Decision Support and Visualization in a Space Situational Awareness C2 Application

TOPIC: Space C2 Applications

Stuart Aldridge
21st Century Systems, Inc.
9830 S Flower Ct
Littleton CO 80127
720-981-8731
720-981-5893 (fax)
stuart@21csi.com

Dr. Alexander Stoyen
21st Century Systems, Inc.
12152 Windsor Hall Way
Herndon VA 20170
571-323-0080
571-323-0082 (fax)
alex@21csi.com

Jeffrey Hicks
21st Century Systems, Inc.
704 Edgewood Blvd
Papillion, NE 68046
402-272-7474
571-323-0082 (fax)
jeff@21csi.com

Dr. Plamen Petrov
21st Century Systems, Inc.
12612 Decatur Street
Omaha, NE 68154
402-384-9893
402-493-3168 (fax)
plamen@21csi.com

© Copyright 21st Century Systems, Inc. 2001-2003

Abstract

As we progress toward a network-centric environment, our dependency upon space assets to meet our force enhancement requirements is expected to grow exponentially. With so many mission assets parked in orbit, it is important for the commander to be operationally aware of the dispositions and movements in this fourth medium. The success of terrestrial forces requires a comprehensive space situational awareness (SSA). 21st Century Systems, Inc. is essentially developing a SSA and decision support client application that, with the proper inputs, will provide an integrated space picture. We call this concept, SituSpace. The visualization uses two and three dimensional depictions to provide the watchstander with rapid, intuitive situational awareness. SituSpace also employs intelligent software agents to provide timely alerts and cogent recommendations. The intelligent agents apply the user's ROE, situation, resources, etc. to rapidly derive actionable recommendations in order to accelerate the watchstander's decision loop. The SituSpace concept gains synergy from combining two complementary ideas. Using a metalanguage as the medium of storage and interprocess communication provides for interoperability, flexibility, and multilevel abstraction in the handling and passing of data elements. And a modular publish-subscribe software architecture which supports intelligent agents provides flexibility and tailorability to the SituSpace concept.

Overview

As we progress towards the network-centric environment envisioned by JV2010 and JV2020, our dependency upon space assets to meet our force enhancement (C4, weather, navigation, intelligence, surveillance, etc.) requirements is expected to grow exponentially. This trend is not only for our forces, but the forces of our potential adversaries, as well. With so many mission critical assets parked in various orbit types above the theater, it is important for the commander and his staff to be operationally aware of activities in this fourth medium, space. The theater commander's battlespace awareness must now extend many miles above the earth's surface into this arena.

Indeed, the task of space control is rapidly becoming a necessity. The success of terrestrial forces requires a comprehensive space situational awareness (SSA) and, very soon, will necessitate a space control capability. 21st Century Systems, Inc. is essentially developing a SSA and decision support client application that provides an integrated space picture capable of being used in the space control arena. We call this concept, SituSpace. The visualization uses two dimensional and three dimensional depictions to provide the theater and space force C2 watchstander with rapid, intuitive situational awareness. SituSpace will also employ intelligent software agent technology to provide timely alerts and cogent recommendations. The intelligent agents will apply the user's rules of engagement, situation, resources, and more to rapidly derive actionable recommendations in order to accelerate the decision loop of the C2 watchstander.

© Copyright 21st Century Systems, Inc. 2001-2003

With the proper inputs, this application will provide a single integrated space picture (SISP) for many C2 users.

Any foray into monitoring the space environment runs into two early challenges. Immediately, there are a great number of objects in orbit and that number is growing. These objects range from operating satellites to derelicts and debris. And there is a growing diversity in the cadre of watchstanders interested in SSA. The first challenge is how to prevent this abundance of objects from overwhelming the user and system. The second challenge is how to develop a system that meets the needs of as many of these users as possible.

The SituSpace concept gains synergy from the combining of two complementary ideas. First is the use of a metalanguage as the medium of storage and interprocess communication (IPC). A metalanguage, such as Extensible Markup Language (XML), provides for interoperability, flexibility, and multilevel abstraction in the handling and passing of data elements. The second source of inherent value in the SituSpace concept is the use of a modular publish-subscribe software architecture which supports intelligent agents. This has allowed us to model and to display synthetic environments with remarkable ease.

This Small Business Innovative Research (SBIR) project has completed its first phase and a proof-of-concept prototype was delivered. This proof-of-concept prototype was further enhanced in the option portion of the first phase. The next two-year-long phase is slated to enhance the decision support aspect and to add terrestrial and extraterrestrial weather into the visualization. This project is supported by the U.S. Army's Space and Missile Defense Command (SMDC).

Space Teems with Life

Even if we consider only objects in close earth orbit and larger than a particular size, there are still a great number of objects that must be accounted for. The USAF tracks more than 8,000 objects in orbit around the earth. Most are debris, such as spent rocket bodies, but a large number are operating satellites, both U.S. and foreign, military and civilian [A98]. And any SSA software application must be able to handle this number of objects (from a performance standpoint), yet be able to filter through the clutter in order to provide rapid situational awareness. The SituSpace project attempts to do just that.

Storage and Distribution

One linchpin of the SituSpace effort is the use of a metalanguage to perform activities such as data aggregation from disparate sources, distribution of serializable objects, and display of space object data. XML is the preferred technology in many information-transfer scenarios because of its ability to encode information in a way that is easy to read, process, and generate. Indeed, the Joint Technical Architecture (JTA) referenced XML for document interchange. The JTA stated, "*XML allows domain specific markup languages and customized, application-specific markup languages to be defined through the use of application profiles using application-specific tagged data items. This allows XML tags to be used to represent concepts at multiple levels of*

© Copyright 21st Century Systems, Inc. 2001-2003

abstraction, facilitate metadata searches, provide direct access to data described by the metadata, and provide information as to how to obtain data that is not available directly on-line. Finally, XML allows new capabilities to be defined and delivered dynamically" [JTA2].

The burning question of this research is "Can XML be used to represent a number of space objects in a software application?" The answer to this question is a qualified "yes." In this particular application, XML would play a role in the representation/storage of the space objects and in the transfer of space object information in a distributed system.

```
<XML ID="identifier">
<space-object>
    <intldesignator>YYYYNNNABC</intldesignator>
    <designation>dmsp_2</designation>
    <coe>
        <eccentricity>0.2</eccentricity>
        <raan>090.23</raan>
        <argofperigee>35.234567</argofperigee>
        <inclination>23.23987654</inclination>
        <epoch>35.234567</epoch>
        <meananomaly>123.23954</meananomaly>
    </coe>
    <mission>
        <operator>NPOESS</operator>
        <missiontype>force_enhancement</missiontype>
        <missiondetail>environmental_monitoring</missiondetail>
        <sensor>
            <sensortype>visible</sensortype>
            <fieldofview>120</fieldofview>
        </sensor>
        <sensor>
            <sensortype>infrared</sensortype>
            <fieldofview>150</fieldofview>
        </sensor>
    </mission>
    <weight>2545</weight>
        .
        .
        .
    <constellation>2</constellation>
</space-object>
</XML>
```

Figure 1 – Rudimentary XML Space Object

Metalanguages are languages to describe other languages and objects. They have a great deal of flexibility. But there is a cost associated with this flexibility in the form of verbosity. The tags that are used to make the data portable also add to the processing time and network bandwidth. The XML object in Figure 1 has approximately 70% tag characters (in yellow) versus 30% actual data characters. Just as overhead associated with headers in network communications is referred to as header tax, I would characterize this as a "tag tax."

This "tag tax" would not be as challenging an issue were it not for the number of objects that will be involved in this application. The sheer number of space objects makes the data element inclusion decision and the naming decision very significant. There is a tradeoff in performance for each additional element of information.

The issue of the transfer of space object information in a distributed system appears to an issue that can be overcome. In corporate America, the most significant use of XML is for data transfer among businesses to replace older, more expensive EDI systems [A00]. Once again, the issue of the "tag tax" must be dealt with. David Hayes, in "The Potential of XML," stated "The XML metadata in the form of data tags creates a transmission overhead. The metadata imposes an added burden on network capacity and processing to parse the message." Mr. Hayes also offered insight into addressing this issue, "Data compression, local computation and manipulation of data, intelligent communication of knowledge as opposed to raw data, and granular updates are mechanisms that mitigate the network load penalty." [H00]

© Copyright 21st Century Systems, Inc. 2001-2003

Clutter Management

The plethora of objects in orbit presents another challenge to providing intuitive SSA. The display, either two dimensional or three dimensional (or both) must strike a delicate balance between providing data and overwhelming the watchstander. Too little data and the watchstander doesn't get an accurate depiction. Too much data and the watchstander "can't see the forest for the trees" or the flood of data delays understanding. Figure 2 below shows the inherent clutter associated with a two dimensional display of only 200 objects.

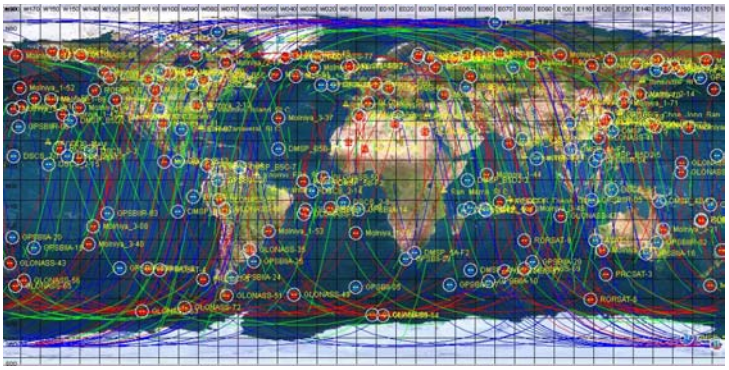

Figure 2 – Cluttered screen with only 200 objects

There are several ways to gain control of the clutter. One such method is to allow the watchstander to define the parameters of the objects that he/she is interested in viewing in the display. The watchstander would select the categories and types of the objects of interest. This is done by filtering out those objects that do not meet these criteria. One advantage to this method is the user can tailor what is displayed to those objects associated with his/her mission. A disadvantage is the watchstander can induce clutter through ineffective filtering by trying to see too much. A typical filtering mechanism is shown in Figure 3.

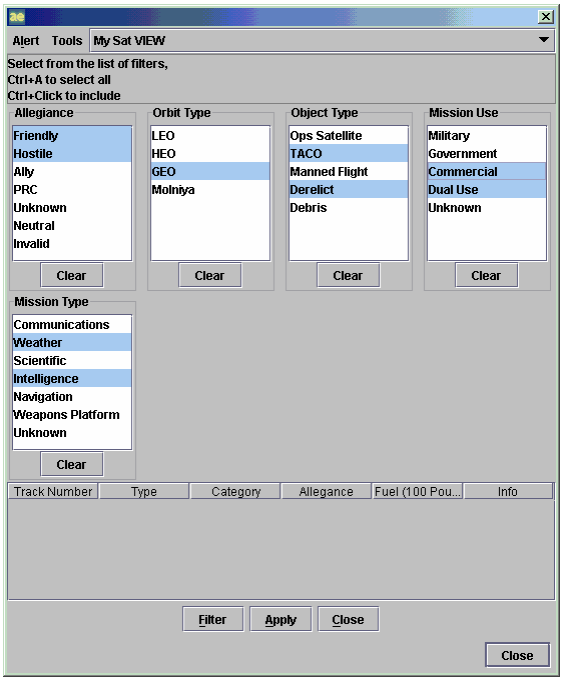

Figure 3 – SituSpace Filter Storyboard

A second, more effective method of controlling the clutter is to use intelligent software agents to aid in the filtering and decluttering process. This works in two ways. Intelligent software agents can be used to "coach" the user through the filtering process in order to reduce the chances of over-filtering and under-filtering. It goes without saying that the watchstander has override authority over the software agent. The software agents can also assist in real time by highlighting objects of interest to the watchstander. Using watchstander input, the software agents can be "trained" to alert the watchstander and to highlight an object of interest when it appears or acts in a particular manner. The watchstander has the option to retain the

© Copyright 21st Century Systems, Inc. 2001-2003

highlight in order to more rapidly acquire the object in future use or to remove the highlight if the object is no longer of interest.

Handling Diversity

While it is traditional for military commands with a space charter to desire a space picture that provides SSA, most military and government entities have more recently come to the conclusion that activities in space (both manmade and otherwise) can have an impact on operations. The average military unit, becoming dependent on force enhancement capabilities provided from orbit (navigation, intelligence, meteorological, etc.), has a vested interest in the space situation. A simple example from a requirements document for a Navy surface combatant:

> *CVNX shall be able to develop, maintain, and display consolidated situational awareness including integrated air/space picture, integrated ground picture and integrated maritime picture including subsurface for any area a carrier could be assigned."*
> *(from ORD for Future Aircraft Carrier, CVNX, 23Feb2000)*

Thus, there is an increasing diversity in the watchstander desiring SSA. In order to meet the challenging needs of this growing and evolving user segment, a mature, flexible software architecture is critical.

Getting an AEDGE on the Challenge

The diversity of the watchstander and his/her environment means a SSA software application must operate differently for different watchstanders. A watchstander who is not "handcuffed" to the console may require verbal alerts from the application (to gain one's attention from across the room). Another may operate in an environment where this would not be practical (i.e., due to noise). Another watchstander may only be interested in the swath or footprint of the satellite's payload on the earth, while another, in a space control role, may be interested in the space environment.

In order to meet the various needs of the potential user base, we required a software architecture where components could be added or removed seamlessly (like in the case of a speech synthesis capability). A publish-subscribe architecture is one such environment. Additionally, since various watchstanders have differing alerting requirements, an architecture that supports easy tailoring of intelligent software agents is desirable.

For a common integrated architecture for our application, we choose to use 21CSI's existing agent architecture, AEDGE™. So the burning question of the moment is, "What are the performance and reliability gains of AEDGE™ over a more mature distributed object technology?"

AEDGE can roughly be classified as middleware built on top of distributed object-oriented technology (so far based on Java, with a number of extensions) with native support for agents, extensible data representations, and flexible user interfaces. AEDGE is a robust, high-

© Copyright 21st Century Systems, Inc. 2001-2003

performance, flexible framework designed specifically for building agent-based, near real-time decision support systems. In order to explain the benefits of this approach, let us review the underlying and preceding technologies.

In the paradigm of object-oriented programming (OOP) languages, distributed computing is usually supported by language extensions for distributed objects. While standard OOP languages are quite powerful in defining control and data structures for single processors, most of them do not feature mechanisms for defining and implementing various object distribution models. Thus, with the development of networked and distributed computing, various extensions and models were developed. Object Management Group (OMG)'s Common Object Request Broker Architecture (CORBA) is now considered a standard object distribution model. CORBA defines object location and method invocation services for remote (as well as local) objects, which supports transparent distributed interactions. Multiple implementations of CORBA and other Object Request Brokers (ORBs) exist, for a variety of languages and operating systems. CORBA's advantages of transparency and wide interoperability come at a high price of considerable overhead and relatively low (and unpredictable) performance, partially due to inefficient implementations and partially due to CORBA's inherent architectural complexity.

Microsoft introduced its own approach to distributed object oriented computing – the Distributed Component Object Model (DCOM) – an extension of COM, which defines services and interactions with remote COM objects. While DCOM addresses some of the performance issues, its availability for platforms other than Windows/PC is limited. The model is inherently dependent on a Microsoft-specific implementation of features and is thus hard to analyze with external tools.

Sun Microsystems's Java was designed and built as an advanced network-friendly OOP language with built-in possibilities for extensions. Java's version of distributed object support, the Remote Method Invocation (RMI), borrows from the ORB model and from the older, well-understood Remote Procedure Call (RPC) model. RMI provides remote object location services and a skeleton-stub implementation of RPC. It is more efficient, more flexible, but less programmer-friendly than CORBA or DCOM.

AEDGE is developed in the object-oriented paradigm. Its basic data units are objects, which are capable of writing themselves to a stream and reading themselves back from a stream (i.e. they are serializable, in the Java terminology). In addition, these generic AEDGE data objects support introspection during run-time, which enables the user to peek into any of them and even change their structure, on the fly. This presents tremendous capabilities in terms of system maintainability and robustness – for instance, the operator of an AEDGE-based system can isolate part of the system, while another part takes over the suspended functions, then modify data structures, rules, even expressions and conditions, then safely bring the modified services back, all while the system is still operating (note that no shutdown, reboot, recompile or even linking and restarting was required). Thus, AEDGE implements a flexible and powerful distributed object model, based on existing services (CORBA or RMI where available) or on its own XML-based services over TCP/IP sockets, where required.

© Copyright 21st Century Systems, Inc. 2001-2003

In AEDGE, agents are represented through the common serializable framework of objects. This means that agents are mobile by design (they can serialize themselves, transmit themselves as "data" in a message and the AEDGE environment will re-instantiate and initialize them on the receiving side). The user is free to specify when, where, and how the agents are to migrate from one AEDGE node to another. An agent can even "bootstrap" the AEDGE environment on a fresh node, provided the necessary system permissions and basic services (such as RMI or CORBA) are present. With that in mind, the AEDGE agents are designed with an inherent balance between flexibility and performance. While the user has the power to alter, redesign and create new agents, the internal execution model is still based on language primitives, which are compiled in one form or another (Java byte-code or native code), which enables the framework to maintain the requirement for high performance much better than purely interpretive or rule-based systems.

Several research communities have modeled distributed computing by studying communication and coordination mechanisms among autonomous software entities, or Agents. Agent-based computing focuses on the interaction mechanisms among agents, which permit a rich set of coordinated activities. Effective models of interaction require the following basic capabilities: 1) a transport mechanism to convey messages in an asynchronous fashion, 2) an interaction protocol, defining the available types of communications and their semantics, 3) a content language providing the base for composition of requests and their interpretation, and 4) an agreed-upon set of shared vocabulary and meaning of concepts (often called on ontology). The most common foundation technology used for such agent-based architectures is the Knowledge Query Manipulation Language (KQML) [LF97]. KQML specifies interaction protocols by defining symbolic performatives to represent information about the purpose of a communication. Since it uses a standardized representation of conversational interactions, KQML is limited by its reliance on a fixed set of atomic performatives. Arriving at just the right set of performatives in the ontology has been a major hurdle in this and other approaches.

Another approach to implementing the fundamental capabilities for Agent-based computing is structuring the agent's activities around the concepts of Belief, Desire, and Intention (BDI) [RG91]. While BDI's emphasis on a higher level of abstraction has been important in giving direction to work on agent-based systems, its applicability may be limited by the structural requirements posed on individual agents. BDI makes stronger assumptions about the knowledge availability and processing within agents, which induces difficulties in operating with largely-legacy systems.

The Open Agent Architecture (OAA) [MCM98] also provides a framework for Agent-based computing. OAA focuses on a balanced set of objectives, including efficient interoperation, autonomy, coordination, flexibility, and extensibility. The architecture incorporates a promising set of new technologies, though it is still under development. Practical applications of OAA will require improvement in scalability and robustness, as well as the construction of new development, testing, and profiling tools.

The AEDGE architecture uses an amalgam of these approaches in a way that maximizes flexibility while not compromising performance. AEDGE's framework does rely on a finite number of primitives (or data abstractions), however it does not limit or constrain those

© Copyright 21st Century Systems, Inc. 2001-2003

primitives – the data framework can change and grow even after the AEDGE-based system has been deployed, even while online. The domain specific knowledge in the system is encoded and preserved in the AEDGE agents. Rules and number-crunching algorithms can be imported from databases and libraries, while the agents know how to access those via bridges and interfaces. AEDGE defines a number of standard database and data format bridges, though we recognize that we cannot possibly provide a complete set of converters, access protocols, and interfaces. This is why AEDGE has been designed to support extensions that will convert new unknown data formats to AEDGE's framework structures that are known to its agents.

Publish-Subscribe for Flexibility

As mentioned above, the ability to easily add or remove sophisticated software components is a must for a SSA application. The needs of the diverse user base drive the necessity to tailor the overall application with differing services. And, it would be cost prohibitive if the software had to be rewritten in each case.

In the AEDGE architecture, components communicate among each other via the Service Provider/Service Requester Protocol (SPSR). Service providers are components that implement an algorithm or need to share their data (data sources). Service requesters are the components that need a function performed for them by some other component or need to import data from another component. Both service requesters and service providers implement remote interfaces, which enables such components to communicate over a TCP/IP network. The remote interface implementation is currently based on Java RMI (remote method invocation, a type of simplified ORB service), though the Architecture is not dependent on this implementation.

The SPSR protocol is based on three data objects: Service, ServiceResult and Message. The Service object encapsulates the class, the type, the required quality of service (QoS) and the parameters of a service request. The ServiceResult object provides a reference to the original

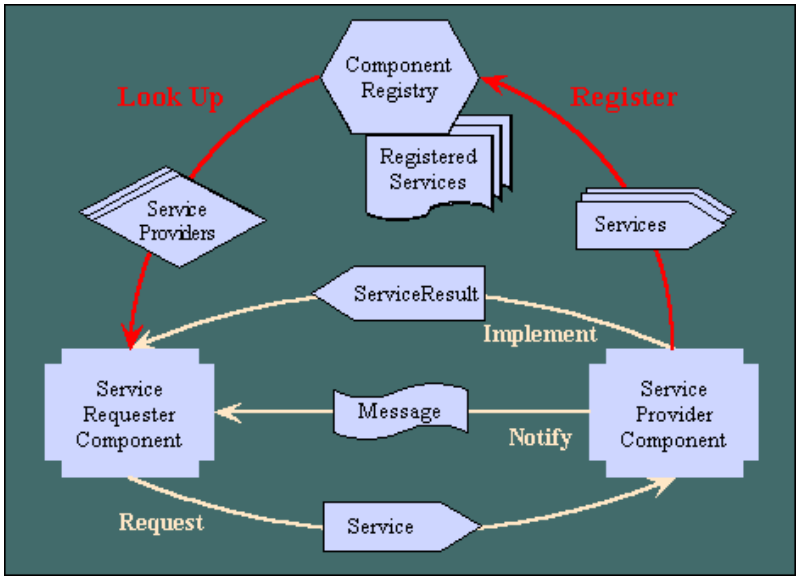

Figure 4 – SPSR Protocol Diagram

© Copyright 21st Century Systems, Inc. 2001-2003

service, a return code (success or failure), a return object (String, Recommendation, etc), and an actual received QoS. Messages provide a way for service providers to advertise the availability of new services and to notify subscribers of new data available.

Service provider components register their location and the services they provide with a Component Registry, which is responsible for tracking and maintaining up-to-date service provider information. Service requesters lookup service provider information from the Component Registry and then establish a direct connection with the providers they wish to engage. A service request (either blocking or non-blocking) is sent from the service requestor to the service provider. The provider then replies (immediately or at some future time) with a ServiceResult.

Tailored to Suit

As mentioned previously, the diversity of watchstanders and missions means meeting a diversity of need. One watchstander may be interested in force enhancement satellites and their impact on terrestrial operations while another may be tasked with maintaining awareness in the space realm itself (as in the space control mission). Each involves monitoring many of the same on-orbit assets, but requires specific, distinct alerting and recommendations. From a software standpoint, this implies the same types of intelligent software agents with specifically tailored differences. This is another reason we went with the AEDGE COTS product.

Agents in AEDGE are specialized components that generate recommendations either in response to a user inquiry or spontaneously, according to their function. Agents are usually organized in agent communities, unified under an Agent Manager component, which is responsible for invoking and synchronizing individual agents.

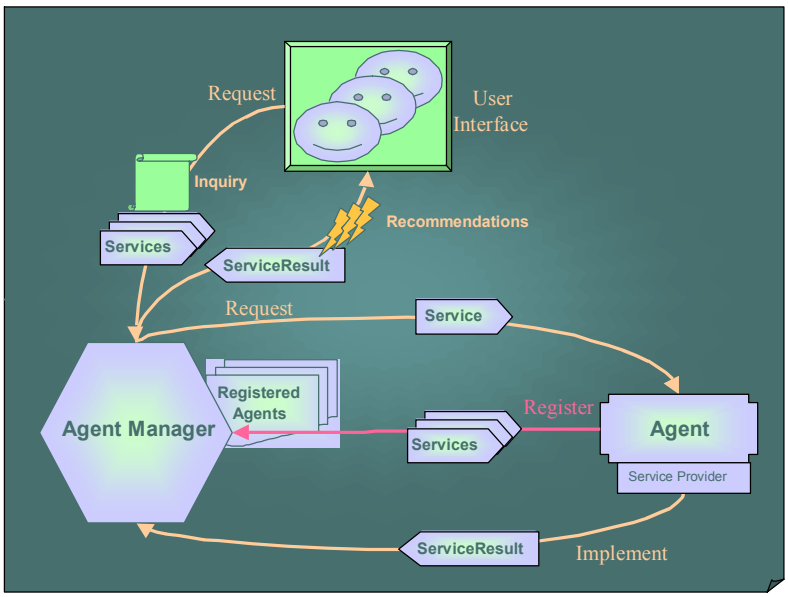

Figure 5 – Agent Components over AEDGE Services

© Copyright 21st Century Systems, Inc. 2001-2003

The Agent Manager interacts with agents via the SPSR protocol, while users (through UIs) interact with the Agent Manager through more user-friendly Inquiry/Recommendation exchange protocol (IREP). The users can query the agent manager by sending context information (entities, geo-references, target information, etc.) and specific requests for recommendations. The query is internally translated to service requests and sent to the Agent Manager. The users are not limited to the IREP – they can use any query representation, such as SQL queries, as long as they can be internally converted to service requests.

Upon receiving a user-level query, the Agent Manager selects and invokes the appropriate agents to perform the desired tasks. The Agent Manager has a table of registered Agents and their capabilities. Thus, the Agent Manager is the one that partitions the problem, sends sub-tasks to the individual Agents, and later combines and deconflicts to reach the solution. After an overall solution is reached, the Agent Manager forms a set of recommendations, which are returned to the User via a ServiceResult object. In essence the IREP is a user-friendly protocol build on top of the SPSR protocol.

The interactions among agents and the Agent Manager are solely based on the SPSR protocol, as these are optimized for efficiency and not necessarily for user-friendliness. Figure 6 demonstrates four different modes of User/Agent-Manager/Agent interactions, described below.

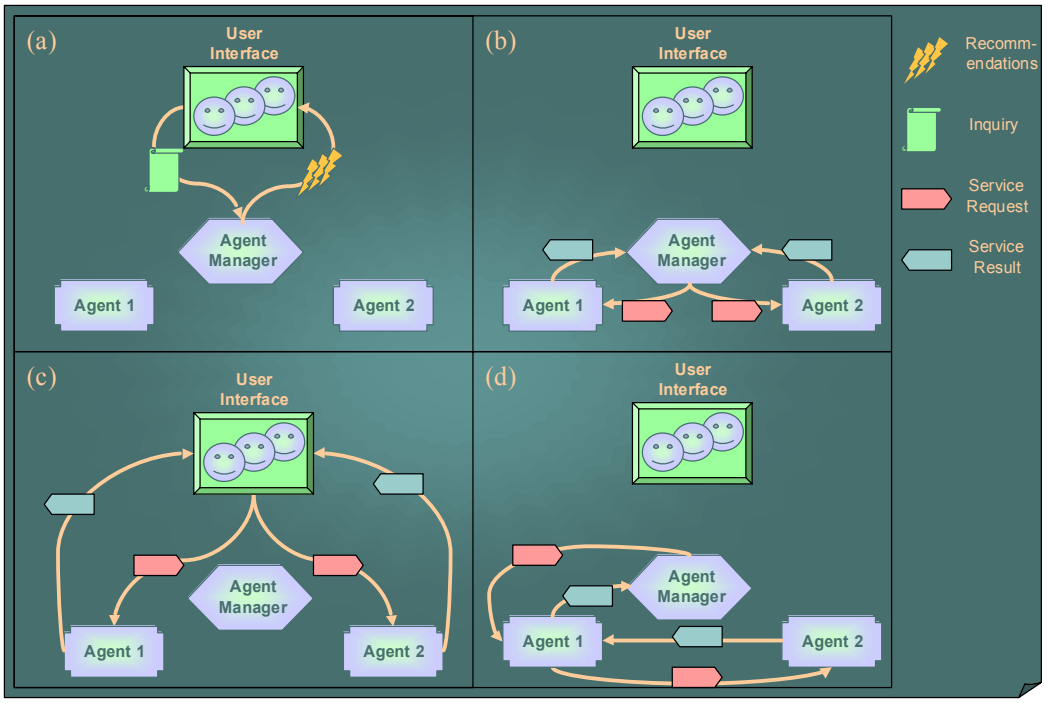

Figure 6 - Different modes of agent interactions: (a) User to Agent Manager; (b) Agent Manager to individual agents; (c) User bypasses Agent Manager; and (d) Agent-Direct interaction.

User to Agent Manager Interactions. Essentially the user sends an inquiry to the Agent Manager, based on the user's current needs and query representation language. The inquiry may consist of a task description (e.g. "Check Fuel 1022" or "Check Range 1022 from 1021"), and optionally a context update, such as platforms, targets, geo-references, etc. The inquiry is internally

© Copyright 21st Century Systems, Inc. 2001-2003

serialized and translated into service requests, which are then sent to the Agent Manager via the SPSR protocol. After the Agent Manager performs the requested tasks, it sends a reply in the form of a set of recommendations. Recommendations are core objects in AEDGE's framework, which represent desired actions and commands. Recommendations may be produced by both Agent Managers and users and are interpreted by Entities to form tasks and orders. In this case Recommendations are generated by the Agent Manager and sent for approval to the User.

Agent Manager to Individual Agents. In this interaction the Agent Manager partitions the task to subtasks for the individual agents, registered under the Manager. Subtasks are then sent to the agents via the SPSR protocol, encapsulated in Service objects. After the individual agents arrive at a solution they respond to the Agent Manager with ServiceResult objects, which are interpreted by the Manager. The Agent Manager performs synchronization and deconfliction of the individual agents' results to ensure that the user will receive a coherent set of recommendations (in the event individual agents provide conflicting information).

User bypasses Agent Manager. The user can interface directly with the individual agents, using the SPSR protocol. If the user process can locate the Service Provider of an agent (via a Component Manager where that agent is registered), the user can send service requests directly to the agent and listen for the ServiceResult object in the reply. This places the burden of locating and interfacing with the agent's service provider on the user, but it provides more flexibility and faster response.

Agent-Direct interaction. Agents can communicate with each other indirectly (through the Agent Manager) or directly (via the SPSR protocol). The Requester agent looks up other agents' service providers from any component manager (including the Agent Manager) and can then send service requests to other individual agents. The Provider Agent handles the service request just like it would handle a request from the Agent Manager. The Requester agent needs to be able to handle the ServiceResult returned by the Provider. Agent-direct interaction provides the flexibility of extending the agent community that belongs to an Agent Manager without having to modify the login of the Manager itself.

The Future

Since Phase I of this project clearly demonstrated technical feasibility of the SituSpace concept, the next step is to take the concept "out of the laboratory" and demonstrate it in the field using real operators [A02]. The Phase II SituSpace application, using inputs from the actual watchstanders slated to use this type of application, will be tailored to provide intuitive space situational awareness and effective decision support for those operators. The Phase II SituSpace application will be extended to integrate atmospheric data (both terrestrial and extraterrestrial). And, in Phase II, we will elicit feedback from the users in order to produce superior information vice superior data. We will continue to focus on SituSpace's critical elements: intuitive visualization and efficacious decision support. [A03]

First and foremost, the SituSpace application must be tailored to the needs of the users … the watchstanders who will use this decision support system to perform their tasks more

© Copyright 21st Century Systems, Inc. 2001-2003

intelligently. A true decision support system should allow the user to think at the higher level by presenting information (vice data), by "handling" the innocuous details, and by allowing the user to address the complex, human-in-the-loop decisions with confidence. Thus, the first step is to obtain watchstander input as to what the system should do for them.

Secondly, SituSpace will be extended to incorporate environmental effects, both terrestrial and extraterrestrial. The user should be kept apprised of the effects earth-bound weather has on mission accomplishment and the effects space weather anomalies have on on-orbit assets.

Lastly, it's one thing to detect, predict, and react to a situation. But, a true decision support system would also provide information as to the possible effect of the actions. This is known as Level 3 fusion and this is derived from applying the parameters of the situation to information typically maintained in a database (troop dispositions, population densities, etc.). SituSpace will be extended to incorporate this type of functionality, as it applies to the user.

© Copyright 21st Century Systems, Inc. 2001-2003

References

[A98] David Atkinson, "USAF Battlelab to Improve Satellite Detection, Defense Weekly 10 August 1998.

[A00] Ahmad Abualsamid, "A Metalanguage For the Ages," Network Computing, April 3, 2000

[A02] Stuart L. Aldridge, "SituSpace-Space Battlespace Awareness Application," Phase I Final Technical Report to US Army Space and Missile Defense Command, August 2002.

[A03] Stuart L. Aldridge, "SituSpace-Space Battlespace Awareness Application," Phase I Option Final Technical Report to US Army Space and Missile Defense Command, January 2003.

[H00] David Hayes, "The Potential of XML," Space and Missile Defense Technical Center Paper, 18 Dec 2000.

[JTA2] Department of Defense Joint Technical Architecture Version 2.0, 26 May 1998.

[LF97] Y. Labrou and T. Finin. A Proposal for a New KQML Specification. Technical Report TR CS-97-03, Computer Science and Electrical Engineering Department, University of Maryland Baltimore County, Baltimore, MD, 1997.

[M00] Brett McLaughlin, Java and XML, O'Reilly & Associates Inc, Sebastopol CA, June 2000

[M99] JP Morgenthal, "Portable Data/Portable Code: XML & JAVA™ Technologies," White Paper prepared for Sun Microsystems, Inc, 1999.

[MCM98] D. L. Martin, A. J. Cheyer, D. B. Morgan. The Open Agent Architecture: A Framework for Building Distributed Software Systems, 1998.

[Oster01] Chad Oster, Improving Satellite Information Displays: A Study of Human Factors in Screen Design, American Institute of Aeronautics and Astronautics Paper 2001-4733.

[RG91] A. Rao and M. Georgeff. "Modeling rational agents within a BDI-architecture." Proceedings of Knowledge Representation and Reasoning, p. 473--484, 1991.

© Copyright 21st Century Systems, Inc. 2001-2003