Chapter 6

LINKING MILITARY SYSTEMS WITH SIMULATIONS AND INTELLIGENT AGENTS THROUGH WEB SERVICES TO SUPPORT COURSES OF ACTION ANALYSIS

Ranjeev Mittu
U.S. Naval Research Laboratory, Information Technology Division, Advanced Information Technology Branch, 4555 Overlook Avenue, SW, Washington, DC 20375

Abstract: The Department of Defense (DoD) has begun to invest resources to support the development of the Global Information Grid (GIG), a plug-and-play Service Oriented Architecture (SOA) whose goal is to enable interoperability between network-centric entities. This chapter describes the current state-of-the-art in web services technology and its role in the GIG. It then discusses a GIG prototype supporting the web-service enabled interoperability between a military system, simulation and intelligent agents for Course of Action Analysis (CoAA). Next, this chapter addresses challenges for agents in the GIG, as well as potential limitations in the use of web services. This chapter concludes with a survey of competing technologies that may help overcome the limitations and provides a brief summary, including future research areas with regard to the GIG prototype.

Key words: Course of Action, Military Systems, Simulations, Intelligent Agents, Multi-agent Systems, Plan Monitoring, Global Information Grid, Web Services, Peer-to-Peer Computing, World Wide Web

1. INTRODUCTION

Web Services technology is gaining momentum and maturing rapidly within the World Wide Web Consortium (W3C) [34], and has the potential to provide the infrastructure necessary to support a SOA such as the GIG. Web services are services that are made available from a business's Web server for Web users or other Web-connected programs. The primary components that comprise web services include the Universal Description and Discovery Interface (UDDI), Web Services Description Language (WSDL) and Simple Object Access Protocol (SOAP). These three technologies are generally used together in a coordinated fashion to support the discovery of, and interaction with, web services. Furthermore, there are a number of supporting technologies, such as the Business Process Execution Language (BPEL)[1] and Ontology Web Language for Services (OWL-S) [1], which complement the primary web service components. These supporting technologies have the potential to add additional value in SOA environments by providing capabilities that enable the management of services. The BPEL provides constructs for composing complex service transactions based on the interactions and linkages between simpler services. The OWL-S, like BPEL, also enables service composition. However, it also provides additional constructs for describing the necessary service semantics in order to intelligently reason about what is being offered by the service. Both BPEL and OWL-S will be described later in the chapter.

The DoD has begun to invest resources to support the development of the GIG [13], a plug-and-play SOA whose goal is to enable interoperability between network-centric entities. These entities will include not only military platforms and supporting software applications, but also intelligent agents, which may be required to assist users/applications in managing the information available on the GIG. The underlying technology that is envisioned to provide the backbone of the GIG will be web services. The GIG infrastructure will enable the dynamic interconnectivity and interoperability between all levels of military entities, and is a shift from more traditional military architectures such as the Defense Information Infrastructure (DII) Common Operating Environment (COE) [8]. The DII COE is considered a "*stovepiped*" architecture, as the interface points between systems or software components are not easily reconfigurable.

There are many definitions of software agents in the literature, but a general definition of a software agent according to [33] is *"a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objective"*. Multi-agent Systems employ groups of software agents that cooperate with each other to accomplish a given set of tasks (see text box on the following page)

[1] Also known as Business Process Execution Language for Web Services (BPEL4WS)

The field of AI can be broadly categorized in terms of three sub-fields: *distributed problem solving*, *parallel AI* and *multi-agent systems*. Distributed problem solving takes a top-down approach; the problem is decomposed into smaller problems, which are assigned to software modules that compute the individual solutions which are then combined by some higher level process into a global solution. The field of parallel AI deals with performance and resource utilization in problem solving. The field of multi-agent systems deals with a bottom-up approach, which assumes that agents will cooperate with each other to negotiate tasks that need to be solved, while cooperating or resolving conflicts. Invariably, there may be many definitions of what constitutes *intelligence*. For example, agents able to reason about their environment or learn through interaction with their environment, other agents or through users might be considered *intelligent*.

The intelligent agents operating within the GIG may be expected to support users and applications in intelligently discovering and processing information, while coordinating with similar agents as necessary to support these processes. It is reasonable to expect that the efficiency of individual agents (in terms of locating and filtering information in the GIG) may be increased through cooperation and subsequent teamwork with other agents.

This remainder of this chapter will be organized as follows: Section 2 will describe the state-of-the-art in web service technology. Section 3 will discuss the development of the GIG, and how web services will be one of the enabling technologies that will be the foundation for the GIG. Section 4 will describe a proof-of-principle that is being developed to showcase the interoperability between a military system, simulation and software agents to support CoAA. The interconnectivity between each of these components is being developed to leverage web service technology. The goal of this prototype is to demonstrate the coupling of simulations with military Command and Control systems to assist in the detection of critical deviations in a plan's execution as reported to the military system. Intelligent agents are responsible for detecting the deviations between reported movements and the simulated movements and alerting the user. The user then has the option to use the services offered by the simulation to spawn multiple "what-if" scenarios to explore CoAA. Section 5 will discuss the challenges agents may face in the GIG. Section 6 will describe potential limitations in the use of web service technology within the GIG, while section 7 provides a brief survey of competing technologies that may help overcome some of the limitations. Lastly, in section 8, we provide a brief summary.

2. WEB SERVICES

Web service technology is rapidly gaining momentum under the auspices of the W3C. The W3C was established in 1994 to help lead the development of standards, specifications, guidelines, software, etc, to promote the evolution and interoperability of the World Wide Web (WWW). Web services technology includes three key components that are used in conjunction with each other. These components include the UDDI, WSDL and SOAP. It should be noted that UDDI is not the only registry standard. For example, the ebXML [9] Registry and Repository Standard is sponsored by the Organization for the Advanced of Structured Information Standards (OASIS) and the United Nations Center for the Facilitation of Procedures and Practices in Administration, Commerce and Transport. The UDDI, however, has emerged as the registry standard for the GIG.

The UDDI is a framework that defines XML-based registries in which businesses can upload information about themselves and the services they offer. An XML-based registry contains names of organizations, services provided by those organizations, and descriptions about service capabilities. XML registries based on the UDDI specification provide common areas through which systems/organizations can advertise themselves and their web services. Attributes that can be registered include the description of the organization that agrees to provide the service as well as information about specific points of contact (including their phone number and email addresses). The UDDI registries also contain information about services as well as service bindings (which are needed to connect with a service). Once a service provider has been located in the registry, a client can then connect to, and interact with, the service based on the services' WSDL document (the UDDI also stores the web address for the WSDL document)[2].

The WSDL is an XML vocabulary standard for Web Services. It allows developers to describe web services and their capabilities in a standard manner. The WSDL helps to expose the web services of various businesses for public access. Generally speaking, programmers develop services based on their language of choice, while supporting software utilities generally provide the necessary conversions to automatically generate the underlying WSDL document. A WSDL document contains information about a web service and the operations supported by the specific service. A web service

[2] It should be noted that UDDI version 3.0 is expected to be extensible in both the UDDI data structures as well as Application Programming Interface (API). So, for example, it will be possible to store a much richer set of service attributes in the UDDI registry as well as access those attributes using the subsequent API. This may make it easier to store the additional attributes associated with OWL-S.

may support multiple operations that can be invoked on that service. Each operation is described in terms of the inputs required by the operation, the outputs generated by the operation as well as the data types for both input and output. Furthermore, the bindings (describing the message format and protocols) are included in the WSDL description.

The SOAP is an XML vocabulary standard to enable programs on separate computers to interact across any network. The SOAP is a simple markup language for describing messages between applications. The SOAP provides a way for developers to integrate applications and business processes across the Web or an intranet, by providing the platform and programming language independence needed to create the business integration of web services. A SOAP message contains an *envelope*, *header* and *body* element. The *envelope* element is the root element of a SOAP message. This element defines the XML document as a SOAP message, the namespaces used in the SOAP document as well as the type of encoding (e.g. the data types used in the document). The optional *header* element contains application specific information about the SOAP message. For example, this element is used to describe whether the receiver of the SOAP message must be capable of understanding any number of elements to be communicated in the transaction. The *body* element contains the message.

Figure 6-1 describes the interaction between UDDI, WSDL and SOAP. A service provider registers the necessary service attributes with a UDDI registry including the location of the WSDL document. The client will then look-up the organizations registered within UDDI and the services they have agreed to provide. If a client chooses to use a specific service provided by an organization, that client will then access the services' WSDL document in order to understand how to access the operations available from that service. The communication between the client, UDDI and web service is via SOAP.

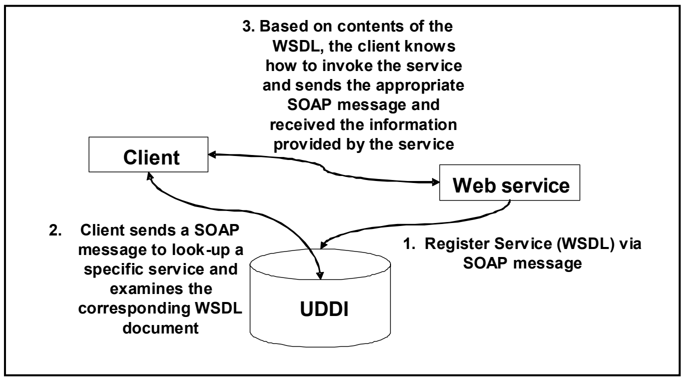
3. Based on contents of the WSDL, the client knows how to invoke the service and sends the appropriate SOAP message and received the information provided by the service
Client
Web service
2. Client sends a SOAP message to look-up a specific service and examines the corresponding WSDL document
1. Register Service (WSDL) via SOAP message
UDDI

Figure 6-1. Interaction between UDDI, WSDL and SOAP

2.1 Web Service Composition and Semantics

Web service languages that support the specification of service composition and semantics are also emerging, and these have a complementary role to WSDL. These languages provide constructs to enable service composition (e.g., the ability to create services with complex behaviors by linking together other services) as well as the semantic tagging of services. The BPEL specification supports service compositions while the OWL-S goes beyond the features offered by BPEL by providing additional constructs for specifying service semantics. The BPEL language is being developed under the auspices of the OASIS, and its potential benefit is that it enables service reusability. The OWL-S is being advanced under W3C, and its potential benefit is that it promotes a more intelligent mechanism for discovery of services.

The BPEL specification is positioned to become the web service standard for composition. The BPEL defines a business process that specifies the execution of web service operations from a set of web services, the data shared between the operations, the partners involved and also includes various exception handling mechanisms. It permits the specification of complex services by wiring together different activities that can, for example, perform web service invocations, manipulate data, throw faults, or terminate processes. These activities may be nested within structured activities that define how they may be run (e.g., sequence, or in parallel). A conceptual view of BPEL is seen in Figure 6-2 [37]. The BPEL derives its features from Web Services Flow Language [35] and XLANG [36], from IBM and Microsoft respectively.

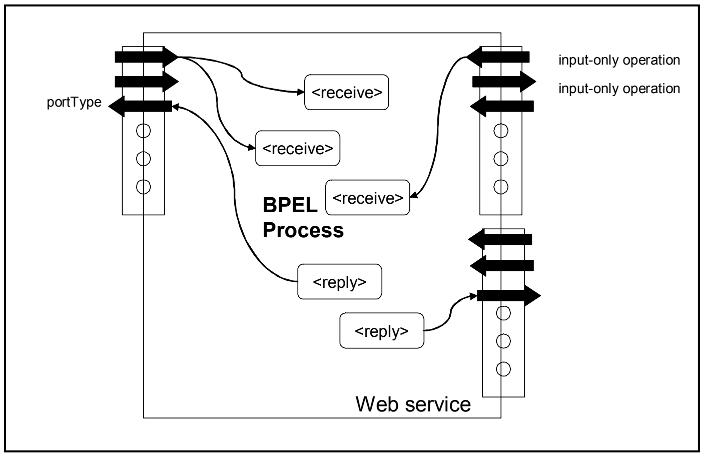

Figure 6-2. Business Process Execution Language (BPEL)

The OWL-S is an example of a semantic web service language [2] and has evolved from the research sponsored by the Defense Advanced Research Projects Agency (DARPA) [7]. Specifically, OWL-S has evolved from the DARPA Agent Markup Language (DAML) [6] and DAML-Services (DAML-S).

The goal of the DAML program (and ontology by the same name) was to develop an XML-based language that describes semantic content to a degree that allows agents to intelligently reason about that content. Traditional markup languages such as the Hyper-Text Markup Language, HTML [16], and the eXtensible Markup Language, XML [10], do not provide sufficient constructs to describe the semantics of information to support intelligent reasoning, being primarily delegated for human consumption. The DAML language leverages concepts found in the Resource Description Framework (RDF) [26] and RDF Schema [27]. The DAML-S was an extension to DAML with the goal of describing semantic content associated with services. The responsibility for evolving the DAML and DAML-S language was eventually given to the W3C, and initial versions have been released under OWL and OWL-S, respectively.

The OWL-S language is described through an ontology that specifies three kinds of knowledge about a service (Figure 6-3). The top level of the OWL-S ontology is the *Service* class, which contains several subclasses. The *ServiceProfile* subclass describes what the service does (e.g., what does the service require of the users and what it provides). This class contains properties that describe the inputs to the service, the output by the service, preconditions that must be valid prior to using the service, and effects the service may have. The *ServiceModel* subclass defines how the service works, and the *ServiceGrounding* subclass specifies how to access the service. Within the *ServiceModel* class there exist constructs for defining atomic services, specifying service compositions as well as for managing flow control (control over how web services are invoked and/or how the information is passed between the services).

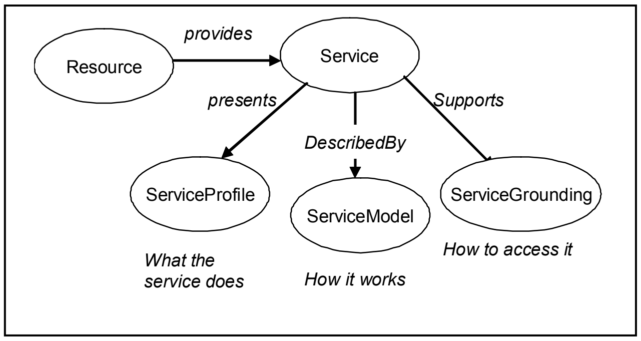

Figure 6-3. Semantic Web Services

2.2 Comparing BPEL, OWL-S and WSDL

The BPEL and OWL-S have broad and somewhat complementary objectives. Both BPEL and OWL-S provide constructs within the language to define complex services in terms of much simpler services, which offers semi-automated processes such as software agents the potential to follow a “recipe’ for interacting with such complex services based on the linkages between the underlying services. The *ServiceModel* class within OWL-S most closely relates to the business process model in BPEL, however, the OWL-S enables the semantic tagging of services, which can help a software agent choose between competing services. For example, within OWL-S, one can specify the preconditions that must exist before the service can be used and the effects of using the service. A frequently used example is that if a user is interacting with a book buying service, then a precondition for using this service is that the user must have good credit if purchasing via a credit card. A second key difference between OWL-S and BPEL is that the former is based on a class typing representation that enables reasoning systems to more readily make higher level inferences about the service. The BPEL, on the other hand, does not support such a representation. Business entities that wish to collaborate with each other using BPEL are restricted by structured XML content contained in the WSDL *PortType* definition.

The WSDL does not provide constructs for defining complex services in terms of smaller compositions. However, as BPEL and OWL-S emerge, they may leverage the existing maturity of WSDL, particularly the representation of service bindings. In fact, the *ServiceGrounding* class of OWL-S does not contain a concrete description of service bindings. This OWL-S subclass relies on WSDL for its bindings, as can be seen in Figure 6-4 [38].

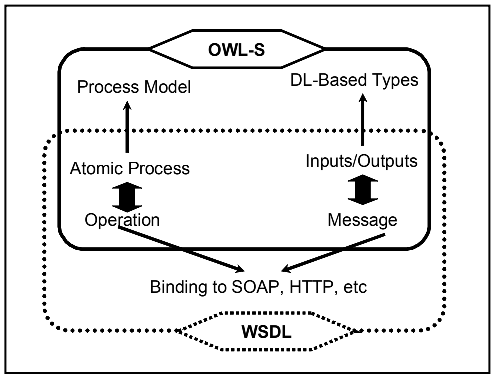

Figure 6-4. Relationship between OWL-S and WSDL

In summary, one of the key differences between OWL-S and BPEL is that OWL-S *ServiceProfile* class provides a much richer set of expressions to support a more intelligent mechanism to interact with a complex service (i.e., inputs, outputs, preconditions and effects). In addition, OWL-S provides the required semantics in order to reason about a service (e.g., based on Description Logic, or DL). The key similarity between OWL-S and BPEL is that both rely to some degree on WSDL. The OWL-S uses the bindings in WSDL to relate the service to a concrete implementation, and BPEL also uses the WSDL specification for its bindings.

The (semantic) web service languages described in this subsection have the potential to empower applications and agents in the GIG to effectively search and utilize services offered by network-centric entities.

3. THE GLOBAL INFORMATION GRID

The DoD is beginning to invest in the transition of architectures such as the Defense Information Infrastructure (DII) Common Operating Environment (COE) to the Global Information Grid (GIG), which is being managed by the Defense Information Systems Agency.

The vision of the GIG is to provide a truly open environment in which net-centric entities such as Command, Control, Communications, Computers and Intelligence (C4I) systems, simulations, sensors, platforms, software agents, etc., can share information in a seamless manner, without the restrictions and limitations imposed by the DII COE architecture, including a requirement placed on system developers to build within a "closed", but interoperable, environment. This limits interoperability across domains, particularly in a dynamic environment in which opportunistic information is readily available, but may not be easily discovered and accessed.

The GIG represents a fundamental shift from these *stovepiped* architectures to a more open architecture, through the reliance on web-based standards and technologies that enable syntactic interoperability. However, syntactic interoperability alone is not by itself sufficient for meaningful information exchange. In order to achieve meaningful interoperability, one must also consider the information from a contextual perspective in order to achieve semantic interoperability. Semantic web services described in the previous section may provide useful capabilities in this regard.

Another fundamental shift within the GIG vision is from a *"process-then-post"* towards a *"post-then-process"* philosophy, whereby an application will be responsible for fusing and converting raw data or information into a form which is most useful for that particular application. For example, rather than one application requesting information that has been processed by a second application, which does not necessarily know the potential uses of that processed information, the GIG vision allows the first application to find the raw data that is most relevant and do that processing locally so that any intermediate information is not lost.

The GIG architectural model is composed of several layers as seen in Figure 6-5. The lowest layer deals with management and administrative functions such as doctrine, governance, policy, standards and architectures. The next layer above this is the transport, which includes the Defense Information Systems Network [14], Joint Tactical Radio System [17] and Transformation Communication Systems and technology. The purpose of this layer is to physically transport information within the GIG. The next layer above this is the GIG Enterprise Services (ES). The GIG ES layer is comprised of the Core Enterprise Services (CES) and Community of Interest (COI) services. The CES will include basic services that will be required by most components, such as discovery services, storage services, etc. The COI services represent those services that are most useful for a specific group of people or applications. The next layer in the hierarchy are the applications that will interact with the lower level services in order to obtain information necessary for the useful functioning of those applications. The topmost layer is comprised of various war-fighting domains that the applications support.

There are several programs with the DoD that are beginning to implement prototype GIG components. The Net-centric Enterprise Services (NCES) [20] Program, for example, addresses the development of the GIG CES while the Horizontal Fusion initiative [15] addresses the means/tools to support the interaction with the GIG services.

Domains	*Examples:* •Warfighting •Business •Intelligence
Applications	*Examples:* •Deployable Joint C2 Program •Business Management Modernization Program
GIG Enterprise Services	*Examples:* •Electronic Mail •Application Hosting •Weapon-Target Pairing
Transport	*Examples:* •Defense Information System Network •Joint Tactical Radio System •Transformational Communication System
Management	*Examples:* •Doctrine •Governance •Policy •Standards •Architecture •Engineering

Figure 6-5. The Global Information Grid (GIG)

Web services technology is expected to provide the underlying mechanism through which information will be shared between the GIG layers. At the time of this writing, the key web technologies envisioned to become a reality in the GIG include UDDI, WSDL and SOAP and to some degree BPEL as they are the most mature technologies. There will be an obvious requirement for users within the GIG to discover and interact with services (whether these be single services or composed of smaller services). However, the WSDL specification does not support the description of semantic relationships, thereby placing a heavy burden on the user/application to determine the appropriateness of the web service for a given usage. Languages such as OWL-S have the potential to make a significant impact to support the intelligent discovery and subsequent interaction with web services by automated software agents. The software agents can interact with an inference engine that has been loaded with the OWL-S ontology, to reason about specific instances corresponding to the ontology.

4. GIG PROTOTYPE

For years, simulations have been used by analysis and planning staffs to develop and rehearse operation plans, analyze results, and develop doctrine. Typically, combat simulations are used most heavily during the planning stages of an operation, prior to battlefield action. However, simulations are increasingly being used *during* operations to perform CoAA (see description in box below) and develop real-time forecasts of future conditions on the battlefield. Recent efforts by the Defense Modeling and Simulation Office (DMSO) to improve the interoperability of C4I systems with simulations has provided a powerful means for rapid simulation initialization and analysis during exercises, and made simulations more useful and responsive as the exercises are executed. The latest DMSO effort involves technology development to support the integration of operational systems, such as those in the Global Command and Control System (GCCS), with simulations such as the Joint Warfare System (JWARS) [21].

Course of Action (COA) [22]**:** (1) A plan that would accomplish, or is related to, the accomplishment of a mission. (2) The scheme adopted to accomplish a mission or task. It is a product of the Joint Operation Planning and Execution System concept development phase.

(Continued from previous page) The recommended course of action will include the concept of operations, evaluation of supportability estimates of supporting organizations, and an integrated time-phased database of combat, combat support and combat service support forces and sustainment. Refinement of this database will be contingent on the time available for course of action development. When approved, the course of action becomes the basis for the development of an operational plan or operational order.

The GCCS [12] is an automated information system designed to support situational awareness and deliberate and crisis planning through the use of an integrated set of analytic tools and flexible data transfer capabilities. GCCS incorporates the force planning and readiness assessment applications required by battlefield commanders to effectively plan and execute military operations. The GCCS system correlates and fuses data from multiple sensors and intelligence sources to provide warfighters the situational awareness needed to be able to act and react decisively. This situational awareness is represented in the Common Operational Picture. It also provides an extensive suite of integrated office automation, messaging, and collaborative applications

The Joint Warfare System (JWARS) [18] is a campaign-level model of military operations that is currently being developed under contract by the U.S. Office of the Secretary of Defense (OSD) for use by OSD, the Joint Staff, the Services, and the Warfighting Commands. JWARS provides users with a representation of joint warfare to support operational planning and execution, force assessment studies, systems effectiveness and trade-off analyses, as well as concept and doctrine development. The JWARS permits studies that require a "balanced representation of Joint Warfare", with models that support 1) the C4ISR systems and processes that are an integral part of US concept of operations; 2) logistics, both strategic and intra-theater in the combat area; and 3) maneuver warfare at the operational level.

The DMSO is sponsoring the integration of JWARS, GCCS and software agents as a proof-of-principle to demonstrate the viability of supporting the interoperability of these three components through the application of web service technologies, Figure 6-6.

4.1 Concept of Operations

The concept of operations of this proof-of-principle evaluation is to initialize GCCS with Unit Order of Battle (UOB) data to represent known locations of forces prior to plan execution. The JWARS is also initialized with the same UOB data to ensure that it is consistent with the force structure in GCCS. Through an artificial mechanism, the GCCS will generate real-time updates to track movements. The reason for the artificial generation of track movements is due to the fact that the demonstration is in a laboratory environment, and hence the system is not integrated with live information feeds; however, this is an assumption that does not invalidate the concept or the application of the technology. The JWARS simulation is capable of generating "expected" movement of the same forces based on its internal models and algorithms.

Both the actual GCCS track data as well as the corresponding JWARS expected track movements will be made available to the software agents, which will compare such things as deviations between real/expected track positions, whether certain tracks enter regions of interest (or, alternatively, fail to do so) in a given time period or time instant, actual versus expected force ratios, etc. The failure conditions, as specified by the user, will trigger the agents to send alerts to both GCCS and JWARS, after which JWARS can be used to spawn additional JWARS simulations to support CoAA in order to correct the failures in the plan.

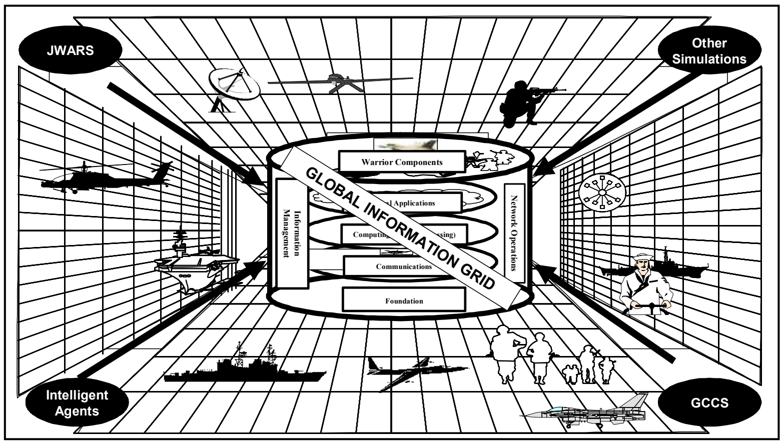

Figure 6-6. JWARS and GCCS interoperability with Intelligent Agents in the GIG

The proof-of-principle demonstration will utilize tracks associated with units (e.g. land and sea) from the GCCS Track Management Server (TMS) as well as air tracks coming from the Theatre Battle Management Core

System (TBMCS). These tracks will be made available to the agents through web service technology.

4.2 System Operation

The basic architecture supporting the proof-of-principle integration between JWARS, GCCS and software agents is seen in Figure 6-7. This architecture leverages the web service technologies UDDI, WSDL and SOAP to enable the syntactic interoperability between each component.

The Army C4I Simulation Initialization System [3] is used to initialize the GCCS-M TMS and TBMCS [31] C4I systems as well as the simulation system (i.e., JWARS). The initialization information contains the current UOB such as organizations, their command relationships, as well as supporting equipment and facilities. A tactical system (in our case, an exercise replay) will deliver the actual data to GCCS.

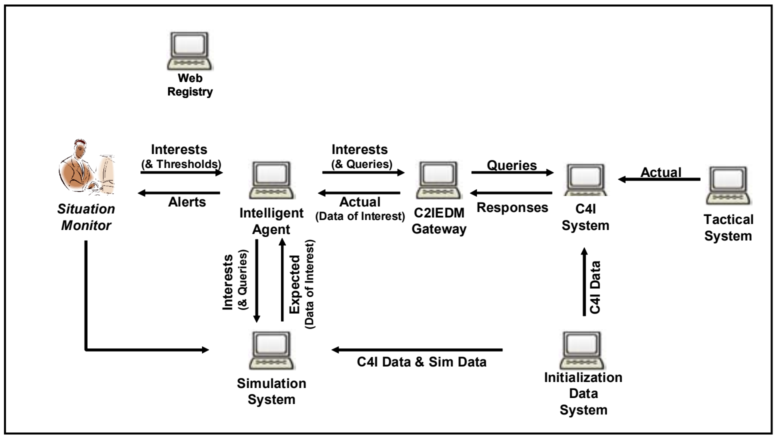

Figure 6-7. The JWARS, GCCS and Software Agent Web Service Federation

The Situation Monitor (SM) is a graphical front end which permits a JWARS user to specify tracks of interest that need to be monitored, and any conditions and corresponding thresholds to which the user would like to be alerted when these conditions are met or thresholds are exceeded. The SM invokes the *subscribeFor* operation of the *TrackMonitorWebService,* in order to send the intelligent agents behind this service the subsequent tracks that the user has an interest in monitoring for deviation analysis.

A small fragment of this web service's XSD file is contained in Table 6-1 and WSDL file is contained in Table 6-2. As can be seen from the WSDL, this operation contains a *subscribeFor* input message and *subscribeForResponse* output message. The input message corresponds to a *TrackRegistration* object described in the corresponding XSD file. The object consists of a list of track identification numbers, criteria for generating alerts and thresholds to be used to detect deviations as received by the web service. This information will be communicated to software agents that will make requests to the C4I system and simulation to obtain the corresponding tracks for subsequent monitoring. The software agents receive the tracks after invoking both the *C2IEDMGatewayService* as well as the *JWARSWebService* (the data from *C4ISystemTrackService* is translated to the C2IEDM interchange format – discussed later – by the *C2IEDMGatewayService*). These agents will compare both the real and simulated tracks using the thresholds to generate alerts, which are sent back to the SM display (again, through the invocation of operations corresponding to the *SituationMonitorWebService*.) The alerts may warrant the exploration of "what-ifs" in order to aid in the analysis and selection of alternative courses of action.

Table 6-1. TrackMonitorWebService XSD (XML Schema Document)

```
***
<xsd:complexType name="TrackRegistration" >
    <xsd:annotation>
***
</xsd:annotation>
  <xsd:sequence>
       <xsd:element minOccurs="1" maxOccurs="1"
           name="wsName" type="xsd:string"/>
     <xsd:element minOccurs="1" maxOccurs="unbounded"
           name="trackIds" type="xsd:string"/>
     <xsd:element minOccurs="0" maxOccurs="unbounded"
           name="criteria" type="xsd:string"/>
     <xsd:element minOccurs="0" maxOccurs="unbounded"
           name="thresholds" type="xsd:string"/>
  </xsd:sequence>
  </xsd:complexType>
***
```

Table 6-2. TrackMonitorWebService WSDL

```
<types>
    <xsd:schema targetNamespace=
           "http://www.TrackMonitorWebService.com/
                  TrackMonitorWebService/xsd"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd
ttp://www.TrackMonitorWebService.com/
                  TrackMonitorWebService/xsd"
      schemaLocation="TrackMonitorWebService.xsd"/>
    </xsd:schema>
<xsd:element name="subscribeFor"
           type="trackMonitorSchema:TrackRegistration"/>
***
</xsd:schema>
</types>
  <message name="subscribeFor">
    <part name="body" element="tmws:subscribeFor"/>
***
<operation name="subscribeFor">
      <input message="tns:subscribeFor"/>
      <output message="tns:subscribeForResponse"/>
</operation>
***
```

Each of the services in Figure 6-7 will register themselves with the UDDI registry. Each component will then do a look-up within UDDI to obtain the WSDL file of the other services, from which the service can dynamically resolve the location of the other services, and subsequently invoke their operations.

The Command and Control Information Exchange Data Model (C2IEDM) [4] gateway will map the information passed between services onto the C2IEDM vocabulary. The C2IEDM was developed under the auspices of the Multilateral Interoperability Programme.

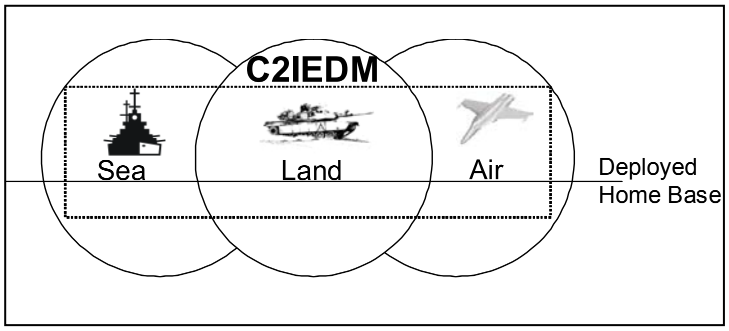

Figure 6-8. The scope of the C2IEDM data model

The C2IEDM is a generic model that can be extended as needed to suit evolving military requirements (e.g., serves as a “hub”; as such, it was originally named the “Generic Hub”, and evolved to Land C2IEDM and eventually C2IEDM to capture other areas including Air and Surface, see Figure 6-8). The C2IEDM is comprised of a conceptual data model, logical data model and physical data model. The conceptual data model represents generalized concepts, while the logical data model represents further details associated with the conceptual data model. The physical data model defines the physical data storage schema. The main purpose of the C2IEDM is to represent Information Exchange Requirements between C2 systems.

The proof-of-principle prototype is currently undergoing experimentation in a laboratory environment and, through feedback from subject matter experts, the capability will be refined. We expect to provide these unique CoAA services to the broader GIG community through participation in integrated experiments in the future.

5. CHALLENGES FOR AGENTS IN THE GIG

There are many challenges in realizing such an ambitious effort as the GIG. One challenge we are faced with in our proof-of-principle implementation is the integration of large legacy systems through web services, which, although maturing at a fast pace, are still evolving. This, by itself, is a tremendous challenge as we are forced to reengineer legacy software to work within a different computing paradigm (the standards for which are continually evolving)! It is envisioned that newer systems in the GIG will be architected to seamlessly work with web services technology.

Having an ability to semi-automatically locate and interact with services will be a key capability, as it will be inefficient to have users in the loop on every transaction to search for web services. Furthermore, systems and components in the GIG may lack the time to form complex search queries. We envision intelligent agents to support this functionality through their abilities to semi-autonomously coordinate with other agents and humans in support of system requirements for information.

Another issue that will inevitably be encountered in the GIG will include the interoperability of systems between Communities of Interest (COI). For example, the Modeling and Simulation COI may rely upon the C2IEDM as the common information exchange model, but this may not be appropriate or adopted for use throughout the GIG. The challenge here will be to develop techniques that map/translate between meta-data or ontologies that are expected to exist across the many COIs. What will be the role of agents in supporting this process, or will this be primarily a manual process?

A key challenge that is certain to arise will be the ability of agents (which understand one ontology) to communicate with other agents (having a different ontological representations). Again, technologies that aid in mapping or translating between ontologies will support the ability of agents to communicate within the GIG environment, which may lead to coordinated agent activity. The field of agent coordination and teamwork is an emerging area of research [25, 30]. To realize the full potential of distributed multi-agent systems, the agents will need to cooperate as part of teams to help the operators (i.e., acting as their proxies) achieve their goals. In the context of our proof-of-principle, teams of distributed software agents with different goals and ontologies may need to coordinate to decompose and relate multiple plans to determine critical points, which may be communicated to a team of agents responsible for monitoring the critical points in the plan's execution.

In the GIG, agents may be required to assess the viability of dynamically composing a service; therefore, it may be necessary to endow these agents with advanced reasoning capabilities. However, there will be a limit in terms of how much an agent is able to practically reason with, hence, additional solutions may be adopted. The additional techniques may include human-agent cooperation (i.e., mixed-initiative approaches), potentially coupled with machine learning techniques in order to create robust, adaptive agents.

6. POTENTIAL LIMITATION OF WEB SERVICES FOR THE GIG

There are several challenges in applying web service technologies in a network-centric environment such as the GIG. The web-service computing paradigm was primarily developed to support Business-to-Business (B2B) commerce in which services offered by businesses could be invoked using web technology. Of course, there are still open questions in using web services in a B2B computing world, such as, for example, payment for services rendered. As the commercial sector is primarily driving the development of web service technology, any solution generated from the commercial sector may have the potential to be used in some form within the military domain. A bigger question, however, concerns reliability. For example, in B2B commerce, it may be acceptable for services to fail quite often, for example, if a service is being upgraded or a computer system goes down. A high level of failure may not be acceptable in a military environment in which the lives of humans may potentially be at stake. How will Quality of Service be guaranteed, what criteria will be used and under what circumstances? This may imply a tight coupling between the upper layers and the lower layers in the GIG.

Another question regarding the use of web services is "how well is the technology suited to the potential bottlenecks associated with the registries?" After all, the client application must know the location of the registries in order to be able to access them and determine what services have been registered and how to access them. What should happen if the nodes that the registries reside on fail, or are bombarded with a potential denial of service attack? Possibly a larger issue to consider is "how will highly mobile platforms and systems interact with registries?" These platforms operate at a very high tempo and connectivity to such registries may be sporadic.

Agent technology has the potential to make a profound impact within the GIG; however, one must also consider whether web services will provide the necessary infrastructure for agent-to-agent coordination. For example, agents may be required to coordinate with each other through bi-directional messaging. This, however, may not be adequately supported through a web services framework. For example, communication between agents does not necessarily fit within the SOA paradigm; an agent's communications capability should not necessarily be categorized as a service that is provided by that agent. Instead, the agents should be able to communicate through a natural metaphor, utilizing web services as needed to perform their functions.

It is apparent that web services will enable much of the interoperability in the GIG, but may not be the silver bullet solution for every situation. Can we assume that web services will be sufficiently mature in the future to address these issues? There is certainly a possibility that web services may not be the only solution, but may be required to work with a variety of supporting technologies that offer a solution to these limitations (a quick look through W3C's activity reveals a heterogeneous mix of technologies being developed which offer varying capabilities suitable for different uses).

7. SURVEY OF COMPETING TECHNOLOGIES

The field of grid computing may be considered a subcategory of distributed computing [11], and may complement web services. There is a subtle difference between grid computing and distributed computing. Generally speaking, the world of grid computing deals with the *large-scale* sharing or utilization of loosely coupled, distributed, heterogeneous resources. Distributed computing, on the other hand, primarily deals with allocating software components on a smaller scale across a network (e.g., to conserve computation cycles on a local machine.) Grid computing holds the promise of taking distributed computing to a new level that enables computing across the internet.

Grid computing, and to some degree distributed computing, may be further characterized as either client-server or Peer-to-Peer (P2P) [23]. Web services technology is most closely related to the client-server model. For example, UDDI registries store information regarding available services, and clients access those registries to determine where the service resides and how to access it.

In a P2P computing environment, there are no centralized registries; a subset of the directory peers maintain a local cache of available service advertisements of peers that choose to register with that particular directory. Any peer requiring a service may dynamically discover and interact with these directories to locate a service offered by other peers. In fact, in a P2P infrastructure, peers are generally dynamically discovered through the interaction with directories that maintain service advertisements. These advertisements allow peers to discover and utilize the services of other peers.

Although the P2P computing landscape is large, the next two subsections will present representative examples of P2P systems. Project JXTA captures some of the primary characteristics of P2P systems, while Neurogrid provides a flavor of intelligent search and discovery in P2P environments. The third subsection will describe the Control of Agent Based Systems (CoABS) Grid. The CoABS grid is not considered a pure P2P system, but is more closely aligned with a client-server model. The CoABS grid is presented because it has been used extensively by the software agent community to federate agent-based systems.

7.1 Project JXTA

Project JXTA [24] is an implementation of P2P computing that is being advocated by Sun Microsystems. JXTA provides an open set of XML-based protocols that allows any device on a given network to communicate and collaborate in a P2P fashion, even when some of the peers are behind Network Attached Devices or Firewalls. The basic concepts supported by JXTA are the *peer*, *peer group*, *network services*, *modules*, *pipes*, *messages*, and *advertisements* which are described below:

- ***Peer:*** A peer in JXTA is any device on the network that supports one or more of the JXTA protocols. There are six protocols defined within JXTA. Peers use these protocols to discover other peers, advertise and discover network resources, as well as communicate and route messages.

- ***Peer Group:*** A peer group is a collection of peers that have an agreed upon set of services. Peers may exist within multiple peer groups simultaneously; however, by default, when peers are instantiated they are joined to the Net Peer Group (all peers are a part of the Net Peer Group).
- ***Network Services:*** Peers generally cooperate and communicate to discover network services. There are two types of services: *Peer services* and *Peer Group Services*. The former type of service is associated with an individual peer while the latter service type is associated with a group of peers, which provides the added advantage of redundancy among the peers in the group (assuming another peer is still able to provide the failed service).
- ***Modules:*** Modules are pieces of code written to represent any kind of behavior, and are described by the *Module Class* (which supports the capability to advertise behaviors), *Module Specification* (which provides support to access a module) and *Module Implementation* (the actual implementation of the module). Network services are the most common forms of behavior that can be instantiated on a peer.
- ***Pipes:*** Pipes support communication between peers. Input pipes are used by peers to receive messages; output pipes are used to send messages.
- ***Messages:*** A message is an object that is transmitted between JXTA peers. Messages may be either in XML or binary form.
- ***Advertisements:*** Advertisements are XML documents that describe peers, peer groups, pipes or services. There are nine advertisement types that are supported in JXTA.

Using the JXTA architecture, peers advertise their capabilities with a rendezvous peer (i.e., directory), which caches the advertisement. The advertisement may include the service offered as well as information about how to connect to the peer that offers the service. If a peer wishes to discover a service, and an advertisement is not found on the local rendezvous peer, then a discovery request is propagated by that rendezvous peer to other rendezvous peers on the network. A rendezvous peer that contains the specific service advertisement provides the pipe advertisement to the requesting peer, which uses the pipe advertisements to connect directly with the peer that offers the service.

Relay peers contain routes to other peers, and are also capable of routing messages to peers. In the example above, if the service is not found on the local rendezvous peer, then a route is needed to other rendezvous peers as well as eventually to the peer that offers a service. The route will be contained as a series of hops through a set of relay peers to the destination. Rendezvous and relay peers may be implemented on the same node.

7.2 NeuroGrid

The Neurogrid [19] environment provides a decentralized, adaptive search system that learns over time in response to user queries. Two main components of Neurogrid that complement one another are semantic routing and learning. Semantic routing refers to the ability to forward queries based on their content, while learning in this context refers to the ability of the nodes to dynamically adjust the meta-data describing the contents of nodes and the files that make up those contents.

The concept behind NeuroGrid is to store the relationship between bookmarked URL's and their relationships to user queries (e.g., keywords) as well as between keywords and other nodes, which then provides a capability to semantically route discovery requests between nodes in order to determine which nodes offer the best response (e.g, URL) to the query (keywords, or metadata, may also be updated at this point). A direct link is also formed between the initiating node as well as the node that returns the response, thereby increasing the connectivity in the network. Neurogrid addresses the issue of how to rank multiple URLs that are associated with the same keyword by not only using the fact that the user has clicked through the URL, but whether it was bookmarked as well. The mathematics behind Neurogrid also takes into account cases where the ratio of recommended bookmarks to that of selected bookmarks, for a given search, is identical.

NeuroGrid, in its current server side implementation is not a pure P2P system in the sense that each node is connected to every other node. It is, however, based around a large number of small servers being linked to one another in a P2P fashion, with each server supporting a small community of users

7.3 The CoABS Grid

The CoABS grid [5] (hereafter referred to as Grid) was developed under the DARPA CoABS program, and arguably provides the most successful and widely used infrastructure to date for the large-scale integration of heterogeneous agent frameworks with object-based applications, and legacy systems. Based on Sun's Jini [29] services, it includes a method-based application-programming interface to register and advertise capabilities, discovers services based on those capabilities, and provides the necessary communication between services. Systems and components on the Grid can be added and upgraded without reconfiguration of the network. Failed or unavailable components are automatically purged from the registry and discovery of similar services and functionality is pursued.

The Grid supports a wide variety of applications, from those that support simple monitoring and information retrieval to complex, dynamic domains such as military command and control. Using the Grid, agents and wrapped legacy systems can (1) describe their needs, capabilities and interfaces to other agents and legacy systems; (2) find and work with other agent components and legacy systems to accomplish complex tasks in flexible teams; (3) interact with humans and other agents to accept tasking and present results, and (4) adapt to changes in the application domain, the task at hand, or the computing environment. The Grid does this by providing access to shared policies and ontologies (mechanisms for describing agents' capabilities and needs), and services that support interoperability among agents and legacy systems with simple or rich levels of semantics—all distributed across a network infrastructure.

Although most agent frameworks provide some of the interoperability and other services that the Grid provides, each framework typically supports specialized constructs, communication, and control mechanisms. This specialization is desirable because particular systems can use mechanisms appropriate to the problem domain/task to be solved. The Grid is not intended to replace current agent frameworks but rather to augment their capabilities with services supporting trans-architecture teams.

The Grid provides helper utility classes that are local to an agent and hide the complexity of Jini. These classes automatically find any Look-up Services in both the local area network and user-designated distant machines. The Grid supports agent and service discovery based on Jini entries and arbitrary predicates as well as by service type. The Grid also provides event notification when agents register, deregister, or change their advertised attributes.

Recently DARPA has conceived a new program within the Information Processing Technology Office (IPTO) called Fast Connectivity for Coalition and Agents Project (FastC2AP). One of the goals of the FastC2AP program is to investigate and build linkages between the CoABS grid and web services. The idea is to make web services easily accessible to software agents on the grid. Programs such as this demonstrate that web service technology is maturing fast and permeating into military applications. However, it is becoming apparent that architectures more suited to large-scale multi-agent systems, such as the CoABS grid, will continue to be used and will therefore be required to work with web services.

8. SUMMARY

This chapter has described the current state-of-the-art in web services technology and how it is being applied to support the development of the GIG. This chapter has also described a proof-of-principle implementation that uses web services to support the interoperability between a military system, simulation and intelligent agents to support CoAA. Future areas to explore in the proof-of-principle include the integration with the eXtensible Battle Management Language [32], which enables access to military Operational Orders (OPORDS) through a web service interface. This will enable the agents to relate the impact of the deviations to the OPORD (particularly whether critical tasks within the OPORD are affected by the deviations). Additional areas include the use of BPEL to configure services from within the JWARS Situation Monitor and exploring techniques for mapping between ontologies to support agent-to-agent communication.

This chapter has also outlined the challenging problems that software agents may be expected to not only face, but also help solve in the GIG. The issues that have been suggested include:

- The integration of agent technology within a web-services paradigm.
- Interoperability of systems between the GIG COI's and whether software agents (which understand one ontology) will be able to effectively (e.g., semantically) communicate with other agents (having a different ontological representation).
- Limitations associated with the reasoning capabilities of software agents in the GIG, and whether human-agent cooperation will be necessary and can agents learn from this interaction?

Lastly, this chapter has outlined the potential limitations of web service technology to support the full operational concept of the GIG, and discussed the role of competing architectures such as JXTA, Neurogrid and CoABS grid. It is unclear how web service technology will evolve to meet the needs of the GIG. For example, industry watchers now proclaim the next big revolution to be grid services, which offers a mechanism to enable, among other things, reliability in accessing services through new WSDL specifications. The continuing evolution of web services and related technology will certainly impact the deployment of software agent technology in the GIG. The big question is "*will there be a single technology that provides the infrastructure for the GIG, or will there be several complementary technologies that also provide better support for software agents?*" If the latter is true, questions of how to best bridge the applications that rely on different technologies will need to be answered?

9. REFERENCES

1. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng, "DAML-S: Semantic Markup for Web Services", In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, July 30-August 1, 2001
2. Berners-Lee, T. Hendler, J. Lassila, Ora. (2001). The Semantic Web. *Scientific American*. 279(5), pp. 35-43 (2001).
3. Carlton, B. Scrudder, R. Black, C. Hopkins, M. Initialization of C4I Systems and Simulation Federations – Today and in the Future, In *Proceedings of the 2004 Fall Simulation Interoperability Workshop* (2004)
4. C2IEDM, http://www.mip-site.org
5. Control of Agent Based System Grid, http://coabs.globalinfotek.com
6. DARPA Agent Markup Language, http://www.daml.org
7. Defense Advanced Research Projects Agency, http://www.darpa.mil
8. Defense Information Infrastructure (DII) Common Operating Environment (COE), http://diicoe.disa.mil/coe
9. ebXML Web Site, http://www.ebxml.org
10. eXtensible Markup Language, http://www.w3.org/XML/
11. Foster, I. Kesselman, K. Tuecke, S. The Anatomy of a Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15(3), pp. 200-222, Fall 2001.
12. Global Command and Control System Common Operational Picture Reporting Requirements, CJCSI 3151-01A (19th January 2003)
13. Global Information Grid (GIG), http://ges.dod.mil, http://www.disa.mil/ns/gig.html
14. Hawkins, J. Defense Information Systems Network (DISN): Policy, Responsibilities and Processes. July 2003, http://www.dtic.mil/cjcs_directives/cdata/unlimit/6211_02.pdf
15. Horizontal Fusion, http://horizontalfusion.dod.mil
16. Hypertext Markup Language, http://www.w3.org/MarkUp/
17. Joint Tactical Radio System, http://jtrs.army.mil
18. Joint Warfare Simulation, http://www.caci.com/business/systems/simulation/jwars.shtml
19. Joseph S. (2002) "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks." In *Proceedings of the International Workshop on Peer-to-Peer Computing* (co-located with Networking 2002), Pisa, Italy, May 2002.
20. Meyerriecks, D. Net-Centric Enterprise Services. *Military Information Technology Online Archives,* **7** (3), 2003.
21. Mittu, R., Walters, J., Abramson, M., "Improving Simulation Analysis through Interfaces to C4I systems and Simulations", In *Proceedings of the 2004 Spring Simulation Interoperability Workshop.*, Crystal City, VA.
22. Naval Doctrine Command. *Naval Doctrine Publication 5: Naval Planning*. 1996
23. Peer-to-Peer Computing, http://www.openp2p.com
24. Project JXTA Web site, http://www.jxta.org
25. Rao A. S. and Georgeff M. P. "BDI agents: from Theory to Practice". In *Proceedings of the First Intl. Conference on Multiagent Systems*, San Francisco, 1995.
26. Resource Description Framework, http://www.w3c.org/RDF/
27. Resource Description Framework Schema, http://www.w3.org/TR/rdf-schema/
28. Sherman, Doron. BPEL Unleashed: Putting a Modern Business Process Execution Standard to Work. *Web Services Journal,* **5** (1), pp. 18, 34-36.
29. Sun Microsystems, Jini Network Technology: An Executive Overview, http://wwws.sun.com/software/jini/whitepapers/jini-execoverview.pdf

30. Tambe, M. "Agent Architectures for Flexible, Practical Teamwork". In *Proceedings of the National Conference on Artificial Intelligence*, 1997.
31. Theatre Battle Management Core System, http://www.fas.org/man/dod-101/sys/ac/equip/tbmcs.htm
32. Tolk, Andreas, Pullen, J. Mark, Sudnikovich, W. Hieb, M. "Developing Battle Management Language into a Web Service". In *Proceedings of 2004 Spring Simulation Interoperability Workshop.* Crystal City, VA.
33. Wooldridge, M. *An Introduction to MultiAgent Systems*, John Wiley and Sons, 2002.
34. World Wide Web Consortium (W3C), http://www.w3c.org
35. WSFL: http://xml.coverpages.org/wsfl.html
36. XLANG: http://xml.coverpages.org/xlang.html
37. http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/
38. DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services",*Proceedings of the International Semantic Web Working Symposium* (SWWS), July 30-August 1, 2001