A SOFTWARE AGENT TOOLKIT FOR EFFECTIVE INFORMATION PROCESSING IN THE BATTLE COMMAND DOMAIN

Mr. Tedd W. Gimber*
Global InfoTek Inc.
Reston, VA 20191

Dr. Michael N. Huhns
Global InfoTek, Inc. and University of South Carolina
Columbia, SC 29208

ABSTRACT

Commanders of combat units have traditionally desired as much information as possible to aid them in making key decisions. Ironically, we have reached the stage where there is now too much information available. A commonly proposed solution is to utilize software agents to collect information, select what is useful, and deliver it to the commander. By their nature, software agents are active, distributed, intelligent, and persistent computations, so they can enable the best information to be made available when and where it is needed. Unfortunately using such software agents effectively requires computer programming expertise not typically available to a battlefield commander. Global InfoTek Inc (GITI) is currently assembling a suite of agent development tools that will enable programmers to develop software agents that can be controlled and manipulated by the commanders in the field.

1. INTRODUCTION

Battlefield commanders have historically faced an insidious enemy, one different than those with weapons facing them from across the field of battle. That enemy is information – too little information; too much information; incomplete information; bad information; information delivered too late; information delivered to the wrong person. This enemy is present not only before the first shot is fired in a battle, but also during the fighting and after the action has completed.

1.1 Historical Example

In July of 1863, Gen. Robert E. Lee faced the information enemy and paid dearly. His forces were massing for what was to be a decisive Confederate victory in Pennsylvania; a victory that Lee hoped would end the Civil War. Instead, Gen. Lee was blind and deaf because his "eyes and ears" were missing. Gen. J.E.B. Stuart, the flamboyant leader of Lee's cavalry, failed to maintain communication with the main body of the Confederate forces and failed to maintain his observation of the Union forces. Thus Lee was unaware of the size and position of the Union forces as they approached him near Gettysburg. He had some intelligence reports from scouts in the field, but without input from his trusted cavalry leader, he could not be certain if those reports were accurate.

1.2 Modern Scenario

Today, US Army commanders directing their forces in a combat environment face information overload. New and emerging technologies under the Future Combat System program, such as sensor networks and autonomous reconnaissance vehicles, greatly increase the amount of information available to the commander. However, timely and proper management of the vast information provided by these networked systems is essential to the success of network-centric missions. Consider this scenario:

> Data from a remote sensor network indicates possible insurgent activity in a valley obscured by forest cover. UAV images confirm the presence of an encampment, but the images are insufficient to determine its exact purpose. A commander decides to utilize a Special Operations reconnaissance team to observe the activity of the camp first hand. In order to plan the operation, data from the sensor network must be quickly examined to determine the suspected size and movement of the forces in the camp. Digital maps must be consulted to decide upon the safest entry and exit points for the team as well as the best route to the valley. Weather data from six different systems is available and has to be considered to determine when the team can operate and under what conditions they will operate. Finally, recent satellite and UAV images need to be analyzed and related to the digital maps to determine the exact location of the encampment. All of this information is constantly changing – weather conditions can rapidly deteriorate, enemy movement may be spotted, and orders from superiors may be modified. The operation planners must stay on top of all of this information to create and update the operation plan.

This scenario illustrates the vast amount of information available to commanders and operation planners that is derived from multiple sources on different heterogeneous systems. Contrasted with the historical example, it is easy to think that we have progressed from having too little information to too much information. But in fact it is not the quantity of information that matters, but it is the quality. The commander needs the right information, at the right time, to make an intelligent decision. It really is a problem of information management.

1.3 Software Agents to the Rescue

A relatively recent solution to this problem is the use of intelligent "agents" that are able to act not only on behalf of their human "masters," but also to take the initiative in gathering information and presenting it in a usable fashion. Research into artificial intelligence during the 1960's and 1970's helped form the basis for the concept of software agents. A vision for such intelligent creatures, crafted in software, first appeared in John McCarthy's seminal work [McCarthy 1979] and has later been termed the *intentional stance*: the philosophical view that cognitive concepts can be ascribed to any physical system and that it is beneficial to do so for complex systems.

This vision has been refined into the current definition that considers agents to be autonomous, distributed, active, persistent, and communicating software components. Consistent with this definition, Tim Berners-Lee has promulgated the notion of software agents working for their human masters to gather information from the World Wide Web [Berners-Lee 2001]. The agents work together to exchange information, make appointments, and generally improve life for humans by using content present on the Internet, thereby rendering the Web as accessible for machines as it is for humans.

These visions present a target for computer scientists, information specialists, and researchers to work towards. And much has been accomplished in the realm of software agents. For example, they are overseeing the supply chains for multinational corporate enterprises, mining the Web for information for intelligence analysts, and managing billion-dollar auctions for energy resources. Yet two major problems still face the potential user of the agents: *interoperability* and *ease of use*.

There are currently several agent systems and agent frameworks in use as research prototypes. But most of the frameworks are built for a single purpose and do not work well together, nor do they work well with legacy software applications. In other words, they lack the ability to interoperate or enable interoperation among other systems. Interoperability is a critical characteristic of any network-based system, and that is especially true in current, emerging, and future DoD systems.

The second major shortcoming of almost all current agent systems is ease of use. Many of these systems sport wonderful Graphical User Interfaces (GUI's) for monitoring and controlling agents. Yet the actual creation and programming of the agents remains firmly in the realm of the computer scientist and software engineer.

It was in part this lack of interoperability and ease of use that lead US Army Communications-Electronics Research, Development, and Engineering Center (CERDEC) to encourage the investigation of the use of a common agent framework that could be combined with current and future agent development tools to create an environment for software agent development. GITI performed that research under SBIR A05-078. During that task we created the concept of an **Agent Development Toolkit**.

The Agent Development Toolkit, shown in Figure 1, consists of a Common Agent Framework, an Agent Factory, and Modifiable Agents. These components are supplemented by additional tools and libraries to provide interoperability with other systems, including legacy systems. The agents are created by software developers using the Agent Factory, but the end users can manipulate and even reprogram those agents via an easy to use and intuitive user interface.

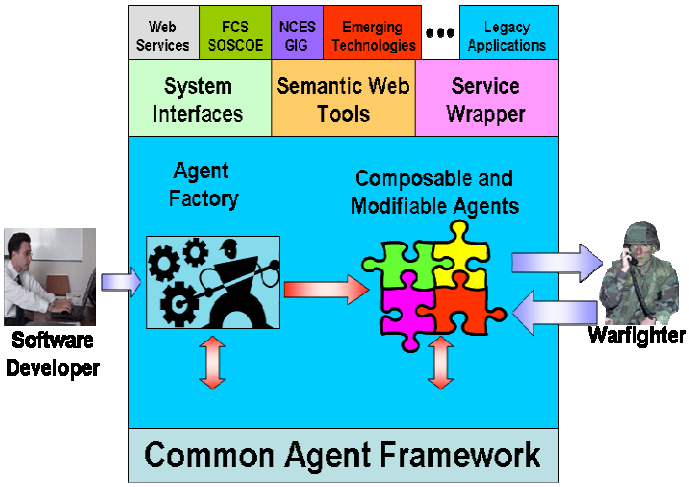

Figure 1 - The Agent Development Toolkit permits non-programmers in the field to access critical information by composing and modifying agents previously created by software developers.

2. THE AGENT DEVELOPMENT TOOLKIT

We conducted our work on the concept of this Toolkit during the Phase I effort of the SBIR task. Our efforts began with the concept of a Common Agent Framework, as define in the Agent Systems Reference

Model (ASRM) [Mayk 2006]. The ASRM was produced by a team from Drexel University as part of their work in support of the Intelligent Agent Integrated Product Team (IPT) working group. That working group is chaired by Dr. Israel Mayk of the US Army Research, Development, and Engineering Command (RDECOM). The working group has analyzed numerous agent-based systems and formulated an understanding of what a typical agent system looks like. The result was the first ever reference model for an agent system. The ASRM is a guideline for what functionality an agent framework should contain. We used the ASRM as the basis for our plans for a Common Agent Framework. The Framework is the foundation component of the Agent Development Toolkit, and one of the first tasks we undertook was determining the requirements of that Framework.

2.1 The Common Agent Framework Requirements

We documented 60 core requirements in 9 different categories. Those categories are: Administration, Security, Mobility, Conflict Management, Messaging, Logging, Interoperability, Directory Services, and Non-Functional requirements. These were based on the major functional areas defined in the ASRM, and represent the core requirements for a Common Agent Framework. Let's take a look at these categories to gain a better understanding of the role they play within the Framework.

The categories of Administration, Conflict Management, Directory Services, and Logging all relate to the management of agents running within the framework. The Administration category describes how agents are started and stopped. Conflict Management deals with agent interaction and the resolution of disputes between agents. For example, two agents might be tasked with obtaining information from sensor, but the sensor interface does not permit concurrent access. Conflict Management determines which agent is given the access, which is denied, and any mediation relating to that denial. The Directory Services category is concerned with how agents are able to find each other in the system. Finally, Logging deals with capturing the activities of the agents, the messages they sent, and so on. This is useful for understanding agent interactions and is a critical part of security.

Security obviously is a major concern to any user of a computer system and thus represents a major requirements category. Networked systems operating within the DoD are some of the most critical information resources in use today, and require extraordinary security protection. Most agent frameworks do not provide ample security, in fact most provide no real security. This is because these systems are being used for research and development. However the Common Agent Framework must address security concerns straight on if there is any hope of using our Toolkit in a Battle Command environment. The Security Category requirements currently cover the basic security needs. These will be augmented by specific security requirements specified as part of Certification and Accreditation process. The basic requirements call for agent identification, authentication, authorization, as well as encryption for messages sent between agents.

Closely related to security is the Mobility requirements category. In many agent systems, agents are able to move from one computer to another. This may seem an odd thing for an agent to do at first, but there are times when agent mobility makes sense. One example is an agent that moves to a different computing platform to be closer to a data source. It is more efficient for the agent to read a database on the database server than to perform queries over a network. The requirements for agent mobility cover how agents may move, how the decision is made, and the security of agent migration.

Of course agents tend to spend more time talking to other agents than they do moving from one computer to another. The category of Messaging specifies how agents communicate with each other. We purposely did not dictate a specific message format such as the Foundation for Intelligent Agent (FIPA) Agent Communication Language (ACL) nor did we specify the means of communication (e.g., Java Message Service). Instead our requirements are very broad and are geared to fostering interoperability by agents using the Common Agent Framework. We require that the framework provide support for synchronous and asynchronous messaging. We also require that the framework allow binary data (such as video) in addition to basic text messages. We require support for XML based messages, but do not require that all messages be sent in XML format. This is to provide a maximum amount of flexibility while still providing interoperability.

The Interoperability requirements category focuses on the ability of users of the Common Agent Framework to exchange information with other agent frameworks as well as non-agent systems. Defining a common messaging format is not sufficient to provide a means of communication between two different agent frameworks. It is necessary to provide a common mechanism for exchanging messages. For example, agents running under different systems could use a centralized database to store messages. In order to achieve interoperability with non-agent systems, we specify requirements to provide an interface to Web services. We also require support for semantic technologies, specifically Resource Description Format (RDF), RDF Schema (RDFS), and Web Ontology Language (OWL). Finally we require a means of wrapping legacy applications to provide a communication channel to the Common Agent Framework.

We also made several non-functional requirements for the Common Agent Framework. We specified that the Framework would function under Microsoft Windows (2000, XP, 2003, etc) as well as the UNIX and Linux platforms. We also specified that the Framework be Java based, specifically that it run under the Java 2 Standard Edition (J2SE) version 1.4 or above. We made Java a standard for the Framework for the following reasons:

- Java has numerous built in security features
- Java has built in support for Remote Method Invocation (RMI) which simplifies distributed processing and mobile agents
- There is a large community of Java developers to provide expertise and support for the Framework
- The majority of agent systems are written in Java
- Java based systems are easily ported to new platforms

Having documented the basic requirements for a Common Agent Framework, we considered how agents would be developed to run on that framework.

2.2 Requirements for an Agent Development Environment

The Agent Development Environment (ADE) encompasses the Agent Factory, as well as the various "tools of the trade" employed by the typical software developer. These include compilers, editors, version control, and other specialized tools. But what makes the ADE special is that it is intended to be used specifically for the development of software agents.

A large number of the requirements that we specified for the Common Agent Framework are repeated for the ADE. For example, the requirement that the Framework provide a means of sending XML formatted messages between agents naturally leads to the requirement that the ADE support the use of XML in general (via libraries perhaps) and that it can create agents that can send XML messages specifically. This is true for most of the agent related requirements previously discussed so we will not repeat those here. Instead we will focus on the more interesting requirements that we determined during our analysis of the ADE.

Probably the most important requirement that we specified was that the ADE will be based on a popular Integrated Development Environment (IDE) such as NetBeans or Eclipse. We further specified that the IDE is to be Java based and run on a variety of platforms, for the reasons laid out in the previous section. We also stipulated that the IDE itself be open source and freely available. This was a specific requirement provided by CERDEC and one that we fully supported. We did not want to base our agent development environment on a product that had expensive licenses which would tax already constrained budgets. We also specified that the IDE needed to be fully extensible via add on modules or "plug-ins" so we could create specialized features for the creation of software agents, namely the Agent Factory.

The Agent Factory is the centerpiece of the Agent Development Environment. We specified in our requirements that the Agent Factory will be a wizard-driven, GUI-based tool for simplifying the creation of software agents. The Factory provides a means for programmers as well as non-programmers to create agents. We do not envision that the commander in the field will use the Agent Factory, but we do believe that the Factory could be used by non-programmers to create agents requested by a commander. This will dramatically reduce the amount of time necessary to develop and deploy software agents in the future.

To support the use of the Agent Factory, we specified that the ADE will include industry standard tools for version control and project building, and those tools will be integrated into the IDE. Two examples of such tools are Subversion for version control and Apache Ant for automating the build process. Both are Java based, open source, freely available tools that can be integrated into an IDE. This will permit the Agent Factory, as part of the ADE, to create agents that are stored in a centralized repository and packaged for deployment as Java Archive (JAR) files.

We also added two additional requirements to the ADE that we feel are critical to the success of developers creating agents: built-in help and working examples. These seem minor, but our experience and review of existing agent development products convinced us these are vital and often overlooked features.

The built-in help functionality is not simply the "search for keywords" type of help available in most modern applications. Instead we provide direct assistance in tasks the user performs (e.g., a Wizard guiding the creation of an agent) as well as context sensitive help such as you get from an IDE like Eclipse when trying to remember the syntax for a "case" statement in Java.

The inclusion of functioning, interesting, and instructive examples in the ADE is likewise important. Such examples are useful for understanding how complex technologies function. This feature allows users to begin with a working model that they can dissect and experiment with. During our review of agent systems, we were impressed with the examples included in Jadex as part of a tutorial. Jadex is a Java based agent framework that is used primarily for researching and experimenting with software agents. Included in the Jadex system are working agent demonstrations for Blackjack, robotics,

and puzzle solving. The robotics demonstration, for example, shows the use of agents to control virtual vacuum cleaners tasked with collecting trash in a room.

2.3 Recommended Tools

Once we completed documenting the key requirements for our Agent Development Toolkit, we focused on evaluating the key components we would need for the Toolkit. We knew that we would need an agent framework to build upon, an IDE for development of the agents, developers' tools for tasks such as version control, and tools or libraries for providing interoperability with other systems.

We used our core requirements to weed out products that were not suited to our needs. For example, when considering IDE's, we quickly set aside Microsoft's Visual Studio product because it does not run under the Linux operating system. We also added additional requirements when we found a particular feature that we felt was important to include. The interesting sample programs included in the Jadex tutorial are one such case.

The purpose for having a Common Agent Framework is that it will greatly simplify the problem of agent interoperability. With the goal of defining what is to be included in a Common Agent Framework, we evaluated numerous existing frameworks. This evaluation of frameworks was aided to a large extent by the survey included in the ASRM. We also looked at additional frameworks, not covered by the ASRM survey, to provide a broad review of technology.

We eventually focused on two different frameworks, the Cognitive Agent Architecture (Cougaar) and the Control of Agent Based System (CoABS) Grid. Both are Java based, and both began life as Defense Advanced Research Projects Agency (DARPA) programs. Review of each showed that they both meet the core requirements for a Common Agent Framework and are generally compatible with the functional areas set out in the ASRM.

We thus had to decide if we would use one of these two products, or if we would build a new Common Agent Framework using the best of all the frameworks we evaluated. In the end we opted to use the CoABS Grid as a starting point and then build upon it. GITI has been involved with the CoABS Grid from its onset and thus we are intimately familiar with it. During our research and evaluations we realized that neither the Grid nor any other single product provides a complete solution to the idealized Common Agent Framework. However, the Grid is lightweight and flexible, and makes an excellent starting point for the Common Agent Framework.

Next we began searching for a proper development tool on which to base our Agent Factory. As stated above, we decided to use a full-featured IDE for that development tool. We evaluated several IDE's including Microsoft's Visual Studio, Borland's JBuilder, NetBeans, and Eclipse. We quickly eliminated Visual Studio and JBuilder because they did not meet our basic requirements of being Java based and freely obtainable. This left us with NetBeans and Eclipse. Each is a solid IDE with features that permit extensions well suited to our Agent Factory. Each has a built-in tutorial plus wizards for performing common tasks (e.g., creating a new Java class). In the end we selected Eclipse because we deemed that it had more popular support (which we grant is a very subjective characteristic) and because Eclipse uses the Open Services Gateway initiative (OSGi) model for its runtime layer [Clayberg 2006]. The OSGi Alliance is a worldwide technology consortium that advocates a standard for a component-based integrated platform to assure interoperability among applications and service. We obtain additional interoperability by building upon this standard.

The Agent Factory consists of tools and libraries in addition to the Eclipse IDE. One tool that works well with Eclipse is Apache Ant. Ant, which was discussed earlier, is a tool for building Java applications and packaging them for distribution. Although Ant is not a scripting language, it can be used to aid in creating agents, by automating the steps needed to build an agent.

Another tool that the Agent Factory requires is a version control tool. This is needed to store versions of agents created by the Factory, so that other developers, and eventually end-users, can call upon that agent. For this we looked at two tools, the Concurrent Versioning System (CVS) and Subversion. Both are freely available, open source, and run on a variety of platforms, including Windows and UNIX. We have decided to include Subversion in the initial Toolkit because it is newer and is an improved version of CVS.

Finally we reviewed numerous tools and libraries that provide interoperability with other systems. We have not specified any tools or libraries specifically because interoperability technologies are still emerging and changing. We have instead included Web services, Semantic Web technologies, and general mechanisms for wrapping legacy systems as key components of the interoperability functionality of the Toolkit.

Web services are intended as a means to support interoperability among multiple heterogeneous platforms (e.g., Windows and Linux) over a common network, such as the Internet. They do this by using a standards based approach to provide common interfaces to the services. The basic language for Web services is the Extensible Markup Language (XML), which is used for basic data exchange. Messaging between the services is typically

done using a standard for messaging such as Simple Object Access Protocol (SOAP). Another standard, the Web Services Description Language (WSDL), provides a common interface for describing a Web service. To advertise and learn about the availability of a Web Service, a directory is used, which is commonly based on the Universal Description, Discovery, and Integration (UDDI) directory protocol.

A rapidly emerging concept for providing interoperability across networks is the Semantic Web. The goal of the Semantic Web is to add meaning to the data available via the Web, including Web services. For example, if we have a description of an individual as a "father," we typically understand that the person is a male who has one or more children. However, in a different context, the "father" being referred to could be a Catholic priest. We as humans have developed an ability to recognize the meaning of words in a given context. However this is much more difficult for computers. This meaning is typically added to the data as metadata – data about data. Additional meaning can be inferred using logic. For example, if one's father has a father, that person can be inferred to be the grandfather. These inferred meanings can be incorporated into the predefined meanings to create a better understanding of the information.

The full Agent Development Environment is shown in Figure 2. As can be seen, the ADE consists of the Agent Factory plus a collection of libraries, application programming interfaces (API's), and tools such as Protégé (an ontology editor used with semantic technologies). The Agent Factory itself includes the IDE (Eclipse), the Java compiler, and developer tools such as Ant. All of this together forms the development environment used to create the agents. So where do these agents live, work, and play? In the run time environment of course!

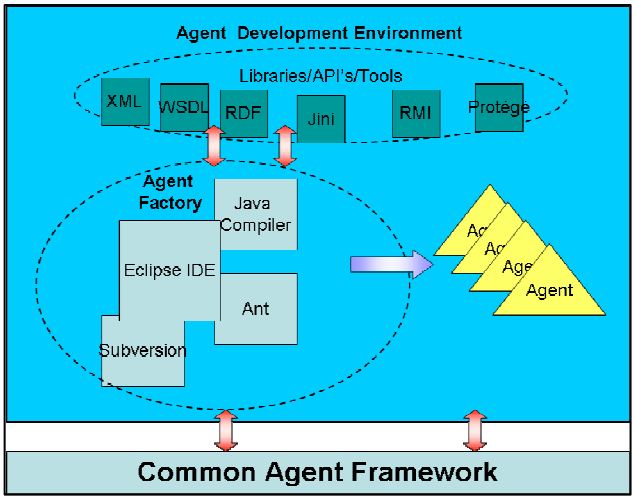

Figure 2 - The ADE will contain the Agent Factory (Eclipse, Java, Ant, etc) as well as libraries for interfacing with external systems. The product is user modifiable agents.

2.4 The Battle Command Run Time Environment

The full Agent Development Toolkit includes the ADE just described, plus a run-time environment where the agents run. The run-time environment also includes a user interface to enable the end-user (i.e., the warfighter) to directly access and control the agents. It is important to differentiate the programming environment from the run time environment for our Toolkit, for we commonly include both when discussing the Toolkit. The ADE is the Java compiler, the editor, and the other developer tools (e.g., Apache Ant). It includes the Agent Factory. Additionally it contains the various libraries necessary to provide interoperability, as previously discussed.

Conversely the Battle Command run-time environment does not contain the Java compiler, or Apache Ant, or the Eclipse IDE. It does include various libraries, including those necessary for interoperability. These libraries will be required for the agents to communicate with other agents, frameworks, and non-agent-based systems. And it should be clear that the Common Agent Framework, which itself is comprised of libraries, will be included in the run-time environment.

The Battle Command run-time environment also includes a specialized user interface for controlling the running of the agents and, more importantly, a mechanism for ***reprogramming*** agents as they run. This is possible with modifiable behavior-based agents. Behavior-based agents contain a core logic that processes instructions received from another agent (including a human agent) via messages. The instructions are queued for execution by the agent, and in effect become the behavior of that agent. New instructions may be received that modify or replace existing instructions, thus providing an agent whose behavior can be changed as it runs, without the need to recompile it.

It is important to understand that these behaviors are programmable and are not simply parameters. For example, we can have an agent that monitors a thermometer. If the temperature exceeds some set value, say 70° Fahrenheit, the agent is to send an alert message. It is a simple thing to change a parameter to say the threshold is now 75°, but the behavior of the agent really hasn't changed in that case; it stills sends a message when a limit is exceeded. But imagine we tell the agent that we now want it monitor the thermometer plus an acoustic sensor and alert us if the temperature changes by more than 10% in 5 minutes and the acoustic sensor registers an increase in sound during the same time. Clearly this is much different than simply changing a parameter.

We are currently using such behavior-based agents for a sensor monitoring project. As can been seen in Figure 3, behavior-based agents have a queue to store behaviors, plus a collection of data about the sensors. This knowledge base is filled with environmental information obtained from "managed elements," which are in fact sensors. New behaviors are sent to the agent, which modify the agent's processing of the knowledge base. The result is the agent produces new messages or takes some other action different than what was originally programmed.

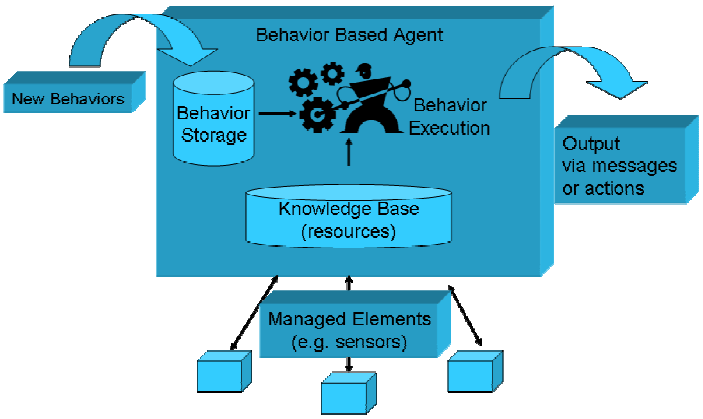

Figure 3 - Behavior Based Agents can be reprogrammed by replacing their existing behaviors, which tell them how to process information such as sensor data.

In order for warfighters to control agents intuitively, our Toolkit will contain a Modifiable Agent User Interface (MAUI). In addition to conjuring up images of white sandy beaches and blue water, MAUI will be the port for warfighters to interact with the agents that are processing their information. It will be a graphical interface, highly configurable and intuitive. In order to be truly useful to warfighters, we intend to develop MAUI using an iterative development approach with frequent feedback from Subject Matter Experts (SME's) experienced with the Battle Command environment.

3. CURRENT ACTIVITIES

Currently our work on the Common Agent Development Toolkit is being performed as an internal research and development project. However, many of the pieces of the Toolkit have already been developed and, in an effort to produce highly flexible products for our customers, we are incorporating some aspects of the Toolkit into our solutions for them.

Recently, GITI acquired the technology of the Valaran Corporation. Valaran had a Service Oriented Architecture (SOA) that was similar to the CoABS Grid, with the addition of advanced security features based on Sun's Jini 2.1 technology. We have incorporated those features with the CoABS Grid to create the Intelligent Service Layer (ISL). The ISL is a services based architecture that forms the basis for our software agent framework.

We are utilizing behavior based agents as part of our current research on Mobile Ad hoc Networks (MANET). A MANET is a network composed of mobile wireless nodes connected in a peer-to-peer fashion using specialized routing protocols to address the dynamic nature of a mobile network. We are using agents to monitor the MANET and take appropriate actions based on the state of the network. For example if the MANET is dense, meaning there are many nodes and the overall throughput of the network is good, our agents will transmit large messages with photographic images. But if bandwidth decreases, our agents will send a text message describing the image. By using modifiable agents with the behaviors described previously, we can provide a highly adaptive and flexible agent environment.

To augment the use of modifiable agents, we have developed an agent planning tool to link agents together to perform complex tasks. The Composable Heterogeneous Agents for Intelligent Notification (CHAIN™) is a tool developed to allow autonomous agents to be created, linked together, and managed in order to provide extensive capabilities to users. CHAIN could be used in conjunction with MAUI to provide a highly configurable agent-based system for the Battle Command environment.

We have also developed automatic generation of software agents as part of CHAIN and ISL. These automatic generation tools are used to create new agents and to wrap legacy systems as agents to permit interoperability. Our recent Valaran acquisition has provided us with an Eclipse plug-in for developing software agents. These capabilities will eventually become the Agent Factory, which will produce agents that can be programmed via MAUI and linked into CHAIN workflow plans.

The ISL provides extensive interoperability capabilities that we are continually extending. The ISL includes a Web service interface that permits our agents to speak to Web services or to act as a Web service. And we are investigating the use of semantic technology to permit communication among heterogeneous systems.

4. FUTURE WORK

We are currently not directly funded to develop the Agent Development Toolkit, but we continue to work on the individual components with the goal of eventually assembling the full version. Our original plan called for

incrementally developing the Toolkit, with each iteration resulting in a new version with more capabilities. We continue to follow that plan, though at a reduced effort.

One of the reasons for developing the Toolkit incrementally is to provide end-users the opportunity to review the product and provide feedback. We currently lack access to Battle Command SME's that can provide us with that feedback. But we are currently taking advantage of the current Grid user base to garner useful feedback, and we are actively seeking new tasks where we will have direct access to Battle Command SME's.

There is continuing progress in heterogeneous system interoperability, particularly in the area of the use of semantics and ontologies to share information. Semantic understanding will be a critical capability for software agents to communicate with other agents across global networks. We are actively involved in this research, because we believe this will be the next major breakthrough in interoperability.

Finally, though the original concept of the Toolkit was created with the US Army Battle Command domain in mind, we see great potential in using the Toolkit in other environments. This includes use in homeland security, network monitoring and management, and home automation systems. We are continually improving the tools we have and creating new ones with awareness of the value of the Agent Development Toolkit in a wide range of domains.

CONCLUSIONS

We have described our concept of an Agent Development Toolkit, and shown how it can be applied to the Battle Command domain to deliver to warfighters the right information at the right time. Developers will be able to create agents that can be utilized and manipulated by warfighters. These agents will assist the commanders and operation planners to make sense of the vast information available to them, and to manage that information effectively.

The Toolkit will consist of a Common Agent Framework, an integrated development environment for creating powerful software agents, tools for version control and packaging agents, plus a run time environment where end-users will be able to access the agents. The agents themselves will be built by developers using a modified and extended version of the IDE which we call the Agent Factory. Those agents will have programmable behaviors, which will allow the agents to be reprogrammed while they run. The control of these agents and the manipulation of the behaviors will be accomplished by commands using a special user interface created for just that purpose.

The Toolkit will be built using industry best practices and defined standards, such as the Agent Systems Reference Model. By selecting Java based tools that are license free, we will create a powerful and flexible collection of tools, integrated into a single package that will run on most if not all computing platforms with a low cost to the government at a time of reduced budgets and demands for greater efficiency.

Although the Toolkit is not a fully functional product at this time, many of the key components exist and are being used today. The GITI Intelligent Service Layer (ISL), CHAIN, behavior-based agents, and other technologies are real and proving themselves daily. Full integration into a comprehensive toolkit is simply a matter of time.

ACKNOWLEDGMENTS

We would like to extend a special word of thanks to the members of the Intelligent Agent Sub-IPT working group chaired by Dr. Israel Mayk of RDECOM CERDEC. Our participation with the working group has been very beneficial. In particular, we thank Dr. William Regli and his team from Drexel University for their work in creating the ASRM.

We would also like to thank Dr. Ray Emami of Global InfoTek Inc for his belief in our work and backing that with internal research and development funding. It is a pleasure to work for a gentleman who values progress and contributions to knowledge above money.

REFERENCES

Berners-Lee T., Hendler, J., and Lassila O. 2001: The Semantic Web. *Scientific American*, May 2001 Issue.

Clayberg E., Rubel D., *Eclipse – Building Commercial-Quality Plug-ins*. Addison-Wesley, 810pp.

Mayk I., Regli W. C., et al, 2006: *The Agent Systems Reference Model* Drexel University.

McCarthy J 1979: Ascribing Mental Qualities to Machines. *Philosophical Perspectives in Artificial Intelligence*, Ringle M., ed., Harvester Press.