Online Knowledge Center Tools for Metadata Management

Galip Aydin, Harun Altay, Mehmet S. Aktas, M. Necati Aysan, Geoffrey Fox, Cevat Ikibas, Jungkee Kim, Ali Kaplan, Ahmet E. Topcu, Marlon Pierce, and Beytullah Yildiz
Community Grids Lab
Indiana University
Bloomington, IN 47404-3730

Ozgur Balsoy
Computer Science Department
Florida State University
Tallahassee, FL

Abstract

We describe the design and implementation of an XML metadata management system for creating, delivering and managing general metadata. We describe message composition wizards, a multipurpose delivery system implementation, and message access role definitions. This system may be used as the foundation for both human readable messaging (such as newsgroups and citation management systems) as well as event-driven application-to-application systems. We describe in detail the design of an Access Control System.

Introduction

XML [1] can be used to provide a computer platform-, programming language-, and application endpoint-independent way for publishing and browsing metadata, or data about data. We may think of these metadata "nuggets" as messages that may be both synchronously and asynchronously communicated to interested and authorized users. Instead of concentrating on endpoint implementations (some of which may be produced by independent developers), we can develop application metadata formats and use common wire protocols (such as HTTP [2] and SMTP [3]) to transport these as messages. These messages then become events in a general purpose message-oriented middleware system.

In this report, we collect, update, and expand previous descriptions of our system, which are available from Refs [4] and [5]. We have built many applications using the basic architecture described here, including a) a newsgroup system that allows users to post messages, which can then be delivered by email, through a generated web page, or both; b) a training registration system that allows students to register for classes and instructors to post classes and course materials for students; c) a bibtex-based reference management systems that support several related schemas for journal articles, book articles, conference proceedings, etc; d) Basic Interoperability Data Model (BIDM) [6] compatible applications, and e) glossary term and acronym managers. Demonstrations and several running examples are available from http://www.xmlnuggets.org. We are also interested in extending the system to support applications in scientific metadata management. Annotating data provenance, in which metadata is used to describe the history of (real or synthetic) data, is one important example.

We identify the following key constituents of our metadata management system design: an XML message composition tool for creating valid, well-formed messages; an architecture and implementation for delivering the messages to the appropriate interested listeners (supporting both push and on-demand pull delivery systems); and a user role/access control system to define various levels of users and their privileges. The architecture for the entire system is show in Figure 1, with each part described in more detail in following sections.

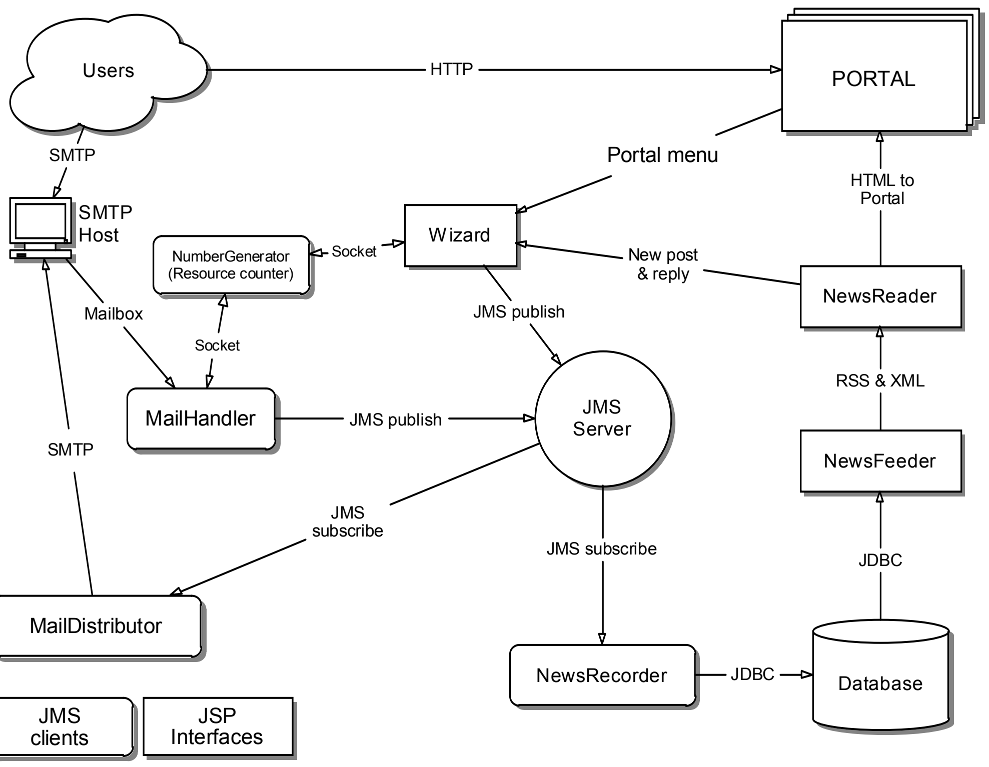

Figure 1 Metadata system architecture

Metadata Composition with Wizards

The first step in our system is creating the XML schemas that define the particular types of metadata instances that we wish to transfer for a particular application. Particular XML messages are instances of these general schemas. The advantage of the XML messaging format is that it only requires the client application to create a valid, well-formed message, which can then be handed off to the system using well established wire protocols. Thus for example a user can create a message posting by hand with any kind of text editor and post through email, as shown in Figure 1. In this case, messages are received by the Mail Handler, assigned a unique ID with the Number Generator, and published to the system through the JMS server.

It is also desirable to allow users to post information to the system through browser interface that ensures that valid schema instances of messages are created. A "wizard" composer can be used to reliably create messages from a particular schema. After defining the necessary schema for a particular application, these schemas have a natural mapping to both user interface components (HTML form elements, Java Swing components, etc.) and data objects (such as JavaBeans). We must thus generate the code for the user interface and the data model.

The process of mapping an XML schema to classes in a particular object oriented programming language is called data binding. This language in our case is Java, and so we wish to cast a particular XML instance into JavaBeans. An XML schema and its instances correspond directly with Java classes and objects. The Java classes have member data corresponding to the schema's elements and attributes and accessor methods ("getters" and "setters") for the data. Tools such as JAXB [7] implementations and Castor [8] for making the conversion, or marshaling, between XML and JavaBeans are available.

The data object bindings may be automatically generated as described above, but we must also develop user interfaces based on the schema that use these data objects. We have initially implemented wizard user interfaces by hand using JavaServer Pages (JSP) [9], mapping the schema elements to corresponding HTML form elements and including corresponding Java data objects generated by Castor in the page.

The next step is to automate the user interface creation. We have developed such a general purpose Schema Wizard system, which we describe in detail in Ref [10] and summarize here. We assume that the content to be generated is based on one or more XML schemas. We provide a set of constraints and directives to schema developers by which they can take control over interface generation. The wizard, then by using built-in mappings from schema elements to HTML form elements, is invoked with schemas to generate user interfaces and necessary source code for data handling and validation. The resulting Web forms are used to interact with users and help generate schema-based and validated XML documents.

The Schema Wizard works by mapping each schema primitive type to a JSP template "nugget" (written in Velocity [11] for scripting) that defines both the user interface and the action. A string element, for example, may be mapped to a text field, an enumeration to radio buttons, and so on. These JSP nuggets can be used to build up displays for more complicated types. The final JSP page for a particular schema is simply the aggregation of all the base JSP nugget types that are needed.

Metadata Publication Mechanisms

XML messages may be posted either through any email client or through the web-based wizard system. The former assumes the user correctly created an XML message in agreement with the schema for the particular application, while the latter (as described in the previous section) creates correct messages. In our system, we use Java Messaging Service (JMS) [12] implementations, with one publisher per message type; i.e. we have a mail handler for incoming email postings and a wizard poster for particular browser wizard applications. As we will describe in the next section, this architecture allows us to decouple message posting and delivery mechanisms and to support multiple delivery of single messages.

The posted message actually consists of more than just the XML message created by the wizard or user. The parts of the final posted message are a) a message header, consisting of the regular email header after being converted to XML; b) the message body; and c) optionally one or more attachments to the posting. Attachments may be binary document files attached to the text message during HTTP upload or email postings. We wrap this entire message (XML posting plus optional email headers and MIME attachments) as a SOAP [13] message with MIME attachments.

Each message is assigned a URI as a unique identifier. This defines a hierarchical, searchable structure for messages: a message can contain child messages (for example, a thread in a news group system). For instance, in a newsgroup application of the messaging system, we can create the reply messages as a child of the message which is replied to. By the URI type unique ID, the attachments can be stored to the unique directory which is related to the unique ID. In this structure, we can also store the messages as an XML file in a directory structure just as we stored attachments. We generate these URIs in an automatic fashion. Top level message names begin with an XML namespace, followed by a number indicating their place in the sequence of posted messages. A child message starts with the name of its parent, followed by a number indicating its place in the sequence of postings to that particular parent.

The message must still be directed to the correct message topic channel. We do this by including a destination tag in the posted message with a URI corresponding to the appropriate message channel (newsgroup topic, for instance).

Metadata Delivery Mechanisms

We need in general to support two sorts of message delivery mechanisms: a "push" model that immediately sends out the message to the subscribing application and a "pull" model that archives posting that can then be recovered on demand. As we illustrate in Figure 1, we use email message delivery as our push system and a database querying system with a browser front end for pull.

As shown in Figure 1, a user may post messages through an email client. This message is received by the Email Handler, which assigns it a unique URI through the ID Generator service and then publishes it to a message distribution hub. Browser wizard postings work similarly. We assign all the channels a unique, hierarchical URI [14]. For example, consider a message group for researcher is Signal Image Processing (SIP). This channel should have gxos://[(unique)destination]/Organization/Newsgroup/SIP. This system can be integrated to the any XML Messaging System. If user want to create new group derived from the SIP message channel, new URI for the new channel is gxos://[(unique)destination]/Organization/ Newsgroup/SIP/SIPArchive created. The administrator of the SIP channel will be automatically assigned to the SIPArchive channel.

We use JMS as our message publishing and subscribing hub. A messaging system will be typically divided into several message channels, such as newsgroup topics or classes in a training system. As shown in Figure 1, we only have one publisher per protocol (message posting wizard for HTTP, for example) and one subscriber per protocol (Email

Distributor for SMTP, for example). These are decoupled, so email postings can be later read through a browser client to the archive. We thus are funneling actual end user publishers and subscribers through a small number of publishers and subscriber proxies. The Email Handler, for example, receives all email postings and is responsible for delivering the posted message to the correct JMS message channel. Thus access control rights are enforced by the publisher and subscriber proxies, which consult an access control service (described in detail in the next section).

Once the message is posted, it will be both immediately sent out and also archived. Immediate notification ("push") is handled by the Email Distributor. The distributor acts as a subscriber to the JMS publishing hub, and contacts the database to get a list of end subscribers that need immediate notification for postings to a particular message channel.

The archival middleware of the messaging system is responsible for recording the messages and providing (feeding) them to the requesters. These two parts of the system are designed to be independent from the message creation part and the message requester part. For that reason, the server should provide us a functionality that the receiver does not need to know anything about the sender. However, the receiver and publisher have to know the message format. The server should also deliver a message to a client only once. JMS provides this functionality for us. The messages are received by the Java Messaging Service in the middleware of messaging system.

The JMS server can be used to communicate events between Java services running on different hosts. These services, such as Email Distributor, may act as bridges to general purpose protocols. The other modules of messaging systems which generate events register as a publisher to the JMS server. The recorder module of the middleware registers to the JMS server as a subscriber to the publication channels which we want to listen. Each message is stored to a database to provide persistency. The unique ID is used to search the database. The Message Recorder module completes its mission by storing the message to the database and the attachment to the directory system. The Message Feeder is completely independent from the recorder part. The requests are received via HTTP GET/POST requests through JSP pages. These requests invoke the feeder to retrieve a message from the database. If the requested information can be found in the database, an XML file is created dynamically. This file includes the information which the requester asked.

The requester makes two kind of request. One is a request which includes the information of the all messages. The response is in RDF Site Summary [15] format. This RSS file includes information such as the link to the original message, the sender name, and the date. The other request is for a specific message, such as a course or a newsgroup posting. To make this request, the requester takes the RSS file and derives the information to request the message body to obtain the desired information.

The Message Displayer uses RSS URI to construct the e-mail/message hierarchy and to get the body of messages. The Message Feeder constructs the RSS file at the request of the message channel. XSL can be used to extract data from XML based message in order to show the required messages to the users. Message Displayer checks the user's access rights by using the database. User access rights allow users to read from and write into message channel topics.

The confirmed message channel can be used to post or read messages. The interface reads the index structure of the message channel by using an RSS Feeder to recover the message IDs in order to index the archived messages.

Channel-Based Access Control

All of our applications are designed around the principle that multiple users will need to access and possibly update various pieces of information in the system, but no two users will necessarily have access to the same sets of data stored in the system. Also, some users must be granted additional privileges for managing access to certain subsets of the data. The channel-based architecture of our system naturally lends itself to this sort of access control, which must be enforced at both the publication points (recorders) and subscription points (readers and distributors) of Figure 1.

Access control systems are important to any system with multiple users, and we highlight some related work. For example, UNIX [16] provides a very simple but powerful form of access control that partially inspires some of our system. Akenti [17] is an access control systems for distributed systems based upon Public Key Infrastructure. An important recent development for Web Services is WS-Policy [18], which provides secure environment for Web Services. Also for Web Services, SAML [19] can be used to convey both authentication and access control assertions in SOAP headers.

Access Control System Requirements

XML metadata may be organized into various categories, or channels. In our newsgroup system, for example, we have many news topics. Each topic includes users with different types of roles, with multiple access privileges. Individuals may also possess different roles in different topics. The default “user” role grants privileges to read and optionally write to one or more message channels. Users have additional options with regard to the choice of message delivery mechanism. That is, a user may request message notification by email, through a web interface, or both. Each channel users is assigned to a role and group. Each role has several privileges (properties). For example, a user may additionally request that attachments to topic postings be sent to him through email. User can make requests to change these properties, which are granted or denied by channel administrators.

Other available roles include “administrator”, and “super-administrator”. Message Channel Administrators have the authority to assign users to a specific message channel. A channel user may have administration privileges over more than one message channel, and a specific channel has one or more administrators. Administrators may modify the access rights of a user, denying a user the privilege of writing to a particular channel, for example. Super administrators manage the entire messaging system. In addition to possessing administrator authorities for all channels, this role has the authority to create new messages channels and assign administrators to them. These roles are summarized in Figure 2.

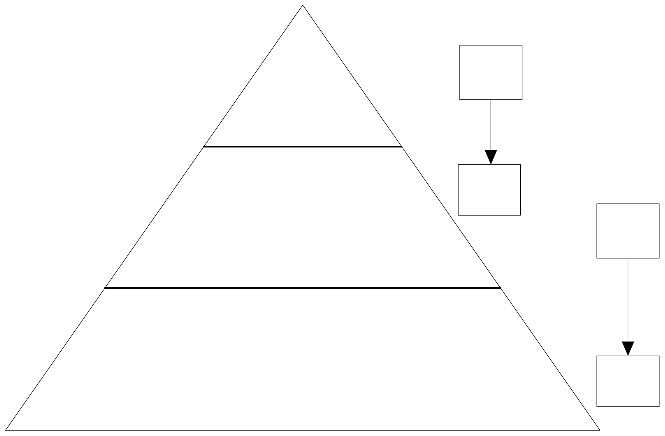

Figure 2 Access roles are arranged in a pyramid structure of privilege.

Figure 3 shows three different use-cases for the access request/confirmation process. The User Request Pool contains requested objects that are initiated by different users of the channels. The Admin Pool captures the request objects and handles the results based on the administrator of the channel. The Result Pool sends users the results of their request. User might have different roles for each news channels. In the figure, for example, Administrator of Ch1 confirmed the user request for both User U1 and User U2 for the channel Ch1. Then, both User U1 and User U2 started to use the channel by having Role R1 in the channel system. However, for the channel Ch2, Administrator of Ch2 confirmed only the User U1's request having a role R1. Administrator of Ch3 having all these control rejects User U3's request for channel Ch3, it is easy to configure user rights for channel, and we have very powerful access control structure for channels having this structure. Different administrators can handle user request objects, which have different roles and different channels.

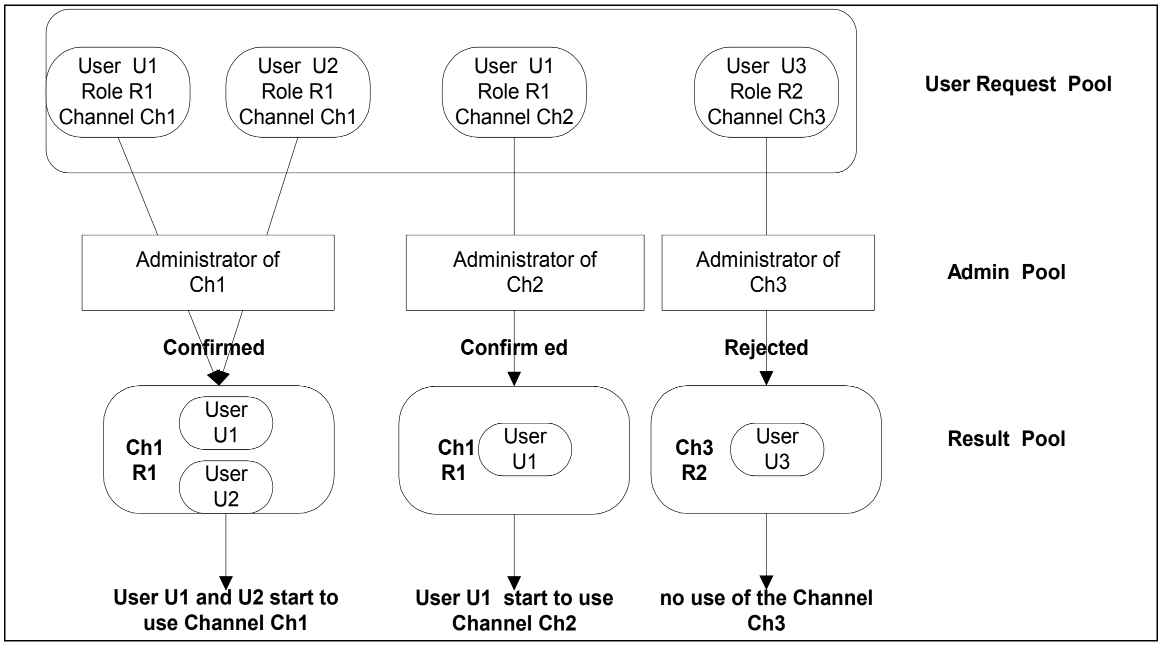
User U1 Role R1 Channel Ch1
User U2 Role R1 Channel Ch1
User U1 Role R1 Channel Ch2
User U3 Role R2 Channel Ch3
User Request Pool
Administrator of Ch1
Administrator of Ch2
Administrator of Ch3
Admin Pool
Confirmed
Confirm ed
Rejected
Ch1 R1 User U1 User U2
Ch1 R1 User U1
Ch3 R2 User U3
Result Pool
User U1 and U2 start to use Channel Ch1
User U1 start to use Channel Ch2
no use of the Channel Ch3

Figure 3 User request and confirmation model.

Web Service for Access Control

Designing systems around a Web services approach has many advantages: protocol independent services, well-defined interfaces for distributed services, and separation of interface from implementation (transparency). Due to transparent services capability of Web services, underlying data-storage may be XML database, relational databases, or flat file system.

The system architecture is shown in Figure 4. The end user sends a request in the SOAP message format. The web server forwards incoming requests to web service engine, which executes the proper web service in the web service pool. The invoked web service ingests the SOAP message, connects to database using JDBC connection, and performs the incoming service request. If there are responses for the incoming service request, it wraps it into SOAP message and passes it to the Web service engine. The Web service engine delivers the output to requestor by passing it to web server.

Let us examine services needed for manipulating our access control model.

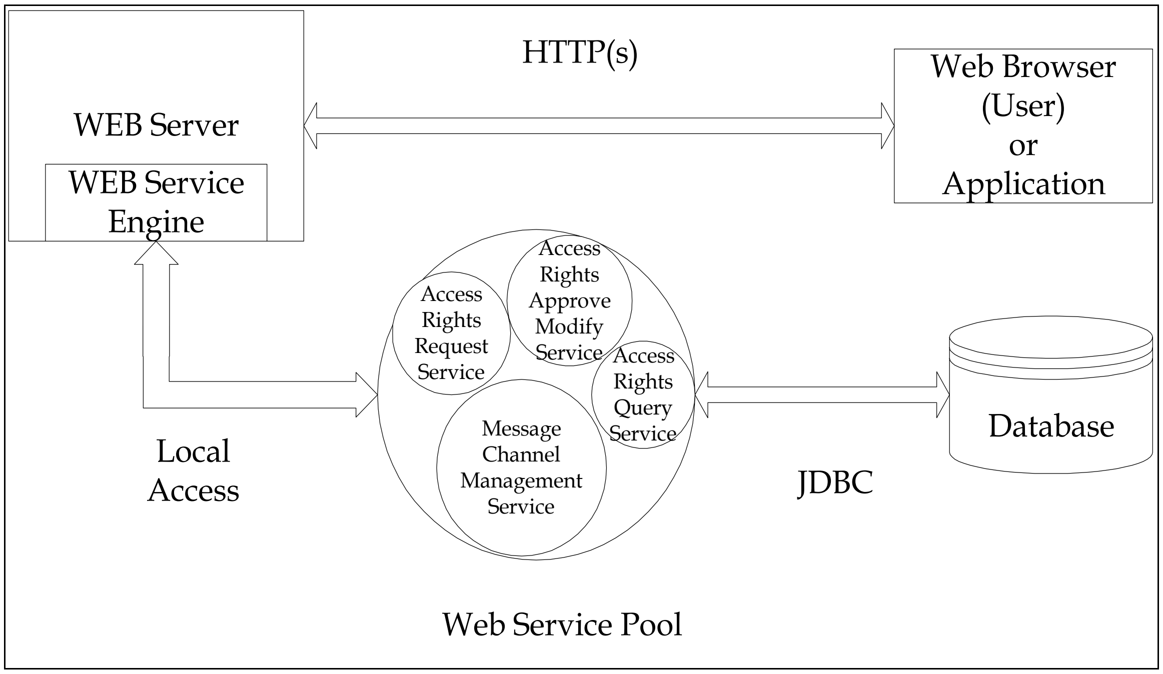
HTTP(s)
WEB Server
WEB Service Engine
Web Browser (User) or Application
Access Rights Request Service
Access Rights Approve Modify Service
Access Rights Query Service
Message Channel Management Service
Database
Local Access
JDBC
Web Service Pool

Figure 4: Interaction of users with the Access Control service.

Access Rights Request Service: This service allows a user to make a request on desired message channel to have proper access rights. For instance, a new user makes a request for read, write, email notify access rights on "java beginners" message channel by calling this service transparently using his/her browser. When Access Rights Request Service has this incoming request, it uses the JDBC to connect to database and enters user's request.

Access Rights Approve, Modify Service: Channel administrators or super administrators use this service. Channel administrator may approve, modify or reject to any request. In addition, he/she may modify the current access rights of the subscribed users by using this web service. Super administrators assign users as a message channel administrators or degrade them by using this service.

Access Rights Query Service: This service is designated for users who do not have right to enter any information (access rights request, modification of access rights, etc.) into database, but they need to access rights information in the database. For example, a message-brokering publisher may need this service to verify the current message sender has write access on given message channel before publishing his/her message. The following are the current methods of this service:
getReadableTopics, getWritableTopics, getAllUsers, getUsersHaveReadAccess, hasReadAccess, hasWriteAccess, getUsersHaveWriteAccess, getAllTopics, hasReadWriteAccess.

Message Channel Management Service: Super administrators use this service. They can add new message channel into system, remove expired message channels from the system and manage their channel administrators by calling this service.

Conclusions

We have presented several aspects of an XML metadata management system: XML instance composition assistance through schema wizards, a system architecture that supports both “push” and “pull” publication mechanisms, and access controls on posting and delivery. We have developed several applications around this basic architecture.

Access Control System is part of a larger security framework, which consists additionally authentication and transport level security. In authentication, the correctness of the user identity is verified. The Access Control System we have presented here depends on an external authentication method. Currently, we implement only HTTP-based authentication. Future versions may incorporate other authentication systems such as PKI, Kerberos or Shibboleth. Data integrity and privacy are provided by transport level security mechanisms such as SSL. Communication between the Access Control System and other entities using the services also needs to use transport level security. Using SSL in all communications provides a sufficient degree of transport level security that concerns about data integrity and privacy.

This work was supported by the US Department of Defense High Performance Computing Modernization Program through the Programming Environment and Training initiative as part of the Online Knowledge Center.

References

[1] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn, “XML Schema Part 1: Structures.” W3C Recommendation 2 May 2001. Available from http://www.w3.org/TR/xmlschema-1/; P.V. Biron and A. Malhotra, “XML Schema Part 2: Datatypes.” W3C Recommendation 02 May 2001. Available from http://www.w3.org/TR/xmlschema-2/.

[2] R. Fielding, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transport Protocol—HTTP/1.1. Internet Engineering Task Force Request For Comments 2616, June 1999. Available from http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[3] J. B. Postel, “Simple Mail Transfer Protocol (SMTP)”. Internet Engineering Task Force Request for Comments 821, August 1982. Available from http://www.ietf.org/rfc/rfc0821.txt.

[4] Galip Aydin, Ali Kaplan, Ahmet E. Topcu, Beytullah Yildiz, Ozgur Balsoy, Marlon Pierce, and Geoffrey Fox ,“An XML Based System for Dynamic Message Content Creation, Delivery, and Control” IASTED International Conference on Information and Knowledge Sharing (IKS 2002) November 18 to November 20, 2002, in St.Thomas, US Virgin Islands.

[5] Ali Kaplan, Ahmet E. Topcu, Marlon Pierce, and Geoffrey Fox, “Access Control System Using Web Services for XML Messaging Systems” in Proceedings of Information Knowledge Engineering (IKE 2003) Las Vegas June 2003.

[6] IEEE Standard for Information Technology—Software Reuse—Data Model for Reuse Library Interoperability: Basic Interoperability Data Mdoel (BIDM). IEEE Std 14201., 1995. See also Basic Interoperability Data Model (BIDM) http://www.nhse.org/RIB/bidm.html.

[7] J. Fialli and S. Vajjhala, “The Java Architecture for XML Binding (JAXB), V 1.0”. See also “Java Architecture for XML Bindings (JAXB)” available from http://java.sun.com/xml/jaxb/

[8] The Castor Project: http://castor.exolab.org/

[9] JavaServer Pages: http://java.sun.com/products/jsp/.

[10] O. Balsoy, et al. “Automating Metadata Web Service Deployment for Problem Solving Environments.” Proc. of *The International Conference on Computational Science,* Melbourne, Australia, 2003.

[11] Velocity Project Page: http://jakarta.apache.org/velocity/

[12] Java Message Service: http://java.sun.com/products/jms/

[13] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer, "Simple Object Access Protocol (SOAP) 1.1." W3C Note 08 May 2000. Available from http://www.w3.org/TR/SOAP/ .

[14] URIs, URLs, and URNs: Clarifications and Recommendations 1.0 21 W3C Note September 2001 Tony Coates Dan Connolly Diana Dack Avaliable from http://www.w3.org/TR/uri-clarification/

[15] G. Beged-Dov, et al. "RDF Site Summary (RSS) 1.0". Available from http://web.resource.org/rss/1.0/spec. See also http://groups.yahoo.com/group/rss-dev/files/namespace.html

[16] F. Grampp and R. Morris, "UNIX Operating System Security", BSTJ, Vol. 62, No. 8, 1984.

[17] S.S. Mudumbai, W. Johnston, M. R. Thompson, A. Essiari, G. Hoo, K. Jackson, Akenti- A Distributed Access Control System, Avaliable from http://www-itg.lbl.gov/Akenti/sc98/akenti.pdf

[18] Web Services Security (WS-Security) Version 1.0 April 5, 2002 Available from: http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-security.asp

[19] SAML Specification Available from http://lists.oasis-open.org/archives/security-services/200106/pdf00002.pdf