

Issues in Providing Quality of Service in a Joint Battlespace Infosphere

Joseph Loyall, Jamie Lawson, Gary Duzan
BBN Technologies, Cambridge, MA
{jloyall, jlawson, gduzan}@bbn.com

Abstract

Military command and control systems rely on information that is timely, accurate, and in the correct form in order to make mission-critical decisions. Therefore these systems rely on having predictable quality of service, or QoS, for information creation, delivery, and processing. The Joint Battlespace Infosphere, or JBI, is an information system architecture being developed to support the military's information exchange for decision making at all echelons. The JBI is based upon a publish, subscribe, and query architecture with support for information fusion and aggregation. This paper discusses the issues of providing QoS in a JBI implementation and describes a prototype JBI implementing publish on demand, an approach to providing QoS managed information through a JBI.

1. Introduction

A large class of military operations, including command and control (C2) systems, involve information exchange that must be timely, accurate, and in the correct form. The right information too late can be useless, as can timely information in the wrong form or with insufficient quality. Information delivery for mission-critical decisions in C2 systems needs *quality of service (QoS)* management because it competes with other interactions taking place, because it must occur in dynamically changing and unpredictable environments, and because it must often take place in resource-constrained situations.

The *Joint Battlespace Infosphere (JBI)* is an emerging information management middleware being developed to provide military users with the information needed to perform their functions, including during crisis or conflict. To do this, the JBI must integrate data from a wide variety of sources, aggregate that

information, and distribute it in the appropriate form and level of detail to users at all echelons [15][16].

For the JBI to be a viable information exchange infrastructure for future military operations, it must be able to support the QoS-managed exchange of information. However, the JBI architecture, which is built upon a publish/subscribe model for information exchange, provides some challenges for QoS management, including the following:

- Decoupling the consumer of information, who determines the QoS requirements, from the provider of information, who determines its form.
- The hiding of the JBI platform and its underlying infrastructure and resources, which must be managed to provide QoS.
- JBI information processing elements, *fuselets*, which can affect the quality of information and the quality of service without the JBI users' knowledge.

This paper discusses the issues in, and explores approaches for, providing Quality of Service (QoS) and Quality of Information (QoI) (which we will collectively refer to as QoS) support in a JBI context. We begin, in Section 2, by giving a brief overview of the JBI, its scope, and its technical approach. In Section 3, we discuss issues in providing QoS management in a JBI implementation. In Section 4, we explore approaches to providing QoS management in JBI implementations. In Section 5, we discuss an approach, called *publish on demand*, that we have prototyped to provide a measure of QoS in a JBI implementation. Finally, in Section 6, we provide some concluding remarks.

2. Overview of the Joint Battlespace Infosphere

The JBI is an architecture for the next generation of command and control systems, intended to dramatically improve the transformation of commander's in-

This work was supported by AFRL under contract number F30602-00-C-0032-P00003. Approved for public release.

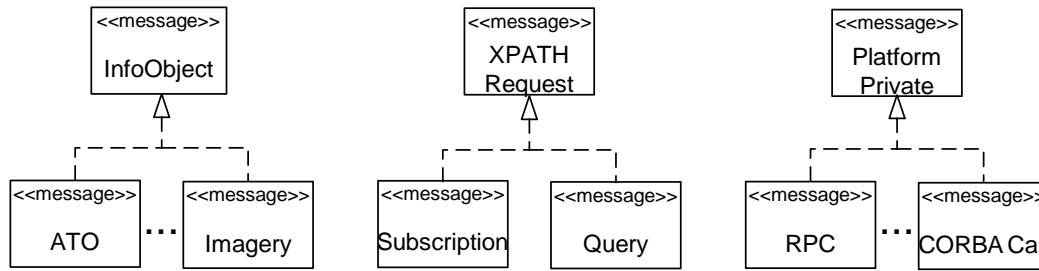


Figure 1. There are three kinds of JBI messages – information objects, XPATH expressions, and messages that are private to the platform.

tent into effective action in the battlespace [15][16]. JBI is based on a foundation of five high-level critical system functions:

1. *Publish*: Making information available to those who need it.
2. *Subscribe*: Fulfilling a need for information whenever instances of that kind of information become available at any point in the future.
3. *Query*: Fulfilling a need for information with all instances of that kind of information that have been made available in the past.
4. *Transform*: Turning low-level data into high-level information and making that high-level information available to those who need it.
5. *Control*: Turning information into action.

The *transform* function is realized with components called *fuselets* [10] which feed on information disseminated through *publish*, *subscribe*, and *query*. These decouple the user of information from its source, or at least make them less tightly coupled and independently (asynchronously) acting, a key underlying principle of the JBI. JBI supports requests based on content, not on the identity of the information provider.

The five high-level critical JBI functions are realized by three essential architectural elements, which must be present in any JBI instantiation:

1. *The JBI platform* provides a query broker, subscription broker, management services, fuselet runtime environment, access control, etc. Different JBI platforms may provide these services in different ways, and the manner in which these services communicate with one another is up to each platform implementation.
2. *JBI clients* interact with the JBI platform, which means that they participate in at least one of the five critical functions.
3. *The JBI Common API (CAPI)* [6] provides an interface for JBI clients to interact with the JBI platform in a platform neutral way.

The JBI's *force template* capability [9] is a place to specify information about a JBI client, useful to the JBI for plugging the client into existing interactions and for controlling the client's behavior with regard to its JBI interactions.

There are three kinds of messages in the JBI: Information objects (IOs), XPATH expressions, and messages that are private to the platform, as illustrated in Figure 1. IOs are the common currency between the clients and the platform shown in Figure 2. XPATH expressions cover subscription and query messages. Messages between platform components might be RMI calls, CORBA calls, SOAP commands, or whatever private communication the platform uses.

An IO consists of three parts: 1) Type; 2) Metadata; and 3) Payload. Associated with each type is XML schema that specifies the form of the metadata. The metadata part of the IO is an XML document conforming to that type. The payload can be in any format, but the metadata has a text field that specifies the format so that a user of the object can make use of the payload.

3. Issues in Providing QoS in a JBI

The overall JBI objectives are paraphrased as “with the right access, provide the right information to the right person in the right form at the right time.” Some consideration for QoS is contained in this expression of goals, especially the phrases “right time,”

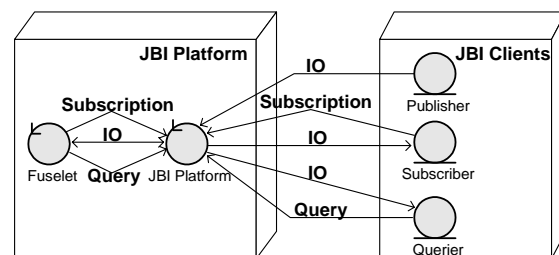


Figure 2. Client to platform communication in the JBI

“right form,” and “right information,” which correspond loosely to documented QoS attributes, such as timeliness, precision, and accuracy [7][17]. “Right access” and “right person” can pertain to security, survivability, and trust issues – all aspects of QoS.

One of the core principles of the JBI is its publish/subscribe architecture that decouples information providers (publishers) from information consumers (subscribers) and both of these from the infrastructure that underlies the JBI. However, providing QoS in distributed systems has traditionally required the following knowledge and control:

- Knowledge of a consumer’s use of information, which places requirements on the quality of the information and its delivery, and system QoS requirements, which might require satisfying and trading off multiple, possibly conflicting, consumer QoS requirements.
- Control over the form and content of the information provided, based on the QoS needs for the way the information is to be used.
- Control over the delivery of the information, including producer to consumer resource management.

The JBI’s complete decoupling of the information providers from information consumers, and hiding the information transport infrastructure makes it difficult to simultaneously capture the QoS requirements, shape the information, and control the resources of information delivery – all elements traditionally necessary for end-to-end distributed QoS.

The JBI as defined provides the following challenges for providing QoS:

- QoS needs to include end-to-end delivery of the right data with the right characteristics (e.g., timeliness, important data content) for whatever the consumer of the data needs to do.
- Requirements driving QoS are typically associated with the consumer end, while the (shared) resources for providing QoS are in the JBI infrastructure, and the ability to modify data to accommodate the resource availability and the consumer needs lie with the data supplier (publisher).
- The interoperation of many information sources and consumers via a JBI, with all of the information directed through, managed by, and made persistent by a JBI, could soon overwhelm that JBI. This becomes obvious when examining the trend in bandwidth requirements for this kind of data. As reported in the *New York Times*: “[A]irwaves over battlefields are becoming worryingly crowded. In the 1991 gulf war, networked com-

puters fed information at a rate of 192,000 words per minute. Soon, the military will be transmitting the equivalent of the Library of Congress each minute, or 1.5 trillion words, says Col. Bruce Sturk, former director of the Air Force Experimentation Office. UCAV’s will eat up a tremendous amount of that bandwidth, and tax already overburdened satellites, leading to the possibility of malfunctions on an unprecedented scale.” [1]

- JBI’s inherently centralized data management worsens some existing information bottlenecks and motivates better management of the shared resources.

Fuselets provide additional challenges to QoS management in a JBI. The primary purpose of a fuselet is to capture information from one or more input publications, transform and combine those sources into new information, and publish that information back to the JBI platform, as illustrated in Figure 3. However, information published with a particular level of quality might have significantly different quality once transformed. For instance, a weather fuselet may publish a weather forecast for a specific target location based on weather forecasts from nearby locations. This forecast is not likely to be as accurate as a weather forecast intended specifically for the target and based on observations at the target, but it may be possible to publish these forecasts more often. If a subscriber is subscribing to weather data for that target and both kinds of information objects are published, it needs to know that the fused weather forecast is less reliable than the actual forecast (though the fused forecast may be better in some respects if it is more current; that is also illustrative of a QoS tradeoff). But since the subscriber doesn’t know which forecast is fused, what it really needs is some kind of reliability contract from each of the information objects – a QoS specification that defines how reliable the data is or needs to be in order to satisfy specific requests for data, and what to do if the specification can’t be met¹.

There can be chains of publish/subscribe interactions and of fuselets, as illustrated in Figure 4. End-to-end QoS needs to address the entire flow of information along one of these chains and mediate the QoS demands stemming from multiple, simultaneously operating chains. Furthermore, in addition to the demands on the infrastructure for QoS of information

¹ These types of QoS specifications, called QoS *contracts*, are a core piece of QoS-aware middleware, such as our open-source *Quality Objects (QuO)* middleware [12], others such as *QoS Enabled Distributed Objects (Qedo)* [13], and QoS standards activities [3]. QuO has been used for providing dynamic, adaptive QoS control in wide-area and embedded environments and underlies the prototype implementation that we describe in Section 5.

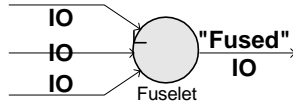


Figure 3. A fuselet “fuses” information from multiple sources and publishes the combined information back to the JBI

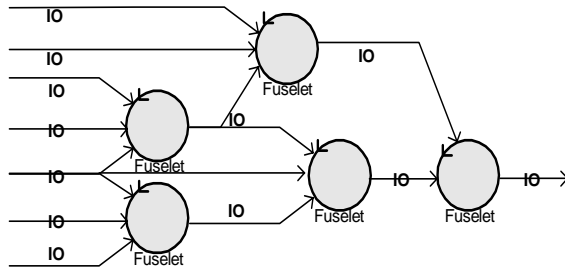


Figure 4. Fuselets are often composed with one another

delivery, the fuselet chains can affect the quality of the content of the information. The quality and reliability of transformed information might get worse at each step through the fuselet chain. This degradation needs to be captured so that the end user of the information can reasonably assess its utility. If the probability distributions of errors in the information are independent, the degradation may scale linearly or even randomly with the number of levels. If the distributions of the errors are dependent (which is likely when the inputs to a fuselet are related), the degradation may scale exponentially with the number of levels, depending on the kinds of information involved. It may be impossible to model this without a substantial amount of metadata about the reliability model for the information. Even if the model information is available, it can be difficult to describe joint probabilities from dependent distributions.

In addition, there could be significant lag time between the IOs that enter the chain at the left to the IO resulting on the far right side of the fuselet chain, based on each fuselet’s publication schedule, the length of the chain, and the QoS of the infrastructure. The “temporal extent” field for the originating IOs could be valid when they are processed by the first fuselets in the chain. The assumptions about temporal currency are pushed forward to the next level in the chain, without any way to trace back to the temporal extent of the original information. In short, fuselets down the chain could make perfectly reasonable decisions about the validity of the input information that are wrong because the temporal validity was lost dur-

ing fusion. There are many ways besides temporal extent that chains can degrade information quality.

Therefore, any solution to providing QoS management in a JBI will need to:

- Consider end-to-end QoS resource and data management;
- Support dynamic conditions (since things will come and go and resource availability, information needs, missions, and information availability will change dynamically);
- Consider tradeoffs (when missions come into conflict, when sufficient resources and/or data are not available; and when changes affect other things).

Providing QoS in a JBI must consider both the quality of information (i.e., application-level QoS) and the quality of the infrastructure underlying the JBI (i.e., network-level and host-level QoS). There are ongoing activities to provide network QoS management for JBI [2] and to provide survivability and security within a JBI [11].

QoS for information exchange, with regard to the mechanics of delivering data from one place to another in a distributed network is frequently described in terms of attributes or properties. For example, Lawrence et al [7][17] define the following three attributes of information QoS:

- *Timeliness* – The time at which information is available as measured against its creation (staleness), transmission (latency), use (deadline), etc.
- *Precision* – The amount of data in the information. For example, the number of bits in an item of transmitted data or the number of significant digits in a measurement.
- *Accuracy* – The amount of error (or, more exactly, the lack of error) in the data. In many cases, this reports how reliably the data that is used represents the data when it was created.

These attributes relate, at least loosely, to some of the JBI goals. The “right time” can be expressed as timeliness, while the “right form” and “right information” relate to the precision and accuracy of the information. For example, these attributes can describe some of the measures of QoS for information transmission across a network: a network can introduce delay (affecting timeliness); lose packets (affecting precision); and introduce noise (affecting accuracy).

In addition, we can identify additional QoS attributes that are important for mediating and adapting information from multiple sources in a resource constrained dynamic environment, such as that provided by the JBI. Two of these that are related to the JBI goals follow:

- *Importance*² – The source and receiver of the data must be able to quantify measures of importance for elements of the data in order to support intelligent adaptation, filtering, and shaping of the data, when circumstances will not enable all data to be delivered. When the network cannot handle the full volume of data, the underlying transport will drop data (e.g., in protocols such as UDP) or introduce delay (e.g., in reliable protocols such as TCP). However, the network transport will do so without regard to the relative importance of data elements (i.e., any bit is equally likely to be corrupted) and without information about the mission priorities. Some protocols, such as Differentiated Services (DiffServ) [4], enable the priority treatment of network traffic. These are at the per-packet level and are just emerging for wireless tactical environments. This requires a notion of application-level and mission-level information importance to translate into packet-level priorities and intelligently filter data when the transport cannot accommodate the full data traffic.
- *Trust* – The confidence that the receiver of the data can have that the data is uncorrupted. This can entail aspects of the accuracy of the data as described above, but also includes how confident the receiver can be that the source of the data is trustworthy. This is especially important in a system like JBI in which the source of data may not be known. Data processing, such as encryption, or additional data content, such as certificates, can increase the measure of trust in particular elements of data, but will usually come with tradeoffs, in the form of reducing the timeliness or increasing the amount of data.

There are scenarios where both infrastructure QoS and information QoS must be considered simultaneously and are at opposing objectives. Consider an example of an airborne sensor on an unmanned aerial vehicle (UAV) providing near real-time imagery to a ground based unit, such as an emergency response vehicle. The UAV is a JBI client capturing imagery for publication to a JBI platform through a low bandwidth connection. The constraints on infrastructure QoS (low bandwidth availability) may force the UAV to publish highly compressed/lossy images. The ground unit/subscriber, on the other hand, may have information QoS requirements for high-resolution imagery. One solution might be for the publisher to deliver only the pieces of the images that the subscriber needs, and

deliver those at higher resolution. But that solution is complicated by the fact that the subscriber can't communicate these needs directly to the publisher. The publisher and subscriber communicate only with the platform so there is no way to explicitly task the UAV while staying within the semantics of JBI.

Since the publisher does not know who will consume its messages, it cannot tune the QoS requirements to those of the subscriber. Similarly, it cannot provide a worst case QoS requirement specification – one that would satisfy all potential subscribers – because different subscribers might have very different needs. Trying to satisfy every QoS requirement simultaneously might mean that no subscriber's needs get fulfilled.

This leads to a question of how to provide QoS in a JBI, how to do so while keeping the information source and its end user decoupled from one another, and what role information objects can play in this. QoS is a concern that crosscuts the publisher (the object that can provide information in various forms), the subscriber (the object that knows what information in what form is important), and the platform (the infrastructure that can control the QoS of information delivery). End-to-end QoS can only be achieved with knowledge of the objectives of the applications (i.e., the source and use of the information) and the mechanisms or infrastructure available to deliver and process the information. This motivates the need for a *middleware*-based solution for QoS, where the crosscutting knowledge of the applications and infrastructure can be utilized, without intertwining it in either.

4. Approaches to Providing QoS in a JBI

One approach to providing QoS in a JBI is through management only at the network layer. This approach would need to cover all implementations of the JBI and all data in the system, including things outside the JBI such as SNMP and NFS traffic. However, this approach pays little attention to the domain interpretation of the traffic contents, does not consider the requirements of the applications using the JBI, and therefore does not solve the problem of what form and with what quality publishers should provide information.

Another approach is to place QoS information in the metadata of objects, which directly addresses domain interpretation. Ideally, a QoS description would be an optional part of the metadata for the *BaseObjectData*, and that description would conform to some well-defined XML schema. One problem with this approach is with inheritance. A new IO type that extends an existing IO type might have very different

² Lawrence also mentions importance, but describes it in terms of maintaining variance in timeliness, precision, and accuracy, rather than as a QoS attribute.

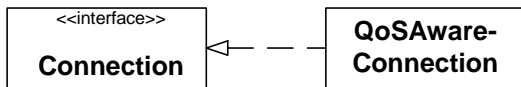


Figure 5. A QoS aware Connection can be provided that looks like a Connection to JBI clients but that manages some QoS aspects.

QoS parameters or different interpretations of the same parameters. QoS information is really a crosscutting aspect of the different IO types, but there is currently no support for this in JBI. On the other hand, if the QoS information is in the metadata, then clients can query it, and they can specify (some of) their QoS requirements in the XPATH request protocols for subscription and query. This offers considerable power with little change to JBI.

The CAPI *Connection* interface is a place where certain types of QoS could be enforced. The CAPI *Connection* interface defines the actual client/platform interaction semantics, through which all client-to-platform and platform-to-client messages are passed. Since the CAPI only defines interfaces with no explicit implementation, they can be realized in any way that satisfies the interface. A *QoS_Aware_Connection*, illustrated in Figure 5, could, for instance, manage QoS for client-to-platform interactions in a way where the client knows nothing about those functions. However, since a given CAPI implementation is likely to be dependent on a particular JBI platform implementation, the portability of such a solution is limited.

Force templates are another JBI feature that can provide some support for QoS. One possibility is that the force template can include the expected resource usage of a particular client. When a client enters a JBI, the JBI will either accept or reject its force template. If the force template is accepted, meaning that the client can connect to the JBI, the JBI creates a Force Template Controller (FTC) that enforces that the client behaves the way its force template describes. If clients can specify the resources that they consume and the QoS with which they can publish information, and the JBI can enforce the templates, then a JBI can do at least some limited amount of resource management.

Getting end-to-end QoS, however, from an information publisher to an information consumer involves both what the clients do and the resources involved with delivering the information from one to the other. Furthermore, the clients' requirements can change over time, especially as mission modes change and as the availability of resources change. To support this, we take an approach using QoS adaptive middleware, spe-

cifically BBN's Quality Objects (QuO) middleware [8][12]. We built a prototype, described in Section 5, that uses QuO to manage QoS and information by shaping the published data and managing the network resources for delivering the data, based upon policies that are dynamic and mission-aware.

As part of the prototyping effort, we developed a *publish on demand* capability in a JBI implementation. This is suitable for supporting the collection and publication of information that has specific QoS demands based upon the way it is being used by the information consumer. It also helps manage the information overload of information sources that have access to large amounts of information (such as surveillance imagery).

In its simplest form, publish on demand consists of an information supplier which subscribes to *requirements* information. When an information consumer publishes requirements information (describing the kind of information needed and the form in which it needs to be provided), the information supplier receives the requirements information, decides how to provide the information, and collects and publishes the desired information. The information consumer then receives the desired information.

This approach obtains the desired results using the existing API and without any additional platform support. However, it requires the subscriber to be aware that certain information may only be available on demand and take additional steps to account for that fact. Another possible solution is that the publisher could publish Information Objects for all its collectable information, but not actually perform any collection until the information from a specific Information Object is requested (using the `getPayload()` method). However, this requires the publisher to publish every possible piece of collectable information, along with every combination of QoS/QoI capabilities.

A variation on this approach is to move part of the publish on demand implementation into the JBI platform. When the platform receives a subscription, it determines if a requirements publication is necessary, extracts the required publication metadata from the subscription's metadata (including any QoS/QoI requirements), and publishes the resulting requirements Information Object. On the publisher's end, a specialized Information MetaObject describes the ability to publish other Information Objects of a particular type and across ranges of attribute values. On receiving the publication of the Information MetaObject, the platform sets up the appropriate subscription to match the range of requirements publications from the subscribers. On receiving the requirements Information Object, the platform makes a callback into the publisher, allowing the publisher to examine the requirements

metadata, collect the information, and publish the resulting Information Object with metadata which matches those required by the subscriber.

This approach has the advantages that it requires no additional action by the subscriber, only affects the publisher if he wishes to provide Information Objects on demand, and leverages existing mechanisms to communicate the required information. It has the disadvantages of requiring some extensions to the JBI API and additional complexity in platform implementations that support the feature. It would also need formalization of how the requirements publication is produced from subscriber metadata and vice versa.

A variation on this would use the JBI to publish and subscribe to meta-data only. The JBI connects up publishers with subscribers that need timely information, and the actual providing of the real-time data is performed through a direct client-to-client connection. This approach requires a back channel, perhaps a client-to-client connection, currently not part of the JBI definition (but likely needed in some cases for ongoing, longer transactions). Here the publisher filters publications based on runtime QoS measurements of the resources available, using consumer's mission re-

quirements to evaluate tradeoffs. We use this approach in our Intrusion Tolerance by Unpredictable Adaptation (ITUA) work [5].

There are a number of potential insertion points for QoS management in JBI, listed in Table 1. For the most part, the approach selected limits the choice of insertion point.

5. Prototype Publish on Demand for JBI

BBN and Scientific Research Corporation (SRC) developed the prototype system and proof-of-concept demonstration illustrated in Figure 6 [14]. In the demonstration, a set of simulated UAVs sends weapon, target, and image data to a simulated Air Operations Center (AOC), which publishes the imagery to a JBI. A simulated Army Reconnaissance Unit subscribes to images covering the area through which it is moving. The prototype uses BBN's QuO middleware [12] and SRC's WARP network management software [14] to provide QoS and data management for the various types of data being transferred between the UAVs, AOC, Army Unit, and JBI: *weapon state frames* which include information about the UAVs (such as location,

Table 1. Potential Insertion Points for QoS Management in JBI

Insertion Point	Platform Dependent	Type of activity	Notes
Between platform components	Yes	Remote method calls between components. Either side of call can control when interaction occurs	May only be applicable in distributed platform implementations. High level of platform dependence may be unattractive.
Between client and platform (metadata)	No	Pub/Sub/Query. Timing is controlled by publishers.	Placing the QoS in the metadata can cause problems if an information object needs to be delivered in different forms to different subscribers. Also, it is meaningless to put infrastructure QoS information in the metadata because the metadata is about the object and not about the infrastructure. So this applies to information QoS only.
Between client and platform (<i>Connection</i>)	Yes	Pub/Sub/Query	Greatest effect on infrastructure QoS.
Within a client (e.g., CPU/bandwidth tradeoff in fractal compression)	No—Done outside JBI	Client keeps information about its environment to determine what information to publish	Largely outside the scope of JBI, but with possible payoff to JBI.
Between clients (e.g., publish-on-demand or back channel)	No, but client dependent	Any kind of client-to-client activity	Potential for significant improvement in information quality.
At the network layer	No—Done outside JBI	Any network activity.	Largely unaware of the importance the end user places in the information.

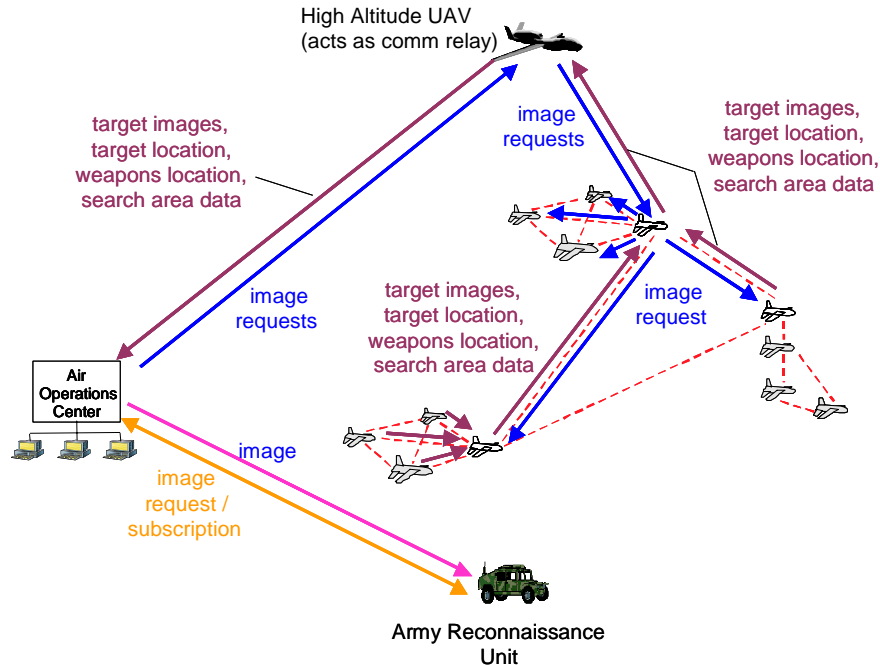


Figure 6. Scenario for the proof-of-concept demonstration

heading, and speed); *target state frames* which include information about targets detected by the UAVs (such as location); and *imagery* collected by the UAVs of the area that they are searching. Weapon and State Frame data are smaller, but higher rate, than the surveillance imagery, which has larger amounts of data but is only published when it is needed (as described below).

As part of the demonstration system, and in order to explore QoS in a JBI context, BBN developed a prototype JBI implementation. The platform was built using CORBA and an adaptation of the draft IDL CAPI specification to allow clients to be developed in different programming languages. The platform implementation provides the basic publish and subscribe functionality, plus an additional *publish on demand* capability. The Information Object metadata is represented using a text representation of variable binding statements (e.g., “variable = value”) instead of XML, and subscription predicates are represented as boolean expressions over the metadata variables (e.g., “variable > value”). The platform implementation language, Python, has built-in support for dynamic evaluation of the bindings and predicates, which greatly reduced implementation time, and retrofitting XML support will require little change to the core platform.

The JBI prototype includes an adapter that publishes imagery from the UAV application to the JBI and a simulated Army Unit that subscribes to imagery for its location with certain QoS requirements. The

goal is to have the UAV/JBI adapter request the transfer of imagery from the appropriate UAV with the Army Unit’s desired QoS. The UAV application then manages the imagery transmission to meet the requirements, if possible, and the results are published to the JBI where the Army Unit receives them through its subscription.

Unfortunately, the basic publish and subscribe operations do not provide the communication channel necessary for a publisher to discover that there are subscribers interested in information that it can offer. To deal with this and similar issues, we added a publish on demand feature to the JBI platform. This required one API extension and some additional platform behavior; no additional hooks into the transport layer were required. The `PublisherSequence` interface was extended to allow the publisher to register an offer to publish. The new method takes a predicate describing the range of offered objects and a callback object that is invoked when a subscription matches the offer.

Internally, registering an offer results in a subscription to request objects for a particular object type. These request objects are created automatically by the platform in response to client subscriptions and include the subscriber’s predicate and sequence QoS attributes. When a request object is received at the publisher (or its agent), the platform tests the subscriber’s predicate against the publisher’s offer predicate and, if they match, invokes the publisher’s callback with a set of variable constraints describing the intersection of the

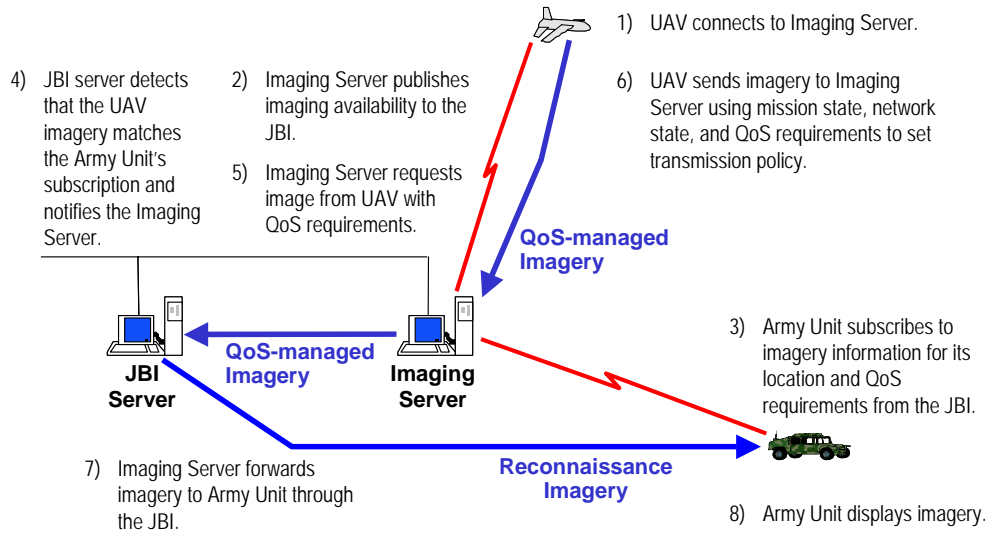


Figure 7. Image Application Data Flow in the Prototype System

predicates and any additional QoS attributes. The publisher can then examine the constraints to see if they can be met, and if so, generate and publish a corresponding Information Object.

For the predicate matching to be effective, the publisher and subscriber need to agree on how to specify QoS and QoI for objects of interest. In the prototype, attributes relating to the data itself, such as size and quality, were included in the metadata and predicate, while information about the timeliness of the data delivery was provided via the subscriber's sequence attributes. Future work will need to explore how to add QoS extensions to the XML schemas in the Metadata Repository.

Figure 7 illustrates the sequence of information flows involved in the publish-on-demand interchange of imagery information in the prototype system. A UAV node publishes information to the JBI indicating its location and the availability of imagery data it has gathered for that location (this is meta-data indicating the location and availability of imagery data; the imagery data is not published yet). Meanwhile, a simulated army unit subscribes to imagery information about the location through which it is moving, including the QoS requirements corresponding to the use of the imagery (e.g., size, resolution). As publishing UAV nodes move over the location occupied by the army unit, the JBI server matches the locations in the UAV publication and the army unit subscription and notifies an imaging server. The imaging server requests imagery from the UAV flying over the army unit location, including the consumer's (army unit's) QoS requirements in the request. The UAV supplies the imagery in the form requested by the army unit and sets

the network priority bits to affect the resources used for delivery. The imaging server forwards the imagery to the army unit.

Notice that this prototype uses QuO and JBI middleware to maintain the decoupling of the UAV imagery publishing from the army unit imagery consumption; to coordinate data and network QoS management to get end-to-end QoS for the image delivery; and to combine priority based techniques with data management to mediate the relative importance of the various information – imagery, weapon state frames, and target state frames. It accomplishes this while providing the imagery data to the “right person” (the army unit) in the “right form” (according to the army unit's requirements) at the “right time” (when the UAV is over the army unit's location).

6. Conclusions

This paper makes a down payment on investigating many of the issues involved in transporting data with stringent QoS requirements via a JBI platform. There are many more issues raised than solved with this work, and additional areas to explore. Foremost among these is further definition and resolution of the meaning of QoS in the JBI definition and how data can be provided with the right quality to consumers when the producers, consumers, and infrastructure are decoupled.

The current JBI definition is still evolving with regard to the QoS needs of many mission-critical military operations. As the definition of the JBI matures, existing features become more firmly defined, and new features emerge, it's important to identify how they

can be used to specify, control, and manage necessary QoS requirements. We have identified three strategies – publish on demand, back channels, and force templates – that can provide QoS for some classes of JBI applications. Middleware-based QoS management, which is being successfully applied to the QoS needs of large classes of military systems, is the right place to look for research that can apply to making the JBI platform middleware more capable for providing, mediating, and managing the QoS demands of a larger class of systems.

Many military operations, such as the detection, tracking, and prosecution of time-critical targets, rely on timely information that is much less useful if it arrives at a consumer too late or in the wrong form. JBI implementations need sufficient control over their infrastructure to provide data with needed QoS for current systems and for the systems that they might be tasked to support in the future.

QoS support in JBI needs sufficient control and management of the following:

- The source of information,
- The infrastructure for transporting information,
- The user of the information,
- The competing demands of multiple information exchanges, and
- Dynamically changing environments, e.g., requirements, mission modes, participants, and resource availability and usage.

The path to providing QoS support in JBI should start by reviewing the space of mission-critical military use cases; their QoS needs; and ongoing research in QoS for distributed embedded military systems. Then enhancements and refinements of the JBI definition to include QoS management as a first-class concern can be proposed, designed, and implemented. This paper takes a first step along this path.

References

- [1] Matthew Brzezinski, “The Unmanned Army”, The New York Times, April 20, 2003.
- [2] http://ceaspub.eas.asu.edu/Ye_Activities/cip_project.html
- [3] Fraunhofer Institute FOKUS. Quality of Service for CORBA Components RFP Initial Submission. Version 1.1, March 29, 2004.
- [4] IETF, An Architecture for Differentiated Services, <http://www.ietf.org/rfc/rfc2475.txt>
- [5] <http://itua.bbn.com>
- [6] JBI Common API, www.infospherics.org
- [7] TF Lawrence. “Quality of Service (QoS) A Model for Information,” *Fourth International Workshop on Object-Oriented Real-Time Dependable Systems*, Santa Barbara, California, January, 1999.
- [8] Joseph Loyall. “Emerging Trends in Adaptive Middleware and its Application to Distributed Real-time Embedded Systems,” *Third International Conference on Embedded Software (EMSOFT 2003)*, Philadelphia, Pennsylvania, October 13-15, 2003.
- [9] Robert E. Marmelstein. “Force Templates: A Blueprint for Coalition Interaction within an Infosphere,” *IEEE Intelligent Systems*, May/June 2002.
- [10] James Milligan and James Hendler. “JBI Fuselet Definition Document”, Draft—February 26, 2003.
- [11] Partha Pal. “Designing Protection and Adaptation into a Survivability Architecture: Demonstration and Validation (DPASA-DV); DPASA Survivability Architecture: Final Design,” July 10, 2003.
- [12] Quality Objects middleware, quo.bbn.com
- [13] QoS Enabled Distributed Objects, qedo.berlios.de
- [14] Pete Sholander, Glenn Frank, Sean Swank, Joseph Loyall, and Gary Duzan. “Multi-Layer, Mission-Aware QoS Management Techniques for IP Applications in a Joint Battlespace Infosphere,” *Military Communications Conference (MILCOM)*, Monterey, California, October 31–November 3, 2004.
- [15] United States Air Force Scientific Advisory Board Report on Technology Options to Leverage Aerospace Power in Operations Other Than Conventional War, Volume 1: Summary, SAB-TR-99-01, February 2000.
- [16] United States Air Force Scientific Advisory Board Report on Building the Joint Battlespace Infosphere, Volume 1: Summary, SAB-TR-99-02, December 17, 1999, Cleared for Open Publication February 2000.
- [17] T Wheeler, et al. “Application of QoS-Driven Adaptive Computing,” *19th IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.