Application of Service-Oriented Architecture to Distributed Simulation

Duncan Thomson*
The MITRE Corporation, McLean, VA, 22102

The paper describes an effort to apply Service-Oriented Architecture (SOA) principles and technologies in a distributed simulation environment. The goal of this effort was to develop and prove concepts for using SOA to increase the flexibility and ease-of-use of a set of corporate assets that are used for analyzing future concepts for the National Airspace System (NAS). These assets include simulations, algorithms, visualization tools, and data archives. The paper provides background information on SOA principles and technology, as well as some general discussion of how SOA can be applied to distributed simulation. Performance issues that arise when using SOA technologies are discussed, and data from a set of performance experiments is presented and analyzed. The paper then describes concepts we developed and prototypes we built to demonstrate and prove these concepts. The concepts include some unique ideas for controlling simulations and managing simulation contexts by creating a simulation-aware extension to a SOA registry, as well as concepts for allowing different assets to be combined without programming.

Nomenclature

Mbps = Megabits per second (1 Mbps = 10^6 bits per second)
ms = milliseconds

I. Introduction

SERVICE-Oriented Architecture (SOA) is an architectural approach in which system components expose functionality and data in the form of "services" that may be accessed by other components using standards-based technologies. SOA is particularly applicable to architecting systems of systems. Distributed simulations can be thought of as systems of systems; therefore, it makes sense to consider how best to apply SOA to distributed simulation environments. We postulated that SOA principles could be used to improve flexibility and allow simulation assets to be reconfigured in new ways to solve problems, and we set for ourselves the challenge of proving this concept through implementation and experimentation. This paper describes the results of this proof-of-concept effort.

This paper focuses on how we applied SOA to MITRE assets such as simulations, algorithms, visualization tools, and data archives, in order to increase our flexibility and speed in applying these assets to solve problems for our sponsors. It should be mentioned that we had another motivation for this work. The Federal Aviation Administration (FAA) is currently beginning to apply SOA principles and technologies to the National Airspace System (NAS) in the System Wide Information Management (SWIM) project, which is a first step towards the network centric operations capability that is fundamental to the Next Generation Air Transportation System (NextGen). The MITRE Corporation's Center for Advanced Aviation System Development (CAASD) is supporting the FAA in these efforts. By applying SOA principles and technologies to interconnect our simulations of the NAS, we expect to gain knowledge that can be used when applying these same principles and technologies to interconnect the systems that make up the NAS. However, this paper focuses on our use of SOA for distributed simulation and analysis tasks; direct application of SOA to the NAS and NextGen is not covered here.

SOA principles and technologies, described further in Section II, are currently being applied by many organizations to a variety of missions and environments, but distributed simulations have some characteristics that

* Senior Principal Network Systems and Distributed Systems Engineer, Dept. F080 – Safety and Performance-Based Services, The MITRE Corporation, Mail Stop N660, 7515 Colshire Drive, McLean, VA, 22102-75397.

are different from the typical business processing environments that are the primary area in which SOA is currently being applied. In particular, simulations often have stringent performance requirements. Some applications of SOA may also have stringent requirements (e.g., stock trading), but in most environments in which SOA is applied delays of seconds, minutes, or even more are acceptable for any one transaction. Section III of this paper discusses the performance issues associated with SOA technologies, and describes experiments we conducted to obtain quantitative measurements, followed by an analysis of the implications for use of SOA technologies for real-time and fast-time simulations.

Section IV describes our prototyping efforts, in which we have created a proof-of-concept SOA Framework for our simulation and analysis environment. In this proof of concept, we implemented services for access to static data archives, dynamic simulation-dependent data, and algorithms. Different types of legacy simulation capabilities were "web service enabled." In addition, we implemented services for simulation management, which allow an analyst to select which assets to include in a simulation, control the execution of a simulation, and access simulation results from visualization tools. Underlying these capabilities, we implemented a simulation-context-aware SOA registry. We believe that our registry implementation and our approach to simulation management through web services are unique, and may be of interest to others working with SOA and distributed simulation.

Section V concludes the paper and summarizes lessons learned.

II. Service-Oriented Architecture Concepts

This section provides some background on SOA concepts and technologies that are available to implement a SOA. Readers who are familiar with the subject matter may wish to skip directly to Section III.

A. Transition from Point-to-Point Integration to Loosely Coupled Services

In the early phases of a SOA transition described in generic SOA roadmaps, an organization is assumed to be starting with an environment in which different systems are interconnected with unique interfaces, engineered specifically for the purpose of interconnecting those specific systems. Transitioning to a SOA in such an environment begins with replacing point-to-point unique interfaces with industry standard protocols and data representation formats for interconnecting systems. In a SOA, interfaces between systems are defined in a standardized manner in terms of services that each system offers to other systems. Loose coupling is achieved through standardizing the syntax and semantics of the service requests and service responses.

In reality, however, things are likely to be somewhat more complex than transitioning from point-to-point interfaces to standard interfaces defined in terms of services. In most cases an organization will be starting with a variety of approaches to interconnecting systems, which may include one-of-a kind “tightly coupled” interfaces, but also probably include one or more generalized interconnection methods that are standardized to some degree. In addition to applying industry standards, an organization may have developed their own local standards for interconnecting systems. Within The MITRE Corporation, for example, we use standards such as Distributed Interactive Simulation[1] (DIS) and High Level Architecture[2] (HLA), in particular an extension of HLA known as AviationSimNet[3]. Within CAASD, we also make extensive use of internally developed local standards such as the Inter-Target Generator Protocol[4] (ITP). Loose coupling is a design goal of all of these standards, even though it may not have been expressed in those terms by the creators of the standards.

So we see that the notion of standards-based interfaces and loose coupling is not really new. Like many organizations, we have been applying these principles for years. What is new is the availability of SOA standards with much broader applicability, and powerful software tooling that supports the standards available from many vendors as well as open source providers. Our initial steps in transitioning to a SOA include application of these new standards, while we continue to use our existing standards such as HLA and DIS, as well as our own “local” standards such as ITP. We have begun to add SOA standards-based service interfaces as we build new components, and we have begun to build “wrappers” which hide the legacy interfaces behind a façade that presents the new, standards-based interfaces. It would not be cost-effective to recode all of our existing applications to conform to new interface standards. However, as we become more accustomed to using the newer standards in our simulation environment, and as our developers become accustomed to the significant software tooling support that is available, we expect our legacy interfaces to be used less, especially on new development, and may eventually be phased out entirely.

In the next section we provide specifics of the standards we have selected and how we are applying them in CAASD’s modeling and simulation environment.

B. Reliable, Discoverable Services

So far we have discussed services primarily as a means of interfacing systems. But a service is more than just the definition of an interface; saying that a system provides a service also implies that there is some location on the network where the service can be expected to be found at any given time, with some known availability and quality of service. In a SOA, consumers must be able to discover what services exist, where they are located, as well as information about the reliability and performance of the service, i.e., quality of service information.

To support these needs, some form of registry[†] is an essential component of a SOA. The registry serves as a central source of information for developers who need to understand how to build interfaces to other systems - serving as a master Interface Control Document (ICD) for the organization. But the registry does more than this. It also defines which services are available at given locations, what those services are expected to be used for, and what quality of service is being provided.

It is worth noting that use of a registry provides another form of loose coupling. In a traditional system-of-systems architecture, if multiple systems need to interact, the system operators must contact each other, exchange information, and then configure and run their systems so that they connect properly with one another. Or one operator may need to bring up multiple systems in one integrated configuration so that they all work properly together. In a SOA, one organizational entity can be responsible for the operation of a system that provides services, and another entity can have its systems connect to and utilize these services, based on information in the registry. Thus the registry facilitates loose coupling of system operation. If necessary to control access to a service (for example, to limit resource consumption in order to ensure performance) then provider-consumer Service Level Agreements (SLAs) may need to be negotiated before a service may be consumed.

C. Composable, Reusable Services

A key idea in SOA is that services developed for one purpose may be reused for other purposes that were not originally anticipated. New applications may be created by composing multiple services. Reuse is, of course, not a new concept. Software and systems engineers have always looked for ways to develop modules and interfaces to maximize reuse. SOA simply continues this process and offers some new approaches that facilitate reuse.

One of the ways that SOA facilitates composition and reuse is that, as discussed above, one system can utilize a service simply by accessing its service interface at a location on the network specified in the registry. The service continues to run on the same servers in the same environment, managed and maintained by the organization responsible for that environment. The user need not be concerned with everything it takes to get this service up and running. This is contrasted with reuse of software source code or object modules, which in order to be reused must be compiled or linked in to run in a new environment, which often creates numerous obstacles.

In addition, service composition may be facilitated in a SOA through the use of off-the-shelf, standards-based Enterprise Service Bus (ESB) components that are specifically intended to facilitate service composition. These components provide functions such as message format translation, content-based message routing, and so on, as well as higher level capabilities for business process management through service orchestration. We have ongoing research into the applicability of ESB technologies within our CAASD SOA Framework; however, discussion of ESB technologies is beyond the scope of this paper.

D. Enterprise SOA

There are several other areas that must be addressed in order to fully transition to an enterprise SOA. Some form of enterprise service management is necessary in order to allow service quality to be monitored and maintained. Security solutions may be needed to provide confidentiality, integrity, and availability of services, data, and systems. Finally, of critical importance, governance is needed to ensure that the SOA evolves in a coherent manner to meet the needs of the various stakeholders and the organization as a whole. We have begun addressing these items as we begin transitioning our SOA proof-of-concept prototype into more general use, however these topics are largely beyond the scope of this paper.

E. SOA Technologies and Standards

SOA is an architectural principle that could in theory be implemented with any number of different underlying technologies, and up to this point we have purposely kept our discussion of SOA concepts independent of any

[†] Some authors distinguish between a registry, which stores metadata about artifacts, and a repository, which stores the actual artifacts. In this paper we avoid making a distinction, and simply use the term "registry" to refer to a capability that provides information about web services within a SOA.

particular SOA technology. However, in practice the choice of technologies is important in determining the ease of implementation and the benefits obtained from a SOA transition. Our approach was to select those technologies that best meet our needs in applying SOA principles in our own environment. Since we wish to maximize our ability to utilize commercial and open source products and tools, one of our primary criteria in selecting technologies was to follow the "mainstream" and select technologies that are widely used and widely supported, and seem likely to continue to be so in the future. Technologies based on open standards are highly preferable, simply because open standards increase the likelihood of interoperable products being available from multiple vendors and the open source community.

The technologies and standards that we have selected as the best candidates for meeting our needs are discussed briefly below.

1. XML-Based Standards

Probably the most fundamental SOA standards have to do with how information is represented. Extensible Markup Language (XML)[5] is the clear choice for a universal standard for representing information. Related standards include XML Schema Definition Language (XSD)[6] and XML Style Sheet Translation (XSLT).[7] XML may be thought of as a basic language structure, but with no defined vocabulary. XSD offers facilities for creating vocabularies for XML by describing the structure and constraining the contents of XML documents. XSLT is a language for transforming XML documents into other XML documents.

The importance of XML and related standards cannot be overstated. The quantity of software tools that can process XML is huge and growing, and the availability of this family of standards and software tools vastly increases the ability of software developers to define and build interoperable interfaces. While the performance impacts from the use of XML are sometimes assumed to be prohibitive in real-time simulation applications, in fact we have found that in many cases the performance impacts are acceptable and outweighed by the advantages of XML. Performance issues are addressed directly in Section III.

2. Protocols and Web Service Standards

While in theory a SOA could be built on any set of network protocols, in practice today, not surprisingly, SOAs are implemented using the Internet Protocol Suite and the protocols used in the World Wide Web (WWW). Services implemented using WWW protocols are referred to as web services. There are currently two approaches to implementing web services. The first approach is known as Representational State Transfer (REST).[8] REST is not a standard, rather it is a style of implementing machine-to-machine interactions using Hyper Text Transfer Protocol (HTTP) in much the same manner as has been used by web browsers and servers for implementing human-to-machine interactions. The second approach uses an XML-based standard known as Simple Object Access Protocol (SOAP)[9] and a related family of web service standards issued by the World Wide Web Consortium (W3C), which we will refer to as WS standards.

With both REST and SOAP, services are invoked by sending a request to a server at a given Universal Resource Locator (URL), and receiving an XML message in response. With SOAP, the XML messages are encapsulated in a SOAP envelope, which may then be transported over HTTP (as well as other transport mechanisms). SOAP provides a standard message structure that includes a header and a body. The SOAP body contains the actual message content being provided as input to, or output from, a web service. The SOAP header can contain additional fields, for example fields containing security information. With REST, the XML message content is transported directly over HTTP.

SOAP messages can also be transported over other message transports. For example, we have done some experimentation using Java Message Service (JMS)-based message-oriented middleware (MOM). At present the web services we are using operate with SOAP over HTTP, but we could easily adopt JMS-based transport in the future if a need arises for some of the functionality provided by a MOM (e.g., publish/subscribe, reliability, persistence.).

The WS standards provide a fairly extensive set of standards for implementing web services using SOAP messaging. One important WS standard we should mention is Web Service Description Language (WSDL). WSDL is an XML-based language (defined in XML Schema Language) for defining web service interfaces. REST and SOAP each have their advantages and disadvantages, which are currently being actively debated. To date we have primarily used SOAP to build web services to interconnect simulation and analysis tools; however, we see no reason to preclude REST-style interfaces in the future.

3. Server Technology

With a web-service based SOA, a system providing services must function as a server, accepting HTTP connections and messages and providing responses. A consumer of web services does not need to function as a server, however in our environment many of our components are both providers as well as consumers of services. Therefore, we needed a way to provide access to our legacy components through web servers. In theory HTTP server logic could be integrated into an application from scratch, however in practice it makes much more sense to deploy the application into an off-the-shelf web server environment. For REST-style services, any web server would do, but for SOAP-based services a server capable of handling SOAP messages is necessary. For this, we are currently utilizing two server environments. One is the open source Apache Tomcat[10] server, with the Axis2 package[11] for SOAP processing. This is a popular open-source combination for developing web services, and it works well. However, we found even greater functionality and software development support was available in the Glassfish package [12]. Glassfish is an open source product, sponsored by Sun Microsystems, that provides a complete Java Enterprise Edition (EE) container with support for both REST-style and SOAP-based web services.

4. Registry Standards

In the area of registry, there are two major standards: Universal Description, Discovery and Integration (UDDI)[13] and Electronic Business using XML (ebXML).[14] Briefly, UDDI is a standard information model for registry information using web services, including a standard SOAP-based Application Program Interface (API) to access this information. ebXML is an extremely general purpose and extensible standard for storing and accessing information for electronic business, which has been proposed as a means to implement SOA registries. We experimented with both of these standards, and initially began building our registry using a UDDI-based product. However, we abandoned this when we found another solution with a more flexible user interface and a simpler API that we could more easily adapt to meet our specific needs. At present, our registry currently supports neither UDDI nor ebXML, but we do see a need to adopt a standards-based interface to our registry in the future, in order to be able to integrate with other products (e.g., enterprise service management products).

5. Enterprise Service Bus Standards

In the area of ESB standards, we see Java Business Integration (JBI) as a useful technology for interconnecting SOA service components within an ESB environment. Business Process Execution Language (BPEL) and Business Process Modeling Notation (BPMN) are standards used in service orchestration. While we have not done much as yet in the area of service orchestration, we expect to begin to apply standards such as BPEL and BPMN in the future to allow us to integrate components that model individual air traffic management functions into a larger model of NAS operations.

III. Performance Issues in Applying SOA Technologies to Distributed Simulation

The issue of performance impacts of SOA technologies is frequently raised and undoubtedly important in many simulation domains. This section of the paper addresses this issue in some depth.

As discussed above, use of XML is fundamental and pervasive in current SOA technology. Because XML emphasizes generality and human readability over compactness, there is a frequently expressed concern that XML is "verbose". That is, information represented in XML takes up more space in memory and on storage media, and more bandwidth to transmit across a network, than information represented using other standards.

In addition to the concerns about the size of XML-based messages, the addition of a web server layer is sometimes raised as having the potential to cause performance problems. The web service layer must perform SOAP and HTTP processing, as well as XML marshalling and unmarshalling. (XML marshalling is the process of transforming data from its internal representation in programming language data structures, to an internal representation of XML structure, and then to serialized XML text. Unmarshalling is the reverse process.)

Several different approaches are possible to assess the performance impacts and limits of a certain technology, including analytical methods, simulation and modeling, and measurement of candidate implementations. Analysis and simulation approaches are undoubtedly valuable, but we considered it necessary to begin by measuring the actual performance of some specific implementations under various loads and in various network environments. This provides baseline data which could be used as input to analytical methods or to calibrate simulation models. We conducted several experiments, two of which we cover in this paper in some depth. The first set of experiments examined request/response services using Apache Tomcat and Axis, and the second set of experiments examined one-way data publishing services using Glassfish. These experiments are described briefly below, followed by some analysis of the results.

A. Request/Response Performance Experiment

In the first experiments, we considered some representative request/response operations that will occur in the NAS, and that also occur in our simulations of the NAS. Two different types of information exchange were considered. One was a flight plan (FP) information service which provided information about active flights in the NAS, and the other was an airspace request (AR) service which would determine which en route centers were responsible for given airspace locations. As shown in Fig. 1, our experiment compared the performance of a "legacy" implementation in which the information exchange occurred in a compact text-based format directly over a TCP socket, versus an implementation in which a web service "front end" was placed in front of the legacy application to enable the information to be accessed using SOAP-based web services. The legacy flight plan service was programmed in C, and the legacy airspace service was programmed in Java. The web service front ends for both were programmed in Java, and implemented using Apache Tomcat and Axis. The service provider applications were run on a workstation-class Dell PC (single 3 GHz CPU, 2 GBytes RAM) and the consumer applications were run on similar hardware (2.2 to 3 GHz CPU, from 512Mbytes to 2 GBytes RAM). The experiments were run in a Local Area Network (LAN) environment. The clients were configured to send multiple requests, and we varied the number of items (flight plans or airspace lookups) that were included in each request. We gathered data on the total bytes transferred across the network (including network protocol overhead) and round-trip end-to-end application level latency, measured in the service consumer from just before initiation of the service request to when the results were received and available for processing.

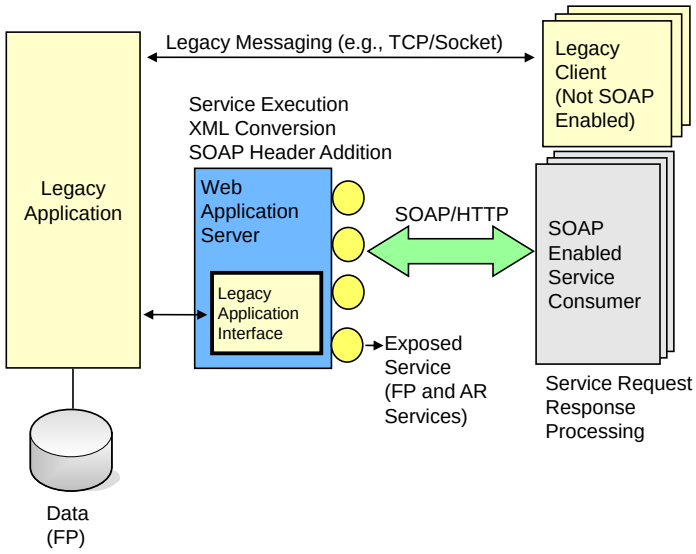

Figure 1. Request/Response Performance Experiment Configuration

The results of the request/response web service experiments are summarized in Table 1. As expected, switching to web services resulted in an order of magnitude increase in network traffic. Latency also increased, by as much as a factor of 15.

Table 1. Request/Response Experiment Results

Information Type	Measurement	Legacy Information Exchange	Web Service Information Exchange	Relative Performance
Flight Plan Information	Total Traffic on Network	91,923 bytes	628,809 bytes	WS = 7 x Legacy
	Mean Latency	2 ms	30 ms	WS = 15 x Legacy
Airspace Request	Total Traffic on Network	28,398 bytes	1,019,704	WS = 40 x Legacy
	Mean Latency	13 ms	47 ms	WS = 4 x Legacy

B. Publish/Subscribe Experiments

In the second set of experiments, we evaluated the use of web services to publish relatively small messages containing aircraft track updates. These experiments used a publish/subscribe message exchange pattern, in which

producers, when driven by events, push messages to one or more consumers. This type of information exchange represents what might be needed in the operational NAS to distribute a common surveillance picture from tracking systems to multiple users. In a distributed simulation environment, this type of information exchange represents dissemination of entity-state information among simulation federates. We were interested in the rate at which updates could be disseminated, and the latency in getting the updates from the producer to the consumer. Unlike the previous experiment, which was performed entirely in a LAN environment, in this case we were interested in evaluating performance in a Wide Area Network (WAN) environment.

The publish/subscribe experiment was run with a single producer and multiple subscriber configuration, and again with a single subscriber and multiple producers. In both cases, we gathered data on the one-way end-to-end latency, measured from when the data was available internally in the producer application, to when it had been marshalled, transmitted, unmarshalled, and made available internally in the consumer application. These measurements were made using the local clocks on the producer and consumer machines, which were synchronized using Network Time Protocol (NTP). We verified synchronization to within 1 ms, so all times reported in this section may be considered subject to a worst-case clock bias error of up to 1 ms.

Figure 2 shows an experiment configuration with one publisher and multiple (up to seven) subscribers. The publisher application generates track updates and publishes them to subscribers. We compared the performance of a Java web-service based implementation with a baseline implementation written in C. The baseline implementation simply wrote the track messages in a raw binary format over a TCP socket. Our initial web service publisher was built using an implementation of WS-Notification, which provides a standards-based mechanism for implementing a publish/subscribe message exchange pattern using web services. However, we chose not to maintain the WS-Notification-based service due to issues with the maturity of the open-source implementation and also concerns about an apparent lack of vendor support for the WS-Notification standard. Instead, for this performance experiment, our publish/subscribe web service used a simple, straightforward, event-driven publish/subscribe mechanism built into the application. This worked as follows: The publisher provides a "Subscribe" service which is invoked by subscribers, providing as an input the URL of the subscribers' "UpdateTrack" service. Whenever the publisher is ready to issue a track update, it simply invokes the "UpdateTrack" service at each of the subscriber addresses, providing the updated track information. Updates to consumers are sent in parallel, using a multi-threaded broker implemented by a Java class embedded within the producer application.

Both publisher and subscriber were implemented in Java and used Glassfish for the application server. Runs were conducted with different bandwidth networks separating the producer from the consumers, ranging from T1[‡] (1.544 MBits/second), through T3 (45 MBits/second), up to Gigabit Ethernet LAN connectivity. In addition, packet delays from 0 up to 40 milliseconds (one-way) were introduced to represent internetwork queuing and signal propagation delay.[§]

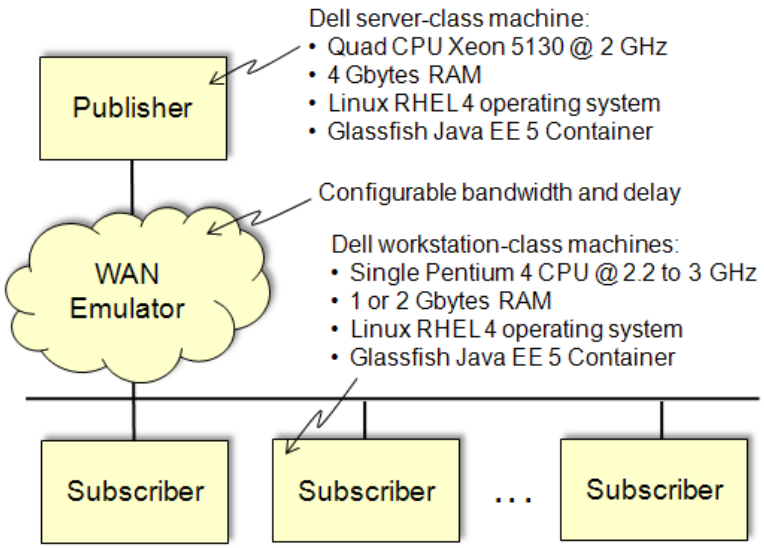

Figure 2. Single Publisher Multiple Subscriber Experiment Configuration

[‡] T1, T3, and OC3 designate commonly used communications link standards that operate at 1.544 Mbps, 45 Mbps, and 155 Mbps, respectively. Gigabit Ethernet is a LAN technology that operates at a raw transmission rate 10^9 bps.

[§] Time to move a packet across a network is the sum of the packet transmission time (packet size in bits times network bandwidth in bits per second) plus the packet propagation time (speed of the signal in the media in meters per second times distance in meters). In an internet, the packet transmission time is incurred at each router "hop," and queuing delays may also be incurred. We represent all of these effects with a WAN emulator that queues each

Table 2 compares the volume of data sent over the network for the baseline TCP socket implementation with the web service publish/subscribe implementation. The results are consistent with our earlier experiments: sending information in XML formats results in roughly an order-of-magnitude increase in the volume of data sent over the network.

Table 2. Track Update Message Sizes (in Bytes)

Implementation	Application Message Content Size	Average Message Size, Including SOAP Headers	Average Message Size, Including SOAP and HTTP Headers	Average Message Size, Including TCP/IP and all Higher Level Headers
Raw binary data over TCP socket	240 (binary)	N/A	N/A	292
SOAP-based web service	1,187 (XML)	1,393	1,681	1,790

Table 3 compares the end-to-end latency and number of updates per second achieved[**] under different conditions. Clearly, the baseline implementation is capable of transmitting huge volumes of data over the LAN with low latency. Interestingly, over a high latency T1 WAN the baseline implementation, while continuing to transmit a relatively high volume of data, created higher latency than even the worst case web service implementation. The explanation for this is relatively simple: the web service implementation runs over HTTP, which requires an "OK" response to each message before the next update is sent, whereas the baseline implementation is run directly over TCP with no application layer acknowledgement for each message, thus allowing multiple messages to be fed into the network transmission "pipeline," maximizing overall throughput at the expense of some additional latency on each message.

Table 3. Summary of Single Producer Multiple Subscriber Latency Results

Implementation	Network Bandwidth	WAN Latency, One Way (ms)	Number of Consumers	Mean End-to-End Application Latency (ms, plus or minus 1 ms)	Updates/Second (each update sent to all consumers)
Raw binary data over TCP Socket	LAN	N/A	1	< 1	166,886
	T3	5	1	12	15,993
	T1	40	1	369	668
Web Services-Based Publish-Subscribe	LAN	N/A	1	1	601
			7	1	447
	T3	5	1	6	77
			7	7	71
		10	1	11	43
			7	12	41
	T1	10	1	20	30
			7	44	12
		40	1	50	11
			7	59	9

packet for a fixed time to represent queuing delays and signal propagation delays, then delays it for a time proportional to packet size to represent transmission time on a constrained bandwidth network access link.

[**] These are the maximum message update rates that could be achieved without dropping messages or incurring unlimited message queue growth.

The second configuration for the publish/subscribe test is shown in Fig. 3. In this configuration the roles of publisher and subscriber are simply reversed from the previous configuration. The results are shown in Table 4 and Fig. 4 and 5.

Table 4 shows the end-to-end latency measured at the application layer as the number of producers is increased under LAN conditions, and also for one particular WAN configuration, selected to reflect "typical" conditions one might expect across a corporate WAN or multi-organization distributed simulation assembly.

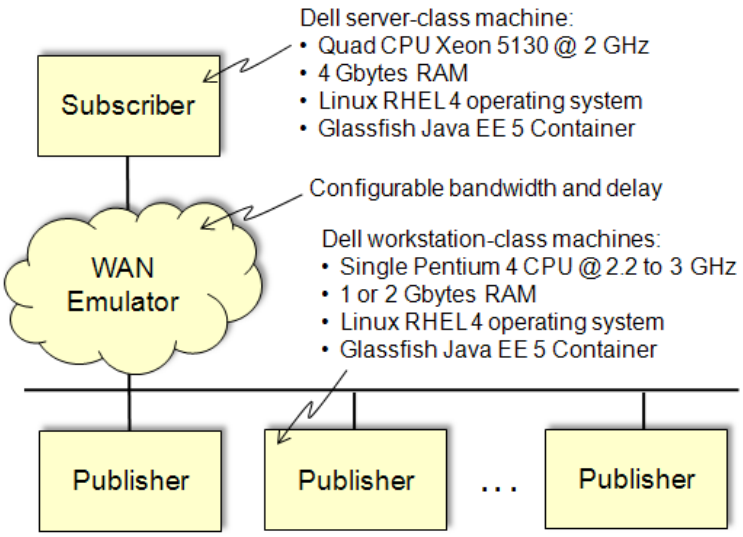

Figure 3. Configuration for Single Subscriber Multiple Publisher Experiment

Table 4. Single Subscriber Multiple Producer Latency Results

Network Bandwidth	Number of Producers	Mean Latency (ms, plus or minus 1 ms)	Standard Deviation of Latency (ms)	Updates Received/Second
LAN	1	0.8	5.3	1,019
	2	1.1	6.5	1,637
	3	1.1	6.9	2,365
	4	1.1	7.6	3,255
	5	1.1	6.3	3,951
	6	1.1	5.9	4,718
	7	1.2	8.8	4,804
	8	1.2	6.3	4,806
	9	1.4	78.5	4,895
	10	1.6	111.3	4,862
T3, 5 ms one-way delay	7	6.3	1.3	572

Figure 4 shows the total number of track updates per second received by the subscriber from all producers, as the number of producers is increased. We notice that the number of updates per second peaks at around 5,000, and does not increase as more producers are added, but at this point the latency of individual updates begins to increase. This raises the question: what is the limiting factor? At this point we are sending about 80 MBits/second across the network, which is a Gigabit Ethernet LAN over which we have observed throughput of up to 400 MBits/second, so network throughput does not appear to be the limiting factor. The most likely limiting factor is the server's ability to accept and process messages. To investigate this further, we examined processor utilization on the server, shown in Figure 5. Processor utilization peaks at only about 50 percent, at about the same point at which our message update rate reaches its maximum value. However, this is a 4 CPU server, so it is possible one or two of the CPUs is reaching 100 percent utilization and the web server application is unable to fully utilize all 4 CPUs simultaneously. With some tuning of the application to better distribute the load across the CPUs it might be possible to optimize performance and achieve significantly higher track update rates with the same underlying server hardware. Another possibility is that the server, or possible an intermediary device such as an Ethernet switch or our WAN emulator, is having difficulty keeping up due to the relatively small packet sizes, in which case throughput could be improved by bundling several track updates into each message, with some corresponding penalty in mean latency.

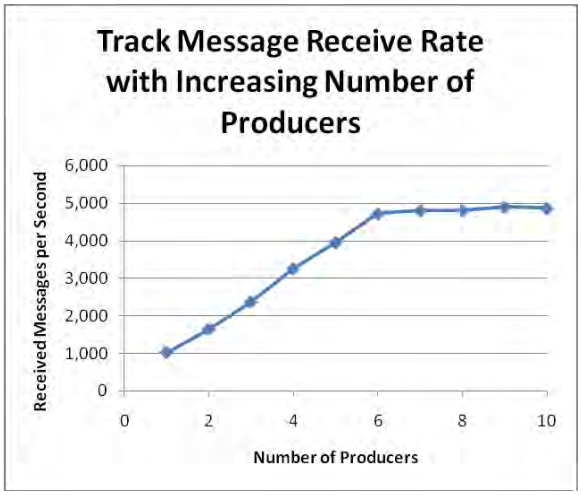

Figure 4. Update Rate in Single Subscriber Multiple Producer Experiment (LAN)

C. Analysis of Performance Experiment Results

In this section we attempt to draw some high-level conclusions about the suitability of SOA technologies to various types of distributed simulation, based on the experiment results presented above. Of course, care must be taken in interpreting the results of any specific test case, as performance results will be highly dependent on the specific hardware, software, and network infrastructure in use. Another important caveat is that we did not investigate techniques such as XML compression that might be used to accelerate performance. Despite these caveats, this set of data does allow us to draw some conclusions, which are reinforced by our qualitative experience obtained from our initial use of web services in distributed simulation, as described in Section IV.

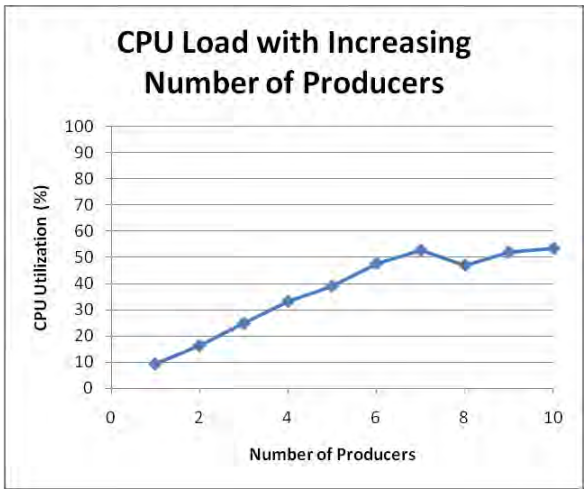

Figure 5. CPU Load vs. Number of Producers

The results of our experiments show that there clearly *is* a performance penalty incurred when exchanging information in XML using web services as compared with sending data in a "raw" format directly over a TCP socket. However, whether there is a performance penalty is not really the important question. What is important is *whether the performance achieved when exchanging information in XML using web services is suitable to meet our needs* for distributed simulation and analysis. The answer to this question is mixed. We believe SOA technologies (in particular XML messages exchanged over web services) will be suitable for our needs in many cases, but not all. These technologies are likely to work well when connecting components representing interacting loosely coupled systems in real-time simulations. These technologies may have more limited use in interconnecting components within fast-time simulations, though they are still likely to be valuable in loading data into such a simulation or providing results for visualization or analysis. Each of these cases is discussed further below.

1. Suitability of Web Services for Real-Time Human-in-the-Loop (HITL) Simulations

Based on the performance experiments described above, as well as our initial experience in utilizing web services to interconnect simulation components, we expect that this technology is likely to meet our performance needs for interconnecting components that are running real-time simulations of different NAS and NAS user subsystems (e.g., en route and terminal Air Traffic Control [ATC] systems, airline operational control facilities, aircraft flight management systems). An example might be a HITL simulation to evaluate how a NextGen concept or a mid-term NAS system improvement would impact air traffic controllers and flow managers. A specific example is described in Reference 12. For this type of simulation, we expect in general to be able to achieve acceptable performance using web services among simulation components at different locations on our corporate network, including components located at different large corporate sites that have high-speed connectivity (T3, OC3, and up), and possibly also including remote sites with lower speed connectivity (e.g., T1 or less), depending on the specifics of the information to be exchanged. For connecting simulation components across the Internet or virtual private networks for multi-organizational simulations, the acceptability of using web services would depend on exactly what services were used, and at what rate. In constrained bandwidth networks, it may be possible to reduce the performance impact of web services by applying compression techniques to reduce the volume of XML-based messages.

To generalize this somewhat, consider the performance requirements for distributed simulations found in the DIS[1] standard, reproduced in Table 5. Without going into detail on exactly how DIS defines these parameters, we can see that, given today's network and processor capabilities, we can easily meet or exceed these performance characteristics using XML information exchanges over web services. The DIS standard was targeted primarily at medium-fidelity HITL training scenarios involving multiple simulation entities connected over a WAN. Therefore, we may conclude that using XML-based information exchanges via web services is likely to provide suitable performance for this type of simulation application.

Table 5. DIS Standard Requirements on Network Performance

Parameter	Default values
Transport-to-transport latency	100 ms – Tightly coupled[††] interactions 300 ms – Loosely coupled interactions
Transport-to-transport jitter	50 ms
Transport-to-physical acceptable latency	10 ms

2. Web Services for Fast-Time Simulations

Within our laboratories we also run fast-time simulations, which are not tied to real time, but may require a very large number of runs to be repeated in order to produce an answer quickly enough to be useful. For example, Zeitlin[16] describes a fast-time simulation used to evaluate potential sensitivities and shortcomings of the Traffic Alert and Collision Avoidance System (TCAS) for use in unmanned aircraft. Another example is provided in Reference 17.

We also use fast-time simulations to run Monte Carlo analyses within decision support tools, for example as discussed by Wanke in Reference 18. Again, potentially huge numbers of simulation runs must be performed quickly enough to be useful within the user's decision-making process.

It is difficult to make concrete statements regarding whether web-service based information exchanges are likely to provide suitable performance for use in fast-time simulations; however, in general fast-time simulations tend to be run within high-performance platforms (or clusters) using shared memory for information exchanges, rather than running on systems distributed across a network and communicating using internet protocols. For such an environment, web services are not likely to be applicable for exchanging entity state information among simulation components, and may introduce unacceptable performance penalty if used in this way. However, web services may still be valuable in fast-time simulations in other ways, for example they might be used to obtain data (e.g., weather data, airspace locations, filed flight plans) prior to commencing multiple runs for a Monte Carlo analysis, or they might be used to request a set of runs and provide the results after the runs have completed.

[††] DIS uses the terms "loosely coupled" and "tightly coupled" to refer to the degree of synchronicity required between components.

IV. CAASD SOA Framework Proof of Concept

This section describes an initial use of SOA technologies for simulation and analysis, which we refer to as the CAASD SOA Framework proof of concept. In this proof of concept, we have implemented services for access to static data archives, dynamic simulation-dependent data, and algorithms. Different types of legacy simulation capabilities with varying interfaces for simulation control and data access were "web service enabled," allowing simulation components to obtain and produce data via web services, as well as allowing simulation components to be controlled via web services. We implemented a registry to allow service discovery, including the ability for simulations to dynamically register services made available by the running simulation, and for service consumers to obtain data within a specific simulation context. We are prototyping tools to allow simulation components to be combined in a "plug-and-play" manner without programming. These tools are intended to allow an analyst to select which simulation assets to include in a simulation, and then launch and control the execution of a simulation. We have designed and implemented a security solution, allowing us to control access to services when needed. We are also exploring the challenges of creating an enterprise SOA, including issues such as enterprise service management and governance. These aspects of the CAASD SOA Framework are discussed in the remainder of this section.

A. Web Services for Access to Simulation Data and Algorithms

One of the most basic aspects of a SOA transition is the use of web services to allow different systems to access data and functionality available from other systems. The most significant services that we have implemented and are currently using are described below, followed by some reflections on our experience in using web services.

1. Simulation-Independent Services

First, we consider services that provide data independent of whether a particular simulation is running. We refer to these as *simulation-independent* or *static* services, since they are intended to be always available. This class of services might include access to data archives or algorithmic functionality. Web services in this category include:

a) *Adaptation data service*. This aeronautical information service provides Air Route Traffic Control Center (ARTCC) and sector boundaries for all 20 ARTCCs in the NAS.
b) *Weather data service*. This gives current and forecast precipitation and echo tops information, as they existed in the NAS at a specified time. This service is based on a local archive of Corridor Integrated Weather System (CIWS)[19] data, which was in turn obtained from web services provided over the internet by MIT Lincoln Laboratory. Archived data is available from March 2007 until the present.
c) *Route blockage service*. Indicates which NAS air routes were predicted to be blocked by convective weather at a specific time, according to MIT Lincoln Laboratory's Route Availability Planning Tool (RAPT [20]).
d) *Trajectory computation service*. Given a flight plan, compute the trajectory along which the aircraft will travel. This is a web-service front end to an existing trajectory modeling library.

2. Simulation context dependent services

Another category of services are those that provide data from within a particular simulation run. We refer to these as "simulation-context-dependent" or "dynamic" services, as they can only provide data when a simulation is running. Also note that, if multiple simulations are running, the consumer must specify the simulation context within which the service request applies. (Our approach to dealing with simulation context is discussed in more detail below.) Web services that we have implemented and are currently using include:

a) *Flight data services*. Provide access to flight data from a running Traffic Flow Management (TFM) Core simulation. (The "TFM Core" is our research platform for modeling TFM algorithms and operations.) Flight data includes complete information about an aircraft flight, including items such as aircraft type, route, departure time and status, trajectory, current position and altitude, assigned altitude, and so on.
b) *Track data services*. Provide access to track data from a running TFM Core simulation. Track data includes information on the location, altitude, heading, and speed of an aircraft being tracked in the NAS, at a particular time. (Earlier versions of this service used a WS-Notification-based publish/subscribe message exchange pattern; the current version uses a request/response pattern.)
c) *Sector Count service*. Provides access to sector counts, monitor alert parameters (MAPs), and weather reduced capacity based on CIWS data and a sector capacity coverage calculation algorithm.
d) *Aviation SimNet Collaborator service*. Provides operations that allow track data and status information to be retrieved from an executing AviationSimNet federation.

The first three of the services listed above provide access to data from specific simulations that run within CAASD's legacy simulation environment, while the latter provides access to data from any simulation running in an AviationSimNet environment.

3. *Software Development Perspective on Web Services*

Our experience in developing software using web services has been very positive overall. There is a significant initial learning curve that must be overcome for developers new to writing code for application servers and producing and consuming web services. However, tool support is advancing rapidly, which will help reduce this barrier to entry for developers. (During the course of this project, new versions of Integrated Development Environments [IDEs] have been released which significantly streamline the development process. We use both Netbeans and Eclipse IDEs.)

The development process in a web services-based SOA environment is illustrated in Fig. 6. Generally, once the need for a new service has been established, the software development process begins generation of the service provider. Services may be developed either "contract-first" or "code-first". In the contract-first approach, the service interface is first defined in a WSDL file. Software tools are then used to generate application code that conforms to the interface. In the code-first approach, the developer creates data structures and interfaces in the programming language of choice (by far the most tool support is provided for Java, but other languages are also supported) and then applies tools to generate a WSDL based on the code. In either case, the tools create all the necessary "stubs" for marshalling and unmarshalling data from native program language structures to and from XML. The automatic generation of interface code is a *huge* advantage, as it eliminates tedious and error prone programming required to add or change application interfaces, thereby significantly increasing the speed at which software can be created and modified for new purposes. Once the service provider has been created, the next step in the process is the generation of service consumers. At this point, the WSDL for the service exists, so developers can import the WSDL and apply tools to generate interface code for consumer applications. Again, the code to translate from internal data structures to XML is produced automatically by the tools, producing the same advantage. Finally, at run time, consumer applications bind to the producer applications over the network and invoke the services using the generated interface code.

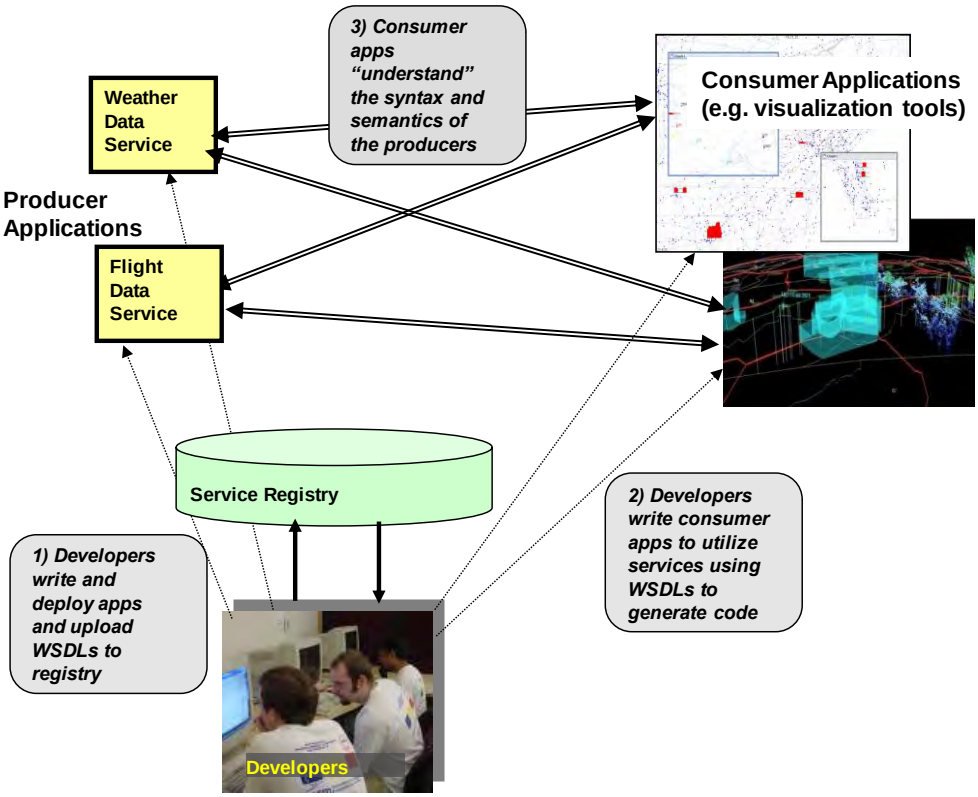

Figure 6. Web Service Development Process

B. Simulation Control

We are also applying web services in the area of simulation control. We defined a *Manage Asset* service for this purpose. The interface contains the following methods:

a.	*Launch*	Execute a simulation application, creating a new simulation run.
b.	*Initialize*	Perform any necessary loading of data or setup prior to commencing the simulation run.
c.	*Go*	Commence the simulation (starts the simulation clock).
d.	*Pause*	Pause the simulation run.
e.	*Resume*	Resume a paused simulation run.
f.	*GetStatus*	Returns information about the current state of the simulation run.
g.	*SetClockRate*	Allows the simulation to be run faster or slower than real time.
h.	*Terminate*	Halts the simulation run and releases all resources.

To date, we have implemented one Asset Manager that provides the *Manage Asset* service interface. This Asset Manager is a web service façade in front of our Central Simulation Manager (CSM). CSM is a legacy simulation management capability which is supported by many different types of CAASD simulation assets. Our Asset Manager enables web services-based simulation control for all of these simulation assets. In the future, additional Asset Managers may be built for additional types of simulation, enabling a single graphical user interface (GUI) to control and combine different types of simulation asset.

To utilize the Asset Manger, we have created a GUI tool called the SOA Simulation Manager (*soa*SimManager). Figure 7 illustrates the use of this tool. The user, having logged in using the MITRE corporate single-sign-on solution, has selected a simulation "mashup" that they wish to run (in this case the TFMCoreDemo mashup), and invoked "Create New" to create a new simulation run. The run is assigned a unique "Sim ID" that will be used to identify the simulation context when accessing web services that are provided by the simulation as it runs. (See subsection D, for a discussion of simulation context.) The user is able to enter parameters such as the real time that the simulation should start, the initial simulation clock value and clock rate, and a description of the run. Other screens (not shown) give the user the ability to select from running simulations and pause, resume, and terminate the simulation runs, subject to access control rules based on the user's group membership and permissions.

Readers may be interested to know that *soa*SimManager GUI is implemented using AJAX[‡‡] technology. The application runs in a browser, making use of the web services provided by the Asset Manager as well as web services provided by our extended registry, which is described in the next section.

Of course, many other configuration parameters are needed to configure a simulation run beyond those shown in the screen capture in Fig. 7. This setup is done in advance, using existing tools and procedures, and the resulting configuration information is referred to as an "asset configuration." Each of the simulation mashups shown in the list in the lower left of Fig. 7 includes one or more of these asset configurations. When the simulation run is started, each of the simulation assets in the selected mashup will be launched, moved in synchronization through any needed initialization stages, and then given the "Go" signal to commence the run at the same time. In this way, our simulation control concept supports combining different simulations (possibly of different types) as well as other assets such as visualization tools, into a larger simulation framework that runs as a whole.

[‡‡] Ajax (asynchronous JavaScript and XML), or AJAX, is a group of interrelated web development techniques used for creating interactive web applications or rich Internet applications.

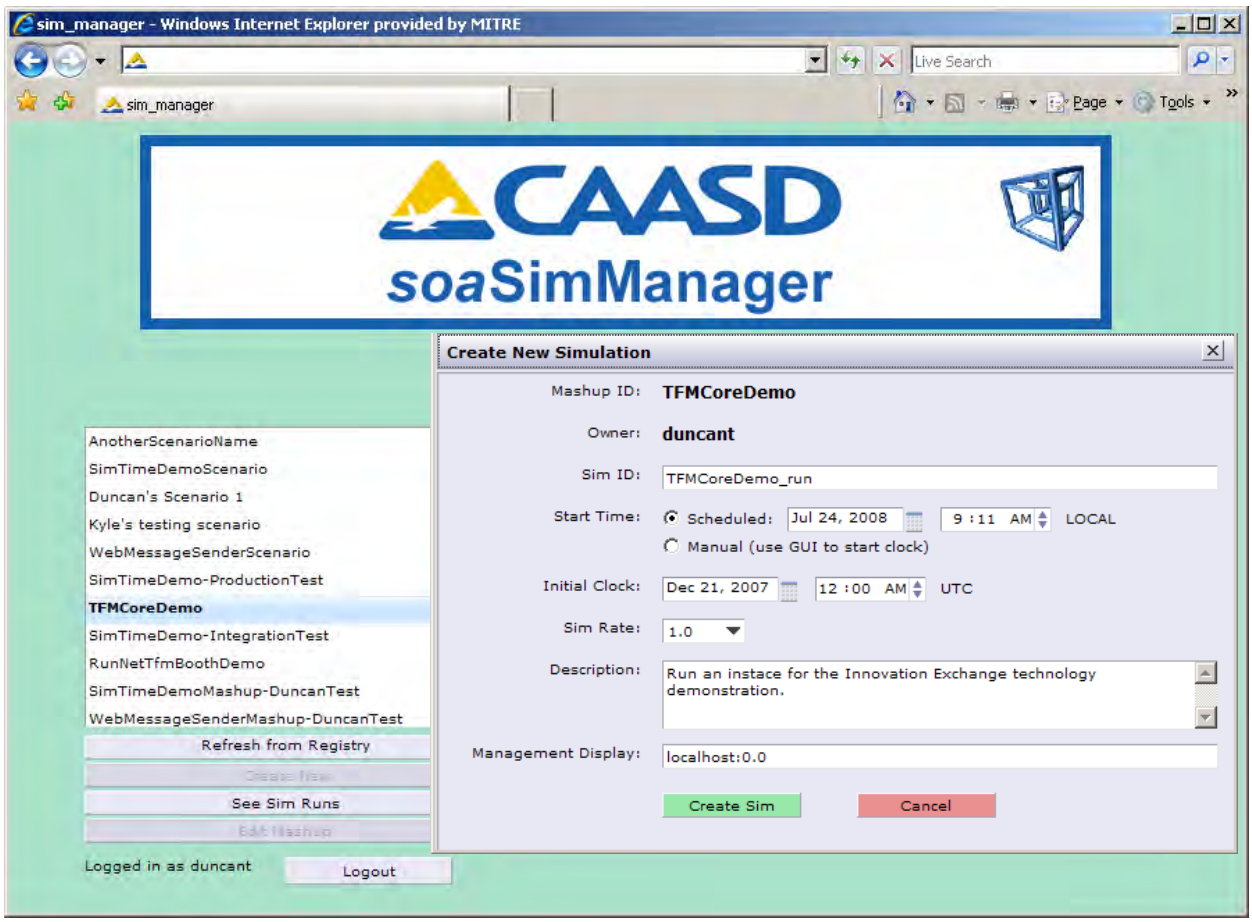

Figure 7. Simulation Control GUI Screen Capture

C. Registry and Service Discovery

A registry is an essential component of any SOA. The registry provides a graphical user interface that allows developers to enter and discover services. Also, through a web service API, the registry allows services to be located at run-time. The user interface and the API of our initial registry prototype are described below.

1. Registry User Interface

Figure 8 illustrates the user interface to the registry, which is accessed using a web browser. User authentication is provided using MITRE's corporate single-sign-on solution. Information about services is provided at three levels. At the highest level, as shown in Figure 8, is a high-level list of services. For each service, the registry stores general information about the service, as well as information about different implementations of the service. Different implementations may represent different versions of the software that provides the service, possibly with different service interfaces. Finally, for each implementation, the registry stores information about deployments of the service. A deployment defines a specific location on the network at which the service may be accessed.

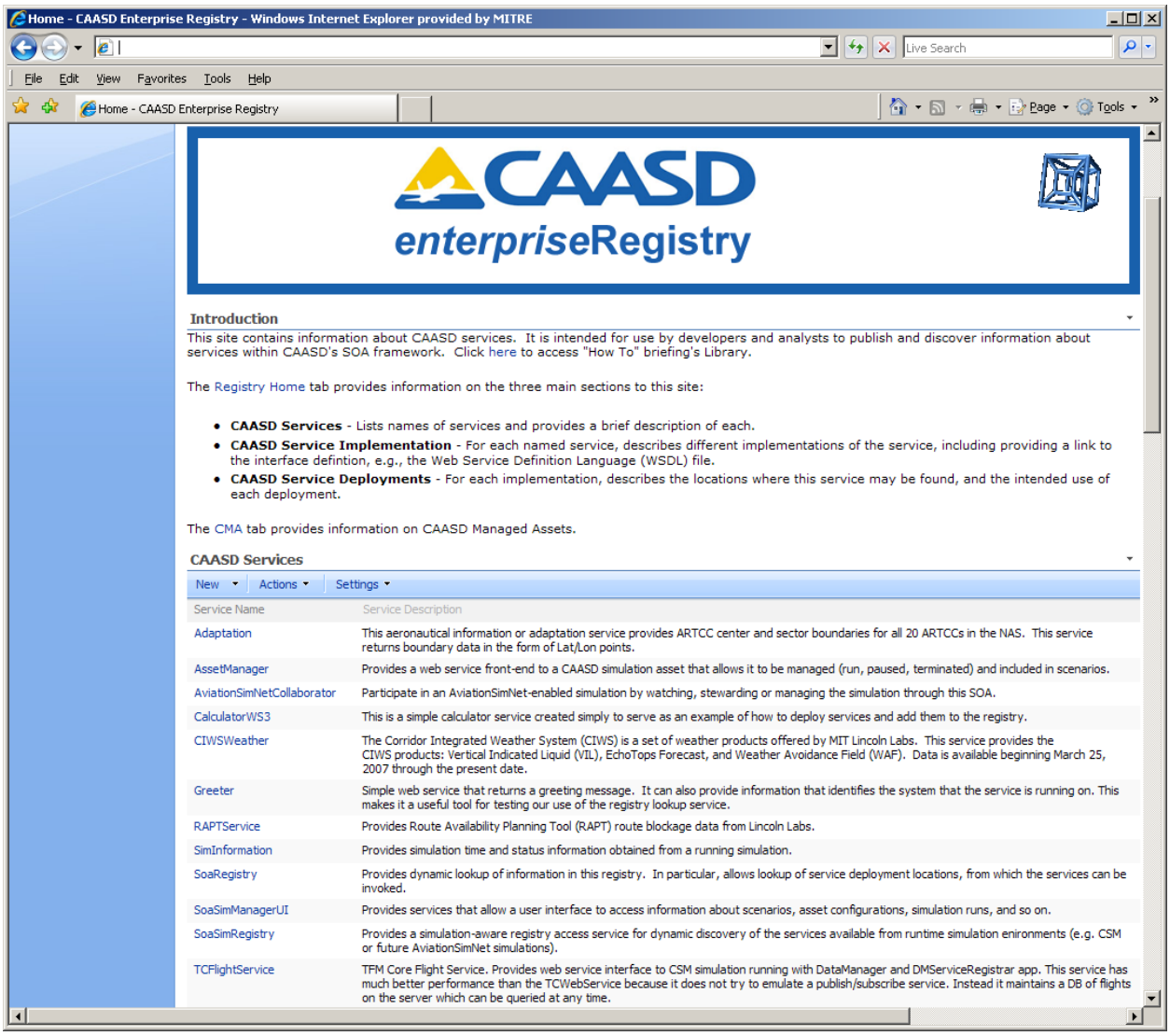

Figure 8. CAASD Enterprise Registry – Top Level Interface

To see how the information in the registry might be used, consider the following use case. Imagine that a user wishing to access a service has client software that is built to interoperate with release 2.1 or higher of the service. The user would access the registry to determine which servers have this version of the service deployed. He or she could then examine the information about these deployments to determine which one to use. Continuing the example, assume that the implementation labeled V3.0-Beta of the service is deployed on the server testbox.mitre.org, for the purposes of testing. Assume further that V2.1-Stable is deployed on prod.mitre.org, intended for general production use, and that V2.1-Stable is also deployed on hitlserver.mitre.org, which is currently being used as part of a HITL experiment to allow controller teams to evaluate a new Air Traffic management(ATM) concept. The user's client could work with any of these deployments, but probably only one choice is appropriate. Based on the information in the registry, the user would know whether to configure their client software to access the service on testbox.mitre.org, prod.mitre.org, or hitlserver.mitre.org, depending on what project the user is participating in.

Figures 9, 10, and 11 are screen captures that provide more details of the information stored in the registry at this time at the service, implementation, and deployment levels. (The registry structure and content is evolving as we develop our concepts of use and governance model.) Figure 9 shows some of the information stored for the *TFM Core Sector Count* service. As can be seen, a high level description of each service is provided, along with links to documentation, and so on.

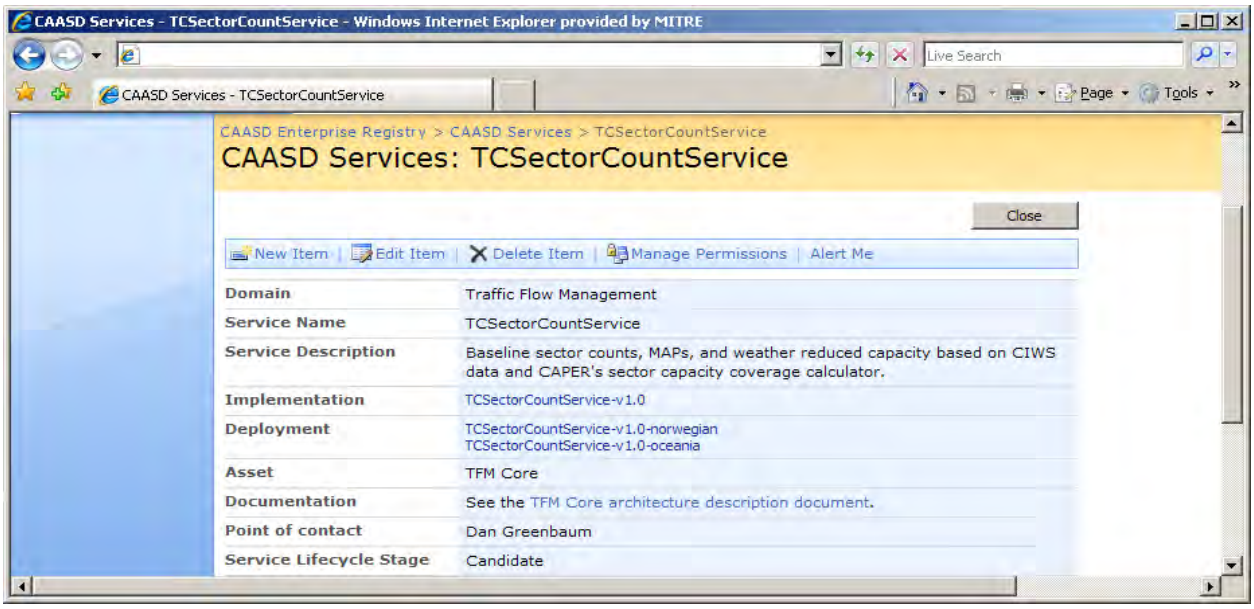

Figure 9. High Level Service Information

Figure 10 shows some of the information stored for a particular implementation of this service. Note that the WSDL file, which defines the details of the interface to the service, is associated with the implementation, allowing different versions of the service to exist simultaneously It is also worth noting that a service may be defined for which no implementations exist, as might be the case when a service is first proposed.

Figure 10. Service Implementation Information

Figure 11 shows some of the information stored for a particular deployment of the service. This entry shows that the service can be accessed on the server oceania, and that this deployment is for experimental use, supporting a particular prototyping effort. Note that a given implementation may be deployed in multiple locations.

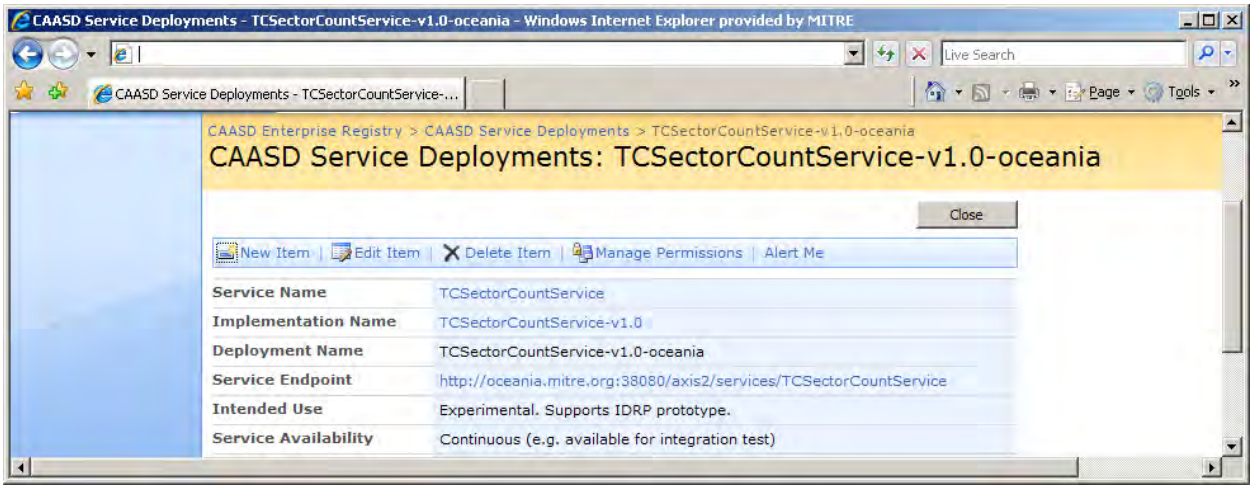

Figure 11. Service Deployment Information

Facilities are provided in the GUI for sorting and searching this list in a variety of ways. For example, as shown in Figure 12, services can be viewed according to domains that are defined in the CAASD Managed Assets (CMA) database, which stores information about all types of CAASD software and data assets (not just web services).

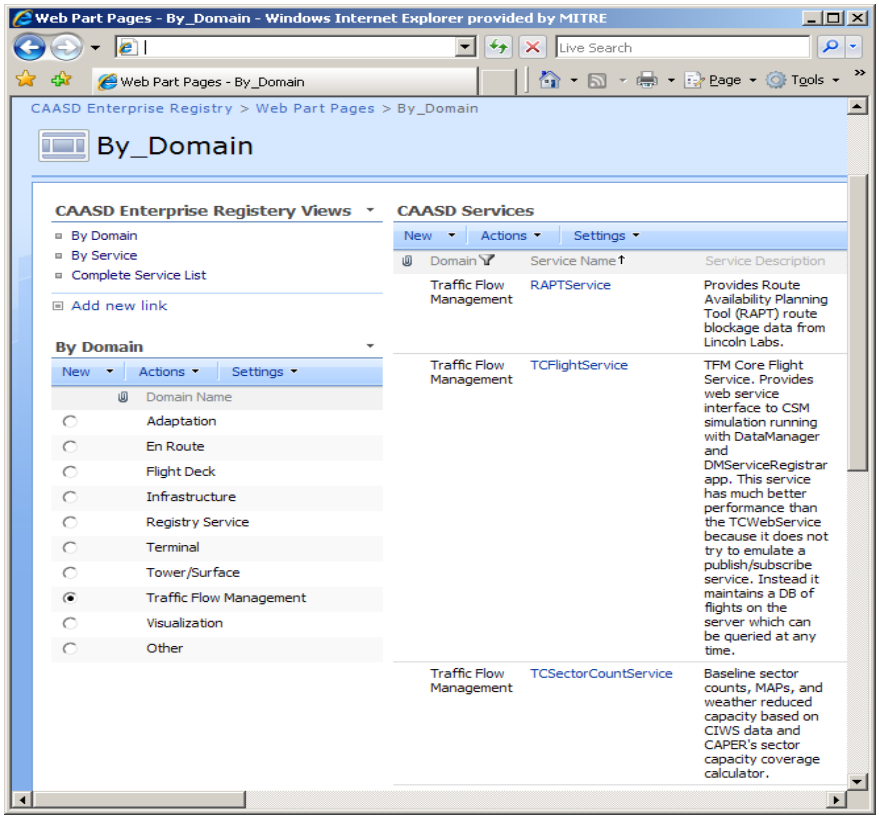

Figure 12. Example of Different Views into Registry

2. Registry Application Programming Interface

In addition to providing a graphical user interface to the registry, we also implemented a set of services that are used to access information in the service registry at run time. Our basic registry API provides a set of simple web services that allow consumers to locate services by service name at run time, returning the URLs and version numbers of the deployments of the service. Use of this service prevents "hard coding" service locations into applications, allowing services to be relocated without breaking service consumers. Additional registry services allow for dynamic registration and discovery of services based on simulation context, discussed below.

Readers familiar with the Microsoft Sharepoint may recognize that we have used this product to implement our prototype registry. Our run-time registry API is a façade in front of Sharepoint's web service API.

At present, our registry does not support UDDI access. We did experiment with a commercial off-the-shelf (COTS) registry product that supports UDDI, but chose to proceed with the implementation described here due to its superior flexibility in customizing views, information stored, and access control capabilities. (As far as we know, this is a unique application of the Sharepoint product.) Especially in the initial stages, as we develop our use cases, this flexibility outweighs the value of a standards-based interface. However, lack of a standards-based interface does limit our ability to connect other COTS products in the future. We are currently investigating a new architecture for our registry solution. Our plan is to keep the existing solution for user access, while incorporating a UDDI-based solution to allow tool access to the registry information using a standard interface.

D. Simulation Context-Aware Service Discovery

In a simulation environment a given service may only be available when a simulation is running, and if multiple simulations are running that can provide the service, some means must be provided to allow a service consumer to specify the appropriate simulation context. Figure 13 illustrates the concepts we have implemented to address these issues. We extended the basic registry API described above with another layer, backed by an Oracle database, and accessed via web services, to implement our "simulation-aware registry" concept.

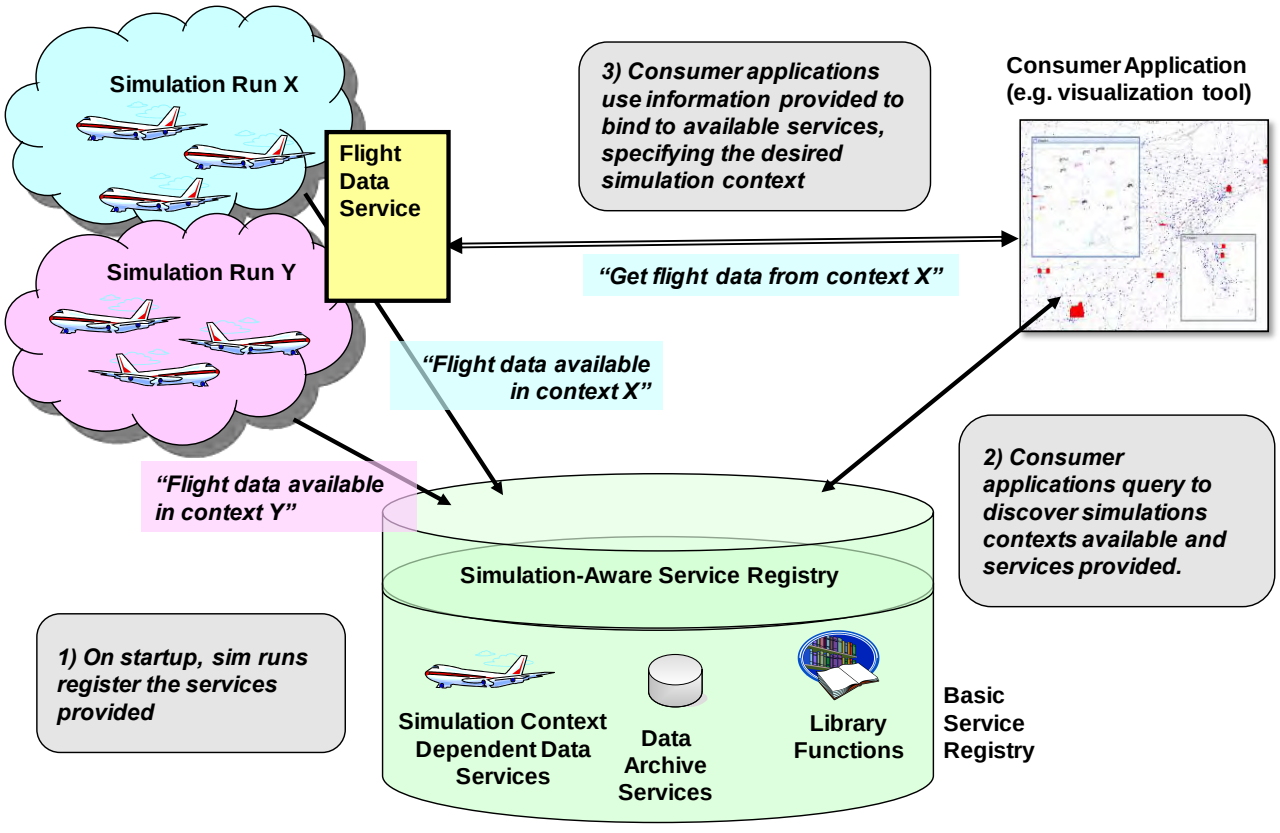

Figure 13. Simulation Context Concepts

As shown in Figure 13, when simulation runs are instantiated the services they provide, and the location of these services, are registered along with a unique identifier of the simulation context. Registration of a simulation run with the simulation-aware registry can occur in two ways. Simulations are launched with the *soa*SimManager tool described above will be registered by the tool. Simulations can also be launched manually or with other tools that are not "registry enabled." In this case, the simulations can use the registry services API to register themselves and list the services they provide. Other tools, for example visualizations, data recording tools, analysis tools, can query the registry to find out what simulations are running and what data services are available from those simulations.

Figure 14 illustrates the simulation context concept in use. The figure shows a simple GUI application (built using the OpenMap[21] package) that is capable of displaying track information from multiple sources. The application has opened a window (in the upper left) which displays the results of querying the simulation-aware registry in a tree form. In this instance, several simulations are running, and the user has expanded the simulation run called "TFMCoreDemo" based on descriptive information about this simulation (not shown on this screen) and has selected the "TCWebService" available from this simulation as a source of data to display. Track data is obtained from this service and shown on the map display. The window in the lower left allows the user to control the rate at which the service is polled and the rate at which tracks are updated on the display. As can be seen in the window to the upper left, at the time this screen capture was made, several other simulations were running, including multiple AviationSimNet federations. Data from these other simulation runs could be brought into the track display by simply selecting the desired simulation run and connecting to the desired track service.

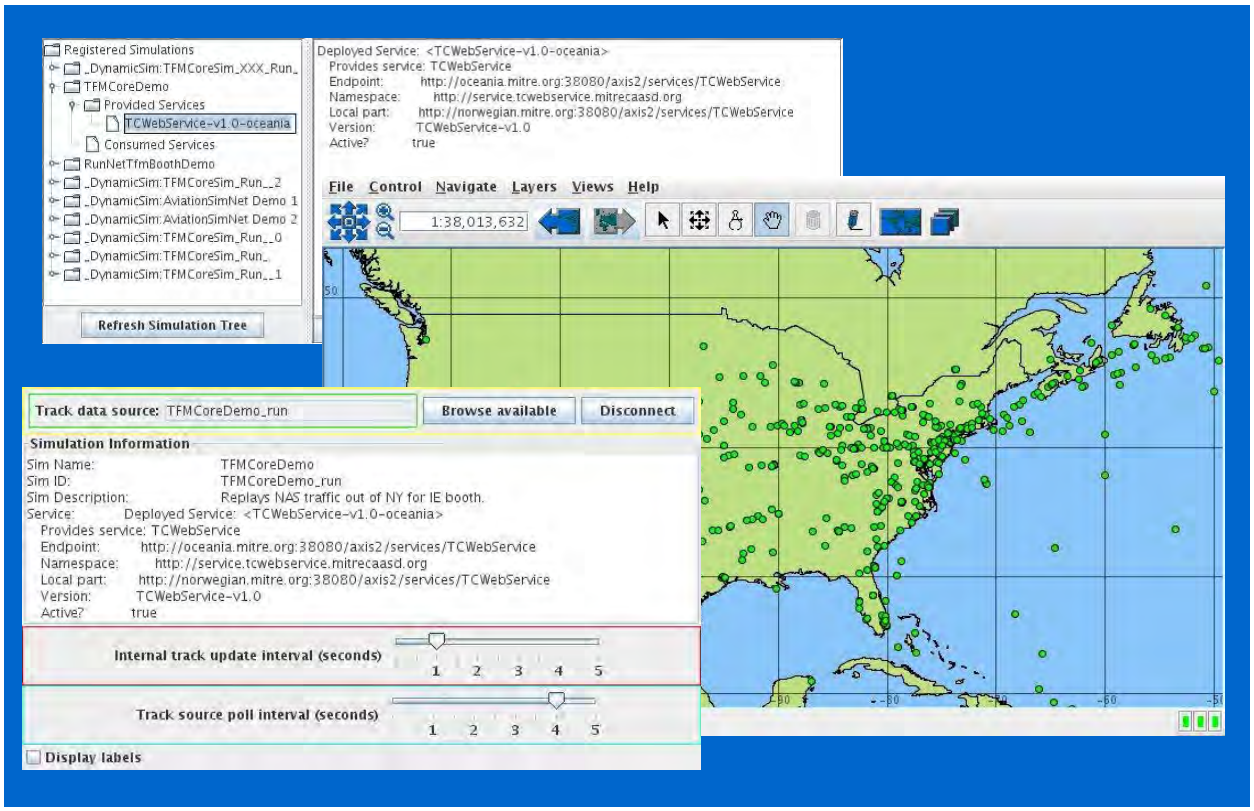

Figure 14. Simulation Context Concepts in Use

E. Enterprise SOA Challenges

In order to transition from prototyping and proof of concept to more general use within CAASD, several challenges remain. Services must be "hardened" and made available for production use, and enterprise service management must be addressed so that service availability and performance can be monitored and maintained. In the area of security, while we have implemented an access control mechanism for our simulation control services, we still need to develop policies and procedures to determine what services need access control, and how we will determine which users will be allowed to access which services. In addition, additional security mechanisms and

approvals will be required if and when we begin to expose any of our services outside of our corporate network. Processes and procedures for adding service information to the registry, and ensuring that the information is kept current, also need to be developed. These and other issues make up SOA governance, which we are now beginning to address within CAASD and across MITRE.

Also, in order to make full use of web services within our enterprise, it may be valuable to introduce and utilize ESB technology. CAASD has ongoing research in this area; results may be published next year.

V. Conclusion and Lessons Learned

Through the performance experiments we conducted, and our prototyping work in creating a proof-of-concept CAASD SOA Framework, we have gained considerable experience in applying SOA principles and web services technologies in our simulation environment. Performance of the new technologies remains an important issue to be considered, but in most cases we do not think performance concerns preclude use of these technologies for real-time, HITL simulations of the systems that make up the NAS. From a software developer's perspective, the tooling support for XML messaging and web services offer a significant advantage in creating new interfaces. From the perspective of analysts and simulation users, we are beginning to see the potential benefits of the shift in paradigm from sharing code and integrating system assemblies as needed, to a service-oriented paradigm in which service providers take responsibility for keeping capabilities available with some established quality of service, and service capabilities can be composed in a "plug and play" manner by the analyst. We are in the process of hardening and refining our software in order to transition this work to general use within CAASD. Significant challenges remain in the area of establishing governance and making organizational adjustments to the SOA approach; nevertheless, we believe we have demonstrated a framework that has clear potential to improve our flexibility and agility in applying and evolving our simulation and analysis assets to meet the needs of the FAA and our other sponsors.

Acknowledgments

The work documented in this paper was a team effort involving quite a few individuals. In particular the author would like to acknowledge Scott Kell and Craig Wanke for their inspiration and leadership. Frank Sogandares helped in concepts development. David Callner, Dan Greenbaum, Eugene Mwendwa, and Aaron Weikle did most of the development work to create web service interfaces to produce simulation data and consume it in visualization tools. Ezra Jalleta and David Callner helped create, run, and analyze data from the performance experiments. Jojo Thoppil and David Bodoh helped design and implement the *soa*SimManager. Kyle Jaranson helped implement the Asset Manager. Amal Srivastava took the lead in designing and implementing our access control solution. Emily Beaton contributed to our initial registry concepts. Finally, and most importantly, all members of the team contributed creative energy and ideas, without which this paper would not have been possible.

Notice

References

[1]Standards Committee on Interactive Simulation (SCIS) of the IEEE Computer Society, *IEEE Standard for Distributed Interactive Simulation Communication Services and Profiles*, April 1996

[2]Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society, *IEEE standard for modeling and simulation (M&S) high level architecture (HLA) - framework and rules*, *IEEE Std 1516-2000* , September 2000

[3]AviationSimNet Standards Working Group, *AviationSimNet Specification, Version 2.0*, Published by The MITRE Corporation, MP 06W0000131, August 2006

[4]Sogandares, F.M., "Stone axes and warhammers: a decade of distributed simulation in aviation research," *Proceedings of 16th Workshop on Parallel and Distributed Simulation, 2002*, pp.114-121, 2002

[5]W3C Recommendation, *Extensible Markup Language (XML) 1.0 (Fourth Edition)*, URL: http://www.w3.org/TR/2006/REC-xml-20060816, August 2006

[6]W3C Recommendation, *XML Schema Part 0: Primer*, URL: http://www.w3.org/TR/2001/REC-xmlschema-0-20010502, May 2001

[6]W3C Recommendation, *XSL Transformations (XSLT), Version 1.0*, URL: http://www.w3.org/TR/1999/REC-xslt-19991116, November 1999

[7]W3C Note, *Simple Object Access Protocol (SOAP) 1.1*, URL: http://www.w3.org/TR/2000/NOTE-SOAP-20000508, May 2000

[8]Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", ch 5, Ph D. Dissertation, University of California, Irvine, CA, 2000

[9]W3C Recommendation, *SOAP Version 1.2*, URL: http://www.w3.org/TR/soap, April 2007

[10]Tomcat, The Apache Tomcat Servlet/Java Server Pages Container, Software Package, Ver .6.0, The Apache Software Foundation, URL: http://tomcat.apache.org/

[11]Axis2, The Apache Axis2 Web Services Engine, Software Package, Version 1.4, The Apache Software Foundation, URL: http://ws.apache.org/axis2

[12]Glassfish, The Glassfish Open Source Java EE Application Server, Software Package, Version v2ur2, The Glassfish Project, URL: https://glassfish.dev.java.net

[13]OASIS UDDI Spec Technical Committee, UDDI Version 3.0.2, URL: http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm, October 2004

[14]OASIS/ebXML Registry Technical Committee, *OASIS/ebXML Registry Services Specification v2.0*, URL: http://www.ebxml.org, April 2002

[15]Smith, E.C., "Assessment of controller situation awareness in future terminal RNAV operations," *Digital Avionics Systems Conference, 2007. DASC '07*, December 2007, pp 6.B.3-1 - 6.B.3-13

[16] Zeitlin, A.D. and Mclaughlin, M.P., "Safety of Cooperative Collision Avoidance for Unmanned Aircraft," *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA* , pp.1-7, October 2006

[17]Callantine, T.J. and Palmer, E.A., "Fast-time simulation studies of terminal-area spacing and merging concepts," *The 22nd Digital Avionics Systems Conference, 2003. DASC '03.*, Oct. 2003, vol.1, pp. 5.B.3-51-11

[18]Wanke, C., Mulgund, S., Greenbaum, D., Song, L. "Modeling Traffic Prediction Uncertainty for Traffic Management Decision Support," *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Providence, Rhode Island, August 2004

[19]Campbell, S., Grappel R., Hoffman, M., Hogaboom, R., Lloyd, R., and O'Rourke, J., M.I.T. Lincoln Laboratory, "Corridor Integrated Weather System (CIWS) Cockpit Weather Display Data Link Demonstration", *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Monterey, California, August 2002

[20]Robinson, M., DeLaura, R. A., and Evans, J. E., "Operational Usage of the Route Availability Planning Tool During the 2007 Convective Weather Season", *13th Conference on Aviation, Range, and Aerospace Meteorology (ARAM)*, New Orleans, LA, 2008.

[21]OpenMap, Open Systems Mapping Technology, Software Package, Ver. 4.6.4, BBN Technologies, URL: http://openmap.bbn.com