

Semantic Message Oriented Middleware For Publish/Subscribe Networks

Han Li ^a and Guofei Jiang ^b

^a Thayer School of Engineering, Dartmouth College, Hanover, NH 03755

^b Institute of Security Technology Studies and Thayer School of Engineering
Dartmouth College, Hanover, NH 03755
{hanli,gfj}@Dartmouth.edu

ABSTRACT

The publish/subscribe paradigm of Message Oriented Middleware provides a loosely coupled communication model between distributed applications. Traditional publish/subscribe middleware uses keywords to match advertisements and subscriptions and does not support deep semantic matching. To this end, we designed and implemented a Semantic Message Oriented Middleware system to provide such capabilities for semantic description and matching. We adopted the DARPA Agent Markup Language and Ontology Inference Layer, a formal knowledge representation language for expressing sophisticated classifications and enabling automated inference, as the topic description language in our middleware system. A simple description logic inference system was implemented to handle the matching process between the subscriptions of subscribers and the advertisements of publishers. Moreover our middleware system also has a security architecture to support secure communication and user privilege control.

Keywords: Publish/subscribe system, message oriented middleware, semantic matching, daml+oil, description logic inference system, secure socket layer

1. INTRODUCTION

Many large-scale distributed systems today need to connect thousands of systems that are widely distributed and change frequently throughout their lifetime. This challenge motivates the demand for middleware that can provide loosely coupled communication models for distributed systems, allowing each component to evolve independently.

While traditional point-to-point and synchronous communication models are popular in rigid and static applications, Message-Oriented Middleware (MOM), provides a versatile middleware system to loosely integrate distributed systems [1]. As the name suggests, a Message-Oriented Middleware system enables distributed applications to communicate by routing their messages through the middleware system. In this system, the client application sends messages to the MOM and the MOM is responsible for delivering the message to remote receivers. There are two broad categories of message-oriented middleware: message queuing and publish/subscribe. The message queuing paradigm provides a point-to-point messaging model, in which typically messages are addressed to their recipients. By contrast, the publish/subscribe paradigm provides a many-to-many communication model so that messages can be efficiently disseminated across a large scale distributed system. In a publish/subscribe system, participant applications have two different roles – publishers that publish messages to the MOM as information sources, and subscribers that subscribe the information of their interests and receive messages from MOM as information consumers. The publish/subscribe paradigm enables communicating parties to be decoupled in communication time and channel and it has gained much attention in last several years.

A fundamental problem in a publish/subscribe system is how to match the interests of subscribers with the available messages from publishers. The publish/subscribe systems can be classified in two categories: topic-based or content-based [2]. In a topic-based system, each message is classified as belonging to one of a fixed set of topics, also referred as groups, channels, or subjects. A publisher labels each message it produces with a particular topic. Similarly, a subscriber targets its subscription with a particular topic, and then receives all messages associated with that topic. In a content-based system, a publisher uses a predefined message schema to create messages, and a subscriber submits the

subscription as a query (or filter, or pattern) against the message schemas. Then the decision as to whom a message is addressed is made on a message-by-message basis. One advantage of a content-based system is that the subscribers have the flexibility to choose filtering criteria along multiple dimensions of the information spaces, instead of being limited to pre-defined topics. The main disadvantage of a content-based system is the burden it places on the underlying system for message matching [3]. The matching problem of content-based publish/subscribe system does not have a known, scalable solution [4] [5].

The research work on publish/subscribe systems mainly focuses on the matching algorithm and scalability issue, such as Siena [6], Gryphon [7], NaradaBrokering [8] and Scribe [9]. However, the interest description and matching between the publishers and subscribers are only based on attributed-based queries and patterns for a predefined information space (for the content-based systems) or keywords and topic IDs (for topic-based systems). None of those systems provides the clients the capability for semantic description and matching. In this paper, we present our implementation of a Semantic Message Oriented Middleware as a publish/subscribe system to meet this need.

2. RELATED WORK

2.1 Java Message Service

The under-layer messaging system of our Semantic Message Oriented Middleware is based on Java Message Service technology. The Java Message Service (JMS) of Sun Microsystems Inc [10] is the specification of a common API that provides a simple but powerful solution for creating a messaging system. JMS supports both the message queuing and the publish/subscribe approaches of messaging systems. The JMS publish/subscribe model is a topic-based messaging model.

As most of other topic-based systems, the limitation of the JMS publish/subscribe model is the limited selectivity of subscription and lacking expressiveness of topic description. In the JMS system, the topic is marked with a plain text name, and all topics should be explicitly created before the messaging system is running. The topic description and matching are only based on the syntax or keyword instead of on the semantics of terms. The publishers and the subscribers need to exactly match their topics to achieve message exchange between them. In other words, all the publishers and subscribers have to pre-agree on all the possible event categories across the entire message system. This requirement is difficult to satisfy in a large-scale distributed system. Our goal is to provide the subscribers with a more flexible way to describe their subscriptions, and also provide the publishers with the capability to capture deep semantics in their advertisements. At the same time, the matching between the subscribers' subscriptions and the publishers' advertisements must be supported.

To solve this problem, we need a language that can express the contents of information spaces and also support inference matching between the advertisements and subscriptions. We adopt the DARPA Agent Markup Language and Ontology Inference Layer (DAML+OIL) [11] as the topic description language of the Semantic Message Oriented Middleware system, since DAML+OIL is a formal knowledge representation language for expressing sophisticated classifications and enabling automated inference. The language and the reasoning supported by DAML+OIL are rich enough for our topic descriptions and for matching between those descriptions.

Another limitation of a JMS publish/subscribe messaging system is that JMS does not provide any security specifications, such as message encryption, identity authentication, or authorization, which are necessary in many application fields in which information security is required, such as the military domain. Some of these security features are provided in our implementation of the Semantic Message Oriented Middleware system.

2.2 Description Logic and DAML+OIL

Description logics are a set of knowledge representation languages that can be used to express knowledge about concepts and concept hierarchies in a structured and formally well-understood way [12]. The basic building blocks of description logic are concepts, roles and objects (or individuals). Concepts describe the common properties of a collection of individuals and can be considered as unary predicates that are interpreted as sets of objects. Roles are interpreted as binary relations between objects. Description logics also define a number of language constructs (such as intersection, union, role quantification, etc.) that can be used to define new concepts and roles [13].

The Knowledge Base (KB) in a description logics system consists of two parts, the Terminologies (TBox) and the World Descriptions (ABox). A TBox stores the conceptual knowledge (vocabulary) of an application domain, while an ABox introduces the assertion knowledge. Whereas TBox restricts the set of possible worlds, ABox describes the specific states of the world by representing individuals together with their properties [14].

Generally, the main purpose of a knowledge representation system in description logics is to reason about the knowledge in the knowledge base. The main reasoning tasks (which are also called standard inferences) are *classification*, *satisfiability*, *subsumption* and *instance checking*. We will mainly discuss the *instance checking* reasoning task.

Instance checking is one of the standard inference problems about ABox. The main task is to check if an individual i satisfies the restrictions of a concept description C . If so, then i is said to be an instance of C .

For example,

Given a TBox as: Parent = Human \cap has-child.Human

Given an ABox as: Human (Tom), Human(Jerry), has-child(Tom, Jerry)

Then we can make the conclusion that Tom is an instance of Parent, say, Parent(Tom).

DAML (DARPA Agent Markup Language) + OIL (Ontology Inference Layer) is an expressive schema language for modeling semi-structured metadata and enabling knowledge-management applications such as the Semantic Web [11]. The DAML+OIL is designed based on the idea of description logics. Hence the semantics of DAML+OIL can be defined by a translation into expressive description logic.

DAML Constructor	DL Syntax	DAML Axiom	DL Syntax
intersectionOf	$C_1 \cap \dots \cap C_n$	subClassOf	$C_1 \subseteq C_2$
UnionOf	$C_1 \cup \dots \cup C_n$	sameClassAs	$C_1 = C_2$
complementOf	$\neg C$	subPropertyOf	$P_1 \subseteq P_2$
One of	$\{x_1 \dots x_n\}$	samePropertyAs	$P_1 = P_2$
ToClass	$\forall P.C$	disjointWith	$C_1 \subseteq \neg C_2$
HasClass	$\exists r.C$	sameIndividualAs	$\{x_1\} = \{x_2\}$
HasValue	$\exists r.\{x\}$	differentIndividualFrom	$\{x_1\} \subseteq \neg\{x_2\}$
minCardinalityQ	$(\geq n \text{ r.C})$	transitiveProperty	$P \in \mathbf{R}$
maxCardinalityQ	$(\leq n \text{ r.C})$	uniqueProperty	$T \subseteq \{\leq 1PT\}$
inverseOf	r^-	unambiguouseProperty	$T \subseteq \{\leq 1P^-T\}$

Table 1 DAML+OIL constructors, axioms and equivalent DL syntaxes

DAML+OIL uses syntax based on the Resource Description Framework (RDF). A DAML+OIL ontology can be seen as a TBox in description logic. In the ontology, the terminologies of the application domain are described in term of classes and properties, in which classes correspond to the concepts and properties correspond to the roles. The marked-up instance data of those classes can be seen as the ABox. As in description logics, there also are a set of constructors in DAML+OIL for building up class expressions. Table 1 shows the set of constructors and axioms in DAML+OIL along with the equivalent description logics syntaxes. [12]

The Semantic Topic Matching in our system is based on the instance checking inference between the topic descriptions of publishers and subscribers.

3. SYSTEM OVERVIEW

The system topology of the Semantic Message Oriented Middle is centralized client-server model. Each client acts as a publisher or subscriber, and a centralized messaging server is responsible for matching and brokering the messages between publishers and subscribers.

3.1 System structure

The central middleware server contains two layers: the semantic broker layer and the JMS provider (J2EE server) layer. The semantic broker runs on the same machine with the JMS provider, and acts as an interface between the JMS provider and the application clients. The semantic broker handles the publish/subscribe requests of the client applications, organizes and maintains the topics of the JMS provider, matches and maps the clients' topic descriptions with the JMS topics, and generates and receives the JMS messages for the JMS provider. The JMS provider takes care of the delivery of the messages that are grouped by JMS topics. The details of the JMS provider, such as the JMS topic names and wrapping and receiving JMS messages, are transparent to the publisher and subscriber client applications. A set of APIs are provided for publisher/subscriber client applications which encapsulate the procedures to make a connection with the semantic broker server, submit advertisement or subscription, and pass and receive messages to the server and from the server. Figure 1 shows the architecture of the whole system.

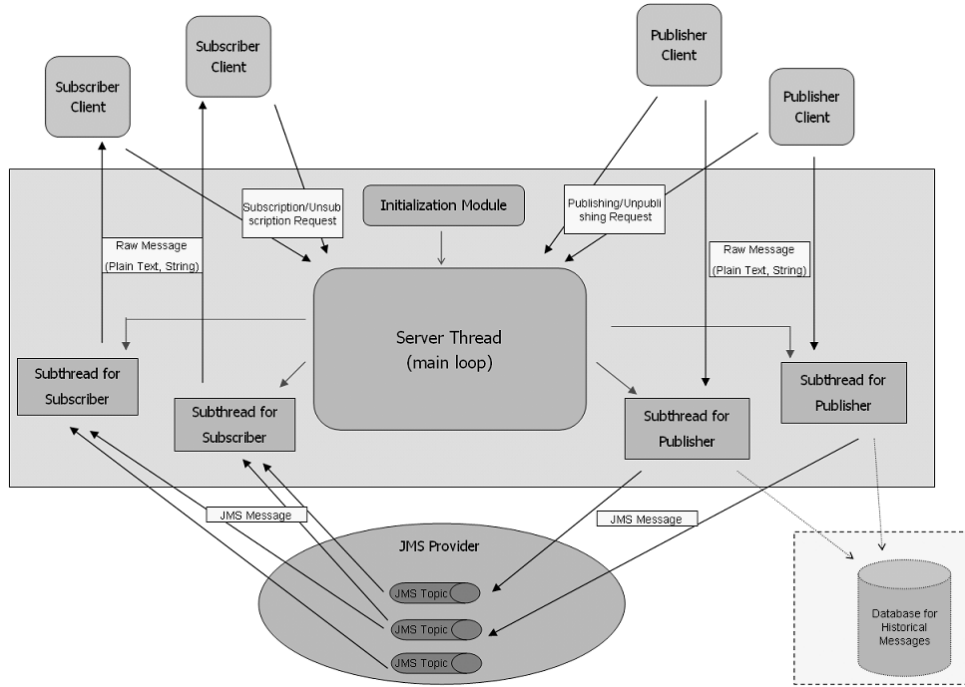


Figure 1 The architecture of a semantic message oriented middleware

3.2 Workflow for a publish/subscribe request

Figure 2 shows the general workflow for a publish/subscribe request. When a client application is going to make an advertisement or a subscription, it connects with the semantic broker server on a specific port. In the semantic broker server, there is a main thread listening to the port, and waiting for a secure socket connection. After the secure socket connection is constructed successfully, the client is required to provide the user identification information. If the user is certificated, the client then can submit the DAML topic description as the advertisement or subscription to the semantic broker. The semantic broker will search other existing topic descriptions in the system for matches, and then map the advertisement or subscription of the current client to the JMS topics. After the mapping and matching process, the main thread of the semantic broker will store information of current new client, and create a new sub-thread to handle subsequent message transfer. For a publish request, the created sub-thread will receive messages from the publisher over the socket connection, wrap them as JMS messages, and publish them to the JMS provider. For a subscribe request, the sub-thread will receive the JMS messages from JMS provider, unwrap the contents from those messages, and send data to the subscriber client. The main thread of the semantic broker will assign each client application a client ID as identification of its session. When a client application wants to stop publishing or subscribing, it needs to send a stop request to the main thread. After receiving the request, the main thread will stop the corresponding sub-thread and turn off the connection of that client application based on the client ID.

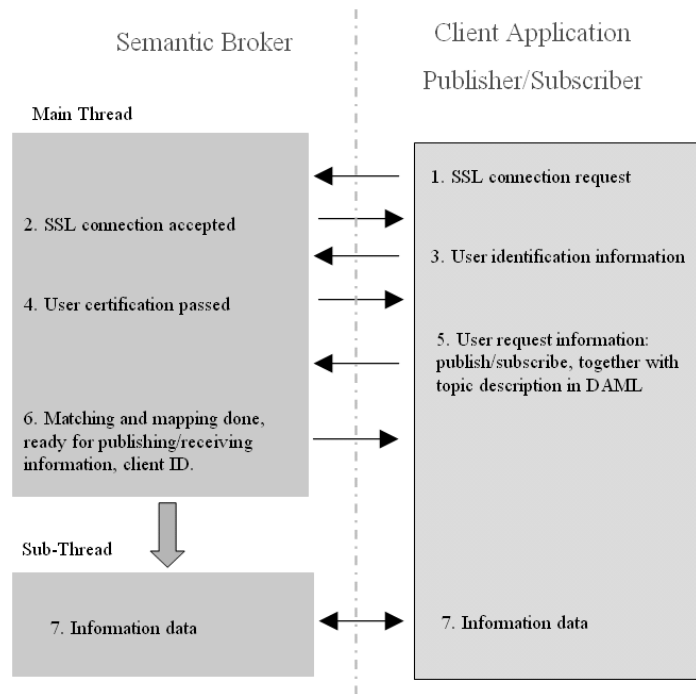


Figure 2: General workflow for a publish/subscribe request

3.3 Message persistence feature

The semantic broker also has the functionality of message persistence. By this feature, all the messages published in the system, together with information such as their publishing time and topic descriptions, will be stored in a database. Then all the information can be accessed in the future, even if the original publishers are no longer available.

4. SEMANTIC TOPIC DESCRIPTION AND MATCHING

4.1 Overview

The semantic topic-matching feature of the Semantic MOM system is implemented as a simple description logic system, which includes two components. The first is the knowledge base, which defines the topics domain. Second is the reasoning engine, which implements the inference service. The knowledge base consists of DAML+OIL ontologies, concept descriptions of subscribers, and instance descriptions of publishers. The DAML+OIL ontologies contain the definitions of concepts and roles of the topic knowledge domain, as well as the hierarchy structure and relationships between those concepts and roles, including a *topic* class definition as the schema of topic description for the publisher. These DAML+OIL ontologies are pre-agreed by the subscribers and the publishers. Each publisher submits an instance description of the *topic* class defined in the DAML+OIL ontology as advertisement. Each subscriber submits a concept description, which is a class definition in terms of DAML+OIL as a subscription. As an expressive schema language, DAML+OIL gives rich means for expressing constraints in schemas, and provides primitives that support the general representation of knowledge. These features give subscribers the capability to express their requests. A simple DAML+OIL reasoner has been implemented for the instance checking inference between each subscriber's class description and publisher's instance description to see if they match.

4.2 DAML+OIL reasoner

As we mentioned before, since the matching workload of a topic-based messaging system is relatively light, our semantic topic matching function only uses a simple table looking up algorithm. The semantic topic matching function is responsible for the *instance checking* inference between each subscriber's class description and each publisher's instance description to check if they match. Specifically, a subscriber's request matches a publisher's advertisement

when the DAML instance description of the publisher can satisfy the restriction of the DAML class description of the subscriber. In our semantic MOM system, the topics of subscribers and publishers are stored separately in two tables. For each new incoming subscriber, all existing advertisements in the publish topic table will be checked to see if the new subscriber's request matches them; for each new incoming publisher, all existing requests in the subscribe topic table will be checked for a match. The DAML+OIL reasoner is implemented based on the Jena Semantic Web Toolkit release 1.6.0, which is developed by HP labs [15].

The subroutine of *MatchDAMLClass* (*damlClass*, *damlInstance*) finds the match relationship between a DAML class expression and a DAML instance expression. A *daml:Restriction* is also a legal DAML class expression, so this subroutine can also handle the reasoning for *daml:Restriction*. The algorithm is showed in Figure 3.

When the DAML class expression is a simple DAML class, the matching relationship is defined whether the RDF type of the DAML instance is the same as or is subclass of the DAML class expression. For *daml:Restriction* expression, the subroutine will parse the corresponding property value of the DAML instance based on the *onProperty* value of the *daml:Restriction* expression, and a matching is found when the RDF type of the instance is the same as or is subclass of the *daml:toClass* of the *daml:Restriction* expression, or the URI value of the instance equals the *daml:hasValue* of the *daml:Restriction* expression.

```

MatchDAMLClass ( damlClass, damlInstance )
{
    if (damlClass instanceof daml:Restriction)
    {
        property = damlClass.valueOf ( daml:onProperty );
        newInstance = damlInstance.PropertyValueOf (property) ;
        toClass = damlClass.valueOf ( daml:toClass );
        hasValue = damlClass.valueOf ( hasValue );

        if (RDFTypeOf(newInstance) equal toClass
            or ( RDFTypeOf( newInstance ) isSubClassOf toClass)
            return true;

        else if ( UriValueOf(newInstance) equal hasValue )
            return true
    }
    else
    {
        if (RDFTypeOf( damlInstance) equal damlClass)
            return true;
    }
}

```

Figure 3: The algorithm of *MatchDAMLClass* subroutine

The subscriber can achieve exact keyword matching with the publisher by specifying the *daml:hasValue*. Otherwise, the type match will determine the overall match.

4.3 User interface of subscription editor

As discussed before, the advertisement of a publisher client is a DAML file that contains instance data of the predefined DAML ontology. Some existing DAML markup tools can be used to markup the advertisement description. In addition, we implemented a subscription editor (shown on Figure 4) with a graphical user interface for the subscriber to create the request description, which is a DAML class expression. The subscription editor has been integrated with our PQS (Process Query System) client user interface application.

In the subscription editor, the structure of the subscription request expression is presented to the user graphically. The user can select the constraint types and set relative values of the expression from the list. The editor also shows available class names in the ontology from which the user can choose, based on the constraint type defined by the user. A DAML file that contains the subscription request expression will be created for submission to the server.

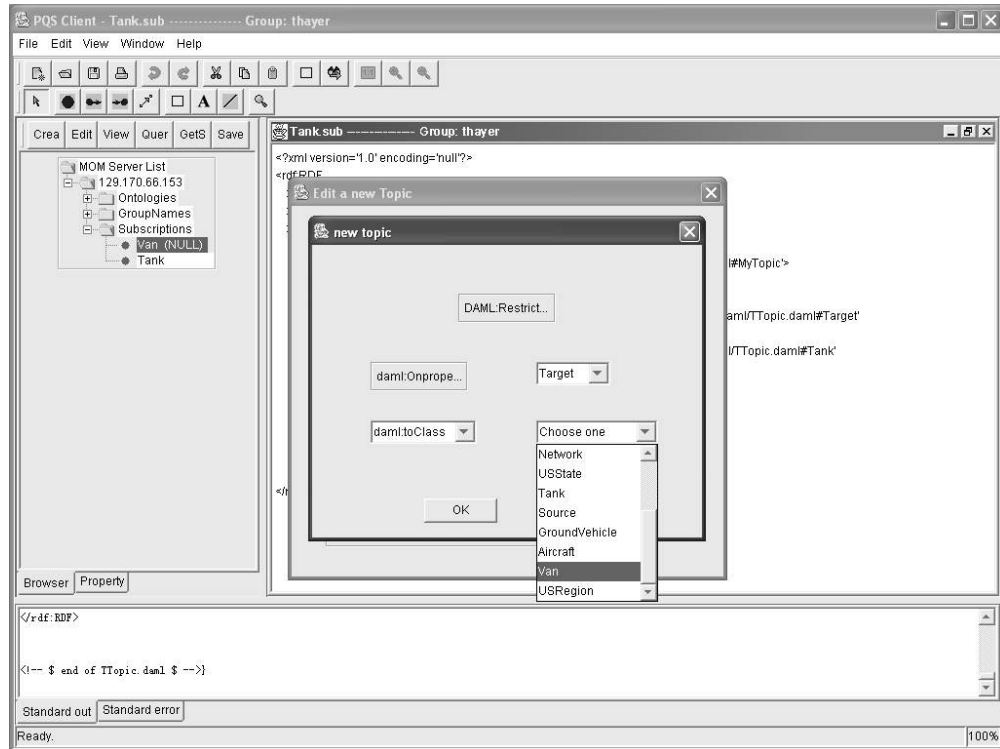


Figure 4 User interface of subscription editor

5. SECURITY ARCHITECTURE

5.1 Security issues

In a large-scale network distributed system, such as Publish/Subscribe Message Oriented Middleware system, information is disseminated across distinct domains, heterogeneous platforms and a large, dynamic population of communication parties. An environment like this raises concerns about the following security issues:

- The identity of each party in a communication should be ensured.
- Network data should not be understandable by an unauthorized third party who intercepts the data.
- Data integrity should be ensured during transit.
- Publisher/Subscriber privilege control.

As introduced before, the JMS does not provide any security features for solving these problems. In our Semantic Message Oriented Middleware system, however the semantic broker server acts as the interface between client applications and JMS provider, which makes the JMS transparent to the client applications. Thus the users of the system do not need to worry about the JMS message delivery.

The security architecture of the Semantic Message Oriented Middleware system is shown in Figure 5, which mainly focuses on providing secure communication between the semantic broker and client applications through a specific port. The information communication between the semantic broker and JMS provider of under layer can be handled inside a private networking environment and protected by other security architectures, e.g. a firewall. By adopting the Secure Socket Layer (SSL) communication, the authentication of semantic broker server side and communication encryption and integrity protection have been achieved. And by username and user information management, we also provide the authorization of the user side and user privilege control.

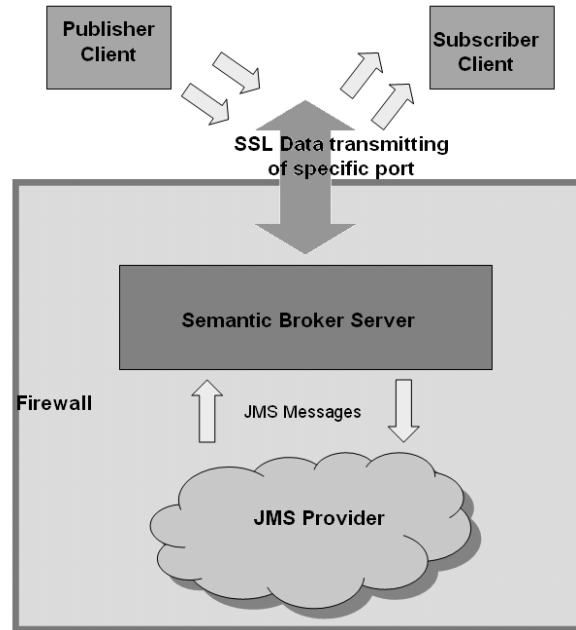


Figure 5: The security architecture of Semantic MOM

5.2 Java Secure Socket Extension (JSSE) and Secure Sockets Layer (SSL)

The Java Secure Socket Extension (JSSE) provides a framework and an implementation for a Java version of the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols, and includes functionality for data encryption, server authentication, message integrity, and optional client authentication [16]. In the Semantic MOM system, we apply the JSSE architecture to implement a secure passage of data between the semantic broker and the clients of publishers/subscribers. Every client communicates with the semantic broker server through a SSL connection by using the public key certificate of the server, so each client can ensure the identity of the server, and then has secure communication.

5.3 Client User Authorization

The server can authorize the user identity by username and password pair. Each client needs to provide its username and password to the semantic broker when it is making a request for publishing/subscribing. The semantic broker will compare them with the information stored in a user file on the server side to check whether the current client is a valid user of the system. If not, the server will deny the request of the client.

Each legal user of the system must have its information, including username and password, recorded in the user file on the server side. The user information file is constructed administratively. The user password is stored in an encrypted form, using the MD5 Message-Digest Algorithm, which makes the passwords unreadable for any unauthorized person who accesses the user information file.

5.4 User Privilege Control

Under some circumstances, users require privacy during the information communication. For example, a publisher may hope that only some specific subscribers can receive his published data, or a subscriber only wants to receive data from some specific publishers. In our Semantic Message Oriented Middleware system, a simple user privilege control feature is provided to solve those problems.

Currently, the user information file on the semantic broker server contains not only information for username and password, but also information for user privilege, which includes accessibility as publisher, accessibility as subscriber, and group names of the groups to which this user belongs. The semantic broker will check that information when handling the request from a client and performing the semantic matching process. The client can only make a request of the same type as its accessibility. Only the publisher and subscriber with the same group name achieve the information exchange between them. The user privilege information is also configured administratively on the server.

6. PERFORMANCE TESTING

Some experiments have been done to evaluate our Semantic Message Oriented Middleware system. We constructed the simulation environment on a single computer that did not connect with other machines by network. It would be ideal to simulate in a large scale real distributed environment, but it is difficult to deploy the system on a significant number of distributed nodes just for simulation, and in an open network environment, and even if we were able to do so, there are other unpredictable factors such as network traffic congestion caused by other network dataflow which can influence the system performance. In our simulation environment, both the messaging server and client applications were running on the computer with different terminal windows, and all the network dataflow was just through this computer itself.

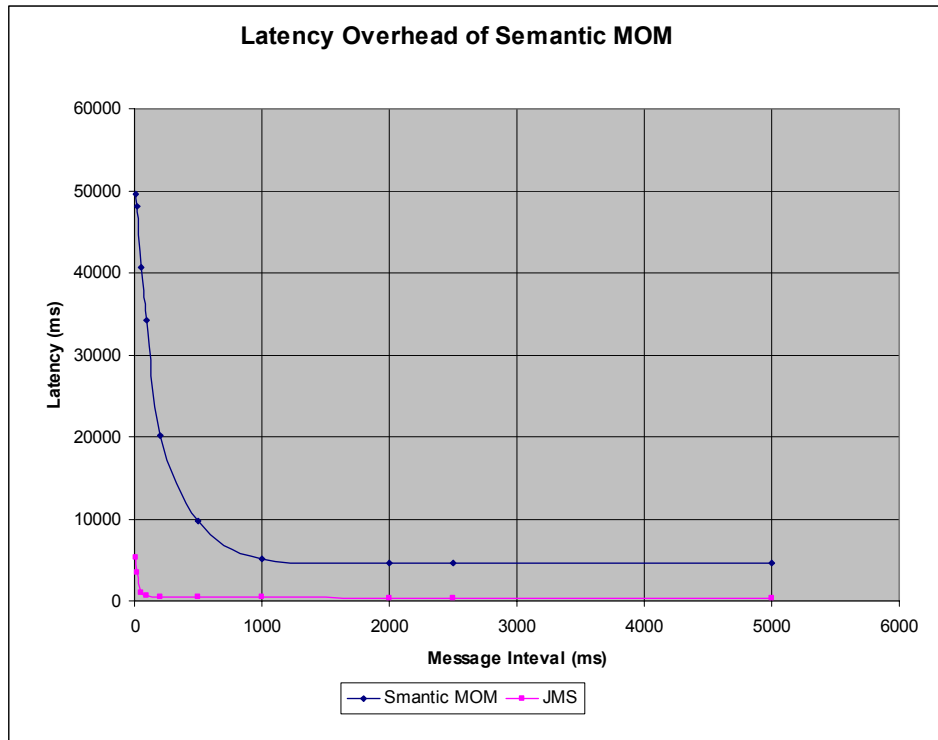


Figure 6: Message transmitting latency of Semantic MOM and JMS

We took average latency of a message from transmission to reception as one measurement of the message delivery capability of the messaging system. Figure 6 shows the simulation data of our Semantic Message Oriented Middleware system and the standard JMS provider (J2EE server) messaging system. The simulations show that there was a message delivery latency overhead of the Semantic Message Oriented Middleware system compared with JMS messaging system. The overhead comes from three main reasons, the first is the extra computation workload for running the semantic broker server beside the JMS provider; second is the workload for maintaining SSL encrypted communication with the clients; The last is the system penalty of multiple tasks running simultaneously within the operation system. The simulation data show that our Semantic Message Oriented Middleware system works stably and provides reasonable performance.

7. CONCLUSIONS

This paper has described our work on the Semantic Message Oriented Middleware system, a subject-based messaging system based on the Java Message Service technology, that provides the capability of semantic topic description and matching, and also provides a specification of message encryption security.

In the semantic broker layer of our system, a simple description logic inference system has been implemented to provide the semantic topic matching service between the publishers and subscribers. By adopting the DARPA Agent Markup Language and Ontology Inference Layer (DAML+OIL) as the topic description language, the event publishers and subscribers can flexibly describe and match their requests based on a pre-agreed ontology that defines the event information spaces. The semantic broker layer also makes the details of the JMS systems transparent to the client applications.

With the Secure Sockets Layer architecture and user information management, the Semantic Message Oriented Middleware system provides secure data communication between the client applications and messaging system.

ACKNOWLEDGEMENTS

This work was partially supported by: ARDA Grant F30602-03-C-0248, DARPA projects F30602-00-2-0585 and F30602-98-2-0107; National Institute of Justice, Department of Justice award number 2000-DT-CX-K001. Points of view in this document are those of the authors and do not necessarily represent the official position of the sponsoring agencies or the U.S. Government.

REFERENCES

1. Brian Whetten, Talarian, *Message-Based Computing: the Fourth Wave of Integration*, <http://www.ebizq.net/>.
2. Roberto Baldoni, Mariangela Contenti, and Antonino Virgillito, "The Evolution of Publish/Subscribe Communication Systems". *Future Directions of Distributed Computing*, Springer Verlag LNCS Vol. 2584, 2003
3. Yin Liu, Beth Plale, *Survey of Publish Subscribe Event Systems*. Technical report, Department of Computer Science (CSCI) at Indiana University, TR574, May 2003.
4. Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, Tushar D. Chandra, "Matching Events in a Content-based Subscription System". *Principles of Distributed Computing*, 1999.
5. Guruduth Banavar, Tushar Chandra, Robert Strom, and Daniel Sturman, "A Case for Message Oriented Middleware". In DISC'99, Bratislava, Slovak Republic, 1999.
6. Antonio Carzaniga, David S. Rosenblum, Alexander L. Wolf, *Design of a Scalable Event Notification Service: Interface and Architecture*. Technical report, Department of Computer Science, University of Colorado at Boulder, August 1998. Technical Report CU-CS-863-98.
7. Guruduth Banavar, Tushar Chandra, Bodhi Mukherjee, Jay Nagarajarao, Robert E. Storm, and Daniel C. Sturman, *An efficient Multicast Protocol for Content-based Publish-subscribe Systems*. In International Conference on Distributed Computing Systems, 1999.
8. Geoffrey Fox and Shrideep Pallickara. "An Event Service to Support Grid Computational Environments". *Concurrency and Computation: Practice and Experience*. Special Issue on Grid Computing Environments.
9. Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. "SCRIBE: A large-scale and decentralized application-level multicast infrastructure". *IEEE Journal on Selected Areas in communications (JSAC)*, 2002.
10. Sun Microsystems, *Java Message Service tutorial*, <http://java.sun.com/products/jms/tutorial/index.html>.
11. DAML language home page, <http://www.daml.org/language/>.
12. Franz Baader, Ian Horrocks, and Ulrike Sattler, *Description Logics as Ontology Languages for the Semantic Web*, Lecture Notes in Artificial Intelligence. Springer, 2003.
13. Description Logics home page, <http://dl.kr.org/>.
14. Daniele Nardi, Ronald J. Brachman, "An Introduction to Description Logics", *the Description Logic Handbook*, edited by F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, P.F. Patel-Schneider, Cambridge University Press, 2002, pages 5-44.
15. Jena Semantic Web Toolkit home page, <http://www.hpl.hp.com/semweb/jena.htm>.
16. Java Secure Socket Extension (JSSE) Reference Guide, <http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html>.