

A Jini-Based Publish and Subscribe Capability

Vaughn T. Combs and Dr. Mark Linderman

Air Force Research Laboratory
Information Directorate
Rome, New York

ABSTRACT

This paper describes a Publish and Subscribe capability developed under the Air Force Research Laboratory's (AFRL) Joint Battlespace Infosphere (JBI) project. The paper will give a brief description of the JBI and its core service components of publish, subscribe and query. A detailed description of the Pub/Sub system design and implementation will then be given describing how and where Java, Jini, and XML technologies were used to describe information objects, match subscribers to appropriate dissemination nodes, and disseminate information objects to subscribing clients. Finally we describe a number of applications that are currently using the Pub/Sub capability.

Keywords: Joint Battlespace Infosphere, JBI, Jini, XML, Publish and Subscribe

1. INTRODUCTION

The Joint Battlespace Infosphere^{1,2,3} (JBI) is an information management concept developed by the Air Force Scientific Advisory Board to address information management challenges in a military environment. A JBI comprises many diverse applications (called clients) and a set of core services that enable the dissemination, persistence and control of information being shared among the applications. One of the key core services provided to disseminate information is publish and subscribe (pub/sub). Pub/sub was chosen because it encourages very loose coupling between applications; it is believed that traditional military systems are too tightly coupled leading to interoperability and extensibility challenges.

However, it was thought that the traditional 'channel-based' pub/sub mechanisms did not provide enough fine-grain control to ensure that subscribers receive only appropriate information. Indeed, while a hierarchical tree of channels is useful to capture the essence of conceptual refinement, it is insufficient to deal with vast amounts of data – all of an identical 'type' or channel. Previous to JMS⁴ 1.0.2, the only solution was to create sub-channels to segregate information, but it was thought that this was too static in nature. For example, partitioning channels by geographical extent works if the regions of interest are static, but if the data represent the forward line of battle, this is constantly shifting. The channel hierarchy cannot be redefined in real time.

To address this issue and to facilitate interoperability, it was decided that information would be structured, and that subscribers would indicate their information needs with predicates over the structured information. However, not all information is naturally structured or likely to be used in predicates (e.g. images), so it was decided that the information would be a) typed, b) have a payload that may contain anything, and c) have structured metadata describing the payload. These three elements combine to form the information object. All applications using the JBI to exchange information must provide that information as typed information objects.

The pub/sub infrastructure is responsible for applying subscriber predicates to the metadata of published information objects. If the metadata of an information object satisfies a predicate, the information object is forwarded to the subscriber. This paper describes Jini-based mechanisms to broker publishers and subscribers efficiently and push information objects from publishers to subscribers.

Efficient brokering is based upon publishers registering before starting to publish. During the registration process, the publisher indicates its *invariant metadata*. Invariant metadata are elements of the information object metadata that will be constant in all subsequently published information objects. This information permits an efficient first-level matching

of subscribers and publishers. Subsequent matching, while more sophisticated, incurs a runtime penalty for subscriber predicate evaluation for each potentially matching subscriber each time an information object is published.

This implementation uses XML and XML Schema to represent metadata. Publishers provide invariant metadata as an XML document at time of registration. The underlying core service uses the schema to identify the invariant portions of the metadata to be used for registration with Jini's broker (lookup service). In addition to brokering based on the invariant aspects of the metadata, the current implementation allows for subscription using an arbitrary XPath or XQL predicate. This capability enhances Jini's equality based matching capabilities with support for inequality, AND, OR, NOT, etc. In addition, when the published information object payload happens to be in XML format (not a requirement), the pub/sub service permits arbitrary content-based filtering as part of the subscriber API. In addition to disseminating published information objects directly into the address space of subscribing applications, the Pub/Sub system also allows for indirect dissemination via email.

The Pub/Sub capability is currently being used by over 30 DoD contractors and is in active use for internal research and development projects. Additionally, this implementation has been included as an adjunct capability in the Control of Agent-Based Systems (CoABS) Grid.

1. Why Publish and Subscribe?

As mentioned above, the Air Force Scientific Advisory Board (SAB) chose a publish/subscribe mechanism to promote interoperability and reduce cost. Why?

When assembling a coalition of allies, as is often the case today, commanders must rely on systems that have the capability to characterize information completely enough so that the right information gets to the right coalition partners at the right time. A publish and subscribe paradigm may be used to richly characterize information object using metadata descriptions. Typically the producer of the information object creates a metadata descriptor associated with a distinct information object instance and then publishes it with the object. The subscriber or consumer of the information submits a subscription predicate defined over the metadata associated with information objects of that type. The underlying core services may then use this information to broker for the delivery of information objects between the producers of the information and the consumers of the information. Such a system allows for disparate military systems to share and disseminate information with no consideration for individual system interfaces.

In addition, most military system architects are coming to the conclusion that building systems that are tightly tied to the most recent substrate technologies makes these systems resistant to change. It is precisely this limitation that has led the JBI group to work toward a set of Common Core Service APIs (these APIs are under development and not presented here). The specification of the Common API is an ongoing DoD community processes which is making considerable progress. This approach would allow developers to design and implement client applications that need not consider what underlying substrate is being used for a particular JBI platform implementation. This notion alone is powerful in the context of military system implementations. Legacy applications that are built on JBI publish, subscribe, and query capabilities would no longer need to reinvest considerable resources to take advantage of the most recent shifts in technology. JBI platform implementers would implement the client API using the new technologies and existing client applications would run with, conceivably, no alteration.

2. The JBI Jini-Based Publish and Subscribe Core Services

The following sections give a brief description of how the JBI Jini-Based Publish and Subscribe core services work. This implementation provides a very convenient set of API method signatures for use by JBI publishing and subscribing client applications. While this implementation does not implement the aforementioned Common API, it does provide an API for publication and subscription that hides all of the underlying Jini specifics from the client developer. The Common API will be implemented on top of the current implementation as a set of veneer classes as the API becomes more mature.

2.1. The Role of Metadata

In a JBI, all information objects are characterized via metadata. In this implementation we have chosen to describe the metadata using XML⁵. Corresponding to each information object type is an XML Schema⁶ that describes the structure of its metadata. The metadata is used by the underlying core services to match subscribers to information objects that are of interest. Figure 1 shows a simple example of a metadata instance that describes a particular information object.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier>JBI000023</JBIIIdentifier>
    <publisher>418th</publisher>
    <keywords>Intel</keywords>
    <language>EN</language>
  </RequiredMetadata>
  <ImageDescriptor>
    <ImageType>IR</ImageType>
    <Area>Kabul</Area>
    <LocationCoord>
      <lat>33.34</lat>
      <latord>N</latord>
      <long>69.98</long>
      <longord>E</longord>
    </LocationCoord>
  </ImageDescriptor>
</metadata>
```

Figure 1. Example of metadata for intelligence imagery.

In the example the information object payload happens to be intelligence imagery in the Kabul area. The *information object type name* is defined to be mil.af.rl.jbi.Intel_Imagery. For each information object in the JBI there exists a unique metadata schema that describes the structure of the metadata for the type. The schema is used by the developer of the publishing application to generate valid metadata describing the object being published. The schema is similarly used by the subscribing application developer to create a predicates over the metadata that describe the subscriber's information requirements.

In our implementation we distinguish between the *invariant and variant* aspects of metadata. In the example above we may wish to consider the attribute "*RequiredMetadata*" subelement to be invariant. In other words, for this session, we could reasonably expect that the type, JBIIIdentifier, publisher, keywords and language would not change for each of the images published. In contrast, we could consider all of the "*ImageDescriptor*" information to be a variant sub-element of the metadata. For example, we would expect that, in general, the

location of the target depicted in the image published would change with each published image. While this would seem to be an unnecessary distinction, we will describe how the identification of the invariant and variant portions of the metadata may be used to increase the efficiency of the brokering mechanisms used in this implementation. Currently we annotate subelements within the XML Schema with comments denoting whether a subelement is variant. In future releases we will allow the user to specify this using XML attributes.

Work has begun to eliminate the additional specification of a subscription template regarding the invariant aspects of the metadata. The schema would then also be used to parse the XPath predicate for equality based expressions. This information would be used in the creation and population of the necessary underlying Entry objects.

2.2. What is Jini?

Jini⁷ is a freely available service oriented architecture that was originally designed as a distributed computing environment to provide networked devices with a network level plug and play capability. The architecture specifies how clients and services find each other within a "community". The service implementers supply Jini and, in turn, clients with a portable Java-based object (proxy) that can be used to allow the client to access and interact with the service. While, in most cases, this proxy uses Java Remote Method Invocation (RMI), any network technology may be used (i.e. CORBA, SOAP, etc.).

When a service joins a network of Jini enabled services, it must advertise its capabilities by publishing the proxy implementing the service's exported API. The proxy is registered with Jini's Lookup Service (LUS). A client

application may then request the proxy for a specific service. A typical example may be to request the proxy for a “Printer Service”. This would probably not be desirable, since you would receive the proxies for all printers that are currently registered. You may, alternatively, wish to interact with a printer within the same building that is capable of printing in color. Jini allows the service to further characterize itself using classes that describe its attribute value pairs. The Jini LUS would then allow the client application to use the additional attributes to specify a limited filtering capability. Currently the Jini LUS only allows for equality based matching with wildcarding. In our previous example one could then easily request the proxies to all printer services that are capable of printing in color and/or all printer services within a specific building or, if desired, all printer services. When the client application receives the service's published proxy object, it will download any code it needs in order to interact with the service.

Jini supports a rather robust set of protocols to enable the spontaneous discovery of services as they enter (and leave) a community. Should a client wish to use a specific service and it is not currently registered within the community, the client may choose to be notified when one becomes available. Analogously, the client may also choose to be notified when a service leaves the community. Additional Jini lookup services may be instantiated within the community to support a collection of lookup services that are tied to specific groups or simply to provide for a certain level of fault tolerance. In this case the clients and services within the community are notified of the new LUS and are afforded the opportunity to register with it.

2.3. Brokering

One of the most essential capabilities within a publish/subscribe system is a well-designed brokering capability. The fundamental responsibility of the broker is to match information objects produced by a publisher with appropriate subscribers. This section describes how the Jini-based Publish and Subscribe core services leverage the brokering capability implemented within the Jini Lookup Service with XML based technologies to provide efficient matching and information object dissemination services.

2.3.1. Using Jini Within the Publish and Subscribe Infrastructure

In a previous section, we touched briefly on Jini's ability to characterize services based on attributes. Jini allows a service to register a proxy with its LookUp Service (LUS). The proxy is ultimately used by remote clients to interact with the service. In addition, the service may associate a number of classes that implement the *Entry* interface with the proxy. These Entry class instances contain simple attribute/value pairs that are used by the LUS for the equality-based matching capabilities described above. If a successful match occurs, the proxy is then handed off to the client and the client may then interact directly with the service.

We use this capability to enable a first level of matching between published objects and subscribers. As mentioned above, we distinguish between the invariant and variant aspects of metadata. The invariant metadata is used to specify attribute value pairs that will be registered with the Jini LUS. Specifically, our implementation uses the schema for the information object metadata and the metadata instance provided by the publishing application to auto-generate, instantiate and populate the Entry classes that we use to register with the Jini LUS. The associated “proxy” that is registered is actually an exported API that will be used by the JBI's subscription side classes to register with appropriate dissemination nodes (delegate model) or publishing applications (peer-to-peer model).

The subscribing application, using the JBI subscription API, may then register a subscription template and predicate. The subscription template (an XML document) is used to specify the first level filtering that we would like to apply over the invariant attributes. Again, unbeknownst to the subscribing application, we generate, instantiate, and populate all Entry classes needed by the underlying Jini substrate and use the classes to register our “subscription template”. We will discuss both the publish and subscribe side interactions in greater detail in a later section.

2.3.2. Using XML within Publish and Subscribe Infrastructure

Previously we have mentioned that this implementation treats variant and invariant metadata differently. Using our previous example the publishing application first would register to publish providing an XML document that specifies

information object type and the invariant element values that are appropriate for the registered sequence of published objects.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier>JBII00023</JBIIIdentifier>
    <publisher>418th</publisher>
    <keywords>Intel</keywords>
    <language>EN</language>
  </RequiredMetadata>
</metadata>
```

Figure 2. Invariant metadata used to populate Entry classes.

The information shown in Figure 2 would be used to populate the generated Entry classes and subsequently register with the Jini LUS. Upon publication, a full complement of metadata would be provided (both variant and invariant) not unlike the example shown in section 2.1. The subscribing client application may provide two pieces of information. First, a subscription template must be defined for the *invariant portions* of the metadata (Figure 3). This template is essentially an XML document that describes the equality based matching with wildcarding semantics that would be provided as our first level match between candidate publication and subscription.

```
<?xml version="1.0" encoding="UTF-8"?>
<metadata xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="Intel_Imagery.xsd">
  <mil.af.rl.jbi.Intel_Imagery/>
  <RequiredMetadata>
    <Type>Imagery</Type>
    <JBIIIdentifier/>
    <publisher>418th</publisher>
    <keywords/>
    <language>EN</language>
  </RequiredMetadata>
</metadata>
```

Figure 3. Subscription template for invariant metadata.

Notice that, in this example, the subscriber has decided that she is interested in all Imagery published by the 418th that is annotated in English. In other words, the client has wildcarded the JBIIIdentifier and keywords fields.

While this first level of matching is very powerful it is inadequate given the matching semantics that are necessary within a JBI (i.e. inequality, AND, OR, NOT, etc.). To support the appropriate level of matching the Pub/Sub core services use XPath or XQL technologies to do additional predicate testing in the dissemination nodes and peer publishers (default implementation uses XPath).

To recap, the first level match is made and, if successful, a registration process occurs that allows the subscriber to register an optional XPath expression. The expression will then be used as a *predicate* that will be applied to the metadata that is published with each of the information objects of interest. An example of the instance metadata may be found in Figure 1. If the metadata describes an object that the subscriber wants then it will be disseminated.

The following is a simple but very powerful XPath expression that is consistent with our example:

```
*[(/metadata/ImageDescriptor/LocationCoord/lat>33.2) and (/metadata/ImageDescriptor/LocationCoord/lat<35.0) and (/metadata/ImageDescriptor/LocationCoord/latord='N') and (/metadata/ImageDescriptor/LocationCoord/long>65.7) and (/metadata/ImageDescriptor/LocationCoord/long<70.0) and (/metadata/ImageDescriptor/LocationCoord/longord='E')]
```

The expression defines a rectangular region in and around Kabul, Afghanistan. If this predicate were specified as part of the subscription then only the intelligence images that are of targets within the defined rectangular region would be disseminated to the subscriber.

Additionally, the subscriber may optionally provide an XPath *filter* that would be applied to the payload of the published information object. If the payload is in XML format (certainly not a requirement) the dissemination nodes will apply the

filter to the payload and disseminate only that part of the payload that is of interest. In contrast to our previous example the payload depicted in Figure 4 contains a target list. Each target is characterized by a descriptor that is a sub-element within the information object payload (XML in this case).

```
<targets>
  <descriptor>
    <name>ALPHA50</name>
    <type>F15E</type>
    <position>
      <lat>35.217</lat>
      <ladir>N</ladir>
      <long>62.267</long>
      <lodir>E</lodir>
    </position>
  </descriptor>
  <descriptor>
    <name>BRAVO51</name>
    <type>A10A</type>
    <position>
      <lat>34.817</lat>
      <ladir>N</ladir>
      <long>67.817</long>
      <lodir>E</lodir>
    </position>
  </descriptor>
  ...
</targets>
```

Figure 4. Example XML payload for a target list.

The subscriber may provide a content-based filter to return only target descriptors that are in a specific rectangular geographical region:

```
//descriptor[(((./position/lat>'33.2') and
(./position/lat<'35.0')) and (./position/ladir='N'))
and (((./position/long<'70.0') and
(./position/long>'67.8')) and (./position/lodir='E'))]
```

Again, the distinction between the XPath predicate and the XPath filter is that the *predicate is only applied to the metadata* characterizing the published information object. The *filter is applied to the payload* (if in XML format) to determine what portions of the payload should be disseminated to the subscriber.

2.4. Interactions Within the Pub/Sub System

This section describes, in greater detail, the interactions that take place within the JBI Jini-Based Publish and Subscription Services.

Our implementation supports both peer-to-peer and delegate implementations. These implementations may co-exist with one another. The client applications interact with the system via three adapter classes. The subscribing application uses a Subscriber Adapter (SA) while a publishing application may use a Publisher Adapter (PA) for peer-to-peer publication or a Publisher Adapter Light (PAL) class when using delegates for dissemination.

Using Figure 5 below, we will quickly walk through the publish and subscribe sequences. We will go into greater detail later in this section. In steps 1 through 4 the publishing client registers its intention to publish with the JBI. It is at this point that the client provides the XML Schema representing the structure of the metadata used to characterize the information object type and an XML document specifying the invariant metadata element values as discussed in the last section. In steps 5 through 8 the subscribing client registers its subscription with the JBI. In this case the subscription is first matched to a candidate dissemination node using the XML document subscription template. The adapter then registers with the appropriate dissemination node(s) optionally providing an additional predicate and/or filter that may be applied to the published instance metadata and payload.

The following discussions describe the use and, at a high level, the interactions specifically when a publishing application uses a Publisher Adapter Delegate (PAD).

A PAD is a dissemination node that handles all publishing related processing on behalf of a publishing application. Currently a PAD is characterized simply via two arbitrary strings that may be used by a publishing application to broker

for a distinct delegate. Any number of delegates may be started within the environment and they may be characterized in any way that is deemed appropriate.

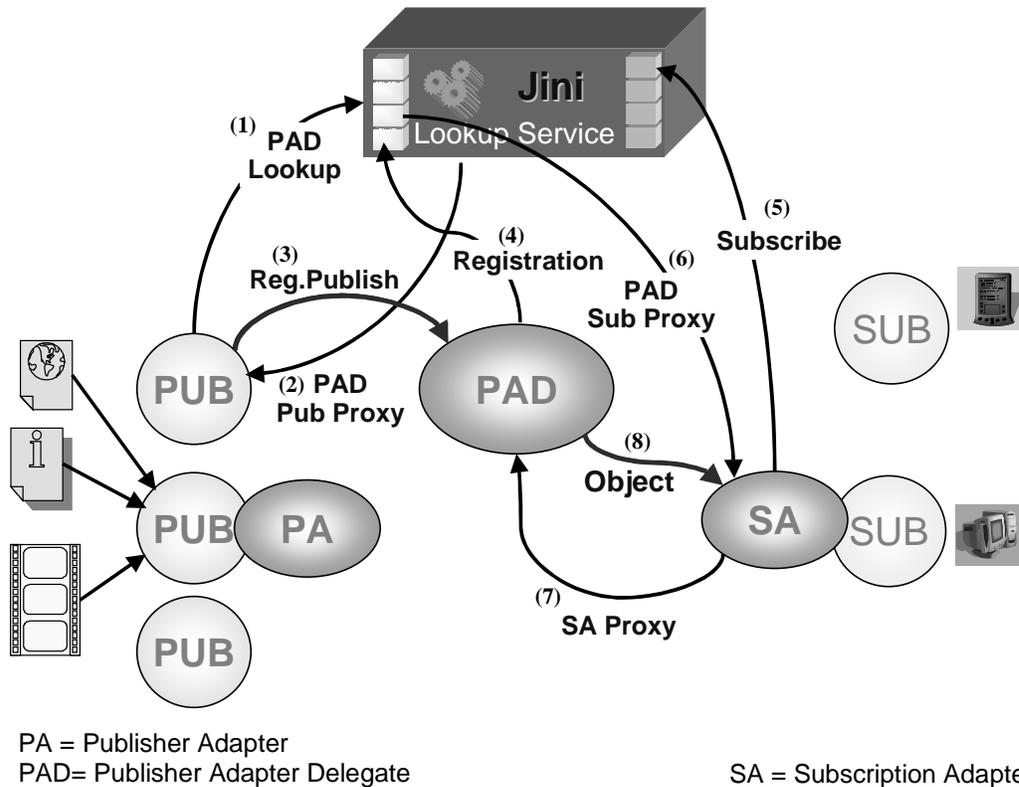


Figure 5. Interactions within the JBI Jini-Based Publication and Subscription Services.

In (1) the publishing application uses the *connectToDelegate* method within the Publisher Adapter Light API to connect to a distinct PAD. This method will return a unique identifier that may be used in subsequent PAL API calls to interact with a distinct delegate. This technique is employed to support the use of multiple delegates within one publishing application.

A great deal of care was taken in creation and management of leases between the PAL implementation classes and a matched PAD. Since the PAD consumes resources on behalf of each of its matched publishers (maintaining publication queues, dissemination threads, registrations with LUS, etc.) it was important to implement a lease between the PAL and matched PAD. The lease times used are all configurable via user properties that may be specified at startup. Should a publishing application lose contact with a PAD or die the PAD will clean up after the publisher in an appropriate manner.

In (2) a proxy is returned that will be used by the implementing PAL classes for all interactions with the matched delegate. The proxy itself is never used by the publishing application. Instead the adapter implementation classes use the proxy to interact with the PAD on behalf of the client application. Moreover, the using application is never aware of the fact that it is using Jini.

In (3) the user invokes the *registerPublisher* method within the PAL API to register a publishable object with the JBI Pub/Sub. The client application provides the information object metadata's XML Schema and the invariant information object metadata attribute values in XML format. The information is then used by the underlying PAL implementation

classes to generate the necessary Jini Entry classes. Subsequently, the classes are used for registration of the invariant metadata attribute values with the Lookup Service. The instantiated Entry class instances are sent to the PAD as part of the registration process. The registerPublisher method returns another UID that identifies this particular publishable information object registration.

While the Entry class code generation software utilizes the XML Schema to define the structure of the classes, for the most part, the code is class-based and hardcoded. In subsequent releases the code generation will be accomplished using the XML Schema and an associated XSL stylesheet. This approach will allow us to flexibly change how the Entry class code is generated from the XML Schema should the overall metadata structure evolve over time. Any overall changes in metadata representation and structure could then be handled without having to alter the underlying implementation code.

In (4) the PAD registers the publishable object's invariant metadata with the Jini LUS for brokering.

At this point the publishing application may use one of the *publish* method signatures supported by the PAL API to publish information objects and associated metadata.

To recap, the publishing application uses variations of three very simple method signatures *connectToDelegate*, *registerPublisher*, and *publish*.

In (5) we begin to look at a subscriber's use of the Subscriber Adapter API. The subscribing application simply uses one of the SA API *registerSubscriber* methods to register a subscription. The method allows for the specification of an XML document that is the metadata template. The template is essentially used to specify the equality based matching criteria over the invariant portions of the metadata. Entry classes that reflect this information are then created, instantiated, populated and used to register this first level of matching criteria with the Jini LUS. It is at this point that the XPath predicate may also be specified as an argument that will eventually be applied by the PAD over the metadata published with a specific information object instance. In addition, it is at this point that a subscriber may specify a content-based filter in XPath as one of the registerSubscriber arguments. The PAD will determine if the payload is well-formed XML and, if it is, it will apply this XPath expression on behalf of the subscribing application and will only send the portions of the payload that the application needs. The subscriber may also specify a notification listener that will be used by the Subscription Adapter to notify the application of information object arrival and loss. Finally, we allow the subscriber to specify both direct and indirect dissemination mechanisms. Specifically, the subscriber may choose to have the information object delivered directly into the application's address space (SA individual subscription queues) or he may choose to have the object delivered to a number of email recipients. The registerSubscriber method returns a unique identifier that identifies a specific subscription and is used in all subsequent SA API calls. This allows the subscribing application to register multiple subscriptions using the same Subscription Adapter (in fact only one SA is allowed in a single address space).

The next two interactions are never seen by the subscribing application. In (6) the Subscriber Adapter receives a proxy to an appropriate PAD where the match has been made using equality based matching over the invariant portions of the information object metadata. In (7) the SA uses the PAD proxy to register the additional subscription information such as the XPath metadata predicate, XPath content-based filter, indirect email addresses, the subscription UID, and, most importantly, a proxy that the PAD may use to contact the Subscriber Adapter. The registered proxy is mostly used to push published objects into the address space of the subscribing application and to determine the health and status of the subscribing application.

We have mentioned earlier that the current implementation also supports a peer-to-peer implementation using the Publisher Adapter classes. The interactions in this case are essentially identical to the delegate discussion with the exception that we do not connectToDelegate and, when matched via the invariant portions of the information object metadata, the Subscriber Adapter is given a proxy to the Publisher Adapter that is running in the address space of the appropriate publishing application. The PA then handles all of the processing that we discussed earlier regarding PADs (i.e. publication queues, management of dissemination threads, metadata predicate processing, content-based filtering, email dissemination, etc.).

3. Application Areas

In this section we will provide a brief description of some of the applications that are currently using this implementation of Pub/Sub.

3.1 Airborne Experiment

The airborne experiment is a joint project involving the Air Force Research Laboratory (AFRL) Information Directorate's Intelligent Adaptive Communications Controller (IACC) and Joint Battlespace Infosphere (JBI) teams.

The IACC project is focused on the requirements of Air Mobility Command (AMC) for global in-transit visibility and seamless, multi-media command and control. IACC provides dynamic network connectivity over multiple resources without control from a user. The in-house IACC team has provided information connectivity to airlifters and other remote/deployed users, by intelligently integrating and managing available communications resources. These resources include military communications such as HF and UHF Satcom, as well as commercial resources such as Global Air Traffic Management (GATM) network, Inmarsat and Iridium. Through integration existing communications media, the system provides for cost effective (no aircraft mods) information capability to deployed and in-transit assets. The IACC system integrates current, stove-piped communications systems to establish a secure, assured, real-time information and communications infrastructure.

The airborne experiment was developed to determine how the JBI Jini-Based Publication and Subscription Services would perform over extremely disadvantaged communications links. The IACC team maintains a testbed capability that allows for experimental transmission and characterization of traffic over IACC tactical links (i.e. UHF LOS, HF, and UHF Satcom). During the first phase of the experiment the team instantiated an "airborne" client that used a metadata template to subscribe to an Air Tasking Order (ATO). An ATO can be rather large so, in order to decrease bandwidth utilization, a content-based filter (XPath expression) was provided at the time of subscription. The filter was applied by the dissemination node so that only mission information that pertained to the airborne node would be disseminated to the subscriber. Since the dissemination node (PAD) was ground based this significantly decreased the bandwidth utilized. During the first phase of the experiment the aggregate bandwidth ranged from 7200 to approximately 23000 bps. The first phase of the experiment proved that the Pub/Sub performed reasonably well over a disadvantaged communications substrate. In the future the team will explore how the Pub/Sub behaves while subjected to various communications degradations and failures. In addition, a study will be made of the control message exchanges spawned by the Pub/Sub infrastructure to determine whether optimizations may be made.

3.2 JView

JView⁸ is a multi-purpose, runtime re-configurable, operator customizable, platform-independent, object-oriented, visualizer. JView has been developed by an in-house research team within the Air Force Research Lab's Information Directorate. Currently the JBI and JView team are using the JBI Pub/Sub to publish simulated UAV telemetry and imagery. JView then subscribes for specific published UAV information and provides the information in both two and three-dimensional renderings within in an operationally meaningful context. In the next year the JBI and JView team will continue to explore how JView may be used to more intuitively metaphor for the subscription of information object data. To that end, JView components will be utilized to provide a means for defining and submitting subscriptions based on type and spatial limitations reflected within information object metadata.

3.3 MCP

The Master Caution Panel (MCP) provides the Joint Forces Air Component Commander (JFACC) with a tailorable, web-based application to manage the Air Operations Center (AOC) as a weapon system. The system is used to further understand how critical command and control systems operate when systems degrade and fail. In addition MCP provides a list of recommended courses of action to mitigate damage to the overall planning, assessment, and execution of the

expeditionary air campaign. In short, the MCP provides a commander with system-wide command and control resource monitoring and centralized, standardized status reporting, automated assessment of mission impact, and intuitive web-based presentation of information that is individually tailorable.

The JBI Pub/Sub is used to instrument “Other Weapons Systems” than Command and Control, such as Unmanned Ariel Vehicles (UAVs) and telephone switches. In the UAV integration experiment JBI pub/sub has been integrated into the MCP Monitoring architecture to support instrumentation of a RC trainer aircraft simulating a Micro UAV. This Micro UAV transmits telemetry from an onboard GPS and Streaming video from an onboard camera to a ground station receiver. The receiver application samples the stream, bundles telemetry data and imagery into a serialized information object. The JBI Pub/Sub is then used to characterize and publish the information object. The MCP monitor utilizes the JBI SubAdapter API to subscribe for the published UAV information objects. MCP then performs an evaluation of the received information in order to provide a user defined representation of the status of the weapons system (in this case the Micro UAV). The status information is then made available as multiple visualizations which are ultimately used by operations staff who use the information to accomplish their mission within an AOC. In addition to this instrumentation the JBI Pub/Sub is being used to correlate the telemetry/imagery objects with targeting information contained in the Air Operations Data Base (AODB). This information forms a Battle Damage Assessment (BDA) object, which is then published in conjunction with the UAV instrumentation. This capability will give the warfighter access to real-time BDA information utilizing a low cost, state-of-the-art, solution.

3.4 COABS Grid Integration

The Control of Agent-Based Systems⁹ (CoABS) is a project that is funded by the Defense Advanced Research Projects Agency (DARPA) to develop and demonstrate techniques to safely control, coordinate and manage large systems of autonomous software agents. In general, the program has invested considerable resources in investigating the use of agent technology to improve military command, control, communication, and intelligence gathering.

The CoABS grid middleware is a service centric implementation that is built upon Jini. It was discovered that a Pub/Sub capability for the dissemination of information would be of great benefit to grid agent developers. Since this capability did not currently exist in the grid and a viable prototype was available in the JBI implementation, it was decided to integrate this capability within the grid. Currently the grid contains a full implementation of version 1.1.8 of the JBI Jini-Based Pub/Sub and is actively being used by a number of grid researchers.

3.5 Joint Experimentation

This publish/subscribe implementation is currently being used to disseminate information from several real and many simulated sensors. These sensors will stress the throughput of the pub/sub infrastructure both from a networking and an XML processing perspective. Simple fusion engines will subscribe to the published sensor ‘reports’ to construct fused (or correlated) outputs. Sensors that respond to tasking will also provide Jini services for this purpose.

4. Future Work

Work has begun to eliminate the additional specification of a subscription template regarding the invariant aspects of the metadata. Ultimately, whether or not a sub-element of the metadata is invariant will be reflected in the metadata schema as an element attribute. This information will be used to allow for the specification of the XPath predicate as the complete subscription. The schema will then be used to parse the predicate for equality based expressions. In addition, work has already started to implement an early version of the Common API on top of the Jini-Based Pub/Sub. This work have already added to enhancements in performance and function within the current underlying implementation.

ACKNOWLEDGMENTS

The authors would like to thank the entire AFRL JBI team. Without the entire team’s diligence, continual support and encouragement this work would not have been possible.

REFERENCES

1. USAF Scientific Advisory Board, "Information Management to Support the Warrior", SAB-TR-98-02, <http://www.rl.af.mil/programs/jbi/documents/IMReport.pdf>, 1998.
2. USAF Scientific Advisory Board, "Technology Options to Leverage Aerospace Power in Operations Other Than Conventional War", SAB-TR-99-01, http://www.rl.af.mil/programs/jbi/documents/TLAP_Final_Volume_1.pdf, 2000.
3. USAF Scientific Advisory Board, "Building the Joint Battlespace Infoshere Volume 1: Summary" SAB-TR-99-02, <http://www.rl.af.mil/programs/jbi/documents/JBIVolume1.pdf>, 1999.
4. Java Message Service Documentation, "Java Message Service Specification Version 1.0.2b", <http://java.sun.com/products/jms/docs.html>, 2001.
5. World Wide Web Consortium, "Extensible Markup Language (XML) 1.0 (Second Edition)", <http://www.w3.org/TR/2000/REC-xml-20001006>, 2000.
6. World Wide Web Consortium, "XML Schema Part 0: Primer", <http://www.w3.org/TR/2001/REC-xmlschema-0-20010502>, 2001.
7. Sun Microsystems Inc., "Jini Architecture Specification", http://www.sun.com/software/jini/specs/jini1_2.pdf, 2001.
8. J. A. Moore, C. Salisbury, "Reconfigurable Simulation Visualizer", *Proc. SPIE* **4026**, pp. 50-54, 2000.
9. Kahn, Martha and Della Torre Ciclese, Cindy. "The CoABS Grid." In Goddard/JPL Workshop on Radical Agent Concepts (WRAC), 2001.