Building Net-Centric Military Applications over Service Oriented Architectures

Ken Birman[*], Robert Hillman[+], Stefan Pleisch[*]

[*] Department of Computer Science, Cornell University, NY
[+] IFSE, Air Force Research Lab, Rome, NY

ABSTRACT

We compare the overall structure of military GIG and NCES architectures with that of the object oriented architectures (CORBA, J2EE and .NET) and of the emerging Web Services architecture. While the match is good in many ways, particularly with respect to Web Services, we also identify a series of shortcomings that could stymie attempts to implement a GIG or NCES system directly on a commercial Web Services platform. Our comparison leads to suggestions for experimental investigations of some topics, but also for more fundamental inquiry in some areas where the scientific base is inadequate. Several issues of the latter sort arise when we consider the mixture of scalability, security, robustness, and time-criticality that must be simultaneously satisfied in demanding military applications.

Keywords: Global information grid, net-centric enterprise systems, web services

1. Introduction

The past decade has seen steadily increased interest in development and deployment of networked information systems for military applications. Contemporary examples range from networked access to military databases and intelligence systems to tactical support for the warfighter in the theatre of operations; looking to the near future, one would add networked support for autonomous vehicles and sensor systems to this list. The terms “Global Information Grid” and “Net-Centric Enterprise Systems” (GIG and NCES) have been used in reference to the emerging standards that will govern systems and applications in this space.

Each of the services has proposed multiple platform standards in the GIG/NCES space. Our research stems from study of the Air Force Joint Battlespace Infosphere (JBI) architecture. The JBI standardizes interoperation between legacy Air Force information sources and a new generation of client systems. It exploits the publish-subscribe communication paradigm, but integrates that functionality with an information repository that can be queried by clients, a “fuselet” mechanism by which related events can be captured and fused to create new high-level events, a sophisticated information ontology and information management architecture, and a comprehensive security model.

During the same period, an analogous dialog has been underway in the commercial sector. This yielded first the object-oriented architectures (for example CORBA [7]), then new programming languages integrated with such architectures (Java and J2EE, C# and .NET), and most recently the Web Services architecture. Web Services (WS) have received enthusiastic buy-in from a wide range of vendors, and key elements of the technology base are already supported from CORBA, J2EE and .NET. Web Service interfaces will be available in a tremendous variety of technologies: hardware components, internal components of the operating system, new servers and major legacy products. Web Services draw on the extensible markup language, XML, to represent information, and impose standards over XML for such tasks as object invocation (SOAP) and service description (WSDL and UDDI).

Here, we ask the obvious question: to what degree can GIG/NCES architectures be mapped onto Web Services? Our investigation yields a mixture of good news and reasons for concern. On the positive side, there is a substantial overlap between the service oriented architectures and GIG architectures like the JBI, on which we focus in this paper. For example, Web Services and JBI both favor XML for information representation, and the WS-Notification layer of the Web Services architecture corresponds closely to the JBI’s publish-subscribe layer (specifically, to what are called “event sequences” in the JBI). Thus, in this

area, one could easily “map” the JBI to a platform supporting WS-Notification. (How well the resulting system would work is another matter: WS-Notification dictates an architecture, not an implementation, hence different products would be expected to have very different behaviors).

On the cautionary side, our study reveals non-trivial deficiencies. For example, Web Services lacks an information architecture analogous to the JBI information management architecture. Here, one might need to “port” ideas from the GIG/NCES world into a Web Services paradigm: a substantial but feasible task.

Finally, we identify some reasons for outright concern. These concerns center on a commonly cited weakness with the Web Services architecture, namely a pervasive lack of support for high availability, real-time or other time-critical event delivery mechanisms, and inattention to issues such as the “life cycle” of information objects and services. These issues also extend to rather practical considerations, such as the feasibility of scaling some of the concepts present in these architectures up to the size of deployments, rates of communication, and levels of disruption that must be anticipated in demanding military settings. Jointly, these kinds of concerns lead to a broader worry: for some time into the future, Web Services platforms may lack many of the features that GIG/NCES systems would be expected to use heavily, and may lack properties that would be important in GIG/NCES environments.

In the remainder of this paper, we elaborate on each of the points just summarized. Section 2 reviews the JBI from a GIG perspective. Section 3 places the Web Services architecture side by side with the JBI. Section 4 fleshes out the findings mentioned above, and Section 5 concludes by pointing to a potential research agenda focused on bridging the gap by developing technologies lacking in the Web Services framework but needed in military GIG settings. To some degree the research required would be low-risk and near-term, but other questions point to deeper challenges requiring sustained attention over many years.

2. The JBI Architecture

The Air Force JBI architecture can be traced to a series of studies conducted by the Air Force Scientific Advisory Board in the period 1998-1999. JBI is best understood as a set of standards governing a range of possible implementations; down the road, one would imagine multiple JBI systems, each having its own strengths and yet united by adherence to a central set of standards. Like the many implementations of the CORBA standard, it is hoped that JBI platforms will interoperate seamlessly, and that JBI-compliant products will be more easily integrated into information fusion applications than has traditionally been the case for military technologies. Figure 1 illustrates some of the major components of the architecture in its present form. A preliminary application programming interface for the architecture was published in 2002, and several implementations of major components already exist.

In the interest of brevity, we limit ourselves to a quick walk-through of what is expected to be a typical use of the JBI. Consider a rapid targeting system providing quick response capabilities when US forces come under fire in a theatre of operations. At the time that the episode begins, one might imagine that military commanders request that information relevant to a response be assembled, a process not unlike building a customized web page, except that in our case, the contents of the web page would be streams of data coming from various sensors, maps and other data pulled up from intelligence databases as the locations of shooters are refined, positions of coalition forces able to respond, casualty reports, and so forth. One can recognize several aspects here: discovery of the appropriate sources of information, construction of the initial “web page”, and then periodic updates as new events occur.

Concretely, one would imagine that a JBI “rapid targeting” application is launched and given initial configuration information concerning the episode. The first challenge for the application is to identify data sources. Accordingly, using the configuration information, the program would search a “meta data repository” to identify classes of servers and types of information potentially available to it, and then query active servers to find the subset with information actually relevant to the engagement. As servers respond, the application “subscribes” to various information resources they make available to the system. Security policies are employed to prevent inappropriate disclosure of sensitive information to unauthorized users.

Notice that the information resources in question correspond to military computing applications of various kinds and will typically have been operational for some time. These applications are assumed to have registered their capabilities and to be publishing information in standard formats that have been registered within the JBI's information architecture. Moreover they were probably designed and implemented long in

the past, without this specific client application in mind. The standardization of data formats and reporting policies is key to the whole approach, because it permits the system to be extended over time with new computing applications, new client systems, and even new security policies.

In addition to discovering useful resources, at the time it first launches an application would often download initial data from the sources: maps, current tactical data, etc. Subsequently, as new events occur, the data sources "publish" summaries: concise objects that describe the available data, but of small size suitable for rapid transmission on a network. Large data objects such as high resolution images, for example, would be identified by a form of URL and could be downloaded separately, but would not typically be "published" directly. As new events are delivered to the application by the JBI, this data can be updated. Each new event is "content filtered" both to enforce security policies and because the application may be forced to subscribe to a category of data only some of which is relevant to the mission. For example, perhaps it subscribed to information on enemy activity in the Faluja region, but is currently only displaying data relating to a particular map quadrant; in this case it might ask the JBI to filter the incoming events to reduce the network bandwidth by not forwarding events unrelated to the quadrant of interest. Notice the important role of "time" in the JBI: this is a point to which we return below.

Some applications will find it useful to retrieve past events, and the JBI provides a mechanism for doing this. The event "repository" collects events and stores them persistently until they expire. During this period, applications can search or query the repository. In our example, if the "quadrant of interest" changes, such a search might be used to rapidly establish context, perhaps by retrieving the last twenty-four hours of events associated with the new quadrant.

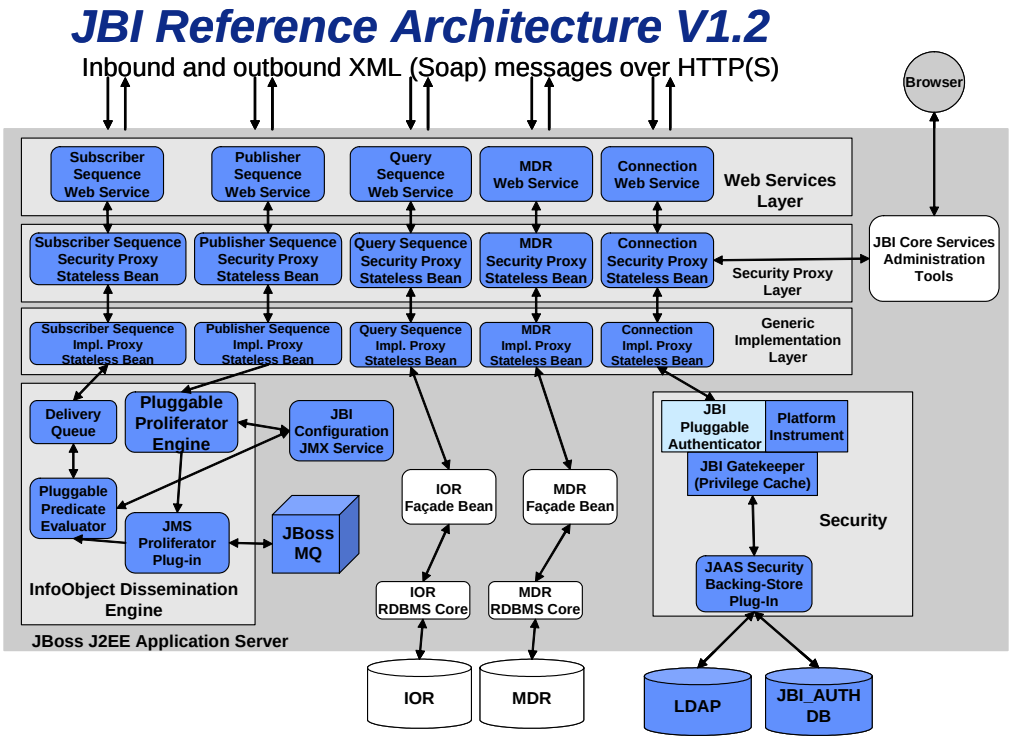

Figure 1 depicts the current JBI architectural structure identifying the functional elements within the associated implementation layers.

Figure 1 highlights the major service-side components of the JBI: a publish-subscribe information bus that supports a content-filtering mechanism, pervasive attention to information security considerations, the role of the repository as a searchable database of persistent events. Note that JBI offers a Web Service front-end. While Web Services are only used as an interface into the JBI, this paper investigates the feasibility of using Web Services as the fundamental architectural paradigm of the GIG (or JBI).

Not mentioned but implicit in our summary are other important aspects of the system. In particular, notice that the JBI functions to broker a communication relationship between a set of information sources and clients that may not have existed or even been contemplated at the time those sources were implemented. To facilitate this type of extensibility, the JBI architecture gives considerable attention to issues surrounding the "information architecture" of the system, a question orthogonal to the "service architecture." In particular, JBI includes an elaborate information ontology subsystem for standardizing the representation of various forms of information. For example, many kinds of devices can capture images, and if connected to a JBI platform, would report the capture of new images through small descriptive messages. The intent is that where possible, products from different vendors would use the same conventions to represent the image size and shape, the time at which it was captured, and so forth. The ontology is extensible, so that a vendor could provide additional data "special" to the product.

The JBI architecture also tackles what might be called "operational" properties of the platform. We've already touched upon security, but other important properties include the ability of the platform to function robustly and to configure itself automatically. The JBI should be self-repairing after disruptions, and should sustain high performance even in very large configurations. Large could mean many things here: the scale of the network itself (a larger network means higher latencies and an increased frequency of disruptive events simply because more components are involved), increased numbers of data producing and consuming "clients", increased rates of client system failures and recoveries, increased load on JBI components, etc. Moreover, the JBI architecture can accommodate varied service expectations on the information streams in question: some kinds of data should be delivered rapidly or not at all; other data may need to be delivered reliably even if delayed by a network failure or overload. Some data should be stored persistently, other data treated as ephemeral. Priorities will be common: some events are of great importance and urgency; others are routine and can safely be delayed or even discarded if load surges.

Even prior to recent research on Web Services architectures, these properties were under intensive scrutiny within the AFRL research community, because they represent significant departures from the properties and requirements of typical commercial platforms that one might use in implementing a JBI-compliant platform. For example, one does not see commercial publish-subscribe technologies with the needed mixture of scalability, robustness and delivery guarantees.

3. Web Services

Figure 2 illustrates the process of interacting with a Web Service in the Web Services architecture, which has been developed by the W3 consortium, bringing together many of the largest platform vendors in the information technologies sector. Web Services is best understood in light of two prior trends.

One trend is the explosive adoption of Web browsers and Internet technologies, which have made HTML and HTTP defacto standards within a tremendous range of products and devices. Web Services build over these standards, replacing the human user of a Web browser with a computer system. Thus where the human clicks on a button after filling in some boxes on a Web page, a Web Services client issues what looks like a remote method invocation; the parameters are marshaled into an XML representation conforming to the SOAP protocol and then transmitted to a Web Services dispatcher, typically using HTTP over TCP (however, other protocols can also be supported). The dispatcher passes the request to the server, then returns a result. Provision is included for transactional operations on databases, and for the use of intermediary processes that can function as message queuing middleware. The reliability model centers on message queuing, persistence and transactional facilities. There are two competing proposals for supporting event streams in Web Service platforms. Here, we'll focus on WS-Notification, which provides for event channels with content filtering.

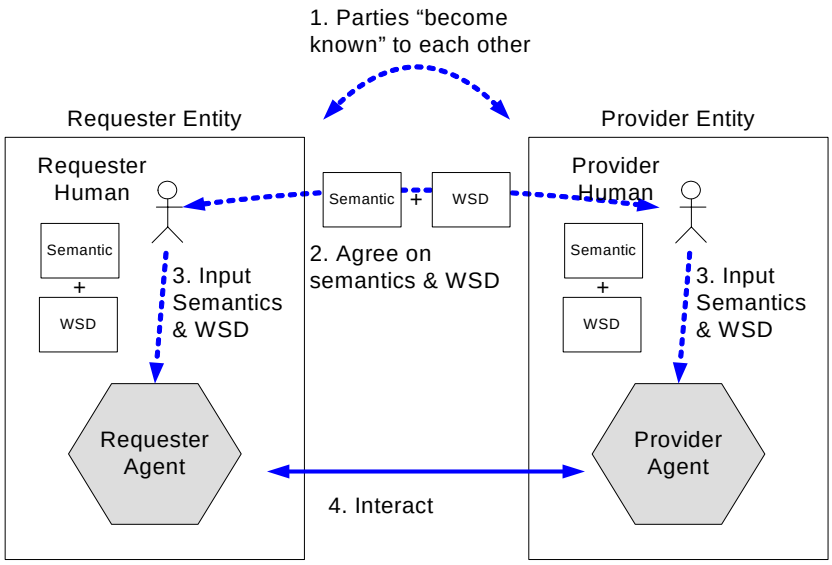

Figure 2 Interaction with a Web Service [9]: The Web Service specification defines a *requester* and a *provider entity*, both of which can represent a person or organization. The corresponding software components are called *requester* and *provider agents*. In a first step, the parties learn of each other, often via a directory service. Then, they agree on their respective semantics and the Web Service description. Finally, the agents can exchange messages.

The second trend involves the popularity of object oriented architectures such as CORBA, J2EE and .NET. Like Web Services, these support method invocations in a request-reply style, but go beyond that to also provide a range of "life cycle" services and a powerful type system. For example, CORBA platforms often include "factory" components that manufacture (or launch) objects on demand, persistence components for long-term storage, and a variety of state management services. CORBA also includes mechanisms for supporting high availability and there has been work on other quality of service options in the architecture. J2EE and .NET share some of these properties with CORBA, although the details differ and each platform has its own "focus". And CORBA is able to do relatively strong type checking across interfaces.

As mentioned, Web Services emerges from these parallel trends. Yet the architecture omits many features of the object-oriented systems, a point to which we'll return repeatedly below. These missing features have led many to suggest that "Web Services are not Distributed Objects" (e.g., [11]), and emerge as a source of concern.

For example, we commented that object-oriented systems typically have relatively strong notions of type hierarchy, inheritance and type checking. This is not the case for Web Services. The Web Services architecture obtains some related features from the so-called "semantic web", which seeks to represent relationships between attributes of XML documents; this superimposes a lightweight type hierarchy on documents. However, the main purpose is to enable automated transformations from one representation to another and to facilitate a type of automated "understanding" of the semantics of attributes within documents. Not only does this limit the kinds of type checking one can do in Web Services systems, it also makes it surprisingly hard to introduce type-like properties to communication channels or services, including high availability guarantees, responsiveness guarantees, quality of service properties, and many others. This reduced attention to the roles and value of type systems, we'll argue, is symptomatic of a broader problem: Web Services are document centric, whereas CORBA is object and interface centric.

Earlier, we commented that GIG/NCES technologies have rather elaborate information architectures. More specifically, we'll see that these technologies have drawn on the object type hierarchy in support of their own information ontologies, and on notions of communication and object "properties" to offer forms of quality of service guarantees. Loosely, these match features seen in CORBA and similar object-oriented systems, and fall squarely into the area where Web Services have departed from object-oriented systems. Thus moving the GIG to what is, in effect, a weaker technology base could be problematic.

4. Mapping GIG Architectures to Web Services

There is an extensive but in some ways shallow correspondence between the features of GIG architectures such as JBI and those of the Web Services architecture. For reasons of brevity we touch on areas of match, while drilling down where we identify important differences.

Generally, GIG standards have tended to emphasize communication architectures, which address the way applications can share information with each other. In contrast, NCES have focused more on database properties. More specifically, the subsystems in an NCES collaborate in a way that preserves the consistency of the involved databases. In this context, database transactions and workflows are important mechanisms. Web services push the transactional options even further; they go beyond what is seen in NCES architectures and thus also far beyond anything seen in GIG architectures.

Figure 3 graphically represents the mapping between GIG and Web Services. The white circle represents the GIG architecture and the gray surface represents Web Services. While the Web Service specification covers some of the needs of the GIG, the converse does not hold: an important part of GIG is not covered by the Web Service specification. At the same time, there are aspects of the Web Services specifications that seem remote from the primary focus of the GIG. For example, Web Services try to treat information systems as databases of documents; one sees this at many levels (including in the documentation and marketing materials for the platforms); GIG systems have more of a communication-centric perspective. The interrupted lines illustrate areas that are significantly different between Web Services and the GIG and that will be the focus of the rest of the paper.

In our figure, we've chosen to include work done by the Globus alliance [5]. Globus extends the Web Service specification to the context of e-science Grid architectures. Although Globus specifications have not received great attention in industry, we do want to note that these tackle at least some of the areas we've highlighted as concerns. For example, e-science systems often work with huge data sets, hence Globus pays attention to issues of data set size that the Web Services community seems not to have considered. However, since the GIG architecture will need to rely on *widely adopted* standards, we do not consider the Globus specification any further in this paper. Similarly, we won't comment on a half dozen other proposed standards that fall into our areas of concern, but seem to lack commercial traction.

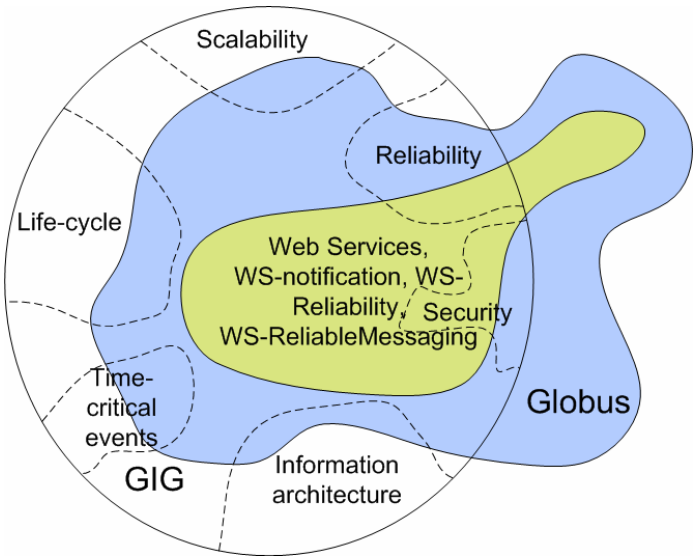

Figure 3 Overlap between the GIG/JBI, Globus, and Web Services standards.

In the remainder of this section, we first discuss the general consequences of the document-centric nature of Web Services. We then elaborate on some specific areas of concern, such as time-critical events, life-cycle management, and operational properties.

4.1. Consequences and Limitations of the Document-Centric Nature of Web Services

One major difference between Web Services and traditional messaging or object-oriented middleware (e.g., CORBA) is the extremely document-centric nature of Web Services platforms [11]. To understand this point, consider traditional middleware systems, such as message-queuing middleware. Here, many properties are defined either on the communication channel or are intrinsic in the communication protocol. Now consider object-oriented platforms. In these, at the center of the system one finds sets of objects, their type hierarchy and relationships, and message-based method invocation protocols that are layered over that base. For example, we might talk about a communication protocol that guarantees such and such a delivery latency bound, or security property, or reliability property. Such a guarantee is a property of the channel, not the messages: the channel might guarantee that 99% of messages will be delivered within 20ms.

In contrast, Web Services define many properties as part of the document, and yet documents are a loose concept – arbitrarily extensible, not "type checkable", amenable to a rather ad-hoc style of creation and manipulation, and often carrying their own property requirements. The services themselves offer interfaces that can "process documents", but nothing in the way of guarantees.

Return, for example, to our communication channel. If one drills down into Web Services specifications such as WS-Reliability, one finds that properties such as reliability are defined on a *per-document* basis, not at the level of services. In effect, there are no "reliable services", although there may be promises that a given document will not be lost as it gets passed around the system. The document itself carries an XML representation of its own reliability needs. The message says "please deliver me with at-least-once semantics", and the various components of the platform are then expected to either reject the request, or to accept a "contract" to perform as required. This is a subtle but profound shift of perspective.

A rather practical issue that arises is that as we move away from strong type hierarchies (which can be enforced by type checkers) and from object-oriented specifications (which can be supported by tools to replicate objects, persist them, fabricate them, etc), we find ourselves in a world of very high message overheads, because optimizations possible on a per-service or per-channel basis become awkward. It becomes hard to see how one might enforce large-scale policies, such as real-time guarantees, high availability guarantees, or certain kinds of security guarantees. These latter goals would require strong statements not about documents, but about services, and this is simply not the primary focus of existing Web Services platform development efforts. Broadly, in adopting a document-centric perspective the Web Services architecture benefits by being closer to the way that Web sites and browsers work, but loses by stepping away from two decades of research that builds "up" from a core notion of objects and types to higher level systems composed of typed objects (and typed communication channels), with high-level properties emerging from that composition.

This is not to say that Web Services can't reestablish the same sorts of useful properties and tools in the new document-centric world. But at the outset, the architecture is missing surprisingly many of the things we take for granted in CORBA and similar object oriented environments.

4.1.1. Information Architectures and Service Integration

Earlier, we pointed out that Web Services lack an "information architecture" analogous to the standards under development in the GIG efforts. This GIG information architecture can be understood as a mixture of a type hierarchy for military data objects (also called an information ontology) and a set of rules of use representing internal standards governing the conditions under which data will be published, the rules for translation between different terms for the same object, semantic properties, etc. Even NCES systems, which generally do presume that there are databases at the bottom, recognize that one can only federate databases to a limited degree, and focus on advertising the kinds of data a given service manages, not on cross-service integration.

This difference of emphasis could be problematic for the military. For example, suppose that multiple vendors offer products that can capture images and annotate them: daylight images, infrared, radar, sonar, etc, with objects identified, tracks for moving vehicles or aircraft, and so forth. Different systems may report the sightings of the same object in different formats, yet we would wish to maintain the greatest possible degree of constancy. Where formats differ but can be related, one would wish that translation be automated, so that components depending on a given type of information can find that information without needing special code for each specific vendor product. Unfortunately, the Web Services architecture does

not help in this task; it merely standardizes the format of the description. Moreover, even the broader notion that a domain might bring an information architecture of its own "to the table" is not one that the Web Services architecture has embraced. Thus we see this as a major area in need of attention.

A similar lack is manifest in the description of services. Web Services are generally described using the Web Service Description Language (WSDL) [6]. While WSDL defines the syntax of a service description, it does not standardize a way in which a service would advertise other properties or other guarantees to a client. More specifically, a service lacks any standard way to inform the client of expected response time or the appropriate recovery action to cope with a timeout or some other fault. Yet such properties matter in the GIG, where applications may be highly sensitive to latency, data rates, priorities, and related issues such as availability guarantees. One might wish for a form of extended WSDL including attributes that could specify these kinds of properties and guarantees, so that clients can know what behavior can be expected by the service. Again, we would call for much more attention to such issues, and to the underlying technology issues that arise as we try to build complex Web Services systems that in fact can offer guarantees.

4.1.2. Difficult to Avoid Message Size Overheads

The specification of properties desired by a document can contribute substantial overhead to the payload of messages. If the payload has a large size (e.g., images of enemy vehicles) this overhead is negligible. However, for small payloads, the additional overhead becomes an important performance factor. While in traditional fixed networks large bandwidth is usually available and this additional message size generally would not overload the network, this is not the case in other settings. Assume, for instance, a mobile ad-hoc network among soldiers themselves and their vehicles. In such a network, because of the shared transmission medium, the probability of interference between two messages increases with increasing message size. Reducing the message size as much as possible thus becomes instrumental in this kind of networks. Yet while Web Services allow individual clients to negotiate non-standard protocols and message representations with individual services, doing so "system-wide" would stray far from the norms.

In contrast, these types of considerations have long been tackled in the context of communication primitives (e.g., group communication, one-way invocations in CORBA) or the properties of a communication channel (e.g., UDP vs. TCP). When such an approach is taken, the resulting properties are "inherited" by the messages sent over a communication channel and thus do not need to be transported with each message (although some control messages or control information may be sent with the message). Strategies such as this, which have given us "real time" CORBA ORBs and multi-level security for some object oriented systems do not map in a direct way to Web Services.

To reduce message size, it would make sense for a document to omit some parts of the complete definition of its properties. Instead, a document might contain a reference to the corresponding definition document, which in turn contains the specification. However, this requires that the property definition documents need to be distributed to all considered servers before the actual sending of the document. Such options seem to go well beyond the existing Web Services specifications.

4.1.3. Lack of Support for Large-scale Policy Enforcement

Web services lack features for large-scale policy enforcement. This omission reflects the document-centric nature of Web Services and the subsequent independence of each server. Moreover, server compliance with the document's properties is not enforced by the underlying infrastructure, as would occur in CORBA as a consequence of type checking.

For example, consider the enforcement of security properties. In Web Services, every server basically implements its own security mechanisms. The client specifies the desired guarantees in the sent document, but has no guarantees that any destination will actually enforce these guarantees. The only way to make sure that policies are enforced is to implement an end-to-end encryption policy, or to send the document only to servers that are otherwise known to comply with the sender's expectations. This requires a mechanism to identify these servers' policies prior to sending the document and to verify that these servers satisfy the required properties. Hence, before even sending the document, the sender has to make sure that the destinations (and all intermediate servers) can satisfy the properties specified in the document. This seems to imply that a sender must know exactly by which intermediate servers his document is routed, which requires intimate knowledge about the application setup, and is completely at odds with the

philosophy of SOA platforms, where implementation details should be opaque to clients! The same issue also arises in the context of reliable many-to-many communication, which we discuss later in the paper.

4.2. Lack of Support for Time-Critical Events/Messages

In military settings, many events are of relevance only during a certain time. After this time, their usefulness expires and they can be disposed of (or archived somewhere for post-action evaluation). Consider, for instance, an artillery targeting system. The coordinates of a target are only valid while the target is stationary at a particular location. Once the target has moved, the coordinates become obsolete. Hence, the communication infrastructure must try to deliver messages within a certain time. After a message has expired, it can be disposed of. Typically, real time systems do not achieve this behavior; instead, to achieve strict guarantees, they slow all messages down. Web Services currently do not support this kind of time-critical events. For use in the GIG, it will be important to extend Web Services.

Indeed, as currently contemplated, a Web Services may delay messages for extended periods of time without notification to client systems. This property is basic to the WS-Reliability model, which centers on queuing subsystems that persist messages even beyond failures of intermediate nodes[1] [2][4]. Consequently, it is not possible to specify an event notification system that reliably delivers events up to a certain time-to-live of the event. After the time-to-live of the event has expired, the event is discarded (or may be logged in a repository for later evaluation).

4.3. Lack of Support for Object Life-Cycle Management

Web Services lack life-cycle features analogous to object factories and persistent object storage in CORBA. While the WS specification postulates that service management in general, including life cycle management, can be a part of the service description, it does not provide details on how this could be done. While one could extend the specifications to add such mechanisms (indeed, Globus does so), this would depart in significant ways from the commercial trends. Moreover, we run back into the same point made repeatedly above: in a technology where properties are associated with documents, not servers, it isn't clear how one can introduce a new kind of server-specific property in an elegant, appropriate manner.

4.4. Inattention to Quality of Service and Other Operational Properties

Operational properties, as cited earlier, include security, robustness, performance, and automatic (re)configuration. These properties are of great importance in military setting such as targeted by the GIG. Indeed, in a military environment, it is highly likely that components of a system are destroyed because of enemy fire. Any system deployed thus needs to be inherently self organized and to respond quickly to failures.

Properties such as scalability and performance have not yet received much attention in the web services specification. Moreover, it is not clear how some of the Web Services specifications interact. As an example, it is not clear how a comprehensive security model can be achieved for WS-Notification [1] by combining it with the Web Services security specification.

In the following, we discuss robustness (Section 4.4.1), security (Section 4.4.2), and scalability issues (Section 4.4.3) in more detail.

4.4.1. Reliable Messaging

The Web Service specification includes an event notification standard, called WS-Notification [1]. WS-Notification defines a content-based publish-subscribe system. The specification explicitly defines an intermediate notification broker, similarly to traditional message-oriented middleware. Consequently, the implementation of WS-Notification relies on federated notification brokers, ruling out implementations based on other architectures, such as peer-to-peer systems, or those using group communication. As yet, there has been no attention to scalability and performance properties that may be supported by a WS-Notification implementation.

Moreover, it is not clear how various specifications interact. Consider, for instance, the example of WS-Notification and WS-ReliableMessaging [4] or WS-Reliability [2]. Both WS-Reliability and WS-

[1] Some publish-subscribe systems suffer from the same limitation.

ReliableMessaging ensure at-least-once, at-most-once, exactly-once, and ordered delivery of messages between two SOAP nodes. At-least-once delivery generally requires persistent storage at the sender side, while at-most-once delivery requires persistent storage at the receiver side. Exactly-once delivery is the logical AND of at-least-once and at-most-once delivery; it can only be supported if both the sender and the receiver implement the needed mechanisms, providing (in effect) an end-to-end guarantee from the sender to the receiver. As detailed in the specifications, protocols implementing these sorts of mixtures of properties would look much like a transaction between sender and receiver, with multiple coordination messages. Such an approach is remote from the protocols known to scale well and perform well in such situations [3], hence we have a case where merely by combining two specifications, WS-Notification and WS-ReliableMessaging, we arrive at a seemingly quite restrictive solution that precludes efficient, scalable, high performance implementation!

And this is anything but the end of the story. By defining reliability guarantees on a per-message basis, Web Services fit poorly with applications that need non-trivial distributed reliability guarantees such as "virtual synchrony" or "multicast atomicity" (common properties of group communication systems [8]). Virtual synchrony orders message delivery with respect to the processes' view of the group membership. Atomic multicast further orders the messages with respect to each other: every process that does not fail receives the same sequence of messages. It is not at all clear that one can implement a Web Services system in which these sorts of properties are offered on a "per-document" basis, as the architecture seems to require. Indeed, this mixture of properties is interesting because it can be proved, formally, that it is not possible to implement them through purely end-to-end mechanisms that operate one-to-one between client and server (or intermediary). Instead, system-wide agreement is needed on such properties as the state (healthy or failed) of processes in the system. Thus there are signs of a fairly deep mismatch between some aspects of the Web Services model and some properties that GIG systems are likely to need.

In a similar sense, it is not clear how probabilistic guarantees can be represented in Web Services. Probabilistic guarantees are important because it has been shown that scalability can be improved by trading deterministic for probabilistic guarantees [3]. Clearly, Web Services simply lacks an acceptable approach to communications reliability, reflecting the different degrees of reliability that the application might need to choose from, and recognizing that some properties must be imposed platform-wide, not just on a point to point basis between client and server.

4.4.2. Security

As mentioned briefly above, Web Services lack a comprehensive specification of security measures as required by the GIG. Indeed, Web services only specify the security on a per-service level. However, the GIG requires a system-wide establishment of security levels associated with roles and particular access rights. Moreover, the corresponding signatures need to be maintained and periodically updated. These requirements go well beyond what is specified in Web Services and require a considerable design effort in the GIG. Reasons of brevity preclude a detailed discussion of this point.

4.4.3. Scalability

In Section 4.4.1, we discussed the impact of the reliable messaging specification on WS-Notification. We argued that Web Services embodies architectural decisions that may be problematic for developers of communication-oriented applications where time-critical event delivery, consistency, or fault-tolerance are needed. But the same point extends to scalability as well. As a research community, we are only just starting to find ways to build large systems that are able to operate robustly despite disruptions such as node crashes, message loss, load surges or dynamic reconfiguration. To build very large Web Services deployments we will need to migrate these best of breed techniques into the architecture and into our platforms. As things stand, the same kinds of architectural decisions that were cited above could limit scalability of Web Services technologies and preclude the sorts of large deployments that will be needed in GIG settings.

4.5. Bridging the Gap

Although we've been critical of a number of shortcomings in the Web Service architecture, we are actually convinced that all of the concerns raised can be overcome. One approach is for the GIG platform developer to extend a Web Services system to have the desired behavior, grafting functionality "over" the standard

platforms. However, this seems unnatural and may even, in some cases, violate the spirit in which the WS specification was written. As an example, consider the properties of reliable messaging that involve multiple receivers such as presented in Section 4.4.1.

From this example and from the discussion above, we see a need for concisely specifying the properties of the service and the communication channels. However, it is not clear how these properties should be defined within the current WS specification. Introducing properties may be particularly challenging for communication channels; such a change would require some fundamental research, and could require non-trivial extensions to the architecture.

As it happens, many of the weaknesses enumerated above are also evident in traditional messaging or object-oriented middleware. In particular, none of these systems generally comes with an information architecture, or specifies the exact behavior in case of failures. Many deal with reliability or scalability in ad-hoc, proprietary ways. Often, this task is left to the developer and, without system-wide guidelines, is specified differently for every server; very much the problem identified above also in the case of Web Services. Thus while Web Services have limitations, it would be unfair to suggest that they are unique in this respect!

Table 1 summarizes the shortcomings and lacks we have identified in the mapping of the GIG to Web Services.

Domain	**What is lacking in the WS specification with respect to the GIG architecture?**
Information architecture and service description	WS provides the data format for the representation of the information architecture and the service description. However, the standardization of the service description is left to the GIG.
Document-centric nature	Dramatically impacts message size. More broadly, points to a lack of large-scale policy specification and enforcement options.
Time-critical events	No support for time-critical events is given by the Web Services specification.
Life cycle management	Not specified comprehensively in WS specification. Needs to be defined by GIG.
Reliable messages	Is limited to one-on-one interaction between a client and a server.
Security	Interoperation with other specifications is not clear. A comprehensive security model is needed for the entire GIG.
Scalability	No experiences yet.

Table 1 Summary of functionality and issues that are cause for some concern or that are lacking in the WS specification and that the GIG architecture needs to provide.

We believe that some of these deficiencies could be addressed fairly easily; indeed, the Globus effort has done so in some areas (such as Life Cycle Management). Others will require more work: reliability, scalability, other kinds of operational properties of service platforms. And still others demand basic research. For example, we are entering a domain of scale never before seen in distributed systems research. We simply lack protocols capable of providing the mixtures of properties that GIG/NCES systems will require, and that same limitation applies to Web Services platforms. The good news is that with attention to the issue, there is every reason to believe that researchers can solve these problems.

5. Conclusions and Directions for Future Work

Charles Holland, Assistant Deputy Secretary of Defense for Research and Engineering has commented that GIG and NCES platforms represent some of the most promising and important technology developments under current consideration. There is strong reason to believe that GIG and NCES solutions could lead to dramatic productivity improvements within the military and, by helping our planners and troops gain better access to information when and as needed, to fight more intelligently and with reduced casualties. The use of standards will reduce the “stovepipe” phenomenon common in military procurement and in so doing reduce the costs and time to developing needed capabilities.

However, the similarity of GIG and NCES architectures to Web Services, and the remarkable rate of update of the latter, suggests that GIG and NCES solutions will increasingly need to be implemented over Web Services platforms (or other SOA technologies). This will not be possible without substantial investment of research energy into open questions, such as the creation of scalable event notification systems able to offer time-critical message delivery, stability under stress, or other key properties. One can make a long list of such technical needs, and the problems aren't shallow: to solve some, we will need major scientific advances of a fundamental nature. On the other hand, not all the problems are of such difficulty; in some cases the primary need reduces to simply demonstrating solutions to GIG and NCES challenges within a Web Services paradigm.

We see particular reason for concern in such areas as reliability, scalability, high availability, real-time event delivery, and the composition of security with these properties. Commercial vendors can afford to build systems that require substantial manual "care and feeding" such as install-time configuration, and that may require manual repair after disruptions or failures, such events being uncommon in commercial settings. In contrast, military systems often must be deployed in great haste, may need to function securely and reliably even when damage has been sustained, and operate within very limited technical oversight.

We have become persuaded that bridging these gaps represents a "must do" objective for the military. But we also see strong reasons that commercial interest should parallel these military needs. Military investment in the areas we've cited should lead to better products with commercial advantages in a diversity of settings. Indeed, one can criticize Web Services precisely for the omissions identified earlier – omissions that will matter in many commercial settings. It is surprising that platform vendors have yet to engage these issues, but we are convinced that as these problems are solved, there will be excellent opportunities to transfer the results back into the commercial mainstream, and indeed to influence the future evolution of the Web Services architecture and standard.

REFERENCES

1. S. Graham, P. Niblet, D. Chappell, A. Lewis, N. Nagaratnam, J. Parikh, S. Patil, S. Samdarshi, S. Tuecke, W. Vambenepe, B. Weihl. Web Services Notification (WS-Notification). IBM, Sonic Software, SAP, HP, Akamai, Tibco. Version 1.0. January 2004.
2. C. Evans, D. Chappell, D. Bunting, G. Tharakan, H. Shimamura, J. Durand, J. Mischkinsky, K. Nihei, K. Iwasa, M. Chapman, M. Schimamura, N. Kassem, N. Yamamoto, S. Kunisetty, T. Hashimoto, T. Rutt and Y. Nomura. Web Services Reliability (WS-Reliability). Fujitsu Limited, Hitachi, NEC Corporation, Oracle Corp, Sonic Software Crop, and Sun Microsystems, January 2003.
3. K.B. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, Y. Minsky. Bimodal Multicast. ACM TOCS. (17) 2: May 1999. Pages 41-88.
4. R. Bilorusets et al. Web Services Reliable Messaging Protocol (WS-ReliableMessaging). BEA Systems, IBM Microsoft, Tibco. March 2004.
5. The Globus Alliance. http://www.globus.org
6. W3C: Web Services Description Language (WSDL). August 2004. http://www.w3c.org/TR/wsdl20.
7. OMG: Common Object Request Broker Architecture (CORBA). http://www.corba.org.
8. K. Birman. The process group approach to reliable distributed computing. CACM (36)12: December 1993. Pages 37-53,103.
9. D. Booth, H. Haas, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web Services Architecture. W3C Working Group Note. February 2004.
10. M. Paolucci, N. Srinivasan, and K. Sycara. OWL Ontology of Web Service Architecture Concepts. http://www.w3.org/2004/wsa.
11. Web Services are not Distributed Objects. Werner Vogels, IEEE Internet Computing, Vol. 7, No. 6, pp 59-66, November/December 2003.