Towards a Reference Model for Agent-Based Systems

Pragnesh Jay Modi, Spiros Mancoridis,
William M. Mongan, William Regli
Department of Computer Science
Drexel University
Philadelphia, PA 19104
{pmodi, spiros, wmm24,
regli}@cs.drexel.edu

Israel Mayk
US Army CERDEC C2D
AMSEL-RD-C2-SS-T
Fort Monmouth, NJ 07703
Israel.Mayk@us.army.mil

ABSTRACT

The current state of the art in agent technology sees that several implementations of agent frameworks exist. However, there is little agreement on the terms and concepts used to describe such systems, which is a significant barrier towards adoption of these technologies by industry, military and commercial entities. A clear definition of terms and concepts at an appropriate level of abstraction is needed to facilitate discussion, evaluation and adoption of these emerging agent technologies. In this paper, we argue that a *reference model* for agent-based systems can fill this need. We discuss what a reference model is, why one is needed for agent-based systems, and our proposed methodology for creating such a reference model. While the complete model is a work in progress, we present a preliminary version to motivate further discussion from the agents community at large. It is our hope that ultimately a wider community of practice will assume responsibility for the standardization similar to the way that the well-known seven-layer Open Systems Interconnection (OSI) reference model was a driving force underlying communications standards.

Categories and Subject Descriptors

H.1 [**Information Systems**]: Models and Principles

General Terms

Standardization

Keywords

Intelligent Agents, Reference Model

1. INTRODUCTION

The ultimate goal of the Agent-Based Systems Reference Model (ABSRM) is to provide a technical recommendation for a reference model for those who develop and deploy systems based on agent technology. The ABSRM should allow for existing and future agent frameworks to be compared and contrasted as well as to provide a basis for identifying areas that require standardization within the agents community. As such, the aim of the ABSRM is to

- establish a taxonomy of terms, concepts and definitions needed to compare agent-based systems;
- identify functional elements that are common in agent-based systems;
- capture data flow and dependencies among the functional elements in agent-based systems;
- specify assumptions and requirements regarding the dependencies among these elements.

As a reference model, the ABSRM will make no prescriptive recommendations about how to best implement an agent-based system; nor is the objective to advocate for any particular approach, architecture or framework. In its broadest sense, an agent-based system for the purposes of the ABSRM simply describes a software platform for building agents and supporting their communications and collaboration. An agent-based system may consist of many different kinds of agents operating across a heterogeneous set of platforms and hosts.

One novel aspect of our approach is to create the reference model based on a forensic analysis of existing agent-based systems. The reference model developed in this document is based on static and dynamic software analysis of existing agent frameworks. Examined frameworks include Cougaar, JADE, RETSINA, and others. Anyone building an agent framework would have to recreate or reproduce some portion of the functionality or components in these existing frameworks (i.e., to enable communications, to enable agent startup and shutdown, etc). By analyzing existing frameworks and the agent-based systems they can be used to build, we can avoid the debate concerning "what is an agent" and simply document the existing state-of-the-art systems that are called agent-based. The model aims to document a superset of the features and functional concepts in the set of existing agent frameworks. Given that there is significant variation between existing frameworks and the functions they may provide, the reference model should describe at an abstract layer the complete set of functional components across all examined agent frameworks.

It is important to note however that the reference model is not confined to being a description of capabilities of existing systems —it serves as a basis for situating the complete set of functions that anyone may want or need to have in an agent-based system. For example, security for mobile agent code is currently a vastly challenging problem that lacks a satisfactory solution. However, the lack of any established, uniform and generally accepted security system for mobile agents does not prevent the reference model for including a description of the security functions and facilities that an agent-based system may provide.

2. WHAT IS A REFERENCE MODEL?

A *reference model* provides appropriate abstractions for facilitating adoption, adaptation and integration of evolving technologies [18]. Any generic model that has specific examples can be considered to be a reference model. Reference models are known to play a key role in understanding a given domain, establishing the domain as a scientific discipline, facilitating collaboration and promoting competition towards maturing technology relevant to the domain. Reference models emerged since the 1980s as a result of the success of the Open Systems Interconnection seven-layer reference model (ISO/IEC 7498-1:1994: Open Systems Interconnection Basic Reference Model) [3] [4] [15] that revolutionized the way communications systems developed. This model was developed as an International Standards Organization (ISO) effort. Another successful example of a reference model that has been developed by the ISO is one for archiving systems and is known as the Reference Model for an Open Archival Information Systems (OAIS) (ISO 14721:2003). In motivating and developing a reference model for agent systems, we draw heavily on these examples of existing successful reference models.

As these existing reference models demonstrate, reference models do not prescribe how functions and systems should be implemented. Instead, reference models provide the patterns of the solution for transforming vague notions into real-world implementation. Reference models simplify problem solving, to enable others to practice their discipline with a solid foundation. Software professionals, in particular use reference models to better understand abstractions and their potential for reuse.

3. WHY A REFERENCE MODEL?

Reference models are a necessity in a confusing, rapidly changing technology environment. As noted in [18], reference models are becoming more commonplace in fields of various human endeavors. The power of compelling reference models of knowledge can not be underestimated as a tool for technical leadership facilitating conclusions from a sound understanding of the problem space and solution domain. Thus, we argue that a reference model would be very useful tool to identify, assess and facilitate R&D and acquisition of agent technology for a wide variety of applications.

One particular application of agent-based systems is in military domains. Agent-based systems are being proposed to enhance and automate applications for collecting, presenting, storing, producing and sharing domain information. A future force is envisioned to be highly autonomous, modular, scalable, and flexible through the agentization of applications and services. This is especially true for complex system of systems (SoS). SoSs are large-scale, net-centric and include a variable mix of multi-department heterogeneous intelligent agents, humans-in-the-loop, and unmanned autonomous components and subsystems. Examples of complex SoSs in the US Army are the Army Battle Command System of the Current Force [1] and the Future Combat System of the Future Force [2]. Systems that would be found in a Joint Service (Army, Navy, Marine and Air Force) SoS are depicted in Figure 1.

Heterogeneous intelligent agents promise to enable conflict resolution between and among applications and services engaged in competition, negotiation, mediation and arbitration, to assist humans-in-the-loop, and to control unmanned autonomous systems and robots.

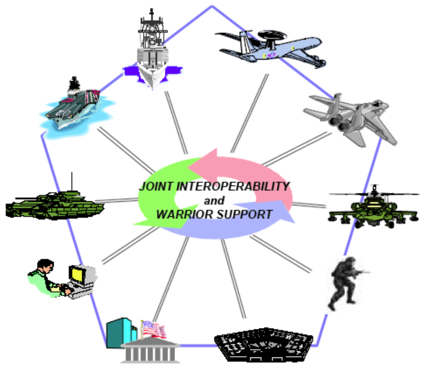

Figure 1: A Joint Service Battle Command is an intended future application for intelligent agent systems.

Agents are anticipated to play an important role in realizing dynamically varying mixed initiative capabilities to command and control manned as well as unmanned assets. As systems become increasingly complex, modularity as promoted by agent technology will become the key to reuse, scalability, and an open architecture. In addition, these design goals are a key to a manageable and affordable transformation from current to future force capabilities.

A reference model for intelligent agents would motivate the benefits of the technology by formalizing key concepts of both behavior and structure essential to enable agent technologies to reduce information collection, storage and sharing latency, workload, presentation overload and clutter for Battle Commanders and their staff. The recognition of the need and the investment to develop a unifying ontology for intelligent agent software across the domains of the systems in an enterprise-wide System of Systems (SoS) are shown to be crucial if not pivotal to the success of such SoS engineering efforts which are inherently multi-disciplinary and collaborative.

4. HOW TO CONSTRUCT A REFERENCE MODEL?

We take a quantitative and evidentiary approach to create a reference model based on a forensic analysis of existing agent-based systems. There are many agent frameworks available today: from companies, from academia and from the open source community. These frameworks have emerged from several large governmental and private research and development programs and have been used in the creation of many successful military and commercial systems. This suggests that a sensible approach towards building a reference model begins with examination of these existing systems and attempts to construct the model in a bottom-up fashion.

Reverse engineering (RE) is an ideal technique for examining and understanding existing systems. In RE, one obtains the software modules that comprise the subject systems by performing software analysis. By grouping, abstracting and querying data in different ways, information can be gleaned that simple observation of code may not find. Conventionally, RE is used to produce data that can allow for documentation and understanding of the software systems and for verification of any existing software documentation. For our purposes, we hypothesize that this data can also be analyzed and abstracted to obtain a reference model. In this section, we describe a specific RE technique we are using called *static analysis*. We also provide an overview of an additional RE technique that we will use in our future work: *dynamic analysis*.

4.1 Static Analysis

Static analysis is the analysis of software using its source code as the primary artifact. The system needs not be executing in order to obtain the appropriate data. Instead, source code or intermediate code is inspected to find the software modules, data structures, data flow, methods and metrics appropriate to the system.

This type of analysis yields many benefits, such as code-rewriting and vulnerability detection, but for purposes of our reference model, the primary goal is to use static analysis to produce a data repository from code that can be queried to find the primary software subsystems. This facilitates the transition from analyzing subject systems to identifying software modules that might fit the overall abstract system defined by the reference model. We describe two of the software analysis tools that we used: Bunch and JadeSniffer.

Bunch[1] is a clustering tool intended to aid the software developer and maintainer in understanding, verifying and maintaining a source code base, especially when when documentation of is nonexistent or outdated. The process proceeds in two steps. First, information from the source code is used to construct a directed graph that represents the key components, and the dependencies between these components. This directed graph, which we refer to as the Module Dependency Graph (MDG) of the system, has nodes as program units such as files or classes, and edges between the nodes as relationships between those modules such as function calls or inheritance relationships.

In the second step, Bunch uses the MDG to find a "good" clustering for the system. In particular, Bunch processes the MDG and produces as output a clustered MDG which includes all of the original source code and dependencies clustered into logical subsystems. Bunch can also use pre-defined clusters to measure or improve the quality of the system's clustering. The output from Bunch can be visualized by several popular graph visualization tools.

The Jade Sniffer[2] is a tool developed specifically to analyze Jade multi-agent systems. Jade is a FIPA-compliant agent framework, so a tool of this type is helpful to analyze behavior patterns and relate them to FIPA-compliant systems. This is accomplished by tracking messages between agents at the Virtual Machine level. The Sniffer itself claims to be a FIPA-complaint agent, like the Jade framework it analyzes. Using this tool together with Jade's logging services, we hope to produce sequence diagrams and message history that occurs between the agents; this provides a detailed view of the agent system behavior from a dynamic analysis perspective.

4.2 Dynamic Analysis

Dynamic analysis also collects data on software systems, but it does so by inspecting the system during execution. This analysis can vary widely by implementation, but one approach is to build a data repository of program behavior. This repository holds information on data flow, object instantiation, the call graph, interprocess communication, network or filesystem I/O activity, and so on. This analysis assists the production of the reference model by providing more sophisticated justification than static analysis alone. For example, dynamic analysis inspects the system as it is running and often breaks the system down into "features", e.g., code components that are actually used to provide some behavior. Moreover, dynamic analysis can obtain data on behavior-specific aspects of the system such as threading and I/O, that could not be found using static analysis. Finally, dynamic analysis can assist in cases where source code is not available for static analysis to be performed.

[1]http://serg.cs.drexel.edu/bunch

[2]http://jade.tilab.com/doc/tools/sniffer/

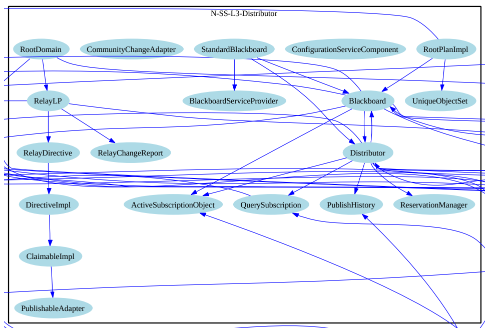

Figure 2: Diagram of Cougaar's Blackboard Distributor obtained using the Bunch reverse engineering tool.

5. EXISTING AGENT FRAMEWORKS

We describe three of the agent frameworks we have investigated to date: Cougaar, Jade and RETSINA. For each framework, we describe several of their functional concepts which were discovered and verified through a combination of static analysis of source code and accompanying documentation.

5.1 Cougaar

Cougaar (Cognitive Agent Architecture)[3] is a Java-based architecture for the construction of large-scale distributed agent-based applications. It was developed over two DARPA-sponsored research programs spanning eight years of effort. The goal of Cougaar is to provide developers with a framework to implement large-scale distributed multiagent applications with modular agent features and requiring minimal consideration for the underlying architecture and infrastructure. Below we describe a few of the key functions of Cougaar.

- *Directory Services:* Each agent is provided a unique name. This name is often similar to a network domain name based on the category and type of role that agent will have (though this is simply by convention). Cougaar provides a White Pages function that maps agent names to their locations. A Yellow Pages was also introduced to handle searches for agents and their locations based on their attributes or capabilities. Yellow Pages is built on top of UDDI, and a number of servers can provide the distributed Yellow Pages service.

- *Messaging:* Cougaar features a distributed blackboard, in which agents in a particular domain communicate with each other. Figure 2 shows a diagram obtained from applying the Bunch reverse engineering tool to the source code of Cougaar's blackboard component. In Cougaar, a software developer creates plugins that use a common language and protocol. These plugins publish objects to the blackboard, and receive objects from the blackboard according to an agent's subscriptions. To facilitate transmitting data to and from other agents, relay attributes are stored to show what information is desired from and appropriate for which agents. Alternatively, instead of storing actual recipient addresses, an agent can address a message based on attributes of the agent within the community.

- *Conflict Management:* To facilitate conflict management, an agent within the Cougaar architecture can form a group with other agents that share a similar goal or function. These

[3]http://cougaar.org

communities together form additional communities or a society. When the Cougaar society is active, new nodes may dynamically join the society, and may leave as well. To coordinate activities, nodes create functional relationships such as producer/consumer (service-based) relationships or superior/subordinate (delegation based) relationships. In order for agent communication to occur, there must exist established relationships between them.

5.2 Jade

Jade[4] is a Java based agent platform that is built upon a number of FIPA standards, including Agent Management Specification [9], Agent Communication Language (ACL), and the ACL Message Structure. Thus, Jade is structured as a FIPA compliant agent platform in which agents, the agent management system and the directory facilitator interact independently with a common message transport system.

- *Directory Services:* Agents are registered, named, and looked up via FIPA standard implementations. An Agent Management System (AMS) is provided for name lookups, and a Directory Facilitator is provided for agent lookups. Based on the location of the agent (local or remote), an agent name lookup will find the physical location of the agent and the Jade subsystem will forward the message over the appropriate I/O channels.

- *Messaging:* A peer-to-peer conversation occurs between agents. This is implemented via a FIPA standard, in which an Initiator and Responder agent communicate with each other through message passing. Feasibility Preconditions are met before the message is sent, and a Rational Effect are defined as to what the Initiator agent should expect (but not assume) upon receipt and processing of the message by the responder. Agents can inform each other or make requests, and they can be responded to with a not-understood, refusal or agreement response, and so on. In addition, agents may subscribe to messages provided by other agents based on their content.

- *Mobility:* Jade supports migration of agents to other hosts. This is done by copying or serializing the agent across the network. The jade-mobility-ontology is provided to ensure that agents are always aware of their physical location. At a minimum, agent mobility is described by an ontology that contains the agent's name and destination. It can also contain an agent profile which contains the OS that the agent resides / will reside within. It can also specify the system and language of the agent.

5.3 RETSINA

RETSINA[5] is developed by the Robotics Institute of Carnegie Mellon University. RETSINA is developed in C++ but is platform independent. Agent systems can be run on a number of different platforms and hosts (including handheld systems) and can be implemented in a number of languages.

- *Directory Services:* The ANS (Agent Name Service) is provided to map an agent by name to a physical location. This location is not bound to a particular physical medium such as TCP/IP, but instead can be any location that physically defines a method to contact the agent. In addition to a naming service, attributes and services of each agent are cataloged when it joins the society by *middle agents* called "Matchmakers." These matchmakers provide agent lookup based on a service query, and can also send notifications when new agents join the society that meet the specified criteria.

- *Messaging:* Messages are either direct or multicast to agents based on their services or attributes. Message passing can be synchronous or asynchronous, and can occur between multiple agents simultaneously via a threaded architecture. RETSINA also separates the physical communications model from the protocol, language and ontology. KQML is used as the agent communication language in RETSINA. As part of the language, each message passed between agents contains both the content and the structure (ontology, language, sender, conversation, etc.) of that message.

- *Semantic Interoperability:* A shared dictionary defines the meanings of content of KQML messages.

6. TOWARDS A REFERENCE MODEL

We describe a preliminary, partial reference model for agent-based systems. The portions of the model we present include a high-level view organized as a set of layers and a functional decomposition of an agent-based system.

6.1 Layered View of an Agent-Based System

An **Agent-Based System** is comprised of agents and their supporting framework and infrastructure which provide fundamental services and operating context to the agents. Our model defines framework, platform and host layers, which mediate between the agents and the external environment. This layered model can be organized vertically as shown in Figure 3. Each layer is described as follows:

- The **Agents** layer consists of agents that perform computation, share knowledge, interact and generally execute behaviors in order to achieve application level functionality. We make few assumptions about the Agent layer except to state that agents are *situated* computational processes—instantiated programs that sense and effect an environment in which they exist. We make no assumptions about the internal processing structures of an agent. An agent could be built with a complex cognitive model or it could be a simple rule-based system. Given the vast array of tasks envisioned for agent systems, it is not the role of a reference model to limit or define what an agent is.

- The **Framework** layer provides standardized functionality specific to supporting agents. A framework typically provides support services such as conflict management, directory and naming services, security, and agent administration services such as monitoring and allocating resources to the executing agents. The major benefit of employing an agent framework is to provide standardization of services and functionality to agents that exist within the framework. The end result is that agents written within a particular framework are easily interoperable with one another. In other agent-based systems, the framework may be trivial or merely conceptual, for example if the services are merely a collection of system calls or are compiled into the agents themselves.

- The **Platform** layer provides more generic computing infrastructure. The platform contains the software components that

[4]http://jade.tilab.com/

[5]http://www.cs.cmu.edu/ softagents/retsina.html

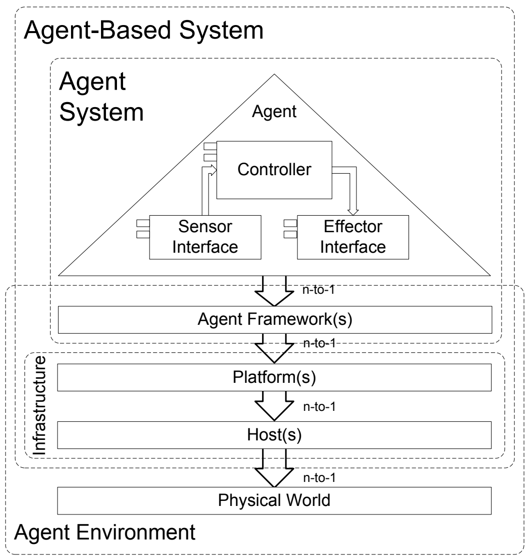

Figure 3: An Agent-Based System is made up of layers.

are available to the agent framework, but are not packaged along with it. Elements such as operating systems, user interface libraries, database software, device drivers, and message transport or socket libraries are in this layer. These services are often provided by third parties and it is unlikely that an agent-based system will provide its own implementation of the platform functions.

- The **Host** layer contains the hardware devices on which the above layers operate. This layer includes not only the physical computing devices such as a desktop computer or handheld device, but also the hardware that provides interaction with the environment such as robot sensors and effectors, cameras, displays, GPS receivers, etc.

- **Environment** is the physical world in which the agent-based system exists and operates.

Each layer can support many entities from the layers above it—many agents may execute on a single framework, many frameworks may execute on a single platform, and so on.

6.2 A Functional View of an Agent-Based System

This section presents a view of an agent-based system as a set of abstract functional concepts that support overall system execution. For example, *administration* and *security* are two abstract functional concepts we define. Our use of the rather abstract term "concept" here is deliberate. The more concrete (and perhaps more familiar) UML term "component" could be used, but we wished to reflect the idea that a function often does not correspond directly to what we might think of as a component, i.e., a clearly delineated piece of the system. Similar to the aspect-oriented software engineering view, we view a functional concept as something that emerges out of complex interactions between pieces of software and hardware located in different layers of the agent-based system.

It is important to note that we make few prescriptions about whether and how each functional concept is implemented. The way in which functional concepts are instantiated may vary significantly in structure, complexity and sophistication across different agent-based system implementations. Indeed, some agent-based systems may not even possess some of the functional concepts we will describe. Our aim here is to describe what the function is in abstract terms so that one can determine if the function exists in a given system, or to verify its existence if it is claimed to exist within a given system.

A functional concept is described through a brief definition and a set of primitive operations that implement the function which we call a Process Model. We define Administration, Security, Conflict Management, and Messaging concepts. Other functional concepts that are part of the model but not described here are Mobility, Semantic Interoperability, Directory Services, and Logging.

6.2.1 Administration

Definition: *Administration functionality a) facilitates and enables supervisory command and control of agents and/or agent populations and b) allocates system resources to agents. Command and control involves instantiating agents, terminating agents, and inspecting agent state. Allocating system resources includes providing access control to CPUs, User Interfaces, bandwidth resources, etc.*

Administration functionality may be implemented in various ways. For example, the framework may perform all the administration functions directly, or there may be (multiple) agent(s) in the agent layer that perform agent administration functions by commanding and controlling other agents, or there may be elements of both approaches in a given system. For convenience of exposition below, we will use the term "administrator" to encapsulate all the administration functions although administration functions may not necessarily implemented with a single administrator.

To further facilitate the exposition of the following process model, consider as an example a hypothetical system that uses agents to monitor message traffic on a communication network. The number of agents required to perform adequate monitoring may be contingent upon the complexity of the network topology or the priority of monitoring relative to other system goals. As both the network topology and priority changes, an administrator is employed to manage the network monitoring agents.

Process Model: Agent administration functionality is described by the following set of processes:

- *Agent Creation* The act of instantiating or causing the creation of agents. In our example, the administrator may determine that there are too few network monitoring agents to adequately maintain a minimum level of security. Therefore, new network monitoring agents will be created.

- *Agent Management* The process by which an agent is given an instruction or order. For example, if it is determined that the greatest security threat is over HTTP traffic, the administrator may request that the network monitoring agents focus their analysis on HTTP traffic.

- *Resource Control* The process by which an agent's access to system resources is controlled. For example, the administrator may determine that security is of less priority than CPU usage. Therefore, it can reduce the available CPU time of the network monitoring agents.

- *Agent Termination* The process by which agents are terminated (*i.e.*, their execution is permanently halted). For example, the administrator might determine that there are too many network monitoring agents and decide to remove those in saturated regions of the network.

6.2.2 *Security*

Definition: *The purpose of security functionality is to prevent execution of undesirable actions by entities from either within or outside the agent-based system while at the same time allowing execution of desirable actions. The goal is for the system to be useful while remaining dependable in the face of malice, error or accident.*

Process Model: Security functionality is described by the following processes.

- *Authentication* A process for identifying the entity requesting an action. Common examples include user name/password credentials and use of public/private keys for digital signatures.
- *Authorization* A process for deciding whether the entity should be granted permission to perform the requested action. A common example in file system security is maintenance of a permission list for each file, which specifies the allowable actions for a given user. Another example includes a web server denying requests when it gets overloaded.
- *Enforcement* A process or mechanism for preventing the entity from executing the requested action if authorization is denied, or for enabling such execution if authorization is granted. A common example for preventing access to information is to encrypt it. Permission to access the information is granted by providing the entity a decrypted copy or providing the entity the means to decrypt it, e.g., the encryption key.

Some general technologies for achieving security include authorization models and mechanisms; auditing and intrusion detection; cryptographic algorithms, protocols, services, and infrastructure; recovery and survivable operation; risk analysis; assurance including cryptanalysis and formal methods; penetration technologies including viruses, Trojan horses, spoofing, sniffing, cracking, and covert channels.

6.2.3 *Conflict Management*

Definition: *Conflict management functionality facilitates and enables the management of interdependencies between agents activities and decisions. The goal is to avoid incoherent and incompatible activities, and system states in which resource contention or deadlock occur.*

As an example, a framework may allow designation of superior/subordinate relationships between agents and provide generic conflict resolution services based on these relationships. The Cougaar framework does this. Similarly, a framework may provide a multiagent task planning language, such as TAEMS [17], that can be used to reason about the interactions between agent actions and to detect plan conflicts.

Process Model: Conflict management functionality is described by the following processes.

- *Conflict avoidance*. A process or mechanism for preventing conflicts. Examples of such processes include multiagent planning algorithms (both on-line and off-line) that take care to produce action plans that do not have conflicts.
- *Conflict detection*. The process by which it can be determined when a conflict is occurring or has occurred. One example includes a plan execution monitoring algorithm that is able to sense when the actions of agents are in conflict. Another example includes performing logical inference over different agents beliefs to determine when they are inconsistent with one another.
- *Conflict resolution*. The process through which conflicts between agent activities are resolved. Negotiation, mediation and arbitration are common mechanisms for facilitating conflict resolution.

Some general technologies for conflict management in agent systems include argumentation and negotiation, distributed constraint reasoning, game theory and mechanism design, multiagent planning, norms, social laws and teamwork models.

6.2.4 *Messaging*

Definition: *Messaging functionality facilitates and enables information transfer among agents in the system.*

This concept is associated specifically with the mechanisms and processes involved in exchanging information between *agents*. Although information exchange via messages can and often does occur between other parts of the system, for example between an agent and its framework, between frameworks, between a host and its platform, etc., such information transfer is not included because it is in a sense at a lower level. Our concept of messaging is at a higher level than that associated with network traffic or interprocess communications.

Messaging involves a source, a channel and a message. Optionally, we can designate a receiver but we admit models in which messages do not have a specific intended receiver. For example, signaling in the environment like smoke signaling, a light flashing morse code, etc, are examples of messaging where there is no designated receiver.

Process Model: The functionality is described by the following processes.

- *Message Construction* The process through which a message is created once a source agent has determined it wishes to deliver a particular message chosen from a finite or infinite set of messages. No commitments are made here in regard to the form, structure or content of a message. For the purposes of this model it is sufficient to discuss messages as an abstract object. The information to be delivered is simply the fact that a particular message was chosen among the set of all messages that could have been chosen.
- *Naming and Addressing* A mechanism for labeling the message with its intended destination or route. Directory white page services are a common mechanism to facilitate this function. Broadcast, multicast and group messaging also all fit within this model.
- *Transmission* The actual transport of the message over the channel. This may be a one-shot transmission or a continuous stream. One common model is messaging an agent on another host involves going through the platform to the host's network hardware, then out into the environment (via wire or air), and back in symmetrically to the receiver.
- *Receiving* The process for acquiring the transmitted information so that is usable by the receiver. This may be as simple as pulling the message off of a queue or more elaborate, e.g., going through a translator.

Some other areas of interest in messaging functionality include notions of best effort delivery, QoS and guaranteed delivery/timeliness.

7. RELATION TO EXISTING STANDARDS

Existing standards for agent-based systems can be used concurrently with a reference model. A reference model describes the abstract functional elements of a system. A reference model does not impose specific design decisions on a system designer. Existing standards such as the Knowledge Interchange Format (KIF), KQML, those developed by the Foundation for Intelligent Physical Agents (FIPA), and even some non-agent specific standards have a place in the reference model and in fact are needed by the reference model. The reference model states that a component needs to exist, the standards tell someone how to design it.

A reference model also does not define an architecture. A reference model can drive the implementation of multiple architectures in the same way that an architecture could drive multiple designs, or a design could drive multiple implementations. In the same way, this reference model is not an attempt to impose (or even define) standards for the implementation of an agent-based system. It is an attempt to describe the abstract layers and concepts that may exist in such a system.

We have classified existing standards for agent-based systems roughly into five categories:

- *General* standards that are higher-level, and less specific. Examples include the FIPA Abstract Architecture [5] and FIPA Agent Management [9] standards.

- *Communication* standards that deal with both the actual logistics of communications and the information being communicated. Examples include KIF [14], KQML [16], FIPA ACL Message Structure [8], and FIPA Message Transport Service.

- *Coordination protocol* standards for coordinating actions of agents. Examples include the FIPA Contract Net Interaction Protocol [10] and FIPA Recruiting Interaction Protocol [13].

- *Representation and Encoding* standards for encoding messages for transmission. Examples include FIPA Bit Efficient ACL [6] and FIPA XML ACL [7].

- *Ontologies* standards for knowledge representation. Examples include FIPA Device Ontology [11] and FIPA QoS Ontology [12].

8. FUTURE WORK

Our goal is to continue refinement of the reference model in consultation with the broader agents community. We will be making available an on-line interactive discussion forum, similar to a Wiki, for facilitating further development of the ABSRM.

Acknowledgments

We thank Christopher Dugan, Moshe Kam, Joseph Kopena, Robert Lass, Jeff Salvage, and Evan Sultanik for significant contributions to this work. We also thank Tedd Gimber, Bernard Goren, Michael Huhns, James Odell, Randy Reitmeyer and Todd Urness for useful discussions.

9. REFERENCES

[1] Army battle command systems (abcs). http://www.fas.org/man/dod-101/sys/land/abcs.htm.

[2] Future combat system (fcs). http://www.fas.org/man/dod-101/sys/land/fcs.htm.

[3] *ISO 7498(Draft): OSI Basic Reference Model*. 1984. Information Processing Systems.

[4] Uyless Black. *OSI: A Model for Computer Communication Standards*. Prentice Hall, 1991.

[5] Foundation for Intelligent Physical Agents. Abstract architecture, December 2002. http://www.fipa.org/specs/fipa00001/.

[6] Foundation for Intelligent Physical Agents. Acl message representation in bit-efficient encoding, December 2002. http://www.fipa.org/specs/fipa00069/.

[7] Foundation for Intelligent Physical Agents. Acl message representation in xml, December 2002. http://www.fipa.org/specs/fipa00071/.

[8] Foundation for Intelligent Physical Agents. Acl message structure specification, December 2002. http://www.fipa.org/specs/fipa00061/.

[9] Foundation for Intelligent Physical Agents. Agent management specification, December 2002. http://www.fipa.org/specs/fipa00023/.

[10] Foundation for Intelligent Physical Agents. Contract net interaction protocol specification, December 2002. http://www.fipa.org/specs/fipa00029/.

[11] Foundation for Intelligent Physical Agents. Device ontology specification, December 2002. http://www.fipa.org/specs/fipa00091/.

[12] Foundation for Intelligent Physical Agents. Quality of service specification, December 2002. http://www.fipa.org/specs/fipa00094/.

[13] Foundation for Intelligent Physical Agents. Recruiting interaction protocol specification, December 2002. http://www.fipa.org/specs/fipa00034/.

[14] M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format, Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Department, Stanford University, Stanford, CA, USA, June 1992.

[15] B. N. Jain and A.K. Agrawala. *Open Systems Interconnection: Its Architecture and Protocols*. McGraw-Hill, 1993.

[16] Yannis Labrou and Tim Finin. A Proposal for a new KQML Specification. Technical Report TR CS-97-03, CS and EE Department, U. of Maryland Baltimore County, Baltimore, MD, USA, February 1997.

[17] V. Lesser, K. Decker, T. Wagner, N. Carver, A. Garvey, B. Horling, D. Neiman, R. Podorozhny, M. NagendraPrasad, A. Raja, R. Vincent, P. Xuan, and X.Q. Zhang. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *Autonomous Agents and Multi-Agent Systems*, 9(1):87–143, July 2004.

[18] Raphael Malveau and Thomas J. Mowbray. *Software Architect Bootcamp*. Prentice Hall, 2001.

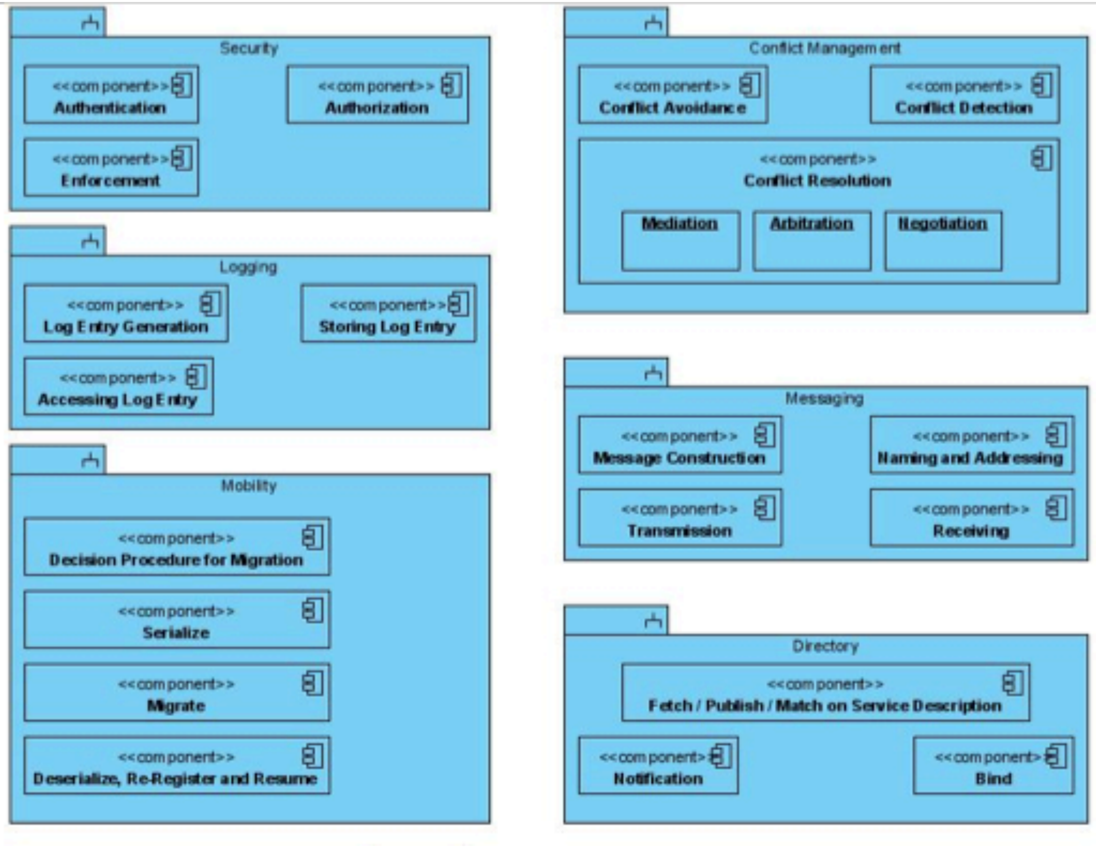
Security
<<component>> Authentication
<<component>> Authorization
<<component>> Enforcement
Logging
<<component>> Log Entry Generation
<<component>> Storing Log Entry
<<component>> Accessing Log Entry
Mobility
<<component>> Decision Procedure for Migration
<<component>> Serialize
<<component>> Migrate
<<component>> Deserialize, Re-Register and Resume
Conflict Management
<<component>> Conflict Avoidance
<<component>> Conflict Detection
<<component>> Conflict Resolution
Mediation
Arbitration
Negotiation
Messaging
<<component>> Message Construction
<<component>> Naming and Addressing
<<component>> Transmission
<<component>> Receiving
Directory
<<component>> Fetch / Publish / Match on Service Description
<<component>> Notification
<<component>> Bind

Figure 4: A Functional View of an Agent-Based System. selected functional concepts shown here in standard UML notation.