QoS Enabled Dissemination of Managed Information Objects in a Publish-Subscribe-Query Information Broker

Joseph P. Loyall[a], Marco Carvalho[b], Andrew Martignoni III[c], Douglas Schmidt[d], Asher Sinclair[e], Matthew Gillen[a], James Edmondson[d], Larry Bunch[b], David Corman[c]
[a]BBN Technologies, 10 Moulton Street, Cambridge, MA 02138
[b]Institute for Human & Machine Cognition, 40 South Alcaniz Street, Pensacola, FL 32502
[c]The Boeing Company, P.O. Box 516, St. Louis, MO 63166
[d]Vanderbilt University, VU Station B 351824, Nashville, TN 37235
[e]US Air Force Research Laboratory, 525 Brooks Rd, Rome, NY 13441

ABSTRACT

Net-centric information spaces have become a necessary concept to support information exchange for tactical warfighting missions using a publish-subscribe-query paradigm. To support dynamic, mission-critical and time-critical operations, information spaces require quality of service (QoS)-enabled dissemination (QED) of information. This paper describes the results of research we are conducting to provide QED information exchange in tactical environments. We have developed a prototype QoS-enabled publish-subscribe-query information broker that provides timely delivery of information needed by tactical warfighters in mobile scenarios with time-critical emergent targets. This broker enables tailoring and prioritizing of information based on mission needs and responds rapidly to priority shifts and unfolding situations. This paper describes the QED architecture, prototype implementation, testing infrastructure, and empirical evaluations we have conducted based on our prototype.
Keywords: Quality of service, information management

1. INTRODUCTION

Information management services have emerged as necessary concepts for information exchange in net-centric operations, growing out of the Joint Battlespace Infosphere (JBI) [10], [11], [12], a US Air Force initiative supporting net-centric warfare concepts. JBI is related to other net-centric warfare initiatives, including Net-Centric Enterprise Systems (NCES) [6], a set of services enabling access to and use of the Global Information Grid (GIG) [7]. The JBI defines an active information management model, in which clients are information publishers and consumers, communicating anonymously with other clients via shared information management services (IMS), such as publication, discovery, brokering, archiving, and querying [5]. Published information is in the form of typed managed information objects (MIOs) consisting of payload and XML metadata describing the object and its payload. Consumers make requests for future (subscription) or past (query) information using predicates, e.g., via XPath [16] or XQuery [17], over MIO types and metadata values. The information management services include *brokering* (i.e., matching MIOs to subscriptions), *archiving* of MIOs, *querying* for archived objects, and *dissemination* of MIOs to subscribing and querying clients.

While information management services have become necessary concepts to support net-centric operations, the decoupled information management and publish-subscribe-query information exchange they provide is not sufficient for dynamic, mission-critical and time-critical operations. Information management services require quality of service (QoS) enabled dissemination (QED) of information. The aim of QED information dissemination is to meet the various quality requirements of users and the missions they are undertaking in a manner that is reliable, real-time, and resilient to the changing, hostile conditions of tactical environments.

To motivate the need for QoS-enabled information management, consider a scenario involving a convoy of troops on patrol in an urban area, as shown in Figure 1. The troops develop their initial route and create subscriptions and queries for information based on their mission and route. The IMS handles the requests by delivering relevant information, both historical and real-time, to the troops, providing situation awareness (SA) and understanding.

This research has been sponsored by the United States Air Force Research Laboratory under contract FA8750-08-C-0022.

As the troops travel along their route, the situation evolves and dynamic events cause urgent changes in user needs. For example, a UAV may publish data that shows the planned route is blocked by a stalled truck. Upon receipt of this information by the troops, it triggers them to replan their route and adjust their subscriptions for information along the new route.

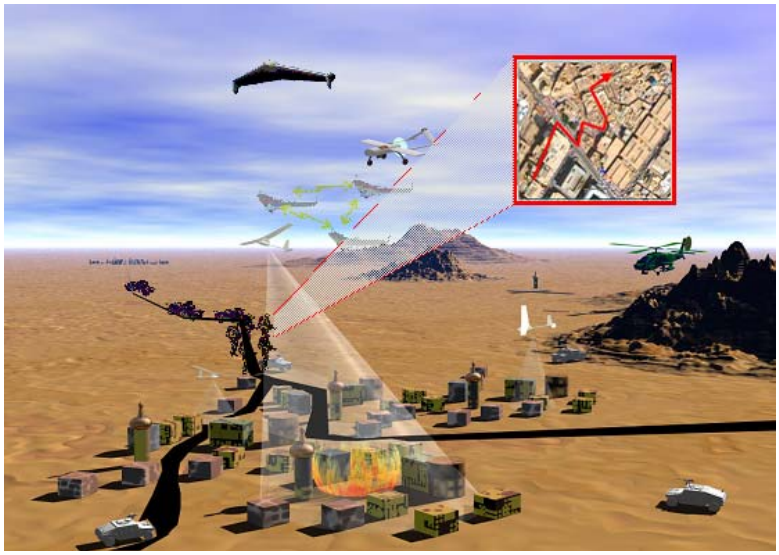


Figure 1. A Motivating Scenario for QoS-Enabled Dissemination (QED)

As the troops advance, the IMS delivers information to the troops of a threat ahead that will jeopardize their on-going operations. Command and Control (C2) publishes orders to provide close air support (CAS) to eliminate the threat. The CAS information and subsequent battle damage imagery are published to the IMS and received by subscribers including C2 and the troops on patrol. To users, it appears that their information needs are being fulfilled by a highly responsive, mission-aware information system. When highly detailed imagery is needed, it appears. When rapid SA updates are important, they arrive with requisite detail and speed.

This scenario is only possible with the capabilities being developed for QED. QED combines technologies addressing mission-based QoS management, aggregate QoS management, resource management, QoS policy, information dissemination mechanisms, QoS monitoring, and disruption tolerance. The QED software system we are developing includes the following capabilities:

- Providing timely delivery of information needed by tactical users in mobile scenarios
- Tailoring and prioritizing information based on mission needs and importance
- Responding rapidly to priority shifts and unfolding situations
- Operating in a manner that is robust to failures and intermittent communications.

This paper describes the QED QoS-enabled information broker we are developing, building upon the *Apollo* [1] IMS baseline. Apollo is a set of information management services developed by the US Air Force Research Laboratory as a JBI reference implementation. Section 2 describes the requirements for QoS management in IMS. Section 3 describes the QED architecture. Section 4 describes the design of individual QED services and components. Section 5 describes the current state of the prototype QED software system and experimental results describing its operation and performance. Finally, Section 6 presents concluding remarks.

2. REQUIREMENTS FOR QOS IN INFORMATION MANAGEMENT SERVICES

The ideal QoS support for IMS would consist of information that leaves a publisher or archive and reaches every consumer client *immediately, completely*, and consistently every time, and that would respond to every request with the smallest amount of information *exactly matching* what the user needs. Of course, the ideal QoS is unachievable. The processing of the brokering and retrieval functions and the delivery over networks introduce *delay* into information dissemination. Resource failures and overload situations can cause information *loss*. Variations in the time to process requests, burstiness of client traffic, and competition with other processes can introduce variation into the system performance (*jitter*). Client requests for information do not always capture exactly the best qualities that they can use, and the attempt to capture these qualities frequently reduces the probability that they will match any published objects at all. Moreover, clients' demands on the information management services can come into conflict, making it impossible to provide high QoS to one without reducing QoS to others.

QoS management services such as QED have to manage the tradeoffs involved in providing higher aggregate levels of QoS across the users of information management services. Higher criticality operations should be provided preferential service. Loss and delay should be introduced where each can best be tolerated and the choice of which to introduce (when both are unavoidable) should depend on which is better tolerated. The pursuit of higher QoS should not introduce thrashing, i.e., higher but unsustainable QoS levels are not necessarily better than slightly degraded but consistent quali-

ty. The best information matches should be preferred, sometimes even over more information matches. Moreover, in those situations where not all information can be delivered, the information delivered should be the most important information to the most important users with respect to the overall mission being performed.

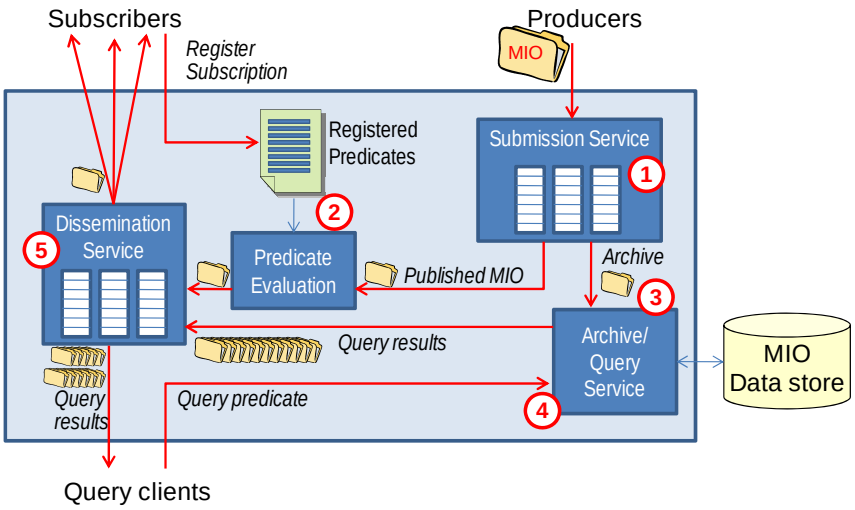


Figure 2. Control Points in the Information Brokering Server

The following are aspects of perceived quality and the way information management services such as QED can affect them:

- *Timeliness* – The speed at which brokering happens or the latency through the information management system. For example, when an information object is published, how fast it reaches subscribers. When a query is issued, how fast responses reach the querying client. In general, the greater the timeliness (and conversely, the lower the latency), the better the perceived QoS.
- *Completeness* – The number of information objects that reach requesters from those available (i.e., published into the information management system) that are relevant (i.e., that match requests).
- *Fidelity* – A measure of the amount of information in each individual information object. Fidelity concerns whether a requester receives the entire amount (metadata and payload) of a published information object.
- *Accuracy* – A measure of the correctness of information delivered, i.e., whether information objects delivered to a requesting client have any errors introduced into them (e.g., during transformation, brokering, or other operations).
- *Smoothness* – The predictability of performance and consistent latency. In many cases, perceived QoS is higher if a user receives a consistent and expected quality than if high quality is interspersed with low quality, to the point where there is a wide variation and a user cannot know what to expect.
- *Suitability* – The better a response matches a user's needs, the higher the perceived QoS. This means that higher resolution and higher precision information objects are generally recognized as higher QoS than lower resolution or precision. Other characteristics of information objects, such as source, currency, content, trust, format, and so forth, can make them more or less suitable for a given request and therefore affect their perceived QoS.

Figure 2 shows the set of core information management services for net-centric operations in a non-QED baseline IMS such as Apollo [1]. These serve as *control points*, i.e., places at which QoS can be affected. Table 1 describes each control point and the ways in which it can affect QoS.

There are no operations in the core information management services that specifically affect the fidelity of individual MIOs. The JBI definition includes fuselets [4], information transformation services that live in the JBI core, which can change the fidelity of MIOs, but addressing them is an area for future research. Likewise, there is nothing in the information management services that affects the accuracy or suitability of MIOS, although certainly errors in MIOs could be introduced accidentally as the result of errors in implementation. Moreover, suitability can be affected by the default semantics of the current implementation. Specifically, the Apollo IMS returns MIOs in the order in which they are published into the system and returned by the database, even if they are less suitable than something else currently queued for brokering or dissemination.

Table 1. Resources Used and QoS Affected at Control Points in the Information Brokering Server

Control Point	Description	Resources Used	QoS affected
1. Submission Service	Receiving information objects entering the system as the result of publishing.	Network bandwidth; memory to store objects until they are processed.	*Timeliness, completeness:* If the rate, number, and size of published objects exceed the bandwidth capacity it will introduce delay or loss; If the rate of published objects exceeds the rate at which they can be processed, delay will be introduced as objects are enqueued awaiting processing.
2. Brokering	Predicate evaluation to match published MIOs with registered subscriptions.	CPU	*Timeliness, smoothness:* Introduces latency; Calls to the predicate evaluator can take differing amounts of time depending on the size and complexity of the metadata and predicates, introducing jitter.
3. Archiving	Insertion of a published MIO into the Information Object Repository (IOR)	CPU to process archive; Disk space to store MIO.	*Timeliness:* Introduces latency
4. Query processing	Evaluation of a query operation and subsequent retrieval of results.	CPU to process the query operation; Memory to store the results (potentially many)	*Timeliness, smoothness:* Introduces latency; Queries can take differing amounts of time and result in different size result sets, introducing jitter.
5. Dissemination	• Delivery of the results of brokering (a single MIO) to matched clients (potentially many) • Delivery of the results of a query (potentially many MIOs) to the requestor (a single querying client).	Memory to store the MIOs being delivered; Bandwidth to deliver the MIOs	*Timeliness, completeness, smoothness:* If the rate, number, and size of MIOs exceed the amount of bandwidth capacity available to send them, delay or loss will be introduced. Since MIOs will vary in size, they will take different amounts of time and bandwidth to send, introducing jitter.

3. OVERVIEW OF THE QED ARCHITECTURE

The QED architecture includes components and services that enforce QoS at each local control point, under management of an aggregate QoS manager that can mediate conflicting demands and enforce system-wide QoS policies. As shown in Figure 3, the QED architecture consists of the following components:

- *QoS administration* – Allows the specification of mission-level policies to set the relative importance of clients and MIOs and specify the aspects and limits of QoS desired by clients.
- *Information space QoS manager (ISQM)* – Serves as an aggregate policy decision point, interpreting QoS policies in terms of the collection of users and the tasks that can be performed (archiving, predicate evaluation, dissemination, etc.). Schedules the tasks and creates local policies (for each control point) and distributes policies to the LQMs.
- *Local QoS manager (LQM)* – Enforces local QoS policies (created and distributed by the ISQM) at each control point, i.e., incoming and outgoing queues, predicate evaluator, and the archiving and query evaluation processes.
- *QoS monitoring* – Gathers information useful for QoS decision making, enforcement, and visualization. This includes information about resource usage and availability, application and data characteristics, and delivered QoS.
- *QoS mechanisms* – Enforcement mechanisms used at each control point. These include differentiated queues for prioritizing tasks and MIOs, ordering and control of predicate evaluation, archiving, query evaluation, and information shaping, such as compression or filtering.
- *Transport protocol* – Serves as the network endpoint, enabling information objects and requests to be sent from the server to clients or from clients to the server, with control over prioritization, reliability, replacement, and ordering.

4. DESIGN OF THE QED COMPONENTS

This section describes the design of the components in the QED architecture in more detail.

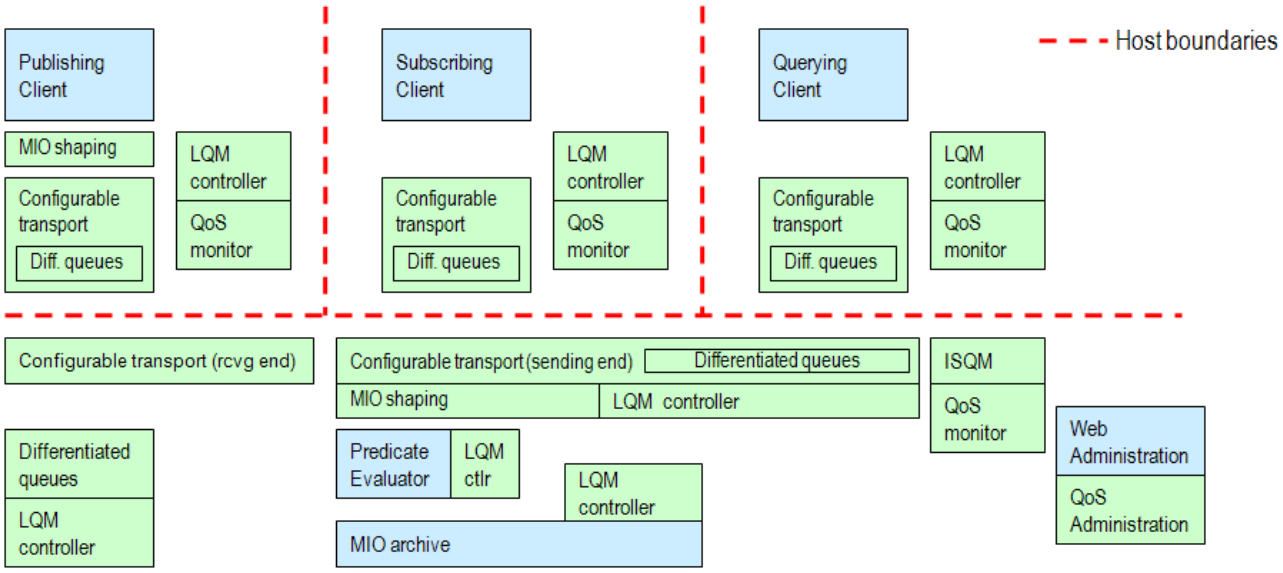


Figure 3. The QED Architecture Includes Components for Aggregate and Local QoS Management, QoS Administration, QoS Monitoring, and QoS Enforcement

4.1 QoS Administration and Mission Management

The QoS Administration service of QED provides a graphical user interface to define client preferences, rules, and other mission-level information useful for managing QoS. The interface enables access to change QoS behavior during run-time, to dynamically adjust to the changing mission parameters. The following are the main goals of the QoS Administration interface:

- *Observability* – The ability to view what is happening in the system using the tools provided, including visibility into delivered QoS provided by the QoS monitoring service (Section 4.4).
- *Understandability* – The system can be understood in the function it performs. Systems that relate to the real world more than a computer language tend to be more understandable.
- *Expressive power* – The flexibility in selecting behavior, based on the system's language to communicate preferences. This language should be as expressive as possible while meeting other goals.
- *Ease of use* – The system is easy to use without extensive training. This involves choosing interfaces and presentation styles with which the users are already familiar.
- *Robustness* – The ability to handle any sequence of actions on the part of the administrator and clients.

The QoS Administration capability enables customization of a set of rules that contain the mission- and client-specific configuration information to make the necessary QoS management decisions. These rules may be added or changed dynamically throughout the lifetime of the system to meet the changing needs of the tactical environment. The rule editing component is extensible so that new rule templates can be defined and added to the set provided. The interface allows the entry of high-level QoS requirements and constraints for mission elements, providing reasonable defaults for data not specifically mentioned in a rule.

QoS Administration also provides visibility into the runtime behavior of the system by displaying information provided by the QoS Monitoring component. By combining and displaying relevant mission and QoS information the administrator can keep track of events to handle, such as extreme overload or doctrine violations.

4.2 QoS Management and Policies

The ISQM translates the mission management information from the QoS Administration service into a set of actionable policies that are enforced by the LQM at each control point in the IMS. While the ISQM provides direction to each LQM, many LQM decisions are parameterized on runtime information such as the size of objects; the expected time it will take to process an object or request; the available bandwidth, threads, or memory; or the size of queues. Therefore,

LQMs make decisions using lookup tables or utility functions, which compute the relative benefit of particular enforcement options within the limits imposed by the ISQM supplied policy. QED utilizes the KAoS Policy and Doman Services [14] for policy management, including the following:

- Policy Representation – the underlying formal and extensible language based on the W3C standard Web Ontology Language (OWL) [15] for representing the rules and preferences defined by the QoS Administration service.
- Policy Distribution – distribution of applicable policies and IMS state information to the local policy decision point associated with each of the QoS Mechanisms described in the next section.
- Policy Reasoning – the automated reasoning necessary to determine the applicable policies at each control point based on the current context.

Each QoS policy associates an action in the IMS with the aspects of QoS required for the action and the relative importance of performing the action.

QoS Policy: <IMS Action, MIO Attributes, Clients, IMS State> → <Importance, QoS Preferences>

The antecedent of each policy describes the applicable context for the policy in terms of the IMS action being performed (e.g., Publish, Archive), the observable attributes of the object (MIO) being processed (e.g., type, size), the identity of the clients involved in the action (e.g., publisher, subscriber), and the current state of the IMS as exposed through the QoS Monitoring component.

The KAoS domain services enable the definition of roles and groups to which clients may be assigned at runtime, thus enabling policies to be pre-defined for classes of clients which are resolved at runtime based on the assignment of clients to such roles and groups. The use of OWL for policy representation similarly enables the definition of class hierarchies for each antecedent term. This allows the definition of policies at an appropriate level of abstraction and helps prevent having to specify a policy for every combination of system conditions. A key enabler for this is the ability to set precedence among policies to override one (typically more general) policy with another (typically more specific) one. For example, the default policy may establish that performing the Publish action for Surveillance MIOs has Low importance, but this policy can be overridden by a policy stating that Publishing Surveillance for clients in a particular group or mission are of High importance.

The consequent of each policy establishes either 1) the relative importance of performing IMS actions that match the policy conditions at a given control point or 2) the QoS preferences describing tradeoffs to be made when service is degraded (e.g., delay before drop). QoS preferences may also establish limits for service degradation (e.g., a deadline after which information is not useful). Actions that are assigned higher importance levels are given precedence over lower importance actions. The importance levels and preferences are enforced at each control point by the LQM and QoS mechanisms described in Section 4.3. As the IMS becomes overloaded the lowest importance processing contexts are degraded first, according to the QoS preferences (e.g., degrade timeliness before completeness). In this way, Importance determines *when* service will be degraded and QoS Preferences determine *how* QoS will be degraded.

4.3 QoS Mechanisms

QED includes QoS mechanisms that are specific to the control points presented in Table 1 and the ways in which QoS can be affected at those points. QED's QoS mechanisms and how they provide control at each control point are described below:

- *Differentiated queues* at the submission and dissemination services (control points 1, 3, and 5) to prioritize incoming and outgoing MIOs according to their relative importance and cost (the time to process or amount of bandwidth needed).
- *Preemptive brokering* – Registered predicates are grouped by type and importance, e.g., based on the client's importance relative to other clients. When an object is ready for brokering (control point 2), the MIO is matched against each set in order of importance. After the completion of each set, the MIO is forwarded for dissemination to any clients that it matches and the predicate matching can be preempted in favor of processing more important MIOs that have arrived in the system. This results in faster servicing of important clients' subscriptions (they don't have to wait for all predicates to be processed), important MIOs are not blocked as long waiting for a previous MIO of lesser importance to be processed, and no additional memory is used (a reference to the MIO – stored in the heap – is all that is transferred to the dissemination service.

- *Preemptive queries* – The underlying information broker utilizes a *cursor-based* query, in which the initial query returns an estimate of the number of results and the results are retrieved by a subsequent sequence of *getNextResultSet* calls, each of which can be scheduled separately by the task scheduler (control point 4, see task scheduling below).
- *MIO shaping* – When there are more MIOs awaiting dissemination (enqueued) than can be sent immediately, but there are extra CPU cycles, the MIOs can be shaped reducing the amount of bandwidth and time needed to send them (control point 5). MIO shaping is type specific and based on client preferences (such as the formats that are acceptable and the ranges of MIO size, resolution, and fidelity requested) and can include compression, cropping, scaling, and content filtering.
- *Task scheduling* – Control points 2, 3, and 4 represent potentially CPU intensive tasks. The baseline information broker services them from a common thread pool in FIFO manner, with no preemption. To support better control, prevent locking up the CPU while higher priority tasks are waiting, and to support the preemption mentioned above for brokering and queries, we have designed a priority based task scheduler, as shown in Figure 4. Publish and query events in the system create tasks that are enqueued. The threadpool manager selects the highest priority task to execute when a thread is available (priority is based on simple importance, weighted importance, or a benefit/cost ratio. Tasks can be preempted if an urgent task is created and there are no threads available. The monitoring capability described in Section 4.4 is used to measure the time needed to execute tasks with particular characteristics to improve task execution cost estimates. Creating these tasks also allows us to include client preferences, such as deadlines, into the QoS management. If a task does not complete before the deadline passes, the thread can be halted and returned to the thread pool.

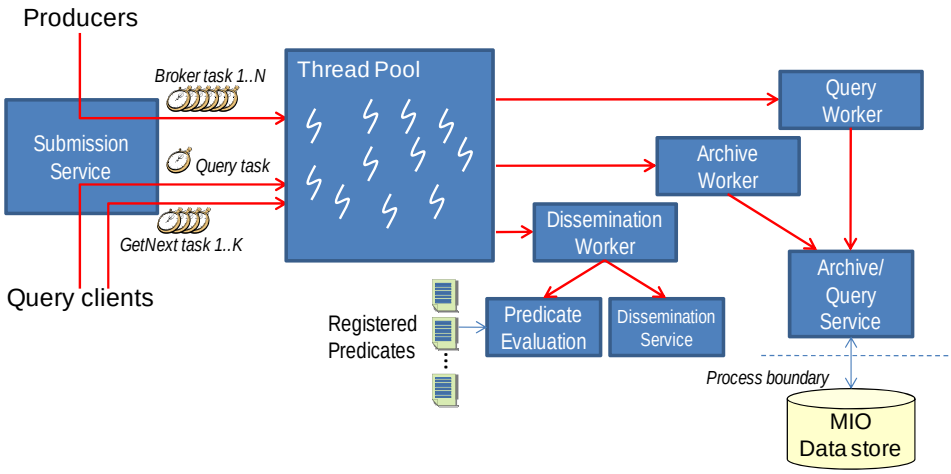


Figure 4. Task Scheduling for CPU Intensive Tasks in QED

4.4 QoS Monitoring

The QoS monitoring service provides a common infrastructure-wide mechanism to store and retrieve system and application metrics useful for QoS decision making and visualization. The server-side QoS monitoring is deployed as a service inside the IMS providing an API accessible from the other services. On the client-side, the monitoring is a Java library loaded as needed to monitor conditions on the client's node.

QoS monitoring relies on three main components shown in Figure 5: the XLayer [3], the XLayer Java Proxy, and the Monitoring service. The XLayer is a cross-layer communications substrate that, among other things, monitors resource conditions. The XLayer Java Proxy allows Java code to access the XLayer substrate functionalities. The Monitoring service provides the API to access stored metrics. Each time-varying metric is stored as a short time series so it can be reported (via the API) by its average value, variance, last value, or trend.

The XLayer has a set of built-in metrics (e.g., CPU and memory utilization, network utilization per interface) which are always available through the monitoring service. In addition to that, the monitoring service provides an interface that enables the development of customized *moni-*

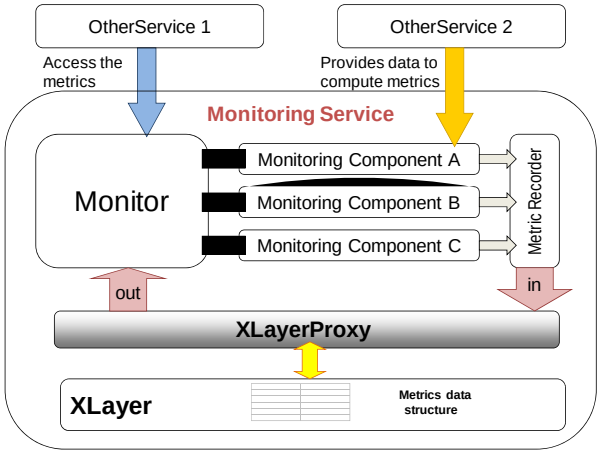


Figure 5. The QED Monitoring Service Architecture

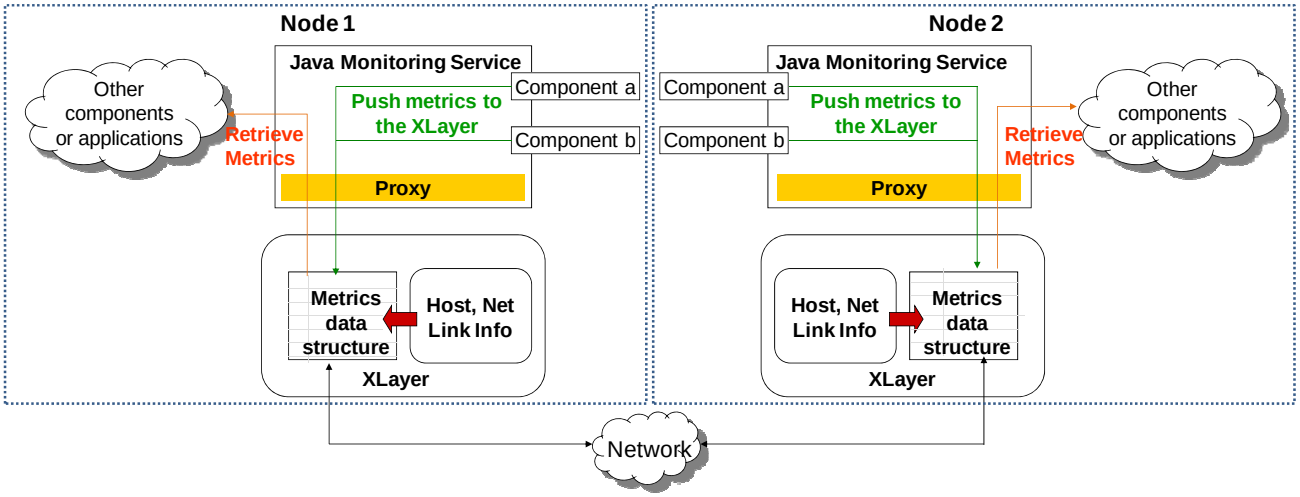


Figure 6. Metrics Propagation across the Network Provided by the XLayer

toring components. A customized monitoring component can be seen as a sensor for a particular metric or set of metrics. When a monitoring component is instantiated it registers itself as a provider for one or more metrics. Other services (or parts of an application) can then be instrumented to get an instance of a specific component and provide values for the registered metrics.

The monitoring service provides two types of subscriptions for metric monitoring, both handled at the level of the cross-layer substrate. The first method is defined as an ongoing subscription, in which the subscriber is notified every time a metric changes. The second is the 'one-time' subscription, which takes an optional threshold as an argument. In this type of subscription, the subscriber is notified the first time (and only the first time) a metric is updated and whenever the new value (or the timed-average) is outside the specified threshold.

An important feature implemented in the QoS monitoring is efficient sharing of statistics between nodes across the network. Each node running the Monitoring service has the ability to query or subscribe for metrics related to the other nodes. The XLayer will take care of the propagation of the metrics between different hosts (e.g., publishers, subscribers and IMS in our case) as shown in Figure 6.

4.5 QoS Transport

The transport capability supports the transmission and reception of control and data between clients and the IMS. The primary goal of the transport protocol is to maximize the timeliness of delivery of data, while minimizing the overhead in providing this delivery capability. The design of the QoS transport is based on the Mockets library [13], which provides reliable and/or sequenced delivery of data, as required by the semantics of the communication. Information and statistics available at the transport layer are exported to the QED monitoring service (described in Section 4.4). The transport layer also provides flexible control interfaces to allow the other components to customize the behavior and operation of the transport layer. For example, the transport layer allows readjustment of priorities of messages, deletion of messages, or replacement of messages even after the messages have been handed off to the transport layer, but are still awaiting transmission or acknowledgement.

The experiment described in Section 5.2 exploits the capability of message replacement, which is a useful capability to address bandwidth constrained links. Under conditions of low bandwidth, data tends to accumulate in queues awaiting transmission, which increases the end to end latency. For periodic data such as position updates (for example, blue force tracking), more recent data invalidates old data. Message replacement is a way to override older data in the queue with newer data, improving the timeliness of the delivery of the data.

Mockets also support the notion of message tagging, which allows an application to differentiate between different types of traffic, e.g., each type of MIO might be assigned a different tag. Alternatively, each subscription from a client can be assigned a different tag. Tagging data allows Mockets to maintain detailed statistics about each type of data (or each subscription), set independent priorities for each type of data, and control bandwidth utilized for each type of data.

5. DESCRIPTION OF QED PROTOTYPE

We have implemented a prototype of the QED design presented in Section 4. This section describes the prototype and presents results of experiments we have conducted using the prototype.

5.1 Description of Current Prototype

The current QED prototype was developed to enhance the US Air Force Research Laboratory Apollo 1.0 IMS [1], which is developed using Java and JBoss [9]. Our QED prototype extends the Apollo baseline by modifying existing Apollo services, adding new services and interfaces, and providing clients of the system with new protocol options. This prototype includes an initial QoS administration interface, aggregate and local QoS managers, differentiated queues for the submission and dissemination services, and client side monitoring of QoS behavior. The following paragraphs describe each of these components.

The prototype administration interface is shown in Figure 7. We extended Apollo's existing administrative interface to include a new section dedicated to QoS management. Apollo's administrative interface is built using Struts [2], an open source Web Services framework, within the JBoss server, and so we followed suit. The administrative interface supports a simple notion of "workflows", which are related groupings of clients and MIOs, defined *a priori* via configuration files. The administration interface provides a means to order the workflows in terms of importance and set an overall policy ("Strict" or "Weighted Fair"). In the *Strict* policy the object of the highest importance is always served, though this can starve other objects (i.e., they never get serviced) if the rate of publication of high importance objects is greater than the rate at which the information broker can service and disseminate them. In the *Weighted Fair* policy, all objects (eventually) get serviced, but higher importance objects get serviced more frequently. The weights attached to importance levels determine the ratio at which objects of each importance level are serviced.

The administration interface updates the policy in the ISQM, which interprets the high-level policies, translates them into an intermediate form, and distributes them to the LQMs. The QED prototype performs a straightforward mapping of importance into priorities for the LQMs and distributes the policies using the Java Message Service (JMS). Currently, the ISQM is a JMS publisher and the LQMs are JMS subscribers.

We have implemented two LQMs, each managing a mechanism that controls the behavior of the system. The first handles the submission queues. Once a published MIO reaches the server, it gets enqueued before the potentially CPU-intensive job of performing subscription matching. We implemented importance-based binned queues with a loop that pops off MIOs from the queue according to the overall policy (strict or weighted fair).

The other LQM we implemented manages a new protocol option for clients of the system. The baseline client-side libraries use standard J2EE mechanisms to interact with the server (RMI over TCP/IP, and in some cases JMS), i.e., all TCP-based. We added the option for the clients to use Mockets, described in Section 4.5, which uses a UDP-based protocol but provides options for ordered, reliable delivery (thus allowing it to be used as a drop-in replacement for TCP). Mockets supports the notion of priority within its own queues, allowing us to control the ordering of MIOs being delivered to an individual client (each subscriber has a single Mocket with its own independent queue). Mockets supports only the weighted fair policy.

All the QoS management we implemented results in client-visible differences from the baseline, so we implemented a subscriber-client to display client-visible metrics. Specifically, we display the end-to-

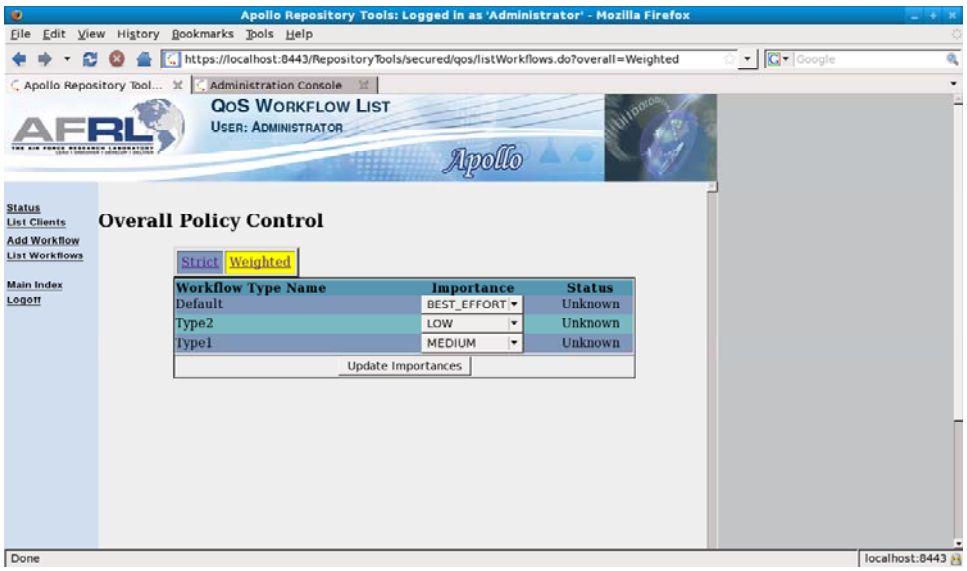


Figure 7. Screenshot of the QED QoS Administration Interface

end latency (of the last IO received) and the current rate (as an average over a 5-second window). We displayed these on a per-publisher basis, as shown in Figure 8. Each MIO contains an opaque publisher-ID as part of its metadata, which allows subscribers to differentiate the sources of MIOs.

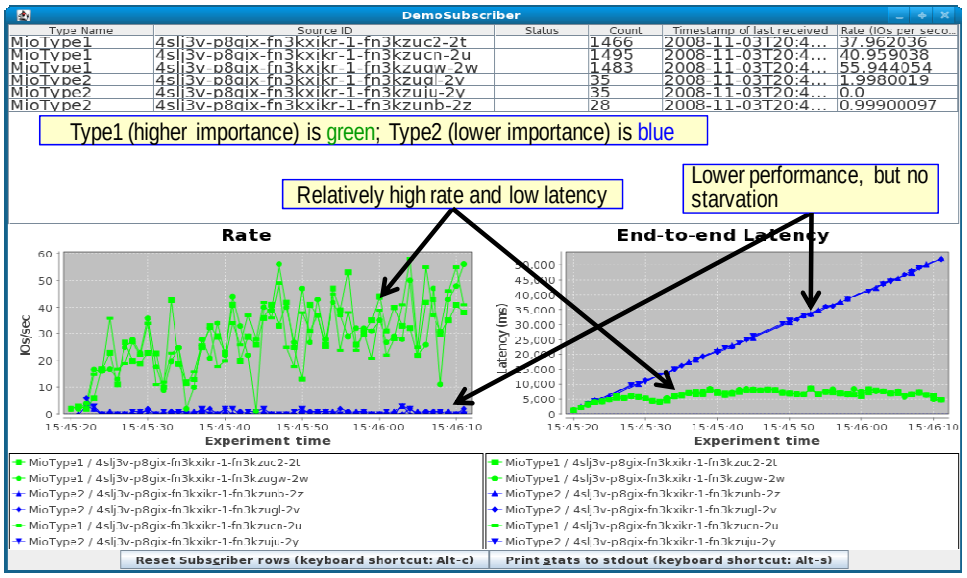


Figure 8. The QoS GUI Displays the Client-visible Behavior Under QED control, including Throughput and Latency

5.2 Experimental Results

The QED testing methodology involves three types of test scenarios. Class 1 scenarios are defined to evaluate specific features of QED under controlled situations. Class 2 scenarios test aggregate QoS behaviors, including scenarios of operationally relevant behaviors, but with fixed policies. Class 3 scenarios also test aggregate QoS behavior, but changes policy dynamically, and are used to test the speed and effectiveness of changing, disseminating, and enforcing dynamic QoS policies.

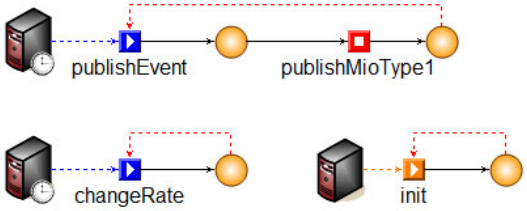


Figure 9. CUTS Model of a QED Test Scenario

Each scenario is created using the *Coworker Utilization Testing Suite* (CUTS) [8]. CUTS uses model-driven engineering tools and emulation techniques to benchmark and analyze the performance of deployments and configurations of distributed real-time and embedded systems, such as QED. The constraint-based CUTS tool enables the modeling of tests in all three classes. For example, Figure 9 shows a model of a Class 1 test with dynamic publishing rates. CUTS supports the rapid modeling of tests, automatic translation into Java code and project build files, and execution of the tests using a set of custom Perl scripts.

The experimental results presented below are for Class 1 scenarios evaluating the effectiveness of QED to manage CPU overload at the submission service, i.e., MIOs published into the information broker faster than the predicate evaluation service can process them, and bandwidth constraints at the dissemination service, i.e., insufficient bandwidth to keep up with the rate of MIOs needing to be sent to subscribers. A burn in period was used to allow the system to stabilize, and then results were recorded for several minutes. These results quantify the effectiveness of the differentiated submission queues, differentiated dissemination service, LQMs, and importance based policies to provide more predictable performance to important information in CPU and network overload situations when compared to the IMS baseline.

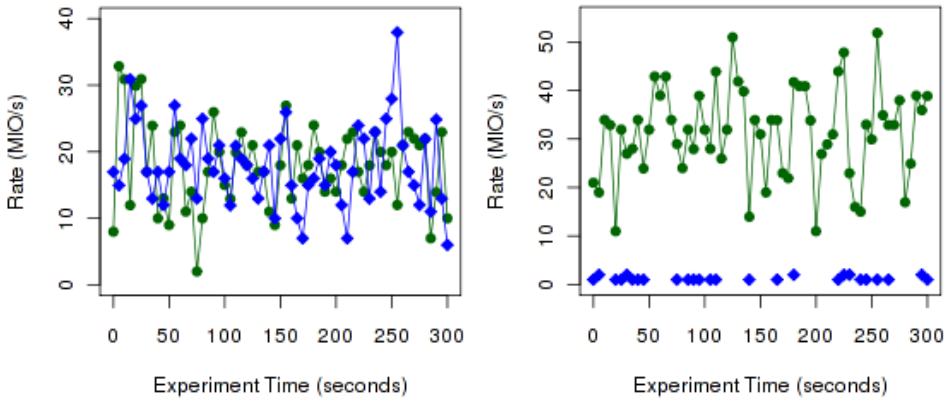
Overview of Test 1. The experimental configuration for the first test involved two workflows, each containing 40 subscribers and 3 publishers. Each workflow had a unique MIO type, which its publishers published as fast as they could. All subscribers had registered predicates, but only one subscriber in each workflow matched the workflow's published MIOs. This configuration resulted in the CPU being a bottleneck since the predicate evaluation service had to evaluate the full set of registered predicates for each MIO published, but only one would match, resulting in the MIO being disseminated to only one client.

In Test 1 the CPU cannot process MIOs as fast as they are entering the system, though there is sufficient bandwidth to deliver all the matched MIOs. We compared the receive rate (i.e., throughput) and latency of the workflows running on the baseline system (*Base,* Apollo with no priority), QED with both workflows having equal importance (*Equal*), QED with workflow 1 having higher importance than workflow 2 and weighted fair policy (*Weighted*), and QED with workflow 1 having higher importance than workflow 2 and using the strict policy (*Strict*).

Analysis of Test 1 results. Table 2 shows the results of running Test 1 for five minutes. QED provides significantly higher throughput (i.e., *receive rate*) to the more important workflow (Workflow 1), while providing the control to either starve (*strict* policy) or provide degraded service (*weighted fair* policy) to the less important workflow (Workflow 2). With equal priorities, the throughput of QED tracks the baseline Apollo closely. QED also provides significantly lower latency to the more important workflow, although the latency improvements of QED with equal importance over the Apollo baseline are due to running with different limits on queue sizes. Figure 10 shows graphs of one publisher from each workflow (a) with equal priorities and (b) with Workflow 1 having higher priority, using the weighted fair policy. The graphs show that when the workflows have equal priorities, their throughput tracks each other closely, but when Workflow 1 (green line with circles) has a higher priority, it gets a much higher rate of MIOs through the IMS, while Workflow 2 (blue line with diamonds) is not starved.

Table 2. Latency and Receive Rate in CPU Bound Test

	Mean Receive Rate (MIOs/sec)				Mean Latency (sec)			
	Base	Equal	Weighted	Strict	Base	Equal	Weighted	Strict
Workflow 1	17.3	17.5	33.8	34.9	8.6	3.2	3.1	3.0
Workflow 2	17.3	17.5	0.5	0	9.0	3.1	> 100	Infinite



(a) Equal priorities (b) Workflow 1 has higher priority

Figure 10. Graphs of Throughput in QED with Weighted Fair Policy

Overview of Test 2. The second test limited the network bandwidth between the subscribers and publishers to 300 kbps. The Test 2 configuration consisted of the same two workflows as in Test 1, but each now has 3 publishers and only 1 subscriber with a predicate that matched every MIO published in the workflow. This configuration kept the CPU from being a bottleneck since there was only one predicate per type to evaluate. Every published MIO, however, must be disseminated to a receiving client, which constitutes a larger amount of data than could be handled by the 300 kbps link. The Mockets transport provided differentiated service to the traffic from the more important workflow when the workflows had different importances. Since Mockets is based on a weighted algorithm, we were unable to set the priority strategy to *strict* for these tests.

Table 3. Latency and Receive Rate in Network Bound Test

	Mean Receive Rate (MIOs/sec)				Mean Latency (sec)			
	Base	Equal	Weighted	Strict	Base	Equal	Weighted	Strict
Workflow 1	3.9	6.4	8.3	N/A	> 100	4.6	2.4	N/A
Workflow 2	3.8	6.1	4.2	N/A	> 100	4.3	62	N/A

Analysis of Test 2 results. Table 3 shows the results from running Test 2 for 5 minutes. The results show that for the network bound case (which is common in a sensor or wireless network) the differentiated queue-based transport provides significantly higher throughput and lower latency to the higher importance workflow. In addition, QED provides significantly better throughput even in the case of equal priorities than the baseline. The increase in *equal* receive rate over the *baseline* is due to the higher performance of Mockets than the JMS used by Apollo. The decrease in latency of the *equal* case over the *baseline* is again due to running QED with different limits on queue sizes.

6. CONCLUDING REMARKS

This paper has presented our initial results creating QED, which provides a QoS management capability for publication-subscription-query information management services. The following are lessons learned from our efforts thus far:

- Our initial results show a significant improvement over the baseline Apollo IMS without QoS. In particular, managing the particular bottlenecks associated with processing and disseminating information, matching the information

processing and rates to the resources available, and prioritizing operations based on mission-level importance all contribute to improved performance and control.

- The IMS is a highly dynamic system, and can be used for scenarios with very different interactions and usage patterns. To realize the goals laid out in this paper—and to realize the QED services that we have designed—we need to develop the interfaces and policies to enable the specification of mission-based groupings and preferences, the translation functionality to refine them into actionable policies, and the runtime monitoring and algorithms to effectively enforce the policies based on current conditions.
- Despite the promise of our initial results, significant R&D challenges remain. In particular, optimization heuristics to achieve aggregate QoS for pub-sub systems, service level agreements (i.e., satisfying client preferences) when clients are decoupled, and capturing mission-level abstractions in policy that is actionable are all significant research topics in themselves.

Our future activities will be to build upon the QED prototype described in this paper to fully realize the QED capabilities and to evaluate them in operationally-based demonstrations and all three classes of experiments.

REFERENCES

[1] Air Force Research Laboratory, Apollo 1.0 User Guide.

[2] http://struts.apache.org/.

[3] M. Carvalho, R. Winkler, C. Perez, J. Kovach, S. Choy. A Cross-Layer Predictive Routing Protocol for Mobile Ad Hoc Networks. Defense Transformation and Net-Centric Systems 2008. Edited by Suresh, Raja. Proceedings of the SPIE, Volume 6981, 2008.

[4] D.H. Chen and P.P. Haglich. Fuselets and Collaboration for Warfighter Decision Superiority. International Symposium on Collaborative Technologies and Systems, CTS 2008. May 19-23, 2008.

[5] V. Combs, R. Hillman, M. Muccio, R.McKeel. Joint Battlespace Infosphere: Information Management within a C2 Enterprise. The Tenth International Command and Control Technology Symposium (ICCRTS), 2005.

[6] Defense Information Systems Agency, Net-Centric Enterprise Services. http://www.disa.mil/nces/.

[7] DoD CIO, Department of Defense Global Information Grid Architectural Vision, Vision for a Net-Centric, Service-Oriented DoD Enterprise, Version 1.0, June 2007. http://www.defenselink.mil/cio-nii/docs/GIGArchVision.pdf.

[8] J. Hill, J.M. Slaby, S. Baker, D.C. Schmidt. Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS. Proceedings of the 12th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 06), Sydney, Australia, August 2006.

[9] JBoss Community. JBoss Administration and Configuration Guide. November 2008. https://www.jboss.org/community/docs/DOC-12927.

[10] M. Linderman, B. Siegel, D. Ouellet, J. Brichacek, S. Haines, G. Chase, J. O'May. A Reference Model for Information Management to Support Coalition Information Sharing Needs. Tenth International Command and Control Technology Symposium (ICCRTS), 2005.

[11] Scientific Advisory Board. Building the Joint Battlespace Infosphere Volume 1: Summary. SAB-TR-99-02, 2000.

[12] Scientific Advisory Board. Report on Building the Joint Battlespace Infosphere Volume 2: Interactive Information Technologies. SAB-TR-99-02, 1999.

[13] M. Tortonesi, C. Stefanelli, N. Suri, M. Arguedas, M. Breedy. Mockets: A Novel Message-oriented Communication Middleware for the Wireless Internet. Proceedings of International Conference on Wireless Information Networks and Systems (WINSYS 2006), Setúbal, Portugal, August 2006.

[14] A. Uszok, J.M. Bradshaw, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, H. Jung. New Developments in Ontology-based Policy Management: Increasing the Practicality and Comprehensiveness of KAoS. Proceedings of the 2008 IEEE Conference on Policy, Palisades, NY, June 2-4, 2008.

[15] World Wide Web Consortium, OWL Web Ontology Language Overview, W3C Recommendation, February 10, 2004. http://www.w3.org/TR/owl-features/.

[16] World Wide Web Consortium, XML Path Language (XPath) Version 1.0, W3C Recommendation, November 16, 1999, http://www.w3.org/TR/xpath.

[17] World Wide Web Consortium, XQuery 1.0: An XML Query Language, W3C Recommendation, January 23, 2007, http://www.w3.org/TR/xquery/.