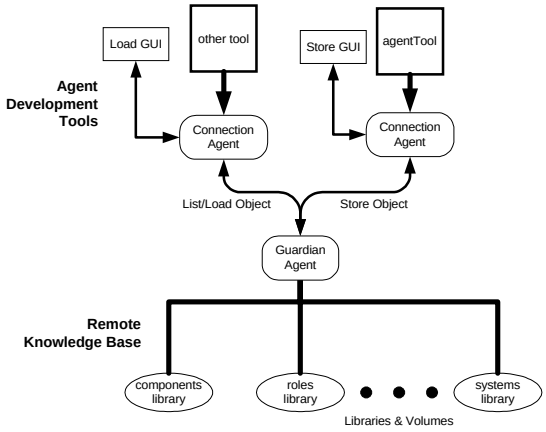

Figure 7. ARAMS Architecture

3. EXAMPLE

The AIM system described above operates smoothly and effectively in a distributed environment. The following describes how specific storage and retrieval operations between agentTool (or another MAML-parsing tool) and ARAMS might proceed. In this example, a multiagent system consisting of two agents and one conversation has been designed. Figure 8 shows the state-diagram associated with the conversation *Register*. The pull down menu with the *Store Object* option selected is also shown. When chosen, this option calls the local Connection Agent into service, which will detect the choice and present a GUI for facilitating a storage conversation. The Connection agent will also call the appropriate knowledge parser methods to generate and retrieve the selected object's MAML representation.

Figure 9 shows the storage interface with its *key* and *description* fields. After the user enters the required data, the conversation is stored in the knowledge base, with the Connection agent establishing a remote connection to the Guardian agent. The Guardian agent determines which library the object should be stored in, which, in this case, is the *conversations* library. If the key already exists, the Guardian notifies the Connection agent who, in turn, prompts the user to enter a new key. This cycle continues until the user enters a unique key.

To reuse the *Register* conversation after it is stored in the knowledge base, another (or possibly the same) user performs a similar series of actions. The user simply creates a graphical placeholder for a conversation in agentTool and chooses *Load Object* from the pull-down menu show in Figure 8. Once again, agentTool signals the local Connection agent, which displays the interface shown in Figure 10. The agent transparently contacts the Guardian, requests the appropriate library listing, and presents the listing to the user. In Figure 10, the user has selected the Register conversation causing the corresponding description to appear on the right-hand side. If this description is satisfactory, the user presses the *use* button and the MAML object is retrieved and parsed into the tool. The conversation replaces the local tool's internal representation overwriting the user's placeholder on the GUI.

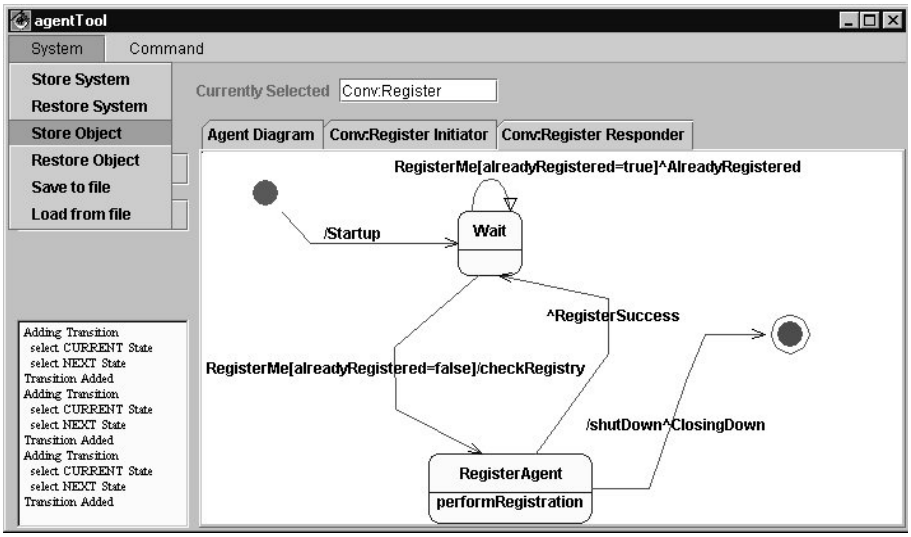

Figure 8. agentTool User Interface

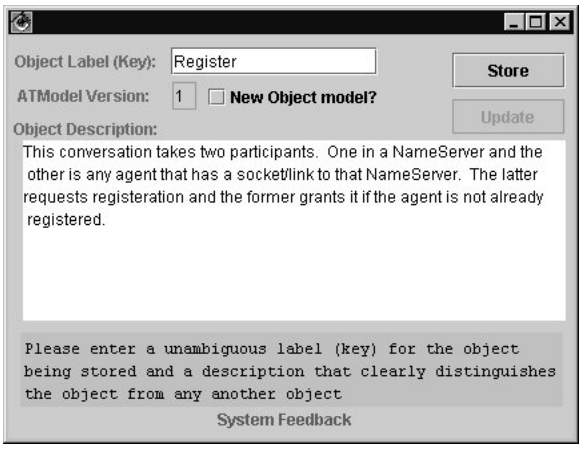

Figure 9. Store Conversation

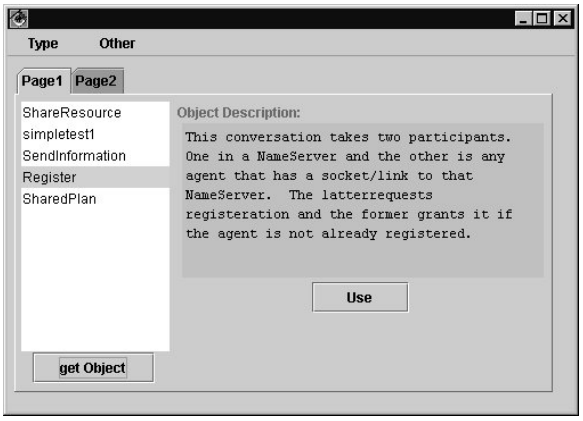

Figure 10. List/Load Conversation

4. CONCLUSIONS

The Agent knowledge Interchange Mechanism introduced in this paper provides a robust method for collaboration using diverse tools. In additional to agentTool, we have also modified the AFIT Wide-Spectrum Object Modeling Environment (AWSOME) to use AIM. The goal of AWSOME is to generate code from general object-oriented domain models. After creating a MAML compliant parser for AWSOME, we have demonstrated transforming MAML compliant multiagent system designs into AWSOME for code generation. We are also considering separating the agentTool verification tool [11], which checks the robustness of an agent design in agentTool, from agentTool using MAML and AIM as an intermediary. Once separated, AIM may be used to maintain interoperability between the two tools while allowing the overall system architecture to be modularized.

While AIM has been successful to date, there are some interesting challenges left. For one, the AIM domain model is likely to evolve as we consider more and more tools for possible inclusion in the suite. Similarly, RAMS has a set number of stored agent knowledge representations, which limits reuse by reducing the likelihood of encountering a pertinent item in storage. However, as with the domain model, this challenge will resolve itself as agentTool and AWSOME are used more extensively.

The distributed nature of AIM is also a possible drawback, particularly in the area of bandwidth allocation and efficiency. Although we believe that AIM interchange may be slowed by competing network applications, we feel that AIM itself will not be a prime contributor because MAML allows for very efficient knowledge transfer.

Another potential challenge of AIM is in MAML. Although MAML is based on industry-standard XML, which is incredibly flexible, newer technologies offer even more promise. XML-Schemas, for example, may replace DTDs and allow for the definition of data types and other useful elements not readily available in XML. Luckily, XML is backwards compatible; scripts can be written to automatically translate XML code (MAML) to be compliant with the XML-Schema specification. The eXtensible Styling Language (XSL) is one XML-related language that can be used to create such scripts. XSL can also be used to translate MAML into a number of forms, including graphical forms for display in a browser.

The final challenge concerns the ontology of the AIM domain model relative to the AIM toolset. Although not mentioned in this paper, capturing diverse tools' ontological differences is a widely known challenge in agent research.

5. ACKNOWLEDGEMENTS

The Air Force Office of Scientific Research (AFOSR) and the Dayton Area Graduate Studies Institute (DAGSI) supported this research. The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the US Government.

6. REFERENCES

[1] M.F. Wood and S.A. DeLoach, "An Overview of the Multiagent Systems Engineering Methodology." in *Proceedings of the First International Workshop on Agent-Oriented Engineering (MOSE).* Limerick Ireland, June 2000.

[2] S. A. DeLoach and M. Wood, "Developing Multiagent Systems with agentTool," in *Proceedings of The Seventh International Workshop on Agent Theories, Architectures, and Languages*, Boston, Massachusetts, July 2000.

[3] C. Iglesias, M. Garijo, and J. González, "A Survey of Agent-Oriented Methodologies," in *Intelligent Agents V. Agents Theories, Architectures, and Languages*, Lecture Notes in Computer Science, vol. 1555, J. P. Müller, M. P. Singh, and A. S. Rao (Eds.), Springer-Verlag, 1998.

[4] D. Hamner, "Using a RandomAccessFile to Build a Low-level Database," *JavaWorld*, 1999. http://www.javaworld.com/javaworld/jw-01-1999/jw-01-step_p.html

[5] M. J. Raphael, "Knowledge Based Support for Design and Synthesis of Multiagent Systems." MS Thesis AFIT/ENG/00M21. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB Ohio, USA, 2000.

[6] FIPA "Architectural Principles Baseline," Foundation for Intelligent Physical Agents FIPA Spec 0-1999, January 1999.

[7] M. Wooldridge, and N. Jennings, "Intelligent Agents: Theory and Practice," *Knowledge Engineering*, 1995.

[8] E. A. Kendall, "Agent Roles and Role Models: New Abstractions for Multiagent System Analysis and Design," *Proceedings of the International Workshop on Intelligent Agents in Information and Process Management*, Bremen, Germany, September 1998.

[9] S. A. Teplyakovsky, "XML Viewer," IBM Alphaworks, 1999.

[10] OMG Agents Working Group. http://www.objs.com/isig/agents.html

[11] T. H. Lacey and S. A. DeLoach, "Verification of Agent Behavioral Models." in *Proceedings of the International Conference on Artificial Intelligence*, CSREA Press, Las Vegas, Nevada, July 2000.

[12] AgentBuilder, "Agent Construction Tools", vol. 2000: AgentBuilder, 1998. http://www.agentbuilder.com/AgentTools/index.html

[13] M. Wooldridge, N. Jennings, and D. Kinny, "The Gaia Methodology for Agent-Oriented Analysis and Design," *Journal of Autonomous Agents and Multi-Agent Systems*. vol. **3**(3), 2000.

7. AUTHOR BIOGRAPHIES

Captain Raphael works for the Air Force Space and Missile Center, in Los Angeles California. He is a leader in the application of modeling and simulation in the acquisition and analysis of future space systems. He completed his Master's Degree in Computer Engineering at the Air Force Institute of Technology in March 2000. His Master's thesis was on knowledge systems for use with MAS development. Before AFIT, Captain Raphael served at the National Air Intelligence Center. Capt Raphael earned his BS in Electrical and Computer Engineering from Brigham Young University in 1995.

Dr. DeLoach is an Assistant Professor of Computer Science and Engineering at the Air Force Institute of Technology (AFIT). His research interests include design and synthesis of multiagent systems, knowledge-based software engineering, and formal specification acquisition. Prior to AFIT, Dr. DeLoach was at the Air Force Research Laboratory from 1996 to 1998. He has also been stationed at Headquarters Strategic Air Command and the Aeronautical Systems Center. Dr. DeLoach received his BS in Computer Engineering from Iowa State University in 1982 and his MS and PhD in Computer Engineering from the AFIT in 1987 and 1996.