

MTR 04W0000092

---

MITRE TECHNICAL REPORT

# Security Guards for the Future Web

## Final Project Report

**September 2004**

**Dr. Nancy Reed, Principal Investigator**

**Research Team: Dave Bryson, James Garriss, Steve Gosnell, Brook Heaton, Gary Huber, Dr. David Jacobs, Mary Pulvermacher, Salim Semy, Chad Smith, John Standard**

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

©2004 The MITRE Corporation. All Rights Reserved.

**MITRE**  
**Center for Integrated Intelligence Systems**  
**Colorado Springs, Colorado**

Approved for public release; Distribution Unlimited  
Case #05-0166

(This page intentionally left blank.)

## **Abstract**

This report documents research efforts for the FY03 and FY04 Mission Oriented Investigation and Experimentation (MOIE) project “Security Guards for the Future Web.” We structured our research into three segments: the browser-based environment, the Web Services environment and the Semantic Web environment. The report includes a description of the prototyping efforts, lessons learned, and recommendations.

**KEYWORDS:** security guard, cross-domain solution, Web Services, Semantic Web

(This page intentionally left blank.)

## Executive Summary

MITRE's clients across the Department of Defense, Intelligence Community, and civilian agencies are migrating to a Web environment as one means to share information. With new and changing missions such as counter terrorism and homeland defense, the number of new mission partners, including foreign partners, is growing dramatically. Traditionally, computer security guards have been used to control what information flows between security domains. Guard technology needs to keep pace with the evolving Web environment.

We conjectured that a family of security guard services would be needed to provide the full range of functionality necessary to support the future Web. We structured our research into three segments: the browser-based environment, the Web Services environment and the Semantic Web.

Our objectives were to investigate how the content and format of data would change in the various Web environments and how mechanisms designed to enforce cross-domain information sharing rules would have to evolve to match that changing content. We also wanted to share lessons learned and recommendations with four audiences: policy makers, guard developers, service and content developers, and cross-domain project management offices.

Our research for the browser-based environment included a requirements analysis and an analysis of alternative architectures for cross-domain information exchanges.

Our research for the Web Services and Semantic Web environments focused on two aspects of guards: the type of data that the guard can transfer and the method used to check the content of items. We decided to focus our Web Services and Semantic Web research on highly structured XML data and guards that would automatically check the content using machine-interpretable information sharing rules. We chose to explore guard designs that supported a wide range of security policies and the ability to easily replace or update the information sharing rules.

Our approach entailed three steps: First, we created an operational scenario for use in our experiment. Second, we built a prototype for experimentation. Last, we conducted tests with the prototype using the generated scenario and captured the lessons learned and results.

We made the following recommendations for the browser-based environment:

- Continue to fund research and development initiatives for the browser-based environment. Focus on three areas:
  - Improvements to reduce the vulnerabilities of browse-down guards, including tools to monitor for possible malicious insiders who are using the browse-down capability to export unauthorized content to a lower domain;

- Improvements to tools that review Web content for possible malicious content; and
- Improvements to tools that make it easier for humans to quickly review Web content for releasability.
- Transform future Web content so that machine review becomes feasible. XML provides the opportunity for achieving this, although schema designers must constrain the XML schemas sufficiently to permit machine review of data.

We made the following recommendations for the Web Services environment:

- Develop schemas for the Web Services requests and responses that are sufficient for guards to use to examine and validate the content of those messages. Security policy makers should provide comprehensive standards and guidelines for the creation of those schemas.
- Maintain a repository of schemas and associated transformation and sanitizing stylesheets that can be shared across the development community.
- Where feasible, urge Web Service developers to avoid designing stateful Web Services.
- Encourage guard developers to take advantage of XML technologies in their future products.

We made the following recommendations for the Semantic Web environment:

- Continue awareness and research in this area.
- Consider research to develop a set of security ontologies.
- Fund research to develop improved ways to translate policy into release rules.
- Encourage ontology developers to defend against ontology corruption wherever possible.
- Consider constraining Web addresses.
- Consider development of a “plug and play” or service-oriented guard.

We believe that the research community should concentrate its efforts in several especially important areas.

- Encourage standards bodies to consider security during their development process.
- Develop outreach programs that educate the Web Service and ontology developer communities on how to maximize their potential for cross-domain utilization.

- Develop tools for policy makers that assist with translating their policies into machine-understandable forms.
- Encourage cross-domain program offices to lead the charge.

## **Acknowledgments**

We received funding for this research from both the Air Force and Army Mission Oriented Investigation and Experimentation (MOIE) programs. Steve Morrissey (ESC/NDT) acted as our Electronic Systems Command (ESC) mission advocate. Doug Poore (AFRL/ IFEB) acted as our Air Force Rome Labs mission advocate. Marianne Bailey and Jeff Duerr (NSA II24) acted as our Army contract mission advocates.

The following people shared information on various government Web Services activities at the November 2003 MITRE Technical Exchange Meeting on Web Services: Earl Dunn, James Garriss, John Kane, Ed Kera, Ron Schaefer, Bud Storck, Julie Surer, Ahn Ta, Larry Thimm, and Jerry Warner.

Jack Sexton helped educate MOIE team members on the use of airspace data and helped develop plausible airspace release rules

Margaret MacDonald provided technical editing support. Wendy Eager provided administrative support in preparing this report.



# Table of Contents

Section	Page
<b>Abstract</b> .....	<b>i</b>
<b>Executive Summary</b> .....	<b>iii</b>
<b>Acknowledgments</b> .....	<b>vi</b>
<b>Table of Contents</b> .....	<b>vii</b>
<b>1. Introduction</b> .....	<b>1</b>
1.1 Problem Statement.....	1-1
1.2 Security Guard Tutorial .....	1-2
1.3 Evolution of the World Wide Web .....	1-3
1.4 Current Situation.....	1-6
1.5 Research Approach.....	1-7
<b>2. Browser-Based Web Environment</b> .....	<b>2-1</b>
2.1 Browser-Based Environment Description .....	2-1
2.2 Investigation Activities .....	2-3
2.3 Browser-Based Results .....	2-4
2.3.1 Need.....	2-4
2.3.2 Observations .....	2-5
2.3.3 Recommendations .....	2-7
<b>3. Web Services Environment</b> .....	<b>3-1</b>
3.1 Web Services Description.....	3-1
3.2 Web Service Guard Approach .....	3-2
3.3 Web Services Design Decisions .....	3-5
3.3.1 Consumer Pull Through Guard .....	3-6
3.3.2 Commingled HTTP/SOAP Agent .....	3-7
3.3.3 No HTTP Confirmation Returned to Senders .....	3-7
3.3.4 SOAP Envelopes Destroyed by the Guard.....	3-7
3.3.5 No Error Returned to Senders .....	3-8
3.3.6 Transport Protocol Support Limited to HTTP.....	3-8
3.3.7 Simple Authentication Mechanism Employed.....	3-8
3.3.8 No Integrity or Confidentiality Mechanisms Employed .....	3-8
3.4 Web Services Mission Use Case .....	3-8
3.5 Web Services Prototype Implementation .....	3-9
3.5.1 GSIX Inbound Agents .....	3-10
3.5.2 GSIX Outbound Agent .....	3-11
3.6 Other Web Services Investigations.....	3-11
3.7 Web Services Results.....	3-12
3.7.1 Needs .....	3-12
3.7.2 Observations .....	3-12

3.7.3 Recommendations .....	3-25
<b>4. Semantic Web Environment.....</b>	<b>4-1</b>
4.1 Semantic Web Description .....	4-1
4.2 Semantic Web Guard Approach .....	4-2
4.3 Semantic Web Design Decisions.....	4-4
4.3.1 Producer Push.....	4-4
4.3.2 GSIX Prototype .....	4-4
4.3.3 OWL-DL Ontology Vocabulary.....	4-5
4.3.4 A Single Input Ontology .....	4-5
4.3.5 Moderate Risk .....	4-5
4.4 Semantic Web Mission Use Case .....	4-5
4.4.1 Process.....	4-6
4.4.2 Release Rules Scenario.....	4-6
4.4.3 Airspace Ontology.....	4-6
4.4.4 Airspace Release Rules .....	4-7
4.5 Prototype Implementation.....	4-10
4.5.1 Design.....	4-10
4.5.2 Tool Selection.....	4-14
4.5.3 Implementation.....	4-16
4.5.4 Testing .....	4-19
4.6 Semantic Web Results .....	4-21
4.6.1 Needs .....	4-21
4.6.2 Observations .....	4-22
4.6.3 Recommendations .....	4-24
<b>5. Information Sharing Activities.....</b>	<b>5-1</b>
<b>6. Summary.....</b>	<b>6-1</b>
6.1 Where We Are .....	6-1
6.2 Future Directions .....	6-2
<b>List of References.....</b>	<b>RE-1</b>
<b>Appendix A - GSIX Description.....</b>	<b>A-1</b>
A.1 Features.....	A-1
A.1.1 Web-enabled.....	A-1
A.1.2 Plug-able Components.....	A-1
A.1.3 Pipeline Content Enforcer .....	A-2
A.1.4 Persistent Logging.....	A-4
A.1.5 Collecting Failed Messages.....	A-5
A.1.6 Testing Messages with the Injector .....	A-6
A.2 Architecture.....	A-7
A.2.1 Registry.....	A-7
A.2.2 Inbound/Outbound Managers.....	A-8

A.2.3 Agents.....	A-8
A.2.4 Handlers.....	A-8
A.2.5 Content Enforcer.....	A-9
A.2.6 Message Queues .....	A-9
A.2.7 Logging and Auditing.....	A-9
A.3 Message Flow .....	A-9
<b>Appendix B - Web Service Prototype Screenshots .....</b>	<b>B-1</b>
<b>Appendix C - Web Service Standards .....</b>	<b>C-1</b>
C.1 XML-Signature .....	C-1
C.2 XML-Encryption.....	C-2
C.3 WS-Security .....	C-3
C.4 SAML.....	C-5
C.5 XACML .....	C-7
C.6 WS-I Basic Profile (BP).....	C-9
C.7 XKMS .....	C-10
C.8 WS-Policy .....	C-13
C.9 WS-Trust.....	C-15
C.10 WS-SecureConversation .....	C-16
C.11 WS-Federation .....	C-17
C.12 WS-Privacy .....	C-18
C.13 WS-Authorization .....	C-19
C.14 WS-Addressing .....	C-20
C.15 XML Technologies for Further Investigation .....	C-21
C.16 List of References .....	C-22
<b>Appendix D - SOAP Header Investigation .....</b>	<b>D-1</b>
D.1 What Are SOAP Headers?.....	D-1
D.2 How Is the DOD Using SOAP Headers in Web Services? .....	D-1
D.2.1 Network Centric Enterprise Services (NCES).....	D-1
D.2.2 Navy Enterprise Single-Sign On (NESSO).....	D-1
D.3 How Does SAML Use SOAP Headers?.....	D-1
D.4 Can SOAP Headers Be Used to Make a Guard State-aware? .....	D-2
D.5 Other Potential Uses for SOAP Headers in the Guard .....	D-3
D.6 List of References .....	D-4
<b>Appendix E - Orchestration and Choreography Investigation .....</b>	<b>E-1</b>
E.1 Difference Between Orchestration and Choreography .....	E-1
E.2 Key Standards.....	E-2
E.2.1 Orchestration.....	E-2
E.2.2 Choreography.....	E-2
E.3 Guarding Orchestrated and Choreographed Services .....	E-3
E.4 List of References.....	E-3

<b>Appendix F - Negotiating Cross-Domain Web Service Information Exchanges .....</b>	<b>F-1</b>
F.1 Questions for the Web Service Provider .....	F-1
F.2 Questions for the Security Managers of the Consumer Environment .....	F-2
<b>Appendix G - Implementation of Airspace Release Rules .....</b>	<b>G-1</b>
G.1 Pruning Queries .....	G-1
G.2 Transformation Rules.....	G-4
<b>Appendix H - Test Cases and Results .....</b>	<b>H-1</b>
H.1 Pruning Supported .....	H-1
H.2 Transformations Supported.....	H-3
<b>Acronyms.....</b>	<b>GL-1</b>
<b>Distribution List.....</b>	<b>DL-1</b>

## Section 1

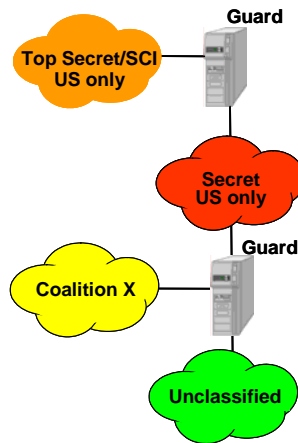
# Introduction

## 1.1 Problem Statement

To accomplish their missions effectively, organizations must often exchange information between security domains (for example, between a Top Secret domain and a Secret domain or between a US Only domain and a Multinational domain). The *Internet Security Glossary* [Shi00] defines a security domain as “an environment or context that is defined by a security policy, security model, or security architecture to include a set of system resources and the set of system entities that have the right to access the resources.” The security policy defines a set of rules governing who and what has access to data and services within the domain.

Our customers operate a number of networks to satisfy mission requirements. Many employ Top Secret Sensitive Compartmented Information (SCI), Secret, and unclassified networks to accomplish their missions. Some organizations also need to operate Multinational networks or connect to networks operated by foreign nations. Historically, organizations operated each of these networks as a system high environment. Within a system high environment, only users with a valid security clearance and formal access approval for *all* information within the domain can access systems within that domain even though they may only have a “need to know” for some of the information contained within those systems. Initially, security accreditors did not authorize any connections between these various system high networks. However, because of critical operational needs to share information between security domains, organizations began to develop and install computer security guards to permit electronic exchange of information *between* security domains. Figure 1-1 depicts the resulting system high, guarded architecture.

To date, attempts to collapse all of the networks into a single Multi-level Secure (MLS) environment have failed. Unless there are **major** breakthroughs in MLS technology, we expect organizations to implement system high, guarded architectures for the near-term future. Organizations may collapse multiple security domains onto a single physical network by relying on encryption to separate the domains. In this case, mechanisms would transfer information between domains by changing the encryption of the information for the appropriate domain(s).

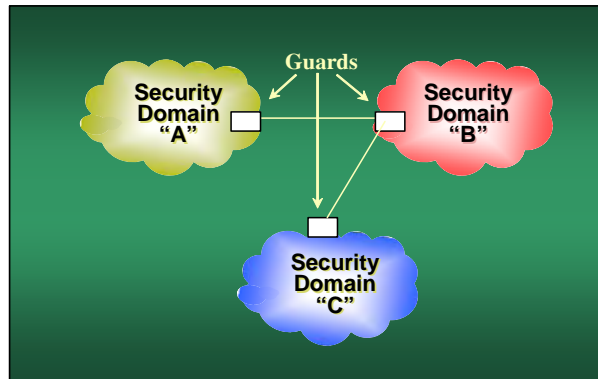


**Figure 1-1. System High, Guarded Architecture**

Within the context of this system high, guarded architecture, organizations are changing how they do business. For years, producers pushed information to specific consumers; security guards supported that “information push” business process. Now, many organizations are migrating to a Web environment as one means to share information. Producers place items in a Web repository and permit consumers to retrieve those items. Consumers want to be able discover information, including information produced within other security domains. Security guards must evolve to support the information sharing policies for that Web environment as well as any changes in the content or format of the data being transferred between domains.

## 1.2 Security Guard Tutorial

As depicted in Figure 1-2, security guards connect two or more security domains to permit transfer of information between domains and enforce information sharing rules associated with the flow of information between those security domains. They also attempt to protect the systems within each security domain from unauthorized intrusion and denial of service attacks. Security guards have more recently been called Cross Domain Solution or Controlled Interfaces. Since we originally used the term “security guard” when we first started the project in 2002, we elected to continue to use that term throughout this report.



**Figure 1-2. Guards Protecting Security Domains**

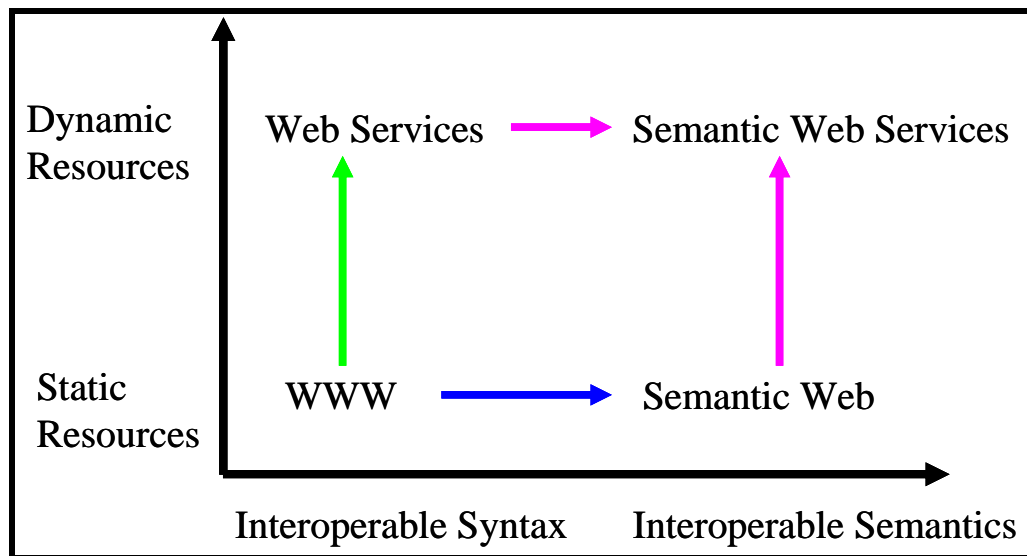
Attributes that characterize guards include:

- **The type of data that can be passed.** Some guards only support transfer of highly structured text; others support transfer of unstructured or semi-structured data.
- **The method used to check the content of items.** Some guards rely on human review of the data contents; other guards use an automated review process.
- **The direction of the data flow.** Some guards transfer data from a lower domain to a higher domain; some transfer data from a higher domain to a lower domain; some transfer data between peer domains; some support bi-directional flows. Guards transferring data from low to high domains are primarily concerned with the introduction of malicious content into the higher domain; guards transferring data from high to low domains are primarily concerned with unauthorized data release. Guards transferring data between peer domains are concerned with both the introduction of malicious content and unauthorized data release.
- **The number of security domains connected by the guard.** Some guards only connect two domains; others connect multiple domains.
- Finally, **the delivery method used to transfer the data.** Some guards use File Transport Protocol (FTP); some use Simple Mail Transport Protocol (SMTP); some use Hypertext Transport Protocol (HTTP); some use other protocols.

### 1.3 Evolution of the World Wide Web

Figure 1-3, derived from *The Semantic Web: A Guide to the Future of XML, Web Services and Knowledge Management* [Dac03], depicts the evolution of the World Wide Web. In the original World Wide Web, depicted in the lower left corner, producers created static Web content using an interoperable syntax called Hyper Text Markup Language (HTML). Human

users interacted with Web servers via client applications called Web browsers. Web servers delivered requested content to the browser using Hyper Text Transport Protocol (HTTP). The Web browsers and Web servers exchanged messages that carried Multipurpose Internet Mail Extensions (MIME)-typed data. In our research, we call this the “browser-based” Web environment. The original World Wide Web provided information sharing opportunities on an unprecedented scale. The World Wide Web’s loosely coupled architecture and its use of standard formats and protocols contributed to its huge success.



**Figure 1-3. Evolution of the World Wide Web**

The green vertical arrow at the left depicts the evolution of the Web from static to dynamic resources. We call this the “Web Services” environment. Web Services permit content to be constructed and tailored on-the-fly, at the time of the request. Web Services adapt the loosely-coupled message-based model for applications which are not browser-based. Web client applications may interact with Web servers without human interaction.

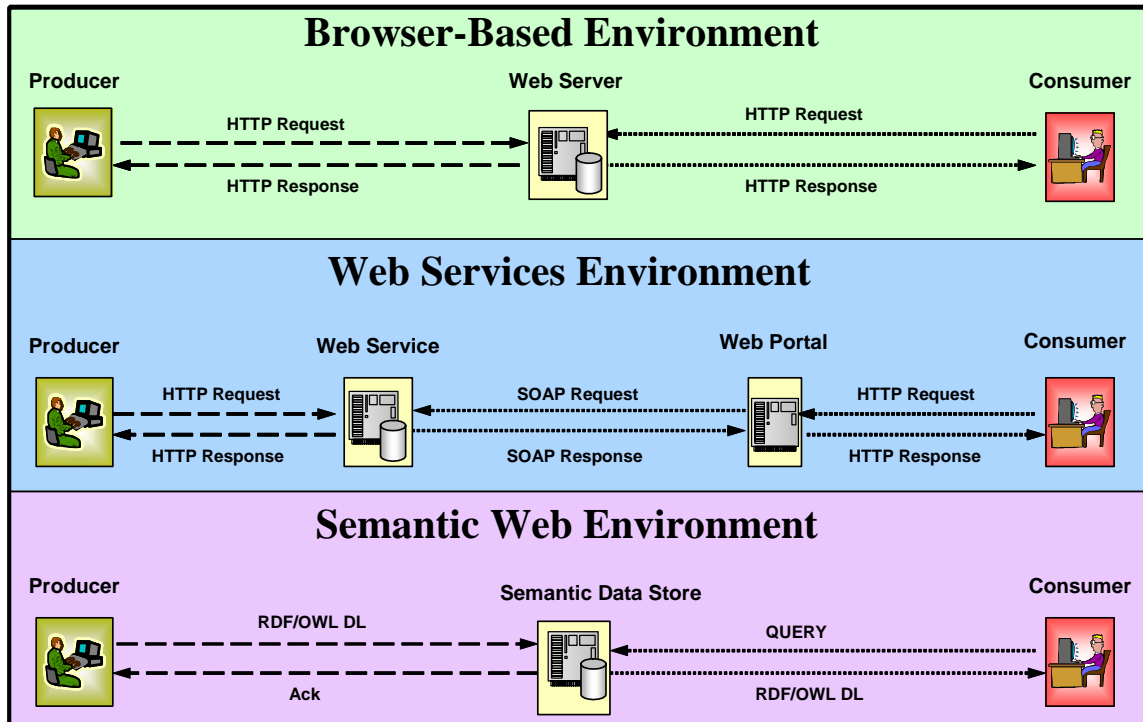
The blue horizontal arrow at the bottom depicts the migration of Web content from interoperable syntax to interoperable semantics, with structure *and meaning* associated with the data. We call this the “Semantic Web” environment. Interoperable semantics permits machine agents to understand and operate on the data with less human intervention. The Semantic Web extends the current Web to give information well-defined meaning, better enabling both people and computers to share and work in cooperation [BHL01]. Programs such as agents, search engines, or service brokers can identify and use Web resources (including both information and services) based on machine-readable representations of their



semantics (meaning). The Semantic Web is a vision of defining and linking information so that machines can display, integrate and reuse data across various applications.

The upper right of the diagram depicts a Semantic Web Services environment with semantically interoperable and discoverable Web Services.

Figure 1-4 depicts the different information flows in the three environments.



**Figure 1-4. Web Information Flows**

The top portion of the diagram depicts the browser-based environment. For our research we assumed that producers used a Web interface to populate the Web server. On the left, a producer submits an HTTP request to post data to the server and receives an HTTP acknowledgement. On the right, a consumer submits an HTTP request to a Web server and receives an HTTP response.

The middle portion of the diagram depicts the Web Services environment. A Web client may interact with the Web Service via a Web portal (as portrayed in the diagram) or may interact directly with a Web Service. We assume that human-to-machine interactions between the consumer and the Web portal continue to use HTTP messages. Machine-to-machine messages between the Web portal and the Web Service are formatted as eXtensible Markup

Language (XML)-formatted SOAP<sup>1</sup> requests and responses. We assume that any human-to-machine interactions between the producer and the Web server continue to use HTTP messages.

The lower portion of the diagram depicts the Semantic Web. Producers populate semantic data stores by creating and posting segments of data formatted using standard ontology vocabularies (e.g., Resource Description Framework (RDF) or Web Ontology Language Description Language (OWL DL)). Users query the semantic data store using standard Semantic Web technologies; their requests refer to a specific ontology. Responses contain segments of data formatted using RDF or OWL DL.

## 1.4 Current Situation

Within the past few years, various government organizations issued directives to “Web-enable” their environments. Organizations responded to those directives by migrating some of their data holdings to a Web environment and by moving human-to-machine interactions to a browser interface. However, the vast majority of Web message exchanges still occur *within a single security domain*. An FY02 investigation into ongoing cross-domain information exchanges within the Combatant Commands [Ree02] identified limited plans to deploy systems which support cross-domain Web exchanges.

A recent survey of Cross Domain Solutions identified a number of products which support the browser-based environment [Ree04]. Some products support transfer of Web content from a producer to a Web repository; these products rely on human review of the content to verify that it can be released. A few Web guards have been deployed which permit consumers sitting in more sensitive domains to browse down and retrieve Web content from less sensitive domains. However, vulnerabilities associated with Web technologies introduce risks. As a result, accreditors have limited deployment of these browse-down guards to bridging domains where risk profiles show limited amounts of risk, or where the operational need is so great that the risk is deemed acceptable. Therefore browse-down products are not broadly available.

Numerous programs throughout the government are beginning to deploy Web Services. However, in a survey of Web Services activities within the government, we only identified a couple of efforts which are beginning to investigate how to extend Web Services to exchange information between security domains.

---

<sup>1</sup> In earlier versions of the SOAP specification, “SOAP” was an acronym for “Simple Object Access Protocol.” SOAP version 2.0 uses “SOAP” as a word, while retaining the previous capitalization.

Academia and industry are actively researching Semantic Web concepts and beginning to build Semantic Web tools. However, in a survey of Semantic Web activities, we identified no efforts that address semantic data exchange between security domains.

Since information exchanges within the Web Services and Semantic Web environments use XML-formatted data, developers could potentially extend guards that support XML to handle Web Services and Semantic Web information exchanges. Products already exist that automatically check highly-formatted text data against machine interpretable information sharing rules. A 2002 survey of products supporting cross-domain information exchanges [Cad02] identified a few developers who were beginning to investigate how existing products could be modified to include mechanisms to support XML-formatted data; none were looking at rules associated with XML-formatted messages used in Web Services or the Semantic Web. Over the past couple of years, developers have started to investigate ways to modify their products to support Web Services.

In FY03, MITRE initiated several projects related to exchanging XML between security domains. A project for the National Security Agency (NSA) examined information assurance issues [Sim03]. A project for the Defense Information Systems Agency (DISA) built a prototype called Guarded Sharing of Information with XML (GSIX), which used XML schemas and Extensible Stylesheet Language Transformations (XSLT) style sheets to enforce information sharing policies for XML-formatted data. An FY04 project for NSA provided guidance on constraining XML Schemas for cross domain solutions [Sim04]. An FY04 project for DISA evaluated commercial XML firewalls and investigated the possibility of combining them with an existing trusted product to provide XML and Web Service guards [ASV04].

## 1.5 Research Approach

Based on our analysis of the evolution of the Web, we structured our research into three segments: the browser-based environment, the Web Services environment, and the Semantic Web environment.

Our objectives and the associated research questions were to:

- Investigate how the content and format of data would change in the various Web environments
- Investigate how mechanisms designed to enforce cross-domain information sharing rules would have to evolve to match that changing content.
- Capture and share lessons learned for four audiences:

**Policy makers** – People responsible for assessing the risk profiles of two domains and determining what constraints must be enforced for a service or information to cross between the domains have a formidable task before them. Our objectives for

this audience were to provide guidance on what risks are mitigated by which constraints, and to make them aware of how these constraints affect the types of service or information that may traverse the guard.

**Guard developers** – Our objective for this audience was to provide guidance on which controls a guard should support to enable enforcement of information sharing policies.

**Service and content developers** – Creators of Web sites, Web Services, and ontological information may inadvertently make it extremely difficult for their information or service to cross a security boundary. Our objectives for this audience were to share information about which practices cause the most difficulty with policy enforcement across domains and to make recommendations on how Web Service or Semantic Web content providers could make it easier for guards to enforce information sharing rules for their data.

**Cross-domain project management offices** – As the overseers of guard development and in some sense clients of service developers and policy makers, we considered the cross-domain project management offices to be in the pivotal position to help disseminate the lessons learned from this research. Our objective for this audience was to ensure that our lessons learned were transmitted to the appropriate groups

Our research for the browser-based environment included a requirements analysis and an analysis of alternative architectures for cross-domain information exchanges.

Our research for the Web Services and Semantic Web environments focused on two aspects of guards: the type of data that the guard can transfer and the method used to check the content of items. We decided to focus our Web Services and Semantic Web research on highly-structured XML data and guards that would automatically check the content using machine-interpretable information sharing rules. In some instances we note where human review of the content would be required.

Our customers live in a dynamically changing environment where information sharing rules might change based on the political situation. We therefore chose to explore guard designs that supported a wide range of security policies and the ability to easily replace or update the information sharing rules.

For the Web Services and Semantic Web environments, our approach to achieving these research objectives entailed three steps.

First, for each Web environment we created an operational scenario for use in our experiment. This meant developing a concept of operations for the scenario that provided enough detail to allow the experiment to explore a wide variety of cross-domain issues. Scenario development included generating sample services or an ontology and a rich set of

security policies to enforce. To support development of the security policies we also created a risk spectrum to help us understand the tradeoffs between security and functionality.

Second, we built a prototype for experimentation. To speed development and make the best use of the time available to focus on new issues, we leveraged existing commercial, open source and related MITRE projects. We surveyed the maturity of relevant existing standards and tools and decided on the platform for experimentation.

Last, we conducted tests with the prototype using the generated scenario and captured the lessons learned and results.

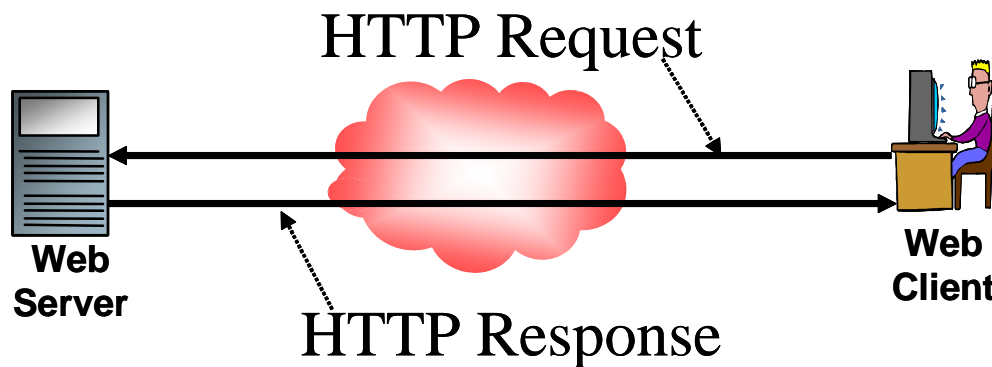
## Section 2

# Browser-Based Web Environment

In this section of the report we describe our exploration of the browser-based Web environment.

## 2.1 Browser-Based Environment Description

Figure 2-1 illustrates a typical browser-based Web environment characterized by human-to-machine interaction. A person using a client application, called a Web browser, initiates a session with a Web server. The Web browser sends an HTTP request to the server and receives an HTTP response. The browser interacts with the server through a series of HTTP requests and associated responses. The browser may ask for information *from* the server. The server response to that request will either be the requested information (or the closest approximation possible) or an error message. The browser may also ask to place information *into* the Web server. The server response to that request will either be an acknowledgement that the information to be posted was received or an error message.






**Figure 2-1. Web Information Exchange within a Single Security Domain**

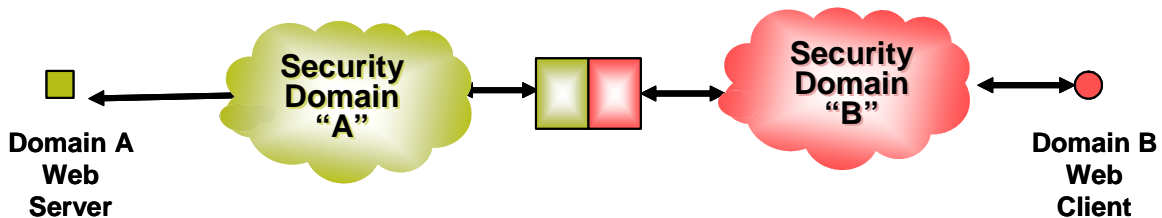
The Web server is an application program that accepts connections in order to service requests by sending back responses. It provides access to Web resources. The *Uniform Resource Identifiers (URI): Generic Syntax* [Uni98] defines a resource as “anything that has identity.” It gives examples of resources such as an electronic document, an image, a service (e.g., “today’s weather report for Los Angeles”), or a collection of other resources. It further explains that a resource is the conceptual mapping to an entity or set of entities, not necessarily the entity which corresponds to that mapping at any particular instance in time. Thus, a resource can remain constant even when its content changes over time, provided the conceptual mapping does not change in the process.

Servers can be configured to support client identification and authentication, access control, and logging of client requests. They can also be configured to provide a secure communications channel between the client and the server to protect the confidentiality of the data being transferred between the client and the server.

In a browser-based Web environment, the Web client hosts a browser application. The browser is a hypertext document display application designed to retrieve data from a variety of types of Web servers. The client may also utilize “plug-ins” to support non-native data types (identified in the HTTP header with codes similar to MIME types). The client also provides an interface for accessibility to other services hosted on the Web server. The client may also support the execution of code (e.g., JavaScript or ActiveX). Content downloaded from a Web server as part of, or under direction of, a Web page executes within the client. This content will usually add visible content or interactive functionality to a Web page.

Figure 2-2 shows a notional architecture to accomplish the flow of information between a Web client in one security domain and a Web server in another domain. The following notations are used in Figure 2.2:

-  Cross Domain Transfer Functionality
-  Web Client Functionality
-  Web Server Functionality



**Figure 2-2. Web Information Exchanges between Security Domains**

For discussion purposes, we call the two domains A and B. We do not distinguish whether there is a security hierarchy or not (i.e., whether one domain dominates the other domain). We merely assume that there are policies that need to be enforced concerning the flow of information between the security domains. Such policies might include access control constraints (who is authorized to transfer data), information flow control (what information can flow), content review policies, privacy constraints, and audit and non-repudiation requirements.

In Figure 2-2, a Web client within Domain B exchanges HTTP requests and responses with a Web server in Domain A. Some mechanism (which we call the cross-domain transfer functionality) establishes the connection between the two security domains, then controls the flow of HTTP requests and responses between the Web client and Web server residing in two different security domains.

## 2.2 Investigation Activities

For our investigation, we analyzed requirements and alternative architectures for cross-domain information exchange in a browser-based environment. We documented our investigations in three reports. In the first document, *Requirements for Transferring Information between Security Domains in a Browser-Based Web Environment* [Ree03a], we proposed operational and security requirements to support a Web-based information exchange. The security requirements were based on the Common Criteria [Com99a-c]. Requirements for later generations of the Web, which could feature machine-to-machine interactions, were outside the scope of this document.

In a second paper, *Testing Security Requirements for Transferring Information between Security Domains in a Browser-based Web Environment* [Gos03]), we provided a generic test plan for the security requirements identified in the requirements document.

In a third paper, *Architecture Alternatives for Transferring Information between Security Domains in a Browser-Based Web Environment* [Ree03b], we identified five notional architectures which could support cross-domain information exchanges, as illustrated in Figure 2-3. We depict two security domains: the red domain in which consumers reside and the green domain in which producers reside.

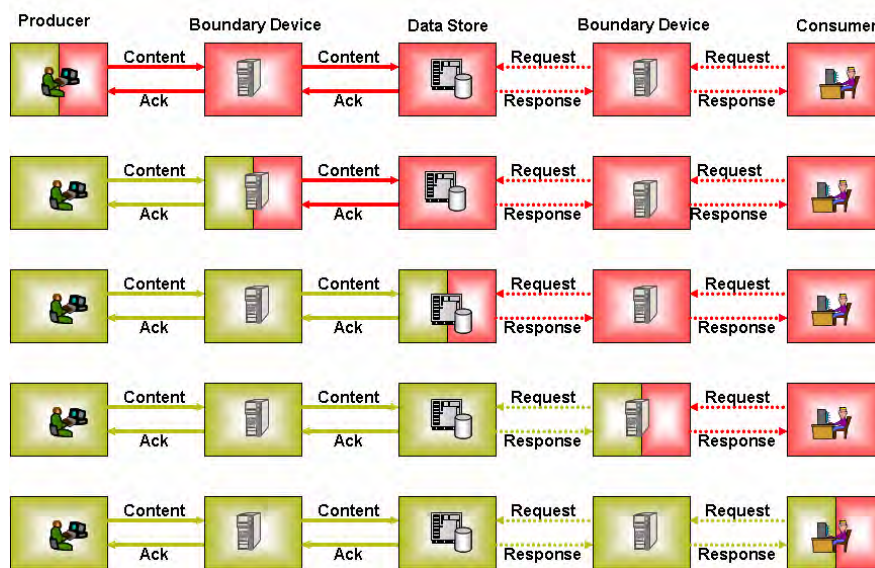
- 1) In the first alternative, producers move information between security domains within their workstations.
- 2) In the second alternative, the information moves between security domains within a boundary device between the producer's environment and a Web repository sitting within the consumer's environment.
- 3) In the third alternative, the information moves between security domains within the Web server (data store).
- 4) In the fourth alternative, the information moves between security domains within a boundary device between the consumer's environment and a Web repository located within the producer's environment.
- 5) In the fifth alternative, consumers move information between security domains within their workstations.

In comparing these alternatives we looked at a number of factors, including:



- 1) Required components within the architecture
- 2) Information flows
- 3) Costs to add a new user (producer or consumer) and to add a new security domain (producer or consumer)
- 4) Advantages and disadvantages related to information assurance, operations, deployment and maintenance.

We also described a variety of additional architectures that combine components from the five basic alternatives.



**Figure 2-3. Architecture Alternatives for Cross Domain Information Exchange**

## 2.3 Browser-Based Results

This section discusses the results of our investigation into the browser-based environment. We divided our results into three categories: needs, observations and recommendations.

### 2.3.1 Need

Today the traditional browser-based Web environment continues to be the most common vehicle for information dissemination. Creating HTML documents has become a ubiquitous enterprise. A tremendous amount of today's knowledge is available in this form. Guards must therefore be developed and deployed to permit exchange of information between security domains in a browser-based environment.

## **2.3.2 Observations**

### **2.3.2.1 Key Requirements**

In our requirements analysis we identified two key attributes that characterize the Web experience for both producers and consumers: the use of HTTP as the transport protocol and the interactive responsiveness of a Web session. We also identified key security requirements related to the following security policies:

- User identification and authentication
- Server identification and authentication
- Access control
- Information flow control
- Privacy
- Audit
- Non-repudiation.

### **2.3.2.2 Multiple Policies**

We concluded that guards may need to enforce multiple policies. If the Web server owner, producers, or consumers belong to different organizations, each organization may have its own policies concerning the use of a particular Web server. One or more guards would therefore have to be deployed to enforce these differing policies.

### **2.3.2.3 Human Review for Release**

For users located in a less sensitive domain to obtain Web content created in a more sensitive domain, the content must be reviewed to verify that it can be released to that lower domain. We concluded that most of today's Web content is not sufficiently structured to permit machine review to determine releasability. Humans must therefore review the Web content to verify that the guard can release it to the lower domain. This introduces delays in the release process and creates a bottleneck in sharing Web content with users in lower domains.

### **2.3.2.4 Malicious Content**

Web content, especially pages with active content, may introduce malicious content into the consumer's environment. Security accreditors will have to carefully balance the benefits of the functionality provided by this content against the risks introduced by permitting that content to be imported into the consumer's environment.

### 2.3.2.5 HTTP Security Vulnerabilities

We identified several possible security issues with HTTP:

- Consumers could use the text of the Uniform Resource Location (URL) as a covert channel to export unauthorized content from their domain to the server's domain.
- Policy makers may wish to limit who is authorized to post information to servers. We originally conjectured that this could be accomplished by limiting the HTTP methods that users would be authorized to employ. For example, producers would be authorized to use the POST method; consumers would not be authorized to use the POST method and would thereby be prevented from uploading information to the server. Unfortunately, HTTP methods provide broad functionality. Although the HTTP GET method is *primarily* used to retrieve information from a server, it can *also* be used to post information to the server. Consumers could therefore utilize this mechanism to export unauthorized content to the server or to export malicious content to corrupt the server.
- If a security policy requires a Secure Sockets Layer (SSL) connection between the Web server and the Web client to protect the confidentiality of the data being exchanged, a guard sitting between the Web server and the Web client would either have to permit the encrypted data to be passed through the guard without inspection or would have to be able to decrypt and examine the content and then reencrypt it. This could introduce a "man-in-the-middle" attack opportunity.

### 2.3.2.6 Architecture Alternatives

In our analysis of architecture alternatives, we concluded that selecting an architecture appropriate for a particular situation was a complex process. There is no clear-cut preferred solution. Some of the factors that affect the decision include:

- The size of the user population and how many users are producers versus consumers. The cost of some solutions increases as the user population increases; in other solutions, the cost is independent of the size of the user population.
- The number of security domains that must be supported and whether they are producer domains or consumer domains. The cost of some solutions increases as the number of security domains increases; in other solutions, there is only a small additional cost to add new domains.
- Which components within the architecture the organization will control. If an organization wishes to enforce specific cross-domain policies, then the architecture alternative it selects should include a cross-domain component that it can control. For example, if a production organization does not control the Web server, but has security policies concerning the release of its data, then that producer organization

should consider implementing either Cross-Domain Producer Workstations or a Cross-Domain Producers' Boundary Device. If a consumer organization does not control the Web server, but has security policies concerning the import of data, then that consumer organization should consider implementing either Cross-Domain Consumer Workstations or a Cross-Domain Consumer's Boundary Device.

### **2.3.3 Recommendations**

#### **2.3.3.1 Research and Development**

The browser-based environment will probably remain dominant for a long time into the future. The government should therefore continue to fund efforts to improve and refine solutions for this environment. Future development activities should focus on three areas:

- Improvements to reduce the vulnerabilities of browse-down guards, including tools to monitor for possible malicious insiders who are using the browse-down capability to export unauthorized content to a lower domain;
- Improvements to tools that review Web content for possible malicious content; and
- Improvements to tools that make it easier for humans to quickly review Web content for releasability.

#### **2.3.3.2 Machine-reviewable Web Content**

Delays associated with human review of Web content will continue to hamper sharing of Web content from a more sensitive domain with a less sensitive domain. Future information sharing approaches must devise ways to permit a greater percentage of information to be reviewed and shared automatically. One possible solution is to transform Web content so that it is more structured, opening the opportunity for machine review and release of that content. The government should therefore actively explore opportunities to migrate Web content from HTML to more structured formats, such as XML-formatted data. Data designers must then find ways to sufficiently constrain those XML schemas to permit machine review of the data.

## Section 3

# Web Services Environment

### 3.1 Web Services Description

Web Services are software services that can be accessed by software clients at a Web address (i.e., a Uniform Resource Identifier (URI)) over the Web. They are based upon XML technologies, including Universal Description, Discovery and Integration (UDDI) for discovery, Web Services Description Language (WSDL) for description, and SOAP for messaging (see Table 3-1). Most Web Services use HTTP for transport, but other protocols such as Simple Mail Transfer Protocol (SMTP) can also be used. Advantages over previous client/server technologies include:

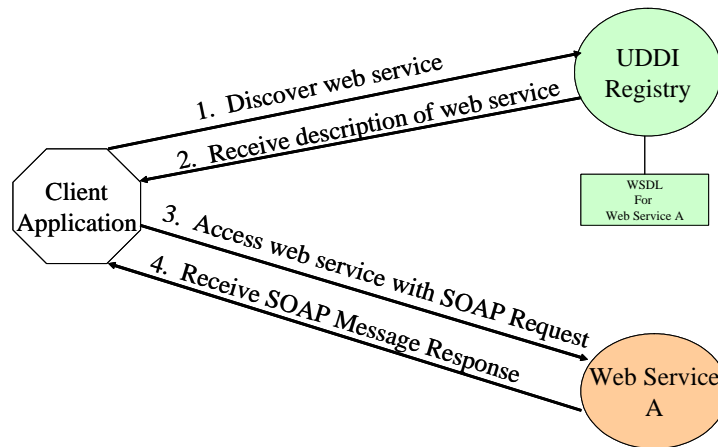
- Loosely coupled – Web Services are independent of operating system, programming language, middleware platform, and application server.
- Standards based – Web Services are based upon Web standards recommended by organizations such as the World Wide Web Consortium (W3C) and Organization for the Advancement of Structured Information Standards (OASIS).
- Broad industry support – Every major company, including BEA, IBM, Microsoft, Oracle, and Sun, supports and/or develops Web Service standards.

**Table 3-1. Layers of Web Services Standards**

	<u>Technology</u>	<u>Overview</u>
Discovery	UDDI	A registry service that allows Web Services to advertise their services
Description	WSDL	A language that describes Web Services and how to access them
Messaging	SOAP	A protocol for accessing a Web Service
Transport	HTTP, SMTP, etc	Transport protocols; HTTP is for networking and SMTP is for email

In our Web Services investigation, we elected to implement a specific transport protocol (HTTP), but we focused our research on an approach where SOAP requests pass from a client through a guard to a Web Service and associated SOAP responses are returned.

Figure 3-1 shows one possible scenario of Web Services in use. A client application, needing some service, searches a UDDI registry. The registry returns information that the application needs to find and use a Web Service with the required capabilities. The client application sends a request to the Web Service, which replies with the desired response.



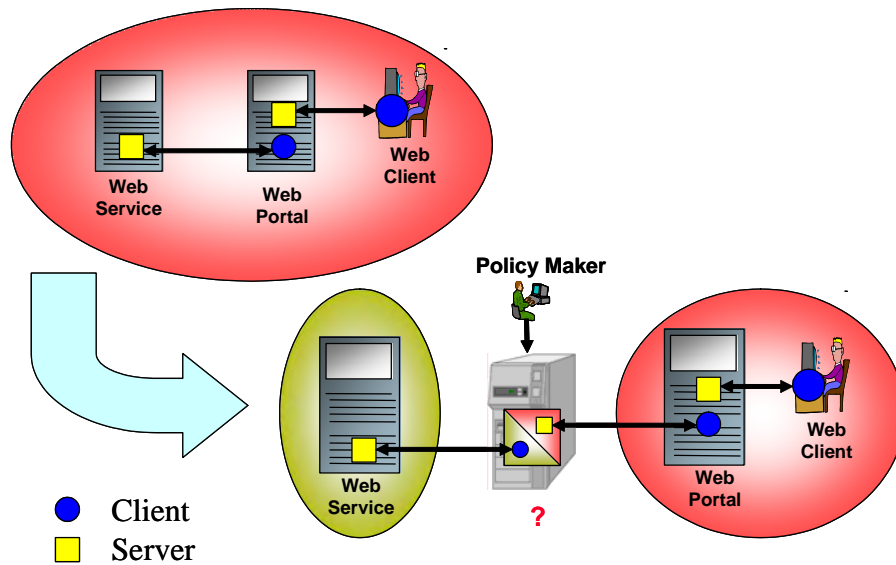
**Figure 3-1. Web Services Scenario**

### 3.2 Web Service Guard Approach

Figure 3-2 depicts an information exchanges for a Web Service within a single domain and then shows how an information exchange changes when the Web Service resides in one security domain and the Web client resides in another.

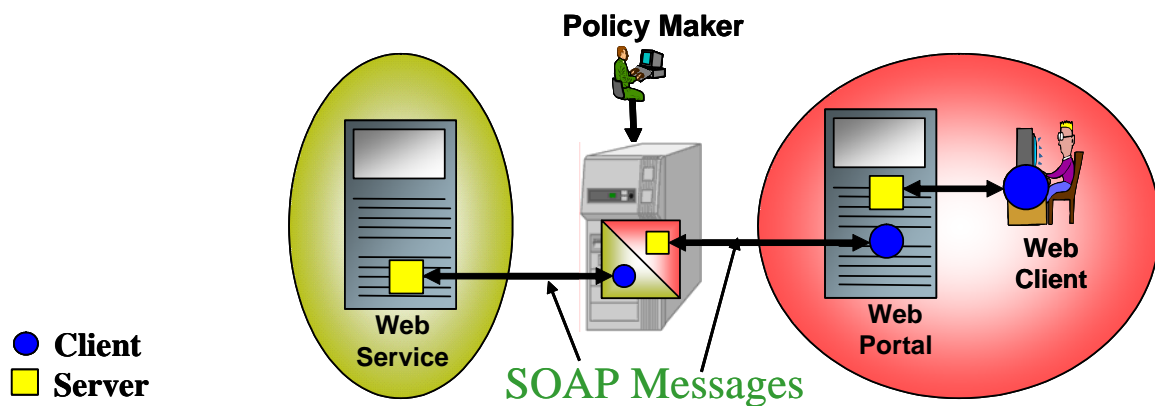
The upper portion of the diagram shows an information exchange within a single domain. A Web client makes a request to a Web portal, which in turn submits a request to a Web Service. The Web Service responds to the portal, which returns the answer to the Web client. Information exchanges flowing between the Web portal and the Web Service are XML-formatted SOAP messages.

The lower right shows the situation where the client resides within a different security domain than the Web Service. We assume that a security guard sits between the two domains that permits the client's request to flow to the other security domain and the response to be returned. A policy maker dictates who can make the requests, which services clients can access, what information can be included in the request, and what information can be returned in the response.



**Figure 3-2. Single Security Domain versus Cross-Domain Web Services**

In our research we explored what happens within that guard if the information exchanges use Web Services. We explored both synchronous and asynchronous Web Services. Figure 3-3 shows the configuration for a guard that would support a synchronous Web Service.

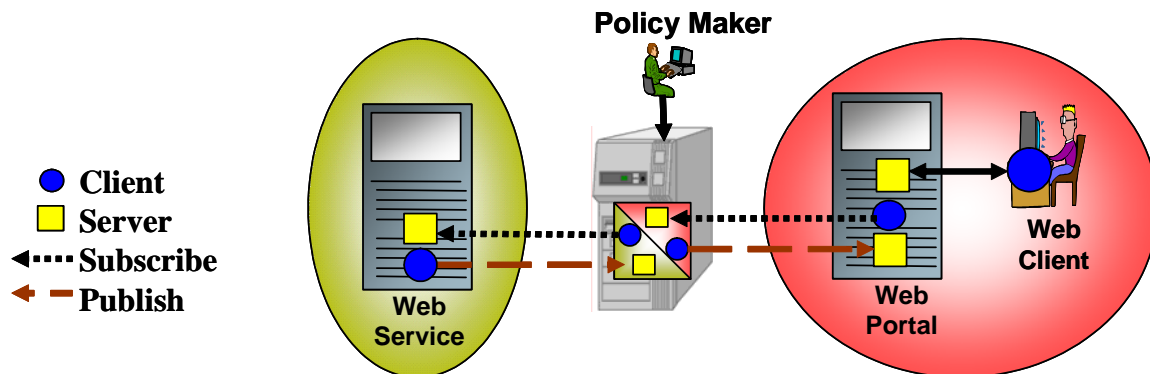


**Figure 3-3. Synchronous Web Service**

In this situation, the client issues a request through the guard to a Web Service in another security domain and receives an immediate response to that request. This is similar to the browser-based Web environment but now SOAP messages are exchanged instead of HTML-tagged data. In a synchronous Web Service clients always initiate the information exchange and the service simply responds to that request. As there are differences in policy and

protection mechanisms depending on the direction of the data flow, we explored synchronous Web Service guards for both a high-side client and a low-side client.

Figure 3-4 shows the configuration for a guard that would support an asynchronous Web Service. In this situation, the client issues a request through the guard to a Web Service in another security domain. At some later point the service issues a response to that request and returns it to the requesting client. This configuration would support a publish-and-subscribe scenario. The client would subscribe to a particular service; the service would return material to the client as it is published. This configuration requires a more complex configuration within the guard, because it must allow sessions to be initiated from either side of the guard and to associate responses with requests. We explored an asynchronous Web Service approach using both a high-side client and a low-side client.



**Figure 3-4. Asynchronous Web Service**

In our research, we identified a set of core components necessary within guarding devices. However, depending upon the particular security policy being enforced, not all components may be needed. These core components include:

- **User identification, authentication and authorization** to identify and authenticate the user making the request and determine whether that user is authorized to make that request.
- **Workstation identification and authentication** to limit which workstations are used for cross-domain interaction. For example, this might be used to permit users to make cross-domain requests from their offices, but prevent them from making the same requests from an Internet café.
- **Server identification and authentication** to ensure that the clients are talking to the server they expect and are not being spoofed by another server.

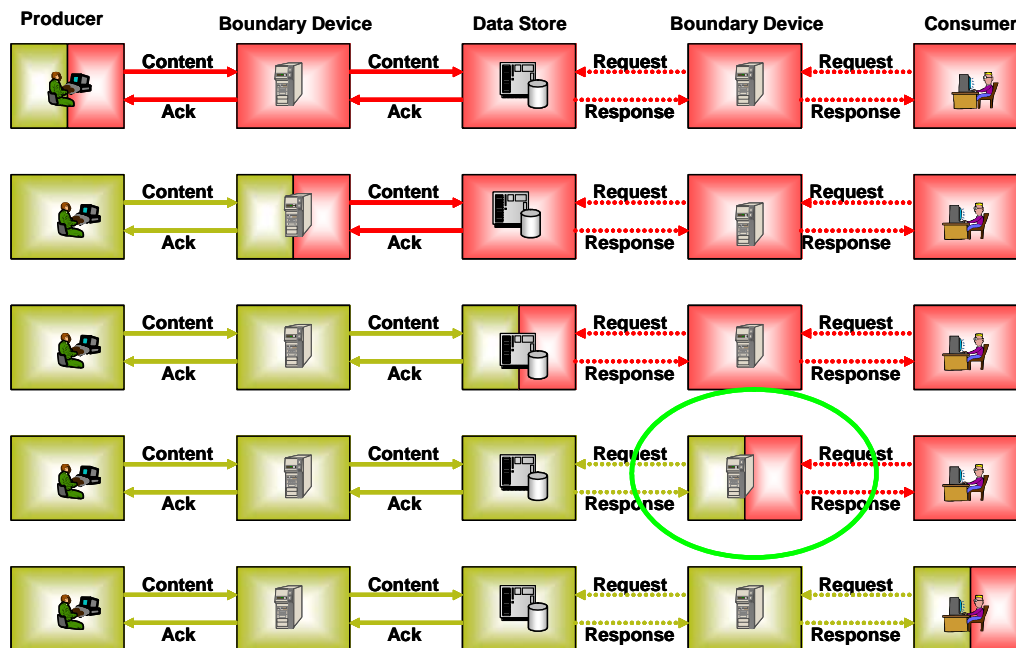


- **Message content validation** to ensure that only appropriate content is moved between security domains. This can be used to prevent data spills for outbound data transfers and limit the introduction of malicious content into an environment for inbound data transfers.
- **Sanitization** to remove information from the package that is not permitted to flow between security domains. The sanitized information may be classified higher than allowed, or it may be malicious content.
- **Namespace and URL changes** to map namespaces or URLs between two different domains, or to hide resources or users from the other domain.
- **Message integrity validation** to ensure that the package of data has not been changed from the time a human reviewed it and determined it was acceptable for release.
- **Transport validation** to limit the functionality provided by the transport protocol. For example, the policy might limit the HTTP methods that are permitted to be used to move information between security domains.

### 3.3 Web Services Design Decisions

The architecture alternatives identified for a browser-based environment also seemed to fit for the Web Services environment. Although a Web portal might be added to the architecture, we presume that any Web portal resides within the consumer's security domain.

From the perspective of the five basic architecture alternatives described in Section 2, we elected to investigate the fourth alternative – where the cross-domain transfer occurs in a security guard between a Web Service and the client (Figure 3-5). We presumed that there might be a production process behind the scenes in which a producer created information and placed it into a data repository. The Web Service might obtain information from that repository or create and deliver content directly to the consumer.



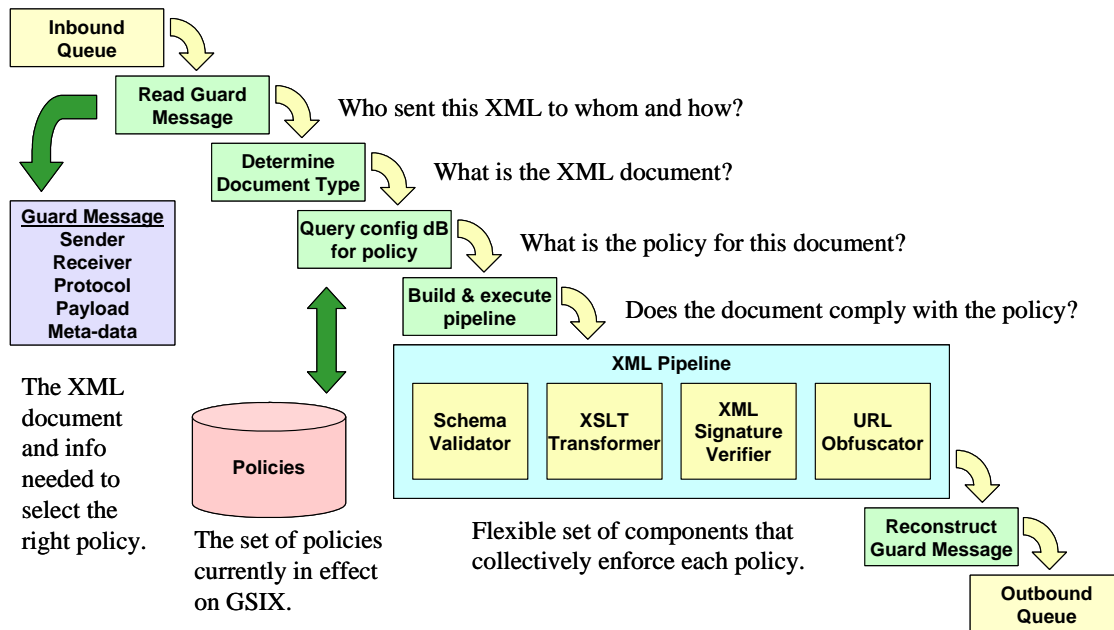
**Figure 3-5 Web Services Architecture Design Decision**

During our first year of research, we built a rudimentary prototype to begin our Web Services investigation. For the second year, we decided to build on the Guarded Sharing of Information in XML (GSIX) framework to develop a more robust Web Services prototype. This prevented the team from constructing a framework from scratch and provided nearly 80% of the needed functionality. GSIX was designed to process XML messages between domains. It provides a loosely coupled architecture where messages are processed as they flow through pluggable components (Figure 3-6). (For more information on GSIX see Appendix A.)

However, GSIX was not initially designed to support Web Services. In order to modify it to meet our needs, we made design decisions that would minimize any unnecessary rework and allow us to focus on the core research questions. The subsections below describe key design decisions.

### 3.3.1 Consumer Pull Through Guard

We focused on the situation where information flowing through the guard originated as a request from an information consumer. Therefore we designed the prototype to proxy the request through the guard, block the response until the request was processed, apply policy to the returned document, and finally send the response back to the original requester (see Figure 3-6).



**Figure 3-6. Architecture of the GSIX Content Enforcer**

### 3.3.2 Commingled HTTP/SOAP Agent

The GSIX architecture did not support the chaining of protocol handlers for this experiment. Therefore, although HTTP is the transport protocol and SOAP is the separate message protocol, we combined the handling of the two into a single agent.

### 3.3.3 No HTTP Confirmation Returned to Senders

The GSIX architecture did not support messages formats other than XML. Therefore, our prototype did not relay back to the client any HTTP-only responses to an asynchronous HTTP/SOAP message.

### 3.3.4 SOAP Envelopes Destroyed by the Guard

GSIX requires complete, well-formed XML documents. In order to apply the guard policy to the actual document, the guard must first extract it from within the SOAP envelope. Therefore, the guard does not transfer the original envelope to the destination domain. Instead, it destroys the envelope upon receipt and recreates a new one on the other security domain so that only the wrapped document crosses the guard. Because of this, SOAP headers in the envelope will not move through the guard.

### **3.3.5 No Error Returned to Senders**

SOAP errors cannot flow through GSIX. Thus, the user sending a SOAP message through the guard does not receive notification of an error from the destination domain that affects the SOAP message.

### **3.3.6 Transport Protocol Support Limited to HTTP**

The prototype used the HTTP protocol for transferring documents across the guard, since it is the most common protocol used by Web Services.

### **3.3.7 Simple Authentication Mechanism Employed**

The prototype provided a simple authentication scheme by requiring a username for the message to move through the guard. However this is not robust enough for an enterprise environment. Future research could focus on the use of XML-Signature and perhaps Security Assertion Markup Language (SAML). Appendix C contains more information on XML-Signature and SAML.

### **3.3.8 No Integrity or Confidentiality Mechanisms Employed**

We elected not to employ integrity or confidentiality mechanisms within our prototype. Future research should focus on the use of XML-Signature and XML-Encryption to provide necessary integrity and confidentiality mechanisms.

## **3.4 Web Services Mission Use Case**

Our mission use case focused on synchronous Web Services. To demonstrate the case of a cross-domain synchronous Web Service, we developed a notional gazetteer Web Service with clients in another domain accessing that service.

Given a location name, the gazetteer service returns data about that location (latitude, longitude, and feature type). Our information sharing policy stated that certain users could access the entire contents of the gazetteer, while other users are allowed to see only a portion of the data.

The purpose of the demonstration was to show how different information sharing policies could be configured within the guard to affect the information flows for the Web Services. We created four notional users, representing four different consumer security domains, each with different authorizations. Those authorizations are summarized in Table 3-2.

**Table 3-2. User Permissions**

<b>User</b>	<b>Access</b>	<b>Access Control Mechanism</b>
User1 – BrookHeaton	Full access	Guard performs no transformation of response—it allows the response to flow through unchanged
User2 – DavidJacobs	Feature type denied	The guard strips the feature type from the XML response using an XSLT configured in the guard for the user
User3 – NancyReed	Latitude/Longitude denied - nothing is mapped	The guard strips Lat. and long. from the XML response using an XSLT configured in the guard for the user
User4 – ChadSmith	Everything denied	The guard denies any access to the Web Service for this user

To demonstrate the guard functions for the above users making requests to the gazetteer Web Service, we developed a Web client with a mapping function. The Web client allows a user to enter a search term, which in turn invokes the Web Service. For example, when BrookHeaton enters a term, the term is sent through the guard as a request to the gazetteer, which generates a response. Since BrookHeaton is fully authorized, the response is returned through the guard to the client with no changes and the data is displayed on the map. NancyReed is not authorized latitude and longitude information. The guard strips the latitude and longitude from the response. The client displays the feature type but does not display the latitude or longitude, thus the client is unable to indicate the location of the feature on the map. Example screenshots from this use case can be found in Appendix B.

### **3.5 Web Services Prototype Implementation**

The GSIX architecture uses software agents to handle incoming and outgoing messages for each protocol. These agents are responsible for translating protocol-specific representations of documents and metadata into and from GSIX guard messages. To add support for HTTP/SOAP, we developed inbound and outbound agents supporting this protocol pair. The working prototype, including the source code and other associated information, is available

to MITRE employees at the Internal Source Forge (<http://developer.mitre.org/projects/gsix/>) as part of the “GSIX3” package. Non-MITRE personnel can request a copy of the code from the author.

### **3.5.1 GSIX Inbound Agents**

Two aspects of the GSIX architecture greatly simplified the implementation of the inbound agent. First, since GSIX did not support the chaining of agents for handling embedded protocols, a single agent had to handle both the HTTP and SOAP. Second, since GSIX already had an embedded Web server with support for Java Servlets, we elected to implement the inbound agent as a servlet. An additional benefit to implementing the inbound agent as a servlet was that it could be integrated into the SOAP exchange as an HTTP proxy. This manner of insertion was minimally invasive to the message exchange and in line with standard industry practice.

Processing for synchronous and asynchronous services differs. However, the inbound agent had no way to identify the message type through examination. We therefore developed two different inbound agents to handle the two cases.

The HTTP/SOAP agent handles inbound messages in a linear fashion. Before any additional processing occurs, the servlet attempts to parse the incoming HTTP request as a SOAP XML message. The parser ensures the message is well formed XML, has the selected SOAP namespace, and has one and only one body. We did not perform a complete validation of the XML document within the inbound agent because the content enforcer would destroy the SOAP wrapper and validate the body of the message.

After the inbound agent parsed the XML portion of the message, it collected required metadata for the GSIX message from the HTTP headers. First, the agent determined the identity of the sender from the authorization header. Then, the agent determined the intended recipient of the message in the destination domain by combining the host and relative URL headers.

With the SOAP message’s body and relevant metadata available, the inbound agent constructed a GSIX guard message, and populated the appropriate fields. The agent then placed this message on the pipeline to the inbound manager. If any of these steps failed, the inbound agent sent a standard HTTP error back to notify the client that a failure occurred.

Up to this point, both the synchronous and asynchronous implementations remained the same. However, as indicated in Section 3.3.3, GSIX did not support sending HTTP-only messages through the guard. After handing the SOAP message off to the inbound manager, the asynchronous agent sent an HTTP 200 response to the client and concluded its processing. The synchronous inbound agent continued to wait for a response message from the Web Service and then transmitted it back to the client. The inbound synchronous agent acted in a similar fashion to the outbound agent by recreating a SOAP message using the

GSIX guard message. However, unlike the outbound agent, the inbound agent returned the response message to the client on the original connection rather than by opening a new connection.

Given the design decision to prevent error messages from being returned to the client, our prototype client had no way to determine if the system failed to generate an expected response message due to protocol failure or denial of release by the content enforcer. Without this error reporting, we assumed that the client would simply time out after a reasonable length of time, concluding that the message had, at some point, failed.

### **3.5.2 GSIX Outbound Agent**

The outbound agent handled SOAP/HTTP messages in a similar fashion to existing outbound agents already within the original GSIX prototype. The outbound agent opened a connection to the intended recipient of the message, whose address was included as metadata in the GSIX message, and transmitted the released message. If a response message was generated, this agent transformed it into a GSIX guard message and submitted it to the content enforcer with the original client noted as the recipient.

The aspect of the SOAP/HTTP outbound agent that differed from the other GSIX outbound agents was that the SOAP/HTTP agent had to recreate the SOAP envelope before sending the message to the recipient. The inbound agent discarded the envelope, so the outbound agent began with an empty SOAP envelope with no header. The outbound agent simply inserted the document from the GSIX guard message as the SOAP envelope's body to reform the SOAP message.

## **3.6 Other Web Services Investigations**

We expanded our research beyond the prototyping effort in several directions. We did not have sufficient time to augment our prototype to incorporate ideas from these investigations.

We examined current and emerging Web Service standards and evaluated their potential impact upon Web Service guards. This research is documented in Appendix C.

We explored various questions related to SOAP headers and their use in a Web Service guard. This research is documented in Appendix D.

We explored Orchestration and Choreography, which are XML-based standards focused on the process of composing and organizing Web Services to accomplish a specific task. This research is documented in Appendix E.

We developed questions that Web Service providers, guard service providers and policy makers from the consumer community would need to discuss when negotiating to permit Web Service information flows through a guard. This research is documented in Appendix F.

## **3.7 Web Services Results**

This section describes the results of our experimentation using a Web Service security guard prototype. We documented our observations and recommendations from our first year of prototyping in *Security Guards for Web Services, Lessons Learned and Recommendations* [Jac03]. For completeness, we include them in this document.

We divided our results into three categories: needs, observations, and recommendations.

### **3.7.1 Needs**

#### **3.7.1.1 Dynamic, Machine-to-Machine Cross-Domain Information Sharing**

Organizations often must move huge volumes of data between security domains. With Web Services, information exchanges can move from human-to-machine interactions to machine-to-machine exchanges, which would increase the amount of data that could be quickly transferred.

#### **3.7.1.2 Machine Review of Content**

For machine-to-machine transfers to be feasible, the requirement for human review of the content must be eliminated from the information release process. To do this, Web Service developers must design their Web Services so that the Web Service requests and responses are sufficiently structured to allow machines to parse and understand those messages. Furthermore, the information sharing rules for those Web Services must be interpretable by a security guard.

#### **3.7.1.3 Streamlined Process for Adding New Web Services**

Introducing policies to a guard associated with a new Web Service would be similar to adding new message types to today's guards that perform automated machine review. The guard administrator would have to add at least two new message types for each Web Service: one message type for the request and another for the response. Today's process for adding a new message type can be complicated and must often be performed by the guard developer. In the future, the process for adding new Web Services to a Web Service guard must be streamlined, otherwise it will overwhelm the guard administrator.

### **3.7.2 Observations**

#### **3.7.2.1 Remote Procedure Call versus Document Style SOAP messages**

When creating a SOAP service, designers can model a message as either an XML document or as a Remote Procedure Call (RPC) – an XML-encoded method signature.



Available programming tools for the RPC method hide SOAP's complexity from the programmer, making creation of these services straightforward. However, the RPC style has limitations. First, because of the tight coupling between client and server, it is harder for programmers to institute changes when they do not control deployments of both the clients and the server. Second, the RPC style has a bias towards synchronous services, which tend to scale more poorly than asynchronous services. For example, consider an implementation of a portal that uses Web Services to collect the information it needs to present to its users. When accessing synchronous services, the portal must leave its resources tied up while waiting for an answer from the server. In the case of slow queries or many parallel queries, this causes a large resource burden.

A related issue for the RPC style is the use of the `soap:encodingStyle` attribute. That attribute indicates the use of a particular scheme in the encoding of data into XML. XML Namespaces can also be used for this purpose. It is preferable to use the literal, non-encoded XML form of RPC.

### **3.7.2.2 Content Validation and Transformation**

The security policies a Web Services guard must enforce on SOAP transactions fall into two categories: those that seek to validate some aspect of the SOAP transaction and those that seek to transform it in some way for security reasons.

Examples of validations include the authentication and authorization of users, client workstations and servers. Examples also include validating message content, message integrity and authorized transports. Examples of transformations include sanitization (e.g., eliminating material not allowed in the other domain), namespace changes (e.g., where a namespace may be classified) and Universal Resource Locator (URL) hiding (e.g., because domain URLs may not be releasable in other domains but are needed for service functioning). The following sections discuss observations about security policy enforcement.

#### **3.7.2.2.1 Authentication**

In a SOAP transaction, three players potentially need to be authenticated and authorized: the user (client), the client workstation, and the server. When policy requires client workstation authentication, guards can maintain lists of authorized workstations based on Internet Protocol (IP) address or Domain Name Service (DNS) names. IP address spoofing could potentially pose a problem with this approach.

When policy requires user and server authentication, the Web today typically uses the SSL protocol, which involves a combination of public-key and symmetric-key encryption. An SSL session always begins with an exchange of messages called the SSL handshake. The handshake allows the server to authenticate itself to the client using public-key techniques, and then allows the client and the server to cooperate in the creation of symmetric keys used for rapid encryption, decryption, and tamper detection during the session that follows.

Optionally, the handshake also allows the users to authenticate themselves to the server using public-key techniques.

The SSL server authentication capability authenticates the server to the user. Although SSL provides a client authentication capability, it places a burden on users, because they must load their private key on every workstation they use. Another common practice for user authentication is passing username/password combinations via HTTP Basic authentication or via a submission from a Web form with SSL used to protect the privacy of the communication.

Implementing SSL communications between client and server in a cross-domain situation is a challenge. First, there is a problem with the keys that would be used for the public-key encryption. Within today's public-key infrastructure, Certificate Authorities only operate within a single security domain. A key generated in one domain cannot be validated by a Certificate Authority in the other domain. If policy requires high-side users to authenticate themselves to low-side servers, they would need valid certificates from the low-side domain. They must then load these low-side certificates into their high-side workstations. If high-side servers need to authenticate themselves to low-side users, the servers would need valid certificates from the low-side domain. In cases where low-side users need to authenticate to a high-side service, policy may prevent loading high-side certificates on the low-side workstations, since the certificates themselves are often considered sensitive. A similar situation would exist for low-side servers that must authenticate themselves to high-side users.

Second, the end-to-end protection afforded by SSL encryption also presents a problem where a Web Service guard intervenes in the information flow between the client and the server. A guard serves as an intermediary between the client and the server. This poses a problem for SSL, which protects itself from man-in-the-middle attacks. Most firewalls today allow SSL traffic to flow through unhindered and unexamined. Obviously, that is not acceptable in a Web Service guard because the guard would be unable to examine the content of the messages to enforce policy on the content.

In examining potential solutions, we ruled out those that give the guard copies of either the server's or the user's private keys or make users ignore their browser's warning that SSL is compromised.

A potential solution for server authentication that we proposed in our 2003 research was to create doppelganger server certificates (i.e., certificates with the same name but with a different private key) in the client domain for each server offering a service through the guard. Doppelganger server certificates issued by an authority that the client's browser trusts must be loaded onto the guard. This allows the guard to present itself to the client as the legitimate server and thereby decrypt the traffic and apply its policies. The guard can then create a separate SSL connection with the actual server.

For SSL's user authentication to function through the guard, doppelganger client certificates issued by an authority that the server trusts must be loaded onto the guard. Clearly, creating doppelganger certificates could become a maintenance nightmare if the authorized servers and clients change frequently.

For transport-independent authentication, SOAP has the ability to use an authenticating token in the SOAP:Header that is signed using XML-Signature. Both users and servers could sign their messages with an XML signature, which provides a security token that could be used to authenticate the user or server. While this option does not have the same problem as SSL with the end-to-end encrypted channel, it relies on a public-key infrastructure for its keys, so it has the same problem with cross-domain validation of keys as the SSL option.

When implementing authentication mechanisms, the Web Service and guard developers must determine who requires the authentication information. For example, in a situation where a high-side consumer wishes to access a low-side service, that low-side service may have no requirement for user authentication. However, security policy within the consumer's environment may dictate that all cross-domain transactions be audited. In this situation, a client's identification credentials would only need to be presented to the guard for auditing purposes; they would not need to be forwarded beyond the guard to the destination Web Service.

Until cross-domain PKI solutions are supported, it will be a challenge to provide credentials using PKI technologies for authentication between clients and servers residing in different domains.

#### **3.7.2.2.2 Content Validation**

The guard reviews and validates the content of the messages to determine that the content is authorized for release. Content validation can also be used to counter threats that might be introduced by that content. These threats come in three forms. First, the message content may make an application perform unexpectedly by providing input that the application does not expect (e.g., buffer overflow). Second, the message may exploit mobile code capabilities of a format (e.g., word macros). Last, the content may subvert an application into providing a back-channel communications medium.

To defend against unexpected inputs, the guard can validate all input against a schema. Unfortunately, most current schemas for Web Service messages are not sufficiently restrictive for use by a guard to permit cross-domain information transfers. A recent study by MITRE provides guidance on restrictions for XML schemas for cross-domain information exchanges [Sim04]. That research also indicates that schema validation techniques alone may not suffice for cross-domain content validation.

Schema developers should strive to keep the schemas unclassified to avoid the maintenance problems of synchronizing multiple schemas across security domains.

Because of the expertise needed to write good regular expressions, schema developers should establish and share a library of tightly constrained types. If the schema needs security attributes, schema developers should use those included in the Intelligence Community Metadata Standard for Publication (IC-MSP). These security attributes have been well thought out and vetted through the community.

Furthermore, XML can support an infinite array of encodings. A Web Service guard must limit allowed encodings and ensure its content checking mechanisms understand the encodings that are allowed. For example the word “BAD” could be put into an XML document as `&#066;&#065;&#068`. The guard’s content checking mechanism must be able to handle this encoding.

One method to mitigate the threat of malicious code is to scan all XML content with a virus checker. However, commercial virus checkers will only recognize common viruses in commercial formats; proprietary/custom formats receive virtually no protection from virus checkers. Because virus checkers only protect against known viruses, Extensible Stylesheet Language Transformations (XSLT) could be used in environments where the risk is unacceptable to eliminate parts of the XML message that are considered active content (e.g., macros in a Microsoft Word document). XSLT provides a powerful implementation of a tree-oriented transformation language for transforming instances of XML using one vocabulary into either simple text, HTML, or XML instances using any other vocabulary.

There are two main methods to mitigate the threat of using a Web Service to open a back-channel communications route. The first is to define the schema so that it severely constrains the amount of information that can be exported through this communications channel. Second, a dirty word search could augment the schema validation process to catch unauthorized material contained within valid elements.

#### **3.7.2.2.3 Availability of Constraining Schemas for Cross-Domain Web Services**

One of the most basic principles of Web Services is that services should be highly accessible. Supporting this principle requires that a service be usable even when no descriptions for it are available. Industry has enabled this flexibility by creating tools that dynamically build SOAP RPC messages. However, these messages either lack associated schemas or the automatically generated schemas are too generic to be of much use for content validation by a guard.

Commercial Web Service developers often do not place any constraints on the XML representing the method signature for the request. Instead, the Web Service application examines the request to determine if all required fields for the call are present and all parameters are sufficient to process the request. As a result, for commercially-developed Web Services, method signature documents can successfully qualify against meaningless or empty namespaces.

This lack of proper schema constraints would be unacceptable in a cross-domain Web Service. Since the guard does not have the same level of understanding about the Web Service that the service provider does, it must rely on standard XML verification mechanisms such as XML Schema to validate the document. If a document cannot be validated, it cannot be released.

For a Web Service to be usable for cross-domain exchanges, programmers will have to examine the messages being generated by the Web Service development tools and manually create a tighter schema. Furthermore, almost every application change will require a corresponding change in the schema used by the guard.

Service descriptors might be viable to use for schema validation. These descriptors, the most notable format for which is Web Services Description Language (WSDL), define the requirements for the contents of the RPC message without dictating its structure. Thus, the guard could validate the message against the requirements for the service to determine whether the message can be released. This type of validation is still in its infancy, but it would place few additional requirements on the service provider or the client. For the client, no changes would be required. Service providers would have to define service descriptors for their Web Services.

Another option is for clients and service providers to qualify RPC messages with a valid namespace, so that the guard can validate the message through standard XML Schema. However, many commercial applications do not support this capability today; custom implementations would be required until commercial practices change. In this option the client must now somehow know the correct namespace to use.

#### **3.7.2.2.4 Validation of Co-Constraints**

XML schemas can be used to determine the validity of an XML document. Sometimes documents can have co-constraints. For example, the policy may state that if a certain element has a certain value, then another element must also exist in the message. If the ability to enforce co-constraints is a requirement, guard developers must consider what technology is most appropriate and whether that technology is already available in commercial applications and can be ported to security guards.

#### **3.7.2.2.5 Binding of Message Envelope to its Contents**

The message envelope may act as more than a transportation wrapper for the content. In many cases, the envelope contains header information that will be required during an inspection of the content. When this is the case, the guard must ensure that the headers of the envelope are easily accessible and reproducible during content examination.

#### **3.7.2.2.6 Validation of Cross-domain Advertisements**

Service registries allow users to advertise and search for Web Services. For most registries, complex replication mechanisms propagate advertisements throughout an enterprise, delivering copies of an advertisement to each registry. Security policies may limit what advertisements can be delivered to various domains. Guards must therefore be able to validate and enforce policies for sharing Web Service advertisements.

#### **3.7.2.2.7 Validation of SOAP Headers**

SOAP headers can be flexible, even to the point of storing information unrelated to the payload. They have the potential to be used as covert channels. The guard must review the contents of every header and verify conformance with an appropriate cross-domain security policy.

#### **3.7.2.2.8 Sanitization of Content**

Security policy may mandate that portions of a message be eliminated before the message can be transferred to the destination security domain. For example, policy may state that certain information must be removed from an Air Tasking Order (ATO) before it can be released to coalition forces. In the intelligence realm, policy may state that source information must be removed from messages before they can be transferred to other domains. Sanitization is the process of removing these unauthorized pieces of information or transforming them into a form that is releasable..

XSLT stylesheets can be used to sanitize an XML document. Sanitization is simplified if schema designers place the filtering values in field attributes or in tags that surround the content to be removed.

Because sanitizing stylesheets may not remove all intended information, schema developers should create a domain-specific schema to double check the sanitization (i.e., to verify that the removed tags/attributes are not in the domain-specific schema).

Given the relative complexity of creating sanitizing stylesheets, schema designers should maintain and share a library of these stylesheets.

#### **3.7.2.2.9 Validation and Sanitization of Metadata**

Most protocols for transmitting documents allow metadata to be appended. In many cases, this metadata describes information required by or important to the transmission. However, metadata may also be irrelevant or malicious.

Guards must examine and enforce security policies on metadata. They must permit relevant metadata to pass through, as the data may be required to process the message. The simplest form of relevant metadata normally applies to the document and specifies the document's

size, type, and filename. The next most common set of relevant metadata describes the sender and is used for identification and encryption. Finally, relevant metadata describing the receiver specifies processing requirements imposed by the sender. If required metadata does not reach the receiver, the message may not be processed in a reliable manner.

The guard must validate and potentially sanitize metadata. The release policy must specify what metadata is required or optionally supported between the sender and receiver. The policy must specify constraints for each piece of metadata so that the values can be validated. With this information, the guard can validate and sanitize expected metadata and remove unexpected metadata.

#### **3.7.2.2.10 Validation and Sanitization of SOAP Attachments**

One standard associated with the SOAP messaging specification allows for additional data to be attached to a SOAP message. Attachments are usually not highly structured data. If a security guard is to support SOAP with attachments, policies on attachments must enforce limitations on what type of attachments are allowed for a given message as well as provide a mechanism for reviewing and releasing any data that cannot be validated through machine review of the content.

Security policies may place restrictions on attachment type. If the guard cannot determine the attachment type, it must prevent its release.

For any attachments that are not highly structured, humans must review the content for releasability prior to delivery of the message to the guard. A human reviewer could digitally sign each attachment in relation to the message to indicate that the content is releasable. By signing the attachment, the reviewer certifies the use of that attachment for that message, thereby attesting that the attachment is both releasable and permitted in the current use.

#### **3.7.2.2.11 Transformation and Obfuscation of URLs**

Information sent across domains may contain URL pointers to resources on the originating domain. These URLs may contain sensitive information that should not be released to the destination domain. Security guards must provide a reliable mechanism to transform sensitive URLs and then map between these sensitive URLs to the proxy URL.

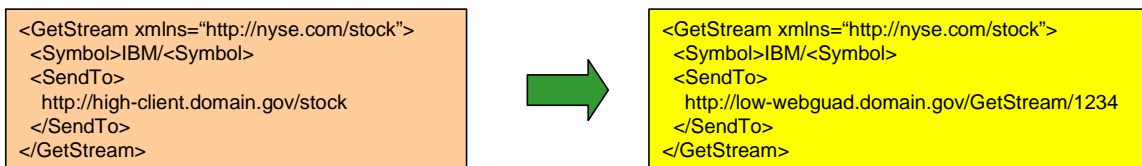
URLs contain fully qualified domain names that typically make it easy to discern their originating domain. Because of this and other information that may be discernable from the URL, security policies often prohibit the use of a higher domain's URLs in a lower domain. However, because URLs are often critical to the workings of Web Services, simply removing them from the document would destroy too much functionality.

For example, if the SOAP message contained the URL <http://www.high-server.gov/abc>, the guard would replace the URL with a generated proxy URL in the destination domain that might look something like <http://Webguard.low-domain.gov/ServiceName/12345>, where

ServiceName is the service to which this message belongs and 12345 is a generated unique identifier. The guard would not modify URLs from domains other than the source domain.

For any messages that contain those obfuscated URLs, the guard must maintain the mapping between the original URL and the obfuscated URL. When the guard processes a message with an obfuscated URL, it must restore the original URL. The guard must maintain a record of those mappings for at least some period of time.

Maintaining those mappings becomes even more difficult for asynchronous services, where the request message provides a URL indicating where to send the response message. The guard must not only obfuscate the URL but also be able to listen for that URL if the guard has been configured for that service. Figure 3-7 provides an example.



**Figure 3-7. URL Obfuscation for Asynchronous Service**

#### **3.7.2.2.12 Transformation and Sanitization of Namespaces**

Information sent across domains may contain sensitive element representations. The guard must transform those elements into a representation approved for release to the destination domain.

Namespaces may need to be transformed. Schemas may include classified elements or attributes (i.e., instances where the attribute or element is classified in addition to its contents). The guard would need to transform the message to a schema/namespace that would be releasable to the destination domain. XSLT could be used to accomplish this.

Namespaces may need to be removed. In some cases a document type in one domain is defined by a Document Type Definition (DTD) and therefore has no namespace while another version of that document type that has a namespace exists in a different domain. DocBook and IC-MSP are examples of this phenomenon. The DocBook specification was created long before XML schemas and hence is defined using DTDs, with no namespace specified. The IC-MSP extended the DocBook standard and placed the results in the <http://www.ic.gov/ic-msp/article> namespace. To move a document that uses the IC-MSP schema to a domain that uses the DocBook DTD the guard must remove the namespace.



#### **3.7.2.2.13 Sanitization of XML Comments**

The XML specification allows for comments to exist in documents in such a way that a schema cannot restrict their occurrences or values. Since schemas cannot be used to validate the contents of these comments, security policies should require the guard to remove them from documents prior to release, unless a human reviews those comments and certifies that they are releasable.

Web Service developers should be made aware of this policy so they do not embed any required information within XML comments. Comments should not contain information used to make release decisions, nor should they be used for auditing of the transactions.

#### **3.7.2.2.14 Content Integrity**

Beyond the normal concerns for data integrity within any environment, the cross-domain environment has two additional reasons to maintain content integrity. First, if the format of the message requires human review to determine its releasability, the message needs to be tamper proof after that human review. Second, any attributes of the message used to sanitize or determine its eligibility for release must also be tamper proof.

SSL can provide integrity for the content between the client and the guard and between the guard and the server. For relatively secure sources (i.e., where the content at rest is considered secure) SSL may be good enough. For the majority of sources where integrity of the content at rest may be less assured, content should be digitally signed using XML-Signature. Using WS-Security, signatures can be included in the SOAP message for the whole message or any parts that require integrity hardening.

#### **3.7.2.3 Bi-directional Conversations**

The use of SOAP and Web Services usually requires at least two messages, a request and response, and often requires many messages passing among many actors. This series of message exchanges, known as a conversation, always occurs in a predictable order as part of a known business process. A Web Service security guard must evolve to support policies that consider conversations and their states when determining releasability of a message.

In many cases, especially in RPCs, any message from one domain to the other may result in a responding or otherwise related message in the opposite direction. This continually related message traffic may flow back and forth between the domains in a conversation, and needs to be handled differently than a one-way push of a message from one domain to another.

Conversations require more work by the guard because the messages occur in some form of sequence. The release policy should be written to take this into account. RPC is the simplest example of this sequencing restriction. Response message traffic should only be released if it has occurred within some time limit after a request and only if it is directed at the originator of the request. More complex cases might involve extremely long-running message

exchanges that have a definite pattern as part of a business process. In this case, the guard must monitor the state of that business process to ensure that only appropriate messages are occurring at appropriate times in the process.

Conversation-aware guards will be complex. First, whenever the releasability of subsequent messages depends on previous content the guard must store the message so that one or more future messages can be examined in relation to it. The guard must be able to refer back to previous messages in the conversation. To prevent the message storage area from filling up, the guard design must place time limits on the length of time it stores the messages. The guard will also have to maintain information about the required sequence of messages, as well as awareness of the current state of the transaction process.

If a Web Service has many micro transactions, developers should examine the Web Service to see if these transactions can be combined into a single larger document that encompasses the complete workflow. For example, many existing Web sites today use a shopping cart model for accumulating items and then a separate action for purchasing. A Web Service that follows this model will be difficult to use through a guard because of the need to track what is happening across many transactions. A much better approach would be to aggregate all the smaller transactions into a single purchase invoice. This greatly eases the guard's policy enforcement and makes the Web Service more scalable as well.

#### **3.7.2.4 Error Reporting**

For SOAP to support RPC and reliable message exchange, error information must be returned to the client. The information presented in the error message allows the client to determine how to handle the error. If security policy prevents error messages from being returned to the client, the benefits of SOAP are greatly reduced and many off-the-shelf Web Services will not function properly.

Many legacy guards pushed files to the other security domain but did not permit any acknowledgment of receipt to be returned to the sender. The document sender was notified of release rejection through another means. Trusted or reliable messaging requires receipt confirmation at the transport layer at a minimum. As a result, the guard must notify a client of release rejection on the same channel used for message submission.

The guard must notify the client about release failures during guarding process as well as about errors during the transmission of the message to the final destination. The guard should also notify the client of a failure to deliver the message to the final destination even if that failure notification is indistinguishable from a failure that occurred during the guarding process. Unless the client has the ability to become aware of delivery failure, reliable messaging across domains cannot occur.

Security policies may dictate the content of the error messages, especially when reporting errors to a low-side client. While more detailed information would assist the client in

decision-making, usually all that is required is minimal information about the nature of the error. The error message must contain enough information for the client to interpret it and reason that a resend of the same data would also fail.

### **3.7.2.5 Web Service Standards**

The Web Services community is actively developing numerous standards. These standards will play an important role in cross-domain Web Services in the future. Based on our study of current and emerging standards that relate to Web Services (as presented in Appendix C), we made the following observations about those standards.

#### **3.7.2.5.1 XML-Signature**

XML-Signature can be used for a variety of purposes within Web Services, including user and server identification and authentication as well as message integrity.

Imagine a scenario where a client uses a Web Service to post data to a database and where the database owner requires a non-repudiation policy for those transactions. XML-Signature could be used to provide the server with the identity of the client. If, however, the client and service are in different security domains with a security guard transferring those posts from one domain to another, the cross-domain security policy may require the guard to strip that XML signature in order to evaluate and transform the content. Stripping the XML signature from the transaction will prevent the database owner from having a means to enforce non-repudiation.

The security guard community must consider how to process documents signed with one or more XML signatures. Alternatives include keeping all signatures, stripping all signatures, or keeping some and stripping others. There may be situations where a given signature is stripped one time and kept another.

The security guard community must also consider which XML Signature binding is most appropriate and which (if any) are not allowed.

#### **3.7.2.5.2 SOAP Headers and Web Service Standards**

Several new and emerging XML standards use SOAP headers to store information. For example, WS-Security stores XML-Signature and XML-Encryption details within the SOAP headers. As stated in Section 3.7.2.2.7 above, the guard must review and validate SOAP headers.

Security policy makers must decide which SOAP headers are allowed and which are not. Guard developers must then develop mechanisms for the guard to identify headers and distinguish valid headers from invalid ones. Developers must also determine if technologies such as XML schemas are sufficient to enforce cross-domain SOAP header policy. If so, the

schema developers must design schemas that cover the entire SOAP message (both header and payload).

Security policy makers must decide what should be done with a message containing non-allowed headers. Alternatives include dropping the entire message or just stripping the inappropriate headers. If the policy mandates that headers must be stripped, then the guard developers must develop a means to strip them. See Appendix D for more information on SOAP headers.

#### **3.7.2.5.3 WS-Security**

The WS-Security standard uses XML signatures to ensure data integrity of messages. XML signatures can be included in the SOAP message for the whole message or any parts that require integrity hardening.

WS-Security does not address how a SOAP client/service and a security guard agree on the nature and characteristics of the security tokens they will use. Simply adhering to this standard does not guarantee interoperability. To use WS-Security effectively, the security guard community must constrain the nature and characteristics of WS-Security tokens. The community must also agree upon a set of tokens to be used.

#### **3.7.2.5.4 SAML**

SAML is an XML-based security specification for exchanging authentication and authorization information. With SAML, any point in the network can assert that it knows the identity of a user or piece of data. The receiving application must then decide if it trusts that assertion. SAML may be useful for identification and authentication of users and servers and conceivably even for describing client and server authorizations.

However, for SAML to be useful in a cross-domain scenario, SAML assertions must be generated by trusted entities. The security guard community must figure out what assertions would be useful to be exchanged between guards and external entities or between components within a multi-component security guard. Once the community decides what assertions would be useful, the guard developers must devise a means for the players (clients, servers and guards) to create and verify those SAML assertions.

#### **3.7.2.5.5 XACML**

XACML is a general-purpose access control policy language. XACML may be flexible enough to express cross-domain policies. If it is, then the security policy makers must determine how to express information sharing rules using XACML. The guard developers must then figure out how to relate the XACML policy to the means used to implement that policy (e.g., schemas and stylesheets).

#### **3.7.2.5.6 WS-I Basic Profile**

WS-I Basic Profile constrains and clarifies a number of core Web Service standards to provide increased interoperability among Web Services. The security guard community should influence the development of the WS-I Basic Profile to ensure it follows good security practices.

### **3.7.3 Recommendations**

From our experimentation, we can offer a number of generalized recommendations to Web Service builders who want to maximize the potential for using their services across security boundaries. The recommendations are as follows.

#### **3.7.3.1 Develop Robust Schema**

For automated release to succeed, schema designers must define robust schemas of the information to be evaluated. These schemas must be highly descriptive and constrain all elements using mechanisms that can be validated by the guard. Defining “good” schemas for Web Service messages is extremely important. Other MITRE research has provided suggested guidelines for constraining schemas [Sim04].

Security policy makers should provide comprehensive standards and guidelines for the creation of schemas. Research is still ongoing on how to adequately restrict schemas for use in cross-domain policy enforcement.

#### **3.7.3.2 Maintain Repository of Schemas and Stylesheets**

Web Services developers should develop and maintain a repository of schemas and associated transformation and sanitizing stylesheets that can be shared across the development community.

#### **3.7.3.3 Minimize Stateful Web Services**

When feasible, Web Service developers should avoid designing stateful Web Services. They should design Web Services that do not require multiple calls to the Web Service to complete a transaction. For example, in the case of an e-commerce style operation, they should use a purchase order paradigm rather than a shopping cart paradigm. This allows a guard to consider each Web Service transaction in isolation and minimizes the amount of stateful information it must maintain.

#### **3.7.3.4 Employ XML Technologies in Future Solutions**

A security guard does not exist or operate in a vacuum. Sometimes it must interoperate with and, to some degree, depend upon components within the infrastructures around it. Some XML technologies—notably SAML, XACML, and XKMS—may not be useful in a cross-domain situation until trusted services and servers appear within these infrastructures (e.g.,

the enterprise services found in Net-Centric Enterprise Services (NCES)). The security guard community should determine when various XML technologies are scheduled to be introduced into the enterprise infrastructure and then plan their future developments to take advantage of them where feasible.

Before using these enterprise services, the security guard community should also consider if services provided by this XML technology can be trusted and whether communications to these services can be adequately protected. Even if the guard community determines that the enterprise services cannot be trusted for use by the guard, they should consider whether these technologies could be employed *within* the guard. The community must also consider the balance between the benefits to be gained by using these XML technologies versus the security issues related to these technologies

## Section 4

# Semantic Web Environment

## 4.1 Semantic Web Description

The Semantic Web is “an extension of the current Web in which information is given well-defined meaning, better enabling computers and people to work in cooperation” [BHL01]. Building upon current Web technologies (Web addressing, universal character set, XML, etc.), the Semantic Web makes Web information meaningful to machines (software) by giving it well-defined meaning (i.e., semantics). These semantics are made explicit through ontologies that software applications can then exploit.

So what is an ontology? A commonly cited definition of “ontology” in the Web community is “the specification of a conceptualization” [Gru93]. In an ontology, concepts and their relationships to other concepts are specified precisely to support machine interpretation. A concept can be thought of as a resource identified by a Uniform Resource Identifier (URI). Resources may either exist on the Web (e.g., a document that can be retrieved) or be represented on the Web (e.g., a person). An ontology captures information about these resources and the relationships between them.

Ontologies are described using Web ontology languages, which are founded on a language called Resource Description Framework (RDF)<sup>2</sup> and its subsequent extension, called RDF Schema (RDF-S). RDF represents resources as sets of triples, where each triple consists of either (Resource, Property, Resource) or (Resource, Property, PropertyValue). These triples collectively constitute a graph. The Web Ontology Language (OWL)<sup>3</sup> is a semantic extension of RDF-S, providing more expressive power. On 10 February 2004, the World Wide Web Consortium (W3C) announced that RDF and OWL were W3C Recommendations, which effectively made them Web standards.<sup>4</sup> Note that both RDF and OWL observe the Open World Assumption, which states that new knowledge can always be added to what already exists.

---

<sup>2</sup> <http://www.w3.org/RDF/>

<sup>3</sup> <http://www.w3.org/2004/OWL/>

<sup>4</sup> <http://www.w3.org/2004/01/sws-pressrelease>

Ontologies provide a mechanism for machines to perform simple inferencing by combining facts to form new facts or conclusions. For example, given that *Mary* is the *spouseOf Jim* and *spouseOf* is a symmetric property, an inference engine can conclude that *Jim* is the *spouseOf Mary* without having that fact explicitly asserted. OWL provides the semantic expressiveness that enables machines to make inferences on the basis of information such as relations between concepts (equivalent, disjoint, etc.), property characteristics (inverse, transitive, symmetric, etc.) or cardinality constraints (e.g., birthmother has exactly one value). Constraints and the capability to combine facts to make inferences allow machines to solve tedious problems, combine facts to discover new information, and help prevent certain misunderstandings. They do not solve all problems.

The Semantic Web is not yet mature. However, while we consider these technologies to be in the “early adopters” stage, there are many indications that these technologies will quickly become intrinsic to commercial applications. First, Semantic Web technologies build upon and blend long-lived concepts from many fields (artificial intelligence, knowledge representation, database management systems, information retrieval, natural language processing, mathematics, logic, etc.) but apply them on a global scale via the World Wide Web (WWW). Second, the Semantic Web is being championed by the World Wide Web Consortium (W3C)<sup>5</sup>, a preeminent organization committed to the evolution of the Web. Third, these technologies are moving beyond the research community. Commercial companies are offering Semantic Web tools and services and large corporations, including Adobe and Sun, are applying them [Bou04]. Finally, the number of conferences and meetings with Semantic Web themes is exploding, further indicating the large and growing interest in these technologies.

## 4.2 Semantic Web Guard Approach

This section discusses our exploration of issues related to sending semantically tagged data across security domains in a Semantic Web environment. We assume that this semantically tagged data is in the form of an ontology or a portion of an ontology. The security guard, or cross-boundary device, would receive this ontology and apply a machine-readable interpretation of the release policy to determine if this data, or some modification of it, may be released. Our approach was to develop a prototype, not as an operational tool to transition, but rather as a mechanism to identify any implications of exchanging ontologies across security domains. It is important to consider how this may be done, as ontologies are used to maintain semantic content. These ontologies may be serialized into XML, but ontologies do

---

<sup>5</sup> <http://www.w3.org/>



not have a prescribed sequence. Therefore, many traditional guard approaches that rely upon highly structured, specifically sequenced information are insufficient in a Semantic Web environment.

As with all forms of cross-boundary devices, a spectrum of approaches and associated risks exists for a Semantic Web guard. Figure 4-1 depicts our view of such a spectrum. It illustrates the ontology approach, policy applied, and the associated risks for each category. The risks listed are additive as one reads from left to right. The most conservative approach assumes that all resources (classes and properties) and all possible values for literals are known ahead of time. This approach is so restrictive that we consider it to be of marginal value. For example, if one defined a resource of “Person” and defined “Person” as having a name, then the guard would have to know every value for “name” in order to release that data. At the other end of the spectrum is the least conservative approach, where both the definition of resources and literal values are unconstrained. While this approach introduces much more risk, it may be feasible in commercial applications.

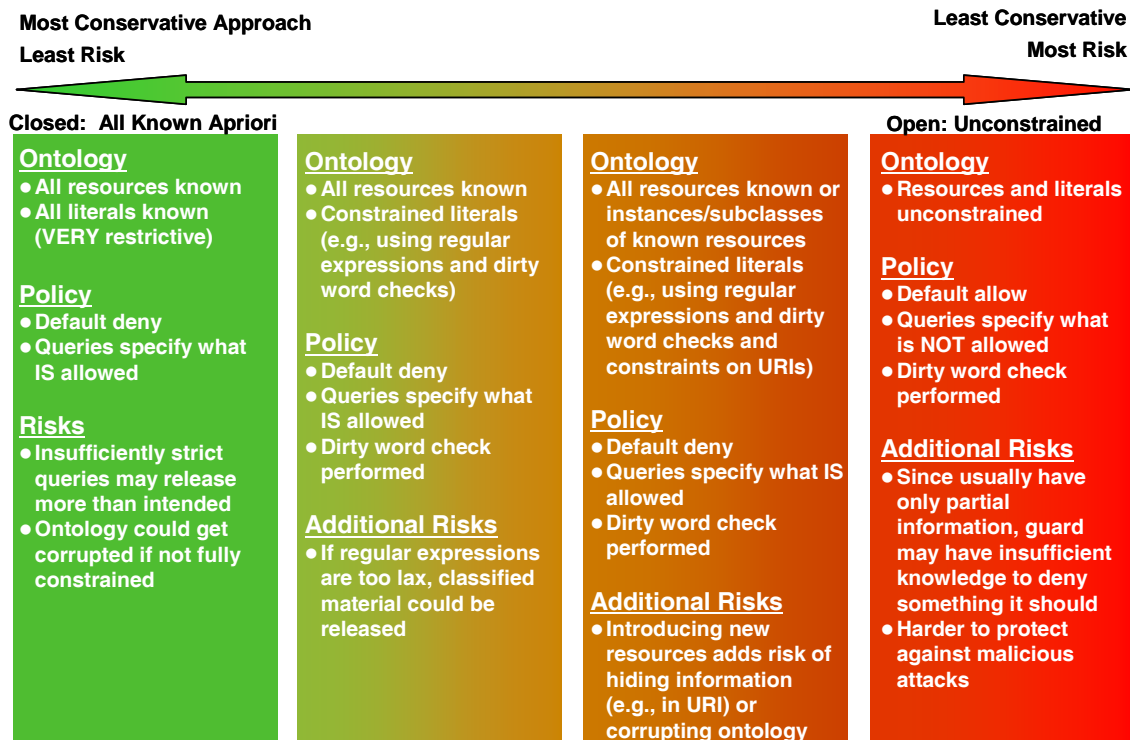


Figure 4-1. Semantic Web Guard Risk Spectrum

### 4.3 Semantic Web Design Decisions

This section discusses five design decisions we made in the implementation of our Semantic Web guard prototype.

#### 4.3.1 Producer Push

The first design decision was to implement a “producer push” scenario. This scenario corresponds to the security guard architecture alternative circled in Figure 4-2. Our Semantic Web guard implementation assumes that the security guard makes a determination of whether information developed at a higher security level may be released to a lower security level. Our implementation is agnostic as to what triggers this machine-to-machine high-to-low data push.

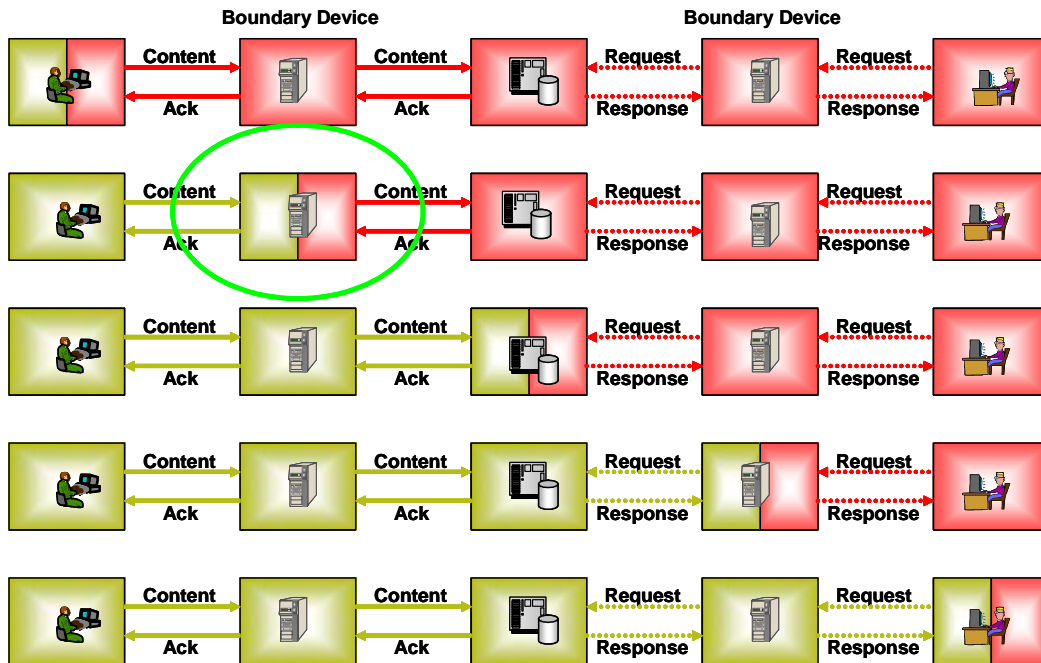


Figure 4-2. Semantic Web Guard Architecture Alternative Selected

#### 4.3.2 GSIX Prototype

Our second design decision was to use the Guarded Sharing of Information with XML (GSIX) prototype in our implementation. We chose this software prototype because it helped us meet our goal of leveraging other MITRE efforts, provided an extensible software infrastructure, and was already being used in our Web Service guard experimentation. This

synergy between our guard experimentation efforts allowed us to focus our limited resources on the Semantic Web guard application rather than on developing infrastructure software.

### **4.3.3 OWL-DL Ontology Vocabulary**

Another design decision was to use the new international standard Web Ontology Language (OWL) to define our ontologies. OWL, which became a W3C Recommendation in February 2004, has three increasingly-expressive sublanguages: OWL Lite, OWL DL, and OWL Full<sup>6</sup>. OWL Lite supports classification hierarchy and simple constraints. For example, while OWL Lite does support cardinality constraints, their values are restricted to either 0 or 1. OWL DL, named for its correspondence to description logics, provides maximum expressiveness while retaining computational completeness (all conclusions are guaranteed to be computed) and decidability (all computations will finish in finite time). OWL Full offers maximum expressiveness but no computational guarantees. We chose OWL-DL because it is the dialect that tool vendors are most likely to implement, and in fact it is supported by our chosen tools.

### **4.3.4 A Single Input Ontology**

For the purposes of our prototype, we assumed that a security guard would operate on a set of knowledge contained in a single ontology. This design decision was a simplifying assumption for our implementation. It implies that an OWL input file would map to a single ontology resident on the security guard. An implication of this design decision is that users who wished to send a collection of information from multiple, unlinked ontologies would have to submit separate input files, each of which maps to a single ontology resident on the guard.

### **4.3.5 Moderate Risk**

Our final design decision was to develop a prototype that we believed represented moderate risk. Our Semantic Web guard approach assumed a policy of “default deny,” where the guard queries the rules to select what data may be released; any other data is discarded (i.e., pruned). Our prototype implementation also assumed that the guard had advance knowledge of all resources (classes and properties) that it would allow to be released.

## **4.4 Semantic Web Mission Use Case**

This section describes the mission use case we used in our Semantic Web Guard prototype.

---

<sup>6</sup> <http://www.w3.org/TR/2004/REC-owl-guide-20040210/#OwlVarieties>

#### 4.4.1 Process

To help achieve our goal of leveraging other related efforts, our first step was to survey activities across MITRE to identify existing ontologies and associated policies to release data mapped to these ontologies across security boundaries. We decided to use an airspace ontology developed for another MITRE Semantic Web research project.<sup>7</sup> As no release rules existed for sharing this data across security boundaries, we worked with an operational domain expert to develop an operational scenario and an operationally realistic set of release rules to use in our experimentation.

#### 4.4.2 Release Rules Scenario

Our scenario centers on the airspace definitions created as part of the airspace planning process on the high side (US Only network). Some of these defined airspaces are shared regularly with coalition partners and may be included in an Airspace Control Order (ACO).

#### 4.4.3 Airspace Ontology

The airspace ontology models a subset of the airspace information defined by the United States Combat Air Force and included in the ACO. The ACO defines and establishes special-purpose airspace for military operations and notifies all agencies of the effective time of activation and the composite structure of the airspace to be used.

Figure 4-3 is a diagram of the simplified airspace ontology we used in our Semantic Web guard experiment, as output by the Network Inference<sup>8</sup> Construct<sup>TM</sup> tool. We defined the Combat Air Force airspace (*CAFAirspace*) as having the attributes of name and description. We used named properties for the airspace shape and the airspace time period.

As shown in the figure, inheritance plays a strong role in this ontology. The solid single-headed arrows show inheritance. For example, *NoFireArea* and *NoFlyArea* are subclasses of *SpecialUseAirspace*. This means that the child resources (i.e., subclasses) inherit the attributes and properties of the parent resource (i.e., superclass). Because airspaces are commonly referred to by their standard acronym, for some classes we created an airspace class named with the standard acronym and made it equivalent to its appropriate class. This allows us to reference that airspace class, its properties, and its instances using either the full

---

<sup>7</sup> The Airspace Ontology was developed by Mary Pulvermacher for use in Netcentric Semantic Linking experimentation as part of Project 0304752600.

<sup>8</sup> <http://www.networkinference.com>

name or the acronym (e.g., either *UnmannedAerialVehicle* or *UAV*). An equivalent class works well where the exact same concept can be referred to in different ways.

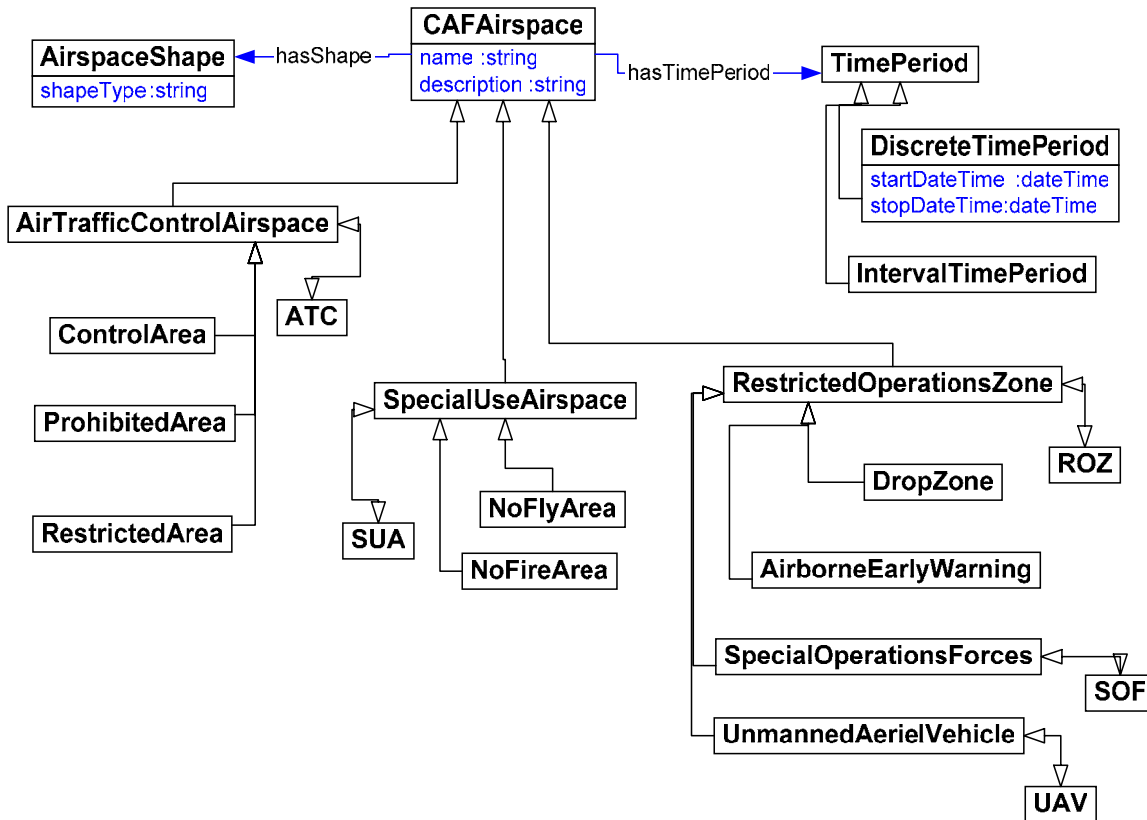


Figure 4-3. Simplified Airspace Ontology

#### 4.4.4 Airspace Release Rules

As mentioned previously, our airspace release rules assume a “default deny” approach. Therefore, our release rules describe what the guard is allowed to release. All other data is assumed to be discarded. Also, these release rules apply to airspace definitions going from the high side (US Only) to a low side (coalition partners). Our release rules are divided into two categories to match the two implementation approaches we used. Thus, we have two sets

of release rules. One set applies to a Semantic Web guard that only supports pruning (i.e., selecting only what is allowed). As discussed later, we implement these release rules using queries. The second set of release rules applies to a Semantic Web guard that allows data to be transformed to make it releasable. Our prototype uses a combination of queries and a rule engine to implement this set of release rules. Therefore, our prototype experiments with sending airspace data through two different kinds of Semantic Web security guards. The two sets of release rules we implemented are listed below, with a mapping of these release rules to the airspace ontology shown graphically in Figures 4-4 and 4-5.

#### **4.4.4.1 Set of Airspace Release Rules Applied for Pruning Guard**

We defined the following rules for the pruning guard:

- Any airspace of type “Special Operations Forces” gets deleted.
- Any “Restricted Operations Zone” airspace with “laser” in the name field has the active time period removed.
- Any “Unmanned Aerial Vehicle” airspace has the name and description field removed.
- Any other airspaces defined in the controlled airspace ontology may be released.
- Only airspace type and properties that exist in the controlled airspace ontology may be released.

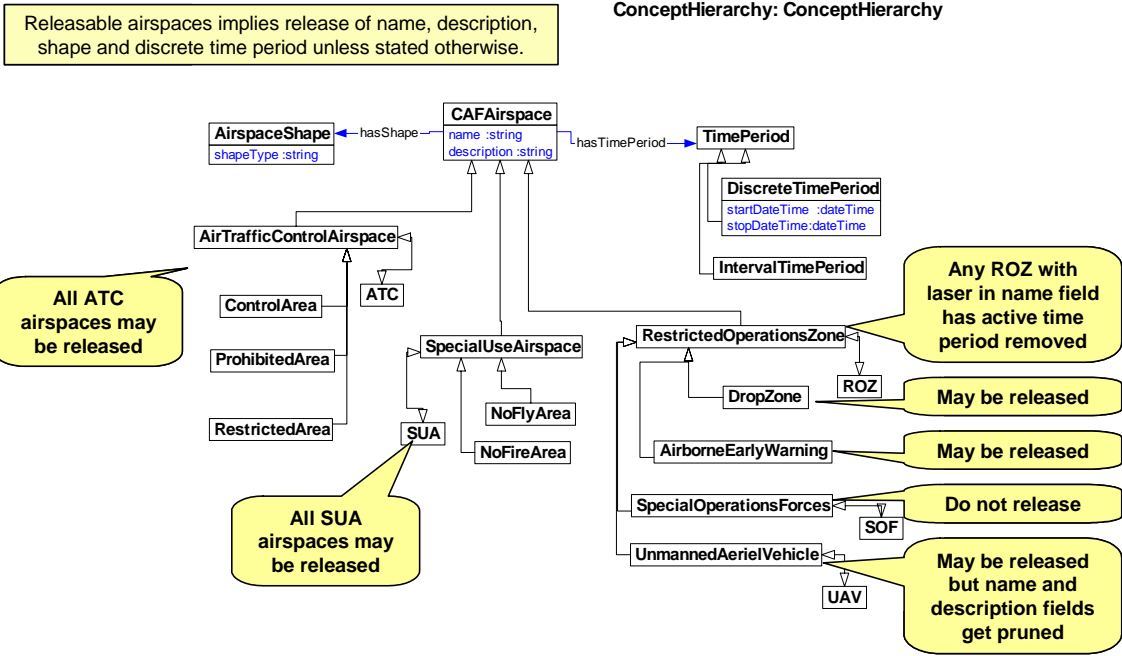


Figure 4-4. Airspace Release Rules – Pruning Supported

4.4.4.2 Set of Airspace Release Rules Applied for Transformation Guard

We defined the following rules for the transformation guard:

- Any airspace of type “Special Operations Forces” gets changed to type “No Fire Area” and name and description fields are deleted.
- Any “Restricted Operations Zone” airspace with “laser” in the name field has the active time period removed.
- Any “Unmanned Aerial Vehicle” airspace gets a generalized name of “UAV” and the description field is deleted.
- Any other airspaces defined in the controlled airspace ontology may be released.
- Only airspace type and properties that exist in the controlled airspace ontology may be released.

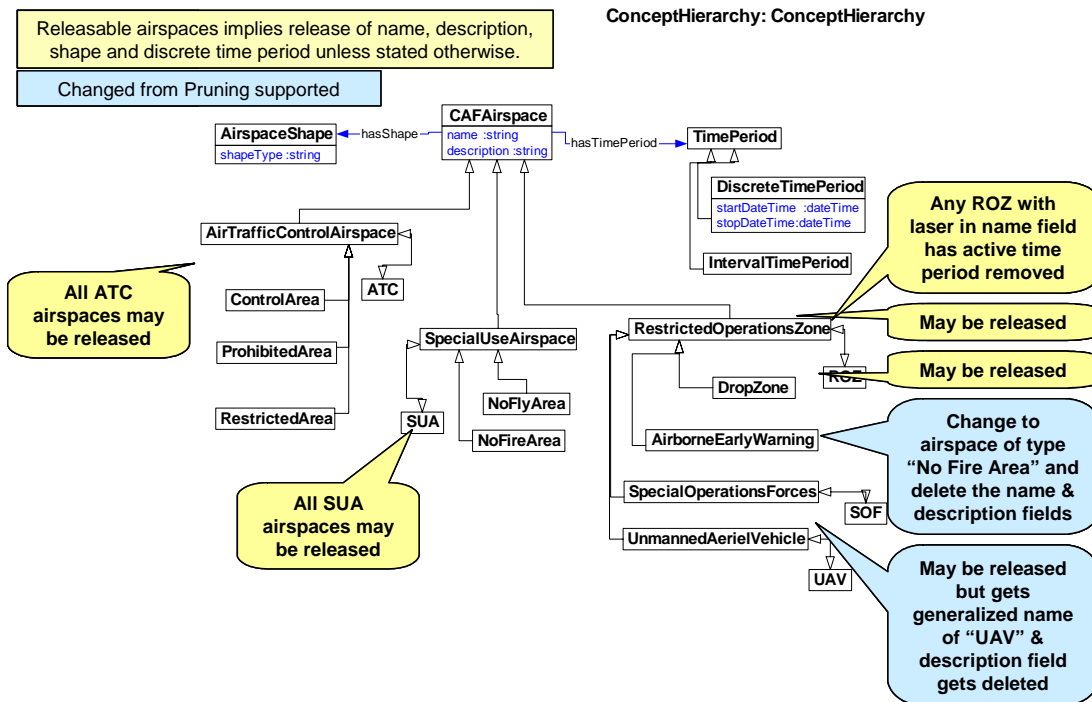


Figure 4-5. Airspace Release Rules – Transformation Supported

## 4.5 Prototype Implementation

### 4.5.1 Design

The goal of this research was to study how one would send ontologies across security boundaries using machines to determine what is allowed to pass through the guard. The prototype development effort served as a mechanism that enabled us to learn about issues associated with such a task. Within this prototype, we considered using ontologies in two ways: (1) sending ontologies across security boundaries and (2) evaluating how ontologies and other Semantic Web technologies may support cross-boundary information sharing through automated machine processing and flexibility in the expression of rules.

The design was motivated by our concept of what would be a realistic capability. Using our knowledge of security guards, we made educated assumptions about what would be acceptable for accreditation. Figure 4-6 provides a high-level illustration of the basic design. An ontology **I** is passed into the guard. The guard, using its inherent capabilities, determines what is allowed to cross the security boundary and returns a potentially transformed subset of **I** across the security boundary, i.e., ontology **O**.





**Figure 4-6. Top Level Design of Semantic Web Guard Prototype**

We based this prototype on emerging international standards, such as OWL, and existing Semantic Web tools. Thus, our design evolved as we learned about the capabilities of these tools and the associated implications for our prototype guard design. It is important to note that, since our design was partially driven by the maturity of existing tools, the design of any Semantic Web guard would undoubtedly evolve as tools become more functional and robust.

We considered several design options for the guard. The first option was to consider each ontology as nothing more than a set of triples. This approach did not necessarily leverage Semantic Web technologies in examining the ontology going across the security boundaries. The process for examining the ontology was to examine each triple sequentially and decide independently whether each triple was allowed to pass. This made for a very simple design, allowing reuse of existing text-processing utilities and a relatively simple algorithm for comparison. However, we did not choose this approach for three primary reasons. First, since triples are not independent, we needed to consider logical groupings of triples, but this design had no easy way to associate triples. Second, rules had to be specified at the triple level, which made it more difficult to express the rules. Third, this approach failed to leverage the power of Semantic Web technology, which, as mentioned above, was one of the primary considerations of this effort.

A second option was to use an inference engine to understand the ontology and determine whether OWL-tagged data would be allowed to traverse the guard. This approach used Semantic Web technology to determine what could pass. The basic idea was to load the ontology under consideration into an inference engine to create a knowledge base and then use queries and rules to apply the policy. Queries would extract the portions of the ontology that would be allowed to pass. Rules would fire based on certain conditions and as a result invoke certain actions to transform the ontology. This approach, in contrast to the previous option, made greater use of Semantic Web technology to facilitate the expression of abstract rules and apply these rules to lower level classes, relationships, and instance data contained

within the input ontology. For example, imagine a policy that states all resources of type *vehicle* are allowed to pass through the guard and the guard has its own ontology that states a truck is a subclass of a vehicle. Now, if an input ontology comes through the guard containing a resource of type *truck*, the guard can use the knowledge represented within its internal ontology to infer that a truck is a type of vehicle, apply the abstract policy, and determine that truck resources are allowed to pass through the guard. We adopted this approach for our prototype design.

As part of this second design, we considered the guard as operating at three increasing levels of complexity, functionality, and flexibility, as shown in Figure 4-6. In the first case, the guard could apply a set of rules against the knowledge base and, if the data met the criteria, allow the information to pass through (shown as Go/No Go in Figure 4-7). On failure, however, since there is no indication of which triples failed to meet the criteria of what is allowed, no portion of the ontology would be passed through.

A second, and slightly more flexible, approach separates those parts of the ontology (i.e., triples) that are allowed from those that are not allowed (pruning). In this case, the guard releases the data subset that is allowed to go through the guard.

Finally, certain triples within the ontology may be allowed to pass after they undergo one or more transformations (Transformation). In the first two cases, this portion of the ontology would simply be dropped. However, a more flexible approach would permit the transformation of the data, based on criteria stated in the rules, and allow this updated data to be released. The three levels of complexity are illustrated in Figure 4-7 where **I** and **O** represent the input and output ontology, respectively.

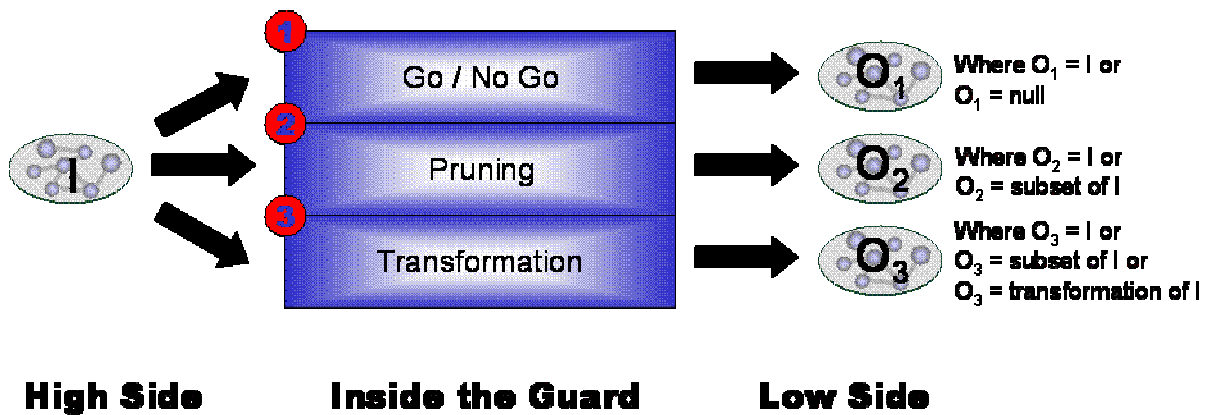


Figure 4-7. Levels of Complexity for a Semantic Web Guard

Another choice we had to make in the second design approach was whether to select those triples that are allowed to pass (default deny) or to select and remove those triples that are not allowed (default allow), leaving the remainder of the ontology to be sent across the security boundary. We chose default deny because this is the more conservative approach and therefore more likely to be acceptable to security policy makers or security guard accreditors. Furthermore, since default deny expresses policies as what is allowed to go through – a finite set, it appears to be a more realistic approach. Given the open-world assumption of the Semantic Web, there is no way of knowing everything that is not allowed without making rules overly restrictive.

Our design made certain assumptions about what resides within the guard. We assumed that the guard knows about what is allowed to traverse it. Our approach was to represent this knowledge as a controlled ontology within the guard. We assumed the input ontology must map to this guard ontology. Any data that did not map to the controlled guard ontology would be discarded. In addition, the guard also contains a set of rules (policy) that expresses what information is allowed to pass through the guard. Figure 4-8 shows the policy and guard ontology outside the guard to make the point that both are configurable. Finally, we assumed the guard contains an inference engine that would understand the input ontology, relate it to the guard ontology, and apply rules against it.

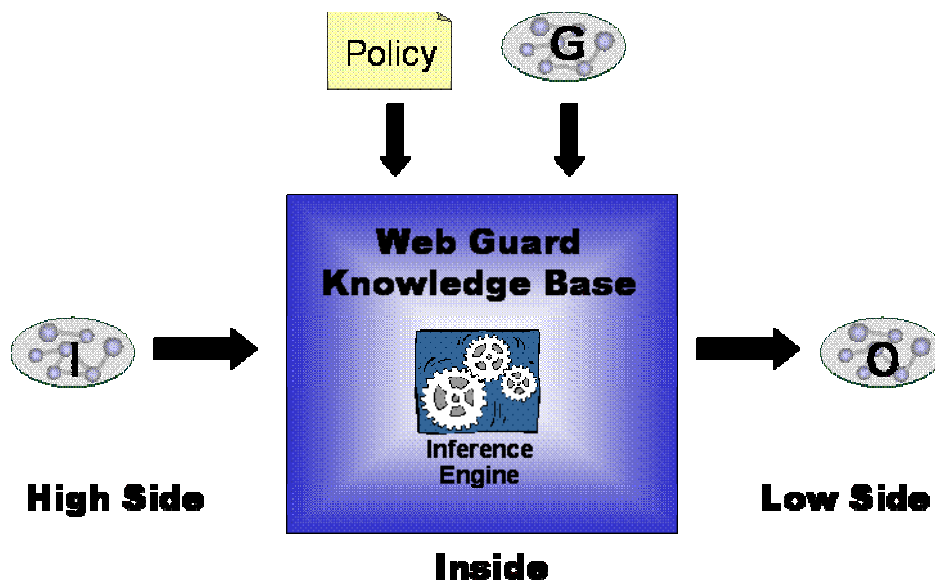


Figure 4-8. Internals of Semantic Web Guard

Finally, another consideration in the design was how restrictive the guard should be with information passing through it. As discussed in Section 4.2, a risk spectrum for a Semantic Web guard can range from allowing no new knowledge (i.e., all is known by the guard a priori) to allowing unrestricted new knowledge. While the former approach seemed too restrictive, the latter exposed security vulnerabilities. We therefore decided on a “middle of the road” approach, where the guard has an internal ontology representing its knowledge, and all knowledge passing through the guard must be associated with the current knowledge of the guard. This implies that each resource (class or property) passing through the guard has to be associated with a resource within the guard’s knowledge base; in other words, no orphan resources are allowed and no new class or property can be introduced. Finally, we decided not to allow any new knowledge, such as inferred knowledge not originally present in the input, to be released.

#### **4.5.2 Tool Selection**

During the development phase of our research we went through a process of selecting tools that would allow us to implement a Semantic Web guard prototype. This process included a survey and comparison of current tools and an evaluation of how adequately they met our requirements. In essence we looked for three capabilities:

- A knowledge base, serving as an ontology repository
- An inference engine, used to infer new facts, based on rules, concepts, and relationships within the ontology
- A query capability that allows users to ask questions of the knowledge base and returns responses to the query.

We found that most tools packaged these components into a single product. Referring to our design, we formulated a set of basic requirements that we used to evaluate the tools. These requirements included:

- An OWL-capable inference engine
- The capability to store and query the ontology
- The capability to change the content of (transform) the ontology
- The ability to integrate our implementation with GSIX.

Our investigation of OWL-capable tools quickly narrowed to three options: SNOBASE, sponsored by IBM; Cerebra, sponsored by Network Inference; and Jena2, an open source tool supported by Hewlett Packard. A more detailed description and evaluation of each follows.

#### 4.5.2.1 SNOBASE

Semantic Network Ontology Base (SNOBASE)<sup>9</sup> is an ontology management system (OMS) supported by IBM under Alphaworks. Its intended functionality includes creating, storing, querying, and manipulating ontologies within its resident knowledge base. It also has a built-in inference engine to support derivation of new facts based on the stored ontologies. SNOBASE provides a Java application–program interface (API) in the form of a Java Ontology Base Connector, which is similar in function to a Java Data Base Connectivity object for relational databases. Furthermore, it provides (or intends to provide) support for DARPA Agent Markup Language (DAML) Query Language (DQL) as its ontology query interface.

#### 4.5.2.2 Cerebra

Cerebra,<sup>10</sup> a Semantic Web product developed by Network Inference, resembles SNOBASE in that it provides the capability to store and query ontologies. Unlike SNOBASE, however, ontologies cannot be manipulated within the Cerebra knowledge base itself. Ontologies are modified outside the knowledge base, either manually via a companion tool called Construct, or through a custom-built automated utility. Construct is primarily used to create ontologies and deploy them to Cerebra. Cerebra uses a query language based upon XQuery<sup>11</sup> to query its ontologies, with results returned in XML. Cerebra does not have an extensively developed, low level Java API. Instead, it uses a Web Service interface for client–server interaction.

#### 4.5.2.3 Jena2

Jena2,<sup>12</sup> the third and final tool we evaluated, is a Java-based, open source ontology framework supported by Hewlett Packard and participating outside developers. Jena differs from the other two tools in that it is less of a COTS tool and more of a framework that can be used to build a custom Semantic Web application. Therefore, Jena has an extensive API with JavaDocs. As part of its API, Jena has both predefined functions to manipulate ontologies

---

<sup>9</sup> <http://www.alphaworks.ibm.com/tech/snobase>

<sup>10</sup> <http://www.networkinference.com>

<sup>11</sup> <http://www.w3.org/XML/Query>

<sup>12</sup> <http://www.hpl.hp.com/semWeb/jena.htm>

and the capability to define custom functions. Jena uses RDF Query Language (RDQL) to query ontologies. Jena's inference support includes predefined reasoners that come with the Jena distribution as well as the capability to plug in another reasoner. Predefined reasoners include a transitive reasoner, RDFS rule reasoner, OWL reasoner, DAML micro reasoner, and a generic reasoner.

#### **4.5.2.4 Tool Evaluation**

Overall, our evaluation of SNOBASE showed that it was relatively immature. While documentation indicated its support for DQL, in actuality the support was minimal at best. Furthermore, upon attempting to use the ontology management system, we realized that the tool required an outside connection to the Internet and did not have support for proxy servers. This meant we could not use SNOBASE behind the MITRE firewall. While the general approach taken to develop the tool appears promising, it was not adequate for our purposes at the time of this research.

Cerebra, while much more mature than SNOBASE, was restrictive in many respects. An important shortfall of Cerebra was its lack of a well-developed API for customization and extension: its API was too rudimentary for us to easily integrate with GSIX. Furthermore, we questioned the maturity of Cerebra's querying capability. Finally, the tool had no ability to change the ontology from within the knowledge base. These challenges led us to decide against using Cerebra for this research.

Based on our requirements and on tool capabilities, we selected Jena2 for use in our prototype development. As mentioned earlier, its flexibility allowed us to develop a custom application, leveraging its ontology toolkit and inference support. While the transitive reasoner appeared useful to run queries as part of our pruning component, the general rule reasoner was suitable for transforming the ontology within our transformation component. Furthermore, Jena had predefined ontology manipulation functions such as intersection and union that were useful in our implementation. While RDQL was not simple to use, it appeared to have sufficient functionality and there were no indications other query languages were easier or more capable.

One of our needs was to have the result of a query be a portion of the ontology that is queried. Natively, RDQL returns value bindings to variables specified in the query. However, we found a utility developed by another MITRE researcher, Amy Kazura, that took the results of the RDQL query and created RDF/OWL-tagged data from it. This provided the results in the format we needed.

#### **4.5.3 Implementation**

The implementation of the Semantic Web guard prototype was driven by our design approach, wherein the guard was developed at increasing levels of complexity (Go/No Go, Pruning, and Transformation). While our design identified three phases of complexity for a

Semantic Web guard, our implementation followed a two-phase approach. As we developed the detailed design for our prototype we discovered that the first two levels of complexity (Go/No Go and Pruning) shared the same implementation details, and therefore we implemented two phases (Pruning and Transformation). We envisioned the prototype implementation as a set of filter components that would be responsible for pruning and/or transforming content as it moved across security boundaries.

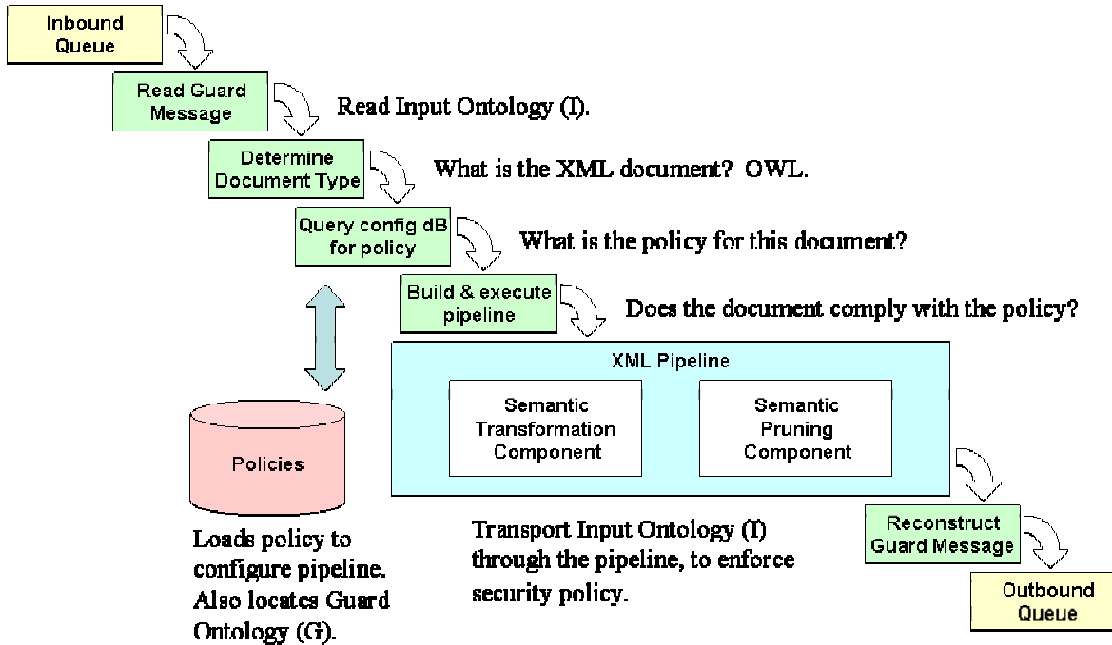
In the first phase, we created a semantic pruning component to remove information that was not allowed to pass through the security guard. In reality this filter queried for allowable content using RDQL. Jena provides the ability to run RDQL queries against an ontology loaded into a Jena inference engine. As previously mentioned, we used Amy Kazura's utility to transform the results back to RDF/OWL-tagged data, and also used a transitive reasoner to infer relationships within the instance data. With this capability we implemented the airspace release rules for pruning in RDQL syntax. These rules selected what was allowed to pass through the security guard. Anything not selected was defined as ineligible to pass through the security guard and was thus discarded.

In the second phase of the prototype development, we created a semantic transformation component to transform content before it was evaluated for eligibility to pass through the security guard. We used the general-purpose rule-based reasoner available in Jena2 to perform the transformations. This allowed us to develop custom rules that were used by the reasoner and applied to the ontology. As in pruning, we had to define airspace release rules for transformations in Jena's rule-based syntax. We used the rule-based reasoner since RDQL was not sufficient to modify semantic content. For example, one transformation rule required "Special Operations Forces" airspaces to become "No Fire Area" airspaces, but the query language could not provide this capability. We decided to pass the ontology through the transformation component before the pruning component at runtime so that instances would be transformed to some allowable type before being pruned in the final processing stage. The complete set of RDQL pruning queries and transformation rules can be found in Appendix G.

To test our semantic filter components we used GSIX. For the Semantic Web prototype, we created two GSIX pipeline components: a semantic pruning component and a semantic transformation component. These components allowed GSIX to operate as a security guard for ontologies. However, in order to take advantage of GSIX we needed to configure it for our needs and create extensions that made semantic pruning and transformation processing possible.

Figure 4-9 illustrates how the semantic filter components fit into the GSIX prototype. After the input ontology is read into GSIX, GSIX interprets its policy file to configure the correct pipeline of components to execute. Note that the GSIX policy file should not be confused with the security policy of the guard (discussed in earlier sections), which determines the release rules applied. GSIX then transports the input ontology through the pipeline. It first visits the semantic transformation component, then the semantic pruning component. Once

fully modified by the security policy, the ontology exits the security guard. A detailed description of GSIX is available in Appendix A.



**Figure 4-9. Semantic Filter Components Within GSIX**

Figure 4-10 shows the data flow through the Semantic guard. Initially, the security guard contains 1) its own ontology, representing the knowledge of the guard, 2) a set of policies, implemented as queries and rules, and 3) an inference engine to reason on the ontology. As an input ontology enters the guard, the data flow is as follows:

1. The input ontology (I) is loaded into the security guard.
2. The guard and input ontology are merged (G+I) and sent to the semantic transformation component. This component loads the airspace transformation rules into the Jena Rules Engine and applies the rules to the input ontology, transforming it (I'), resulting in a modified ontology (G+I').
3. The resulting guard and input ontology (G+I') is sent to the semantic pruning component. This component loads G+I' into the Jena transitive reasoner. It then loads the RDQL input file and proceeds to apply the airspace queries to the data. The results of the queries, which contain the data allowed to traverse the security guard, are merged. The final results of this step are the guard ontology and further modified input ontology (G+I'').



4. The transformed input ontology ( $I'$ ) is then intersected with  $G+I''$  to strip out the guard ontology data and any inferred data before releasing it from the security guard.

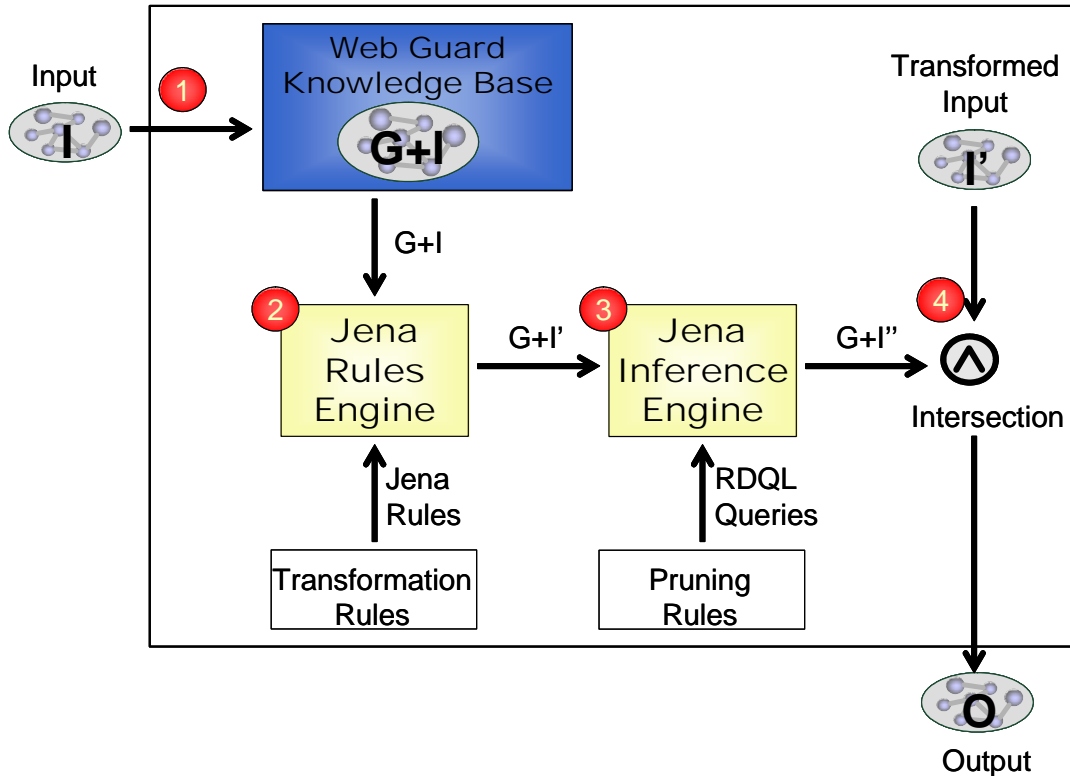


Figure 4-10. Data Flow Through Semantic Web Guard

## 4.5.4 Testing

### 4.5.4.1 Testing Strategy

Once we finished our prototype implementation, our next step was to test the Semantic Web guard to examine its function as a security guard using simulated operational data. As in our implementation approach, we evaluated the guard in two stages. First we assessed how the security guard performed the pruning function only. Second we assessed how the guard performed, first in transforming the instance data and then in pruning it in a two-stage pipeline (as described above). Examining the airspace release rules, we created a list of test cases that would thoroughly analyze all aspects of the release rules and potential interactions between the transformation and pruning components. To create our twenty-one test cases we

used the Network Inference Construct™ tool. The tool allowed us to load our guard ontology and create a range of input ontologies from the simple to the complex. Construct™ also allowed us to visually modify ontology diagrams and export the resulting ontology in OWL format. For a complete listing of the test cases refer to Appendix H.

#### 4.5.4.2 Test Results

In general the actual results matched our expected test results. However, we did identify some problems.

First, we noticed potential problems with our pipeline implementation of the security guard due to fundamental interdependencies between pruning and transformation. We discovered potential interactions between the transformation rules and the pruning rules. For instance, if the pruning component is supposed to remove an attribute of some specific type of airspace, but that airspace has previously been modified in the transformation component, the potential exists for the security guard to release unauthorized content. For example, one of the pruning rules states “Any ‘Restricted Operation Zone’ (ROZ) with *laser* in the name field has the active time period removed” and one of our transformation rules states “Any airspace of type ‘Special Operations Forces’ (SOF) gets changed to type ‘No Fire Area’ (NFA) and name and description fields get deleted.” Because SOFs are a subclass of ROZ, SOFs inherit rules applied to ROZs. Therefore, a ROZ with *laser* in the name field should have the active time period removed. However, if a SOF airspace is first changed to an NFA, the NFA rule applies when that airspace arrives at the pruning component and the active time period is not removed.

We used the pipeline approach because it was relatively easy to implement these two components in GSIX. This problem could be overcome by creating a more complex component that performed both pruning and transformation in the same filter.

Second, we identified some minor issues related to the Jena2 toolkit. One problem we found involved the query language used in Jena2, RDQL. RDQL had a limitation in that the triples specified in the query must exactly match those in the ontology for a resource to be returned in the result set. For example, if airspace A only contains a name and the query asks for name and description, A will not be returned in the results. What is needed is an *OR* condition, which would allow triples that match a subset of the conditions specified in the query to be returned.

While testing we also found a bug in the latest release of Jena version 2.1. This bug caused the transformation component to work improperly by not modifying attributes consistently. Eventually, we found a patch released for this bug, but it was too late in our effort to incorporate the patch into our prototype.

Finally, we found that the intersection method within Jena did not deal well with anonymous classes and would assign names for the internal representation. This caused a problem when

we performed the intersection to remove the guard ontology and any inferred data. Our work-around was to disallow anonymous classes in our instance data. We mention these problems to emphasize that development tools in this area have not yet completely matured. For a complete listing of the test results refer to Appendix H.

The working prototype, including the source code and other associated information is available to MITRE employees at the Internal Source Forge site (<http://developer.mitre.org/projects/semanticguard/>).

## 4.6 Semantic Web Results

This section discusses the results of our experiment using a Semantic Web guard prototype. We divided our results into three categories: needs, observations, and recommendations.

### 4.6.1 Needs

#### 4.6.1.1 Opportunities Offered by Semantic Web Technologies

Today the large amount of data that requires reliable human review creates a huge bottleneck in information sharing. Semantic Web technologies are intended to allow machines to make more decisions. The machine review inherent in these technologies provides a tremendous opportunity to increase the amount of data that can be shared. In our research, we considered how Semantic Web technologies could be applied in two ways: 1) to describe and maintain semantic context for data being sent through the security guard (as demonstrated with our test cases), and 2) to use as tools within the guard. In this second role Semantic Web technologies can provide context for:

- *Information producers or information consumers.* An ontology could be used to categorize information producers or consumers (people or systems) to allow more generalized policy to be applied. For example, a policy could state that a certain set of information from a specific class of producers can be sent to coalition partners. The ontology could specify which destinations or consumers are currently classified as coalition partners.
- *Policy.* An ontology can be used to provide context for the policy itself. For example, an ontology could capture information that determines which policy applies for which type of information.
- *Information traversing the guard.* An ontology could be used to characterize the data objects that will be sent through the guard to more generally and flexibly apply the release rules that represent the policy. For example, in our prototype the policy stated that all Special Use Airspaces (SUA) were releasable. The guard-controlled ontology specified that an NFA was a type of SUA. Therefore, the inference engine accurately

applied the general policy to all NFAs and all other types of SUAs without having to enumerate them.

- *Standard approach for security classification.* A standard security classification ontology that defines security levels and their relationships could be used by information producers and information consumers, as well as the guard itself.

#### **4.6.1.2 Approach for Sharing OWL-tagged Data**

Users are already adopting Semantic Web technologies, and we anticipate that the rate of adoption will accelerate continuously. These technologies offer too much potential to be ignored. This means that an approach for sharing OWL-tagged data across security domains is needed, because otherwise such exchanges will have to rely upon human review. Considering the operational tempo desired by users, it is unrealistic to expect that human review could keep pace with the needed cross-boundary information flow. Furthermore, even if human review were used to determine the releasability of data mapped to ontologies, better tools would be needed. For example, one compelling need would be for better tools to visualize ontologies. With current tools, it is very difficult to examine a file of OWL-tagged data and quickly discern its meaning.

### **4.6.2 Observations**

#### **4.6.2.1 Agile Business Rules**

Agile business rules will be key to successful security guards and other applications. Gartner predicts that “externalization of business rules will be a major focus of new and agile applications” through 2010 [Gar03 p.3]. Our approach separated the release rules from their processing to allow for a more agile Semantic Web guard. However, we found that implementing release rules was challenging and we believe that managing dynamic release rules would be even harder. Our simple prototype release rules were harder to implement than expected because of tool limitations and the unanticipated interdependencies between the data transformation and the data pruning steps in our implementation. In fact, we found that these steps could not be performed independently and therefore our sequential processing model was flawed.

#### **4.6.2.2 Translation of Policy**

Understanding a policy maker’s intent and translating this intent into policy are huge challenges, especially when the policy must be described in a manner precise enough to be interpreted by machines. The goal is to hide the complexity of the policy implementation but create a mechanism that allows the policy manager to effectively express information sharing rules and modify them when the situation changes. We believe that a method to simplify creation of rules that implement the policy would be the key element in developing improved

cross-boundary devices, whether they are for ontologies or other types of data. Without good policy expression, the best policy enforcer in the world is of marginal value.

#### **4.6.2.3 Rules for Ontologies**

Implementing machine-readable release rules is difficult when dealing with traditional, highly structured data but is even harder for ontologies. Ontologies provide more expressive power but also more complexity. First, there is no prescribed sequence for the data triples within an ontology. Second, there is no standard rule or query language to use with OWL ontologies; moreover, we found that the current rule and query languages can be complex and non-intuitive and lack necessary capabilities. Finally, it takes significant skill to implement release rules for data mapped to ontologies. One must understand the policy, the policy's intent, and the ontology in detail. One must also anticipate potential ways to circumvent the policy. For example, one concern we identified was the use of relations between resources to purposefully or inadvertently allow disallowed data to be released. One example of this would be to define an instance of a resource as a member of both an allowed class and a disallowed class. If the ontology does not specifically state that instances may not be members of both classes (i.e., these classes are disjoint) it may be possible to release disallowed data inadvertently.

#### **4.6.2.4 Insufficient Technology**

Developing a Semantic Web Guard with the tools currently available is challenging and would require a significant amount of special purpose code. More mature Semantic Web tools are needed to visualize ontologies, to visualize the effects of applying policy to ontologies, and to interrogate ontologies for vulnerabilities. These tools must not only provide the desired functionality, but they must also be thoroughly tested to understand their vulnerabilities if used within a security guard. We also believe that a standard and robust query language and rule language to use with OWL ontologies would simplify the implementation of a Semantic Web guard.

#### **4.6.2.5 Implementation of a Semantic Web Guard**

Implementing a cross-boundary sharing approach for Semantic Web technologies is non-trivial, in part because there is an inherent contradiction between security guard and Semantic Web environments. Security guards, by their nature, are restrictive. They are built to protect against inadvertent sharing of data and corruption of protected data. The Semantic Web, on the other hand, is built to be open and is perceived as a world-wide repository of semantically tagged data. Placing tight restrictions on sharing OWL-tagged data could limit the power of the Semantic Web. One difficult question addresses the appropriate balance between information sharing and information protection.

We also found that implementing information sharing policies on ontologies and data mapped to these ontologies was difficult. Our pruning component used RDQL to query the ontology for releasable data. We found that RDQL lacked the capabilities we needed and we believe that other existing ontology query languages would have similar problems. Because our transformation component required the ability to change values in the ontology, we used a combination of queries and a rule language, but this made it difficult to handle the needed interactions between these two approaches.

#### **4.6.2.6 Quantifying Risks**

As indicated in Figure 4-1, the design decisions made when implementing a Semantic Web guard reflect the amount of risk one must be willing to accept. These risks are difficult to quantify. How does one measure the risk of having a guard that provides tight constraints on data but has limited functionality versus a guard that flexibly allows data to be released but increases the chance of protected data being inadvertently released? Further, an operational community's willingness to accept risks will depend upon many factors, not all of which are technical. For example, operational needs may dictate that a community be willing to accept more risk in the short term in order to expedite information sharing.

### **4.6.3 Recommendations**

#### **4.6.3.1 Continue Awareness and Research**

While Semantic Web technologies are not yet mature, they are developing rapidly and are already beginning to be adopted. We recommend that the government monitor progress on these technologies and continue to invest in research to advance them. Specifically, we recommend the government serve as advocate for more mature tools that could be exploited to help the cross-domain problem, such as:

- Better tools to visualize ontologies, rules, or queries,
- A standard OWL query language, and
- A standard rule language to use with OWL ontologies.

Further, we recommend that the government participate in standards bodies to ensure that cross-domain security concerns are considered as the standards evolve.

#### **4.6.3.2 Consider Research to Develop Ontologies**

As mentioned previously, ontologies can be used not only to describe the data intended to pass through the security guard, but also to provide capabilities within the guard. We recommend that the government fund investigations into the development of a set of security ontologies that could be used by the guard to describe information producers and information

consumers (people or systems) to include attributes such as their mission role; security classification levels; environmental attributes that affect sharing (time, location, etc.); and the categorization of mission data objects that would be allowed to traverse the guard.

#### **4.6.3.3 Fund Research on Translating Policy into Release Rules**

We believe that the greatest need is for improved ways to translate a policy maker's intent into release rules that machines can interpret, especially when the data to be released is in the form of an ontology. We recommend the government fund research into tools to help express, visualize, and modify these data-sharing rules.

#### **4.6.3.4 Defend Against Ontology Corruption**

We recommend that ontology designers develop domain ontologies on the assumption that these ontologies and the instance data mapped to them will be shared across security domains. Therefore, the designers should restrict the data as much as is reasonable. This includes ensuring that the specified domains and ranges for resources are as restrictive as possible and defining classes of resources to be disjoint where appropriate. Steps such as these allow the inference engine to identify contradictions and can help protect against ontology corruption.

#### **4.6.3.5 Consider Constraining Web Addresses**

In the Semantic Web, all resources are uniquely identified with a URI. One concern we identified is that long URIs could be used to hide data. Therefore, we recommend that policy managers consider mitigating this risk by placing constraints on URIs (e.g., constraining their length).

#### **4.6.3.6 Consider Developing “Plug and Play” or Service-oriented Guards**

A major result of this entire security guard research effort is our belief that the implementation of release rules should be very loosely coupled to the enforcement of those rules. A loosely coupled approach would allow the same policy enforcer to be used on multiple sets of data and their respective release rules. Therefore, we recommend that the government consider the development of a “plug and play” security guard. For Semantic Web technologies this implies an approach whereby a policy manager need only “plug in” an accredited domain ontology and its associated release rules for a security guard to be ready to operate. We also recommend that the government consider creating security guarding services. In this approach, a cross-boundary data request would trigger a service request to the appropriate security guard service (i.e., the service associated with that data steward).

## Section 5

# Information Sharing Activities

One of our objectives was to capture and share lessons learned with several audiences. We have shared our research with the following government organizations:

- AF Scientific Advisory Board
- AF Rome Lab's Information Support Server Environment (ISSE) Guard program office
- AF Rome Lab's Collaboration Gateway program
- AF Electronic Systems Command (Bedford, MA)
- AF Electronic Systems Command CPSG/NIS (formerly NI7) (San Antonio)
- DISA Cross Domain Solutions Branch
- DISA's Center for Joint C2 Capabilities (Coalition Information Assurance Common Operational Picture (C-IA COP) ACTD and sponsor of GSIX)
- Intelligence Community Chief Information Officer (IC CIO)
- Intelligence Community Multi-Intelligence Acquisition Program (IC MAP)
- Intelink Management Office
- Joint Analysis Center (JAC)
- Joint Intelligence Center, Pacific (JICPAC)
- Missile Defense Agency
- Navy Space and Warfare (SPAWAR) Center, San Diego
- Navy Task Force Web, Norfolk, VA
- NSA I1232 Test Organization (formerly V43)
- NSA I124 Cross Domain Solutions Office (formerly V24)
- NSA Pacific
- Office of Naval Intelligence (ONI)'s Extended Intelligence Guard for ONI Replication (XIGOR) program manager
- US European Command



- US Joint Forces Command
- US Northern Command
- US Pacific Command
- US Strategic Command

We shared information on our research with the following developers:

- Boeing (SNS developer)
- Akimeka (Security Agent Broker developer)
- Dolphin Technologies (ISSE Guard developer)
- Lockheed Martin (Radiant Mercury Guard developer)
- Lockheed Martin (associated with Missile Defense Agency (MDA) effort)
- Trident (Collaborative Gateway developer)
- Triton (XIGOR developer)

We shared information on our research with the following MITRE personnel and forums:

- Technology Symposia (McLean, Bedford, and San Diego)
- Technology Subcommittee of MITRE Board of Trustees
- Army Contract Research and Technology Committee
- ESC Information Warfare/Information Security Council
- Cross Boundary Information Sharing (XBIS) Technical Exchange Meeting
- Web Services Technical Exchange Meeting
- G050 Technology Conference (in both 2003 and 2004)
- Principal Investigators from other MOIE projects
- Personnel supporting Command and Control (C2) Constellation Program
- Personnel supporting Synchronized Air Power Management (SAPM)
- Personnel supporting Defense Strategic Integrated Decision Environment (D-SIDE)
- Personnel supporting to Coalition Information Assurance Common Operational Picture (C-IA COP) ACTD
- Personnel supporting Command and Control (C2) Constellation Program

- Personnel supporting Net-centric Enterprise Services (NCES) Program
- Personnel supporting Multi-sensor Aerospace-ground Joint ISR Interoperability Coalition (MAJIIC)
- Periodic briefings to MITRE staff (Bedford, McLean, Colorado Springs, Hawaii, JAC Molesworth, Norfolk, VTCs to other sites)

## Section 6

# Summary

### 6.1 Where We Are

It seems that not a week goes by without the announcement of another initiative targeted at improving information sharing across government agencies and with our allies. With the ever-growing mountain of information that is collected and produced, requiring human review for information sharing has become a critical problem.

Today the most common vehicle for information dissemination is the traditional browser-based Web environment. Creating HTML documents has become a ubiquitous enterprise with a tremendous amount of today's knowledge available in this form. Unfortunately, machine understanding of natural languages has not advanced enough to reliably extract the semantic meaning of all this free text.

This severely limits how much protection a guard can provide when connecting traditional Web sources from one domain to another. As a result, Web guards that support cross-domain Web browsing are restricted to bridging domains where risk profiles show limited amounts of risk, or where the operational need is so great that the risk is deemed acceptable for a period of time.

Web Services, which are now gaining traction in the DOD environment, communicate using more semantically rich structured information than browser-based HTML. Because all communications take the form of XML documents, new validation techniques using schemas and style sheets may be applied to enforce policies. This improves the policy enforcement landscape and increase the array of situations where cross-domain sharing can be deployed.

Web Services are not a panacea, however. When reviewing the results of our Web Services research, it becomes apparent that many issues such as immature standards, poor Web Service designs, and poor schema designs and authentication still make policy enforcement a daunting task. Active education on the issues is needed for all parties involved to maximize the cross-domain sharing potential of Web Services.

The Semantic Web, still in its nascent stage, has stable core standards but supporting standards and tools are just coming to the fore. Because the Semantic Web literally codes all semantic information into a machine-understandable form, it offers the promise of supporting policies that analyze the content's meaning, which would allow much richer information to cross security boundaries in an automated fashion. The challenge, rather than being one of education as with Web Services, is one of continuing research.

## 6.2 Future Directions

We believe that the research community should concentrate its efforts in several especially important areas.

**Encourage standards bodies to consider security** during their development process.

**Develop outreach programs** that educate the Web Service and ontology developer communities on how to maximize their potential for cross-domain utilization. Provide examples of successful implementations for others to copy.

**Develop tools for policy makers** that assist with translating their policies into machine-understandable forms. Policy makers have the Herculean task of mapping their policies into rule languages and formal logics. This effort is fraught with peril and quickly becomes the domain of a few gurus who are adept at the process. Research is desperately needed to determine ways to help policy makers visualize and maintain their security policies so that they are both understandable to the policy maker and the machine.

**Encourage cross domain program offices to lead the charge.** Because the security offices were behind the power curve when Intelink was created, they provided little guidance on Web site creation. This resulted in security issues being largely ignored in the early days of Intelink. The security offices have been playing catch up ever since. While security should not act as an impediment to deploying new capabilities, providing security tips and guidelines in the design stage of development would greatly improve the potential of cross-domain utilization down the line.

## List of References

- [ASV04] D. Amos, R. Simmons, and A. Vincent, *Assessment of XML Firewall Appliances for Use with a Cross Domain Solution, Volume 1: Test Results*, MP 04W0000191V01, August 2004
- [BHL01] T. Berners-Lee, T., J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, vol. 284, no. 5, pp. 34–43, May 2001.
- [Bou04] C. Boulton, “Semantic Web to Take Center Stage at WWW2004,” May 16, 2004, <http://www.internetnews.com/dev-news/article.php/3354651>.
- [Com99a] Common Criteria for Information Technology Security Evaluation; Part 1: Introduction and general model, Version 2.1, CCIMB-99-031, August 1999.
- [Com99b] Common Criteria for Information Technology Security Evaluation; Part 2: Security Functional Requirements, Version 2.1, CCIMB-99-032, August 1999.
- [Com99c] Common Criteria for Information Technology Security Evaluation; Part 3: Security Assurance Requirements, Version 2.1, CCIMB-99-033, August 1999.
- [Dac03] M.C. Daconta, L. J. Obrst, K. T. Smith, *The Semantic Web: A Guide to the Future of XML, Web Services, and Knowledge Management*, Indianapolis, IN: Wiley Publishing, Inc., 2003.
- [Gar03] J. Fenn, Y. Natis, J. Sinur, A.Linden, *The Integrated Enterprise From 2003 to 2012*, Gartner Research Note SPA-18-8139, 3 December 2002.
- [Gos02] S. Gosnell, *Testing Security Requirements for Transferring Information between Security Domains in a Browser-Based Web Environment*, MP03B000003, November 2003.
- [Gru93] T.R. Gruber, “A Translation Approach to Portable Ontology Specifications,” *Knowledge Acquisition*, vol. 5, issue 2, pp. 199–220, June 1993.
- [Jac03] D. Jacobs, *Security Guards for Web Services: Lessons Learned and Recommendations*, MP 03W0000188, September 2003.
- [Ree02] N. Reed, *Survey of Unified Command Requirements for Transfer of Data between Security Domains*, MP02B 000000024 R1, September 2002.
- [Ree03a] N. Reed, *Requirements for Transferring Information Between Security Domains in a Browser-Based Web Environment*, MP 03B0000039, September 2003.

- [Ree03b] N. Reed, *Architecture Alternatives for Transferring Information Between Security Domains in a Browser-Based Web Environment*, MP 03B0000039, September 2003.
- [Ree04] N. Reed, *Cross Domain Solutions (CDS) Technology Survey*, Version 1, MP04W0000153, June 2004.
- [Shi00] R. Shirey, *Internet Security Glossary. Request for Comments 2828*. May 2000.  
<http://www.rfc-editor.org/rfc/rfc2828.txt>
- [Sim03] R. Simmons, *XML-Based Enforcement of Cross-Domain Security Policies*, MITRE Technical Report MTR03W 0000045, October 2003
- [Sim04] R. Simmons, *XML Schema Guidance for Cross Domain Security Policy Enforcement*, MP 04W0000204, September 2004
- [Uni98] *Uniform Resource Identifiers (URI: Generic Syntax)*, Request for Comments Number 2396, August 1998

## Appendix A

# GSIX Description

## A.1 Features

Guarded Sharing of Information with XML (GSIX) is a prototype that demonstrates new ways to enforce security policies on information contained in XML documents exchanged between security domains. It uses XML technologies – such as Extensible Stylesheet Language Transformations (XSLT), XML Schemas, and XML-Signature – to validate and alter the XML information. This section provides an overview of some of the key features of GSIX.

### A.1.1 Web-enabled

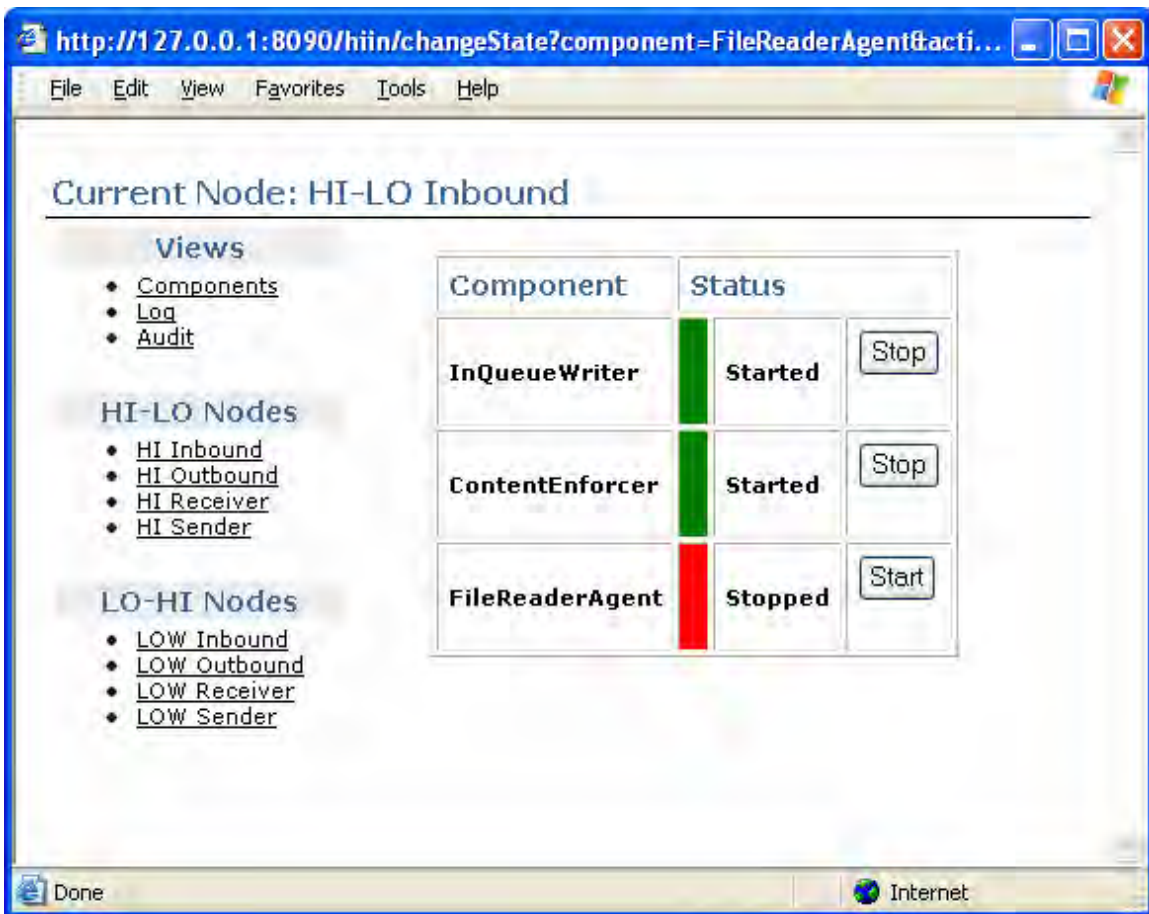
Each node of GSIX contains an embedded Web server engine. The Web server controls the initialization and lifecycle management of the components running in the node, along with an interface to access auditing and logging data. This allows administrators to visit each node, manage components, view data, and trace a message via a Web browser as it passes through the guard.

### A.1.2 Plug-able Components

To allow extensibility and reusability, the architecture is built around a registry that controls access to the components in that node. Components are Java objects that implement a particular interface and are configured via an XML file.

When a user starts the guard, an embedded Web server is executed for each node. Once the Web server is initialized, it will execute a registry that is responsible for starting all components registered with it. At this point each component is running in its own thread to allow for asynchronous execution. This is especially important for the components that work directly with the messaging service to transfer messages between nodes.

The lifecycle of each component can be managed via the Web interface (Figure A-1). Because components run independently of each other, it is possible to stop portions of the guard while leaving others running. This is useful in making policy changes, as one could stop the guard component to change a policy while leaving the in-processing component running to continue to collect incoming messages.



**Figure A-1. Components**

To extend the capabilities of the guard one would only need to develop a new component, add it to the component XML file, and restart the guard.

### **A.1.3 Pipeline Content Enforcer**

The first version of GSIX contained a content-enforcer that was hard-coded with the processors used to handle an XML message. Although this approach worked well, we recognized the need to configure the guard to handle very complex messages that may involve several processing steps. What was required was the ability to create reusable message processors that can be configured in a flexible manner. This led to our design of the pipeline component.

The pipeline uses an assembly line approach to process messages. Each processor in the pipeline is specialized at performing one task well. When a particular processor completes its



task it takes the resulting output and forwards it to the next processor in the pipeline. A pipeline is configured via an XML file (Figure A-2) and gives an administrator the ability to assemble processors to perform complex processing on a message.

**Figure A-2. The Configuration File**

The current implementation of the pipeline is configured via the base guard configuration file. It also allows a fine level of granularity to be defined. For example, for a given sender and message type, each intended receiver can have a different pipeline.

A-3

### A.1.4 Persistent Logging

GSIX contains an embedded database to provide the ability to collect and store data for various analyses, such as the number of documents processed in the last hour or the average message size. Data can be viewed via a Web interface (Figure A-3).



**Figure A-3. Audit Logs**

An embedded database runs within the same process as the node and does not require a separate server. Auditing and logging data are stored separately in different tables and contain different structures based upon the data collection needs. For example, logging data collects system-level information such as when a component starts, while the Auditor collects information specific to processing a message through the guard. The tables below provide an outline of the data collected:

**Table A-1. Description of the Auditor Table**

<b>Field name</b>	<b>Description</b>
Sender	The sender of the information
Started	When the message entered the Guard
Filename	The name of the message file
File size	The size of the message
Message Type	The message type sent to the Guard
Auditing level	Information, Warning, Severe
Transaction ID	Unique ID assigned to each message
Message	Text message

**Table A-2. Description of the Logging Table**

<b>Field name</b>	<b>Description</b>
Error Level	Informational, Warning, Severe
Message	Text message
Source class	The class being logged
Source Method	The method being logged
Thread ID	ID of the current executing thread
Time entered	Time stamp of the entry

#### **A.1.5 Collecting Failed Messages**

Collecting information about documents that fail as they pass through the guard was recognized as an important requirement after JWID 2003. Although GSIX collects auditing information we believed that additional information was needed, including the original message. To address this need we added the Error Bin.

Any message that fails while in the content-enforcer will be routed to the Error Bin. The Error Bin collects metadata about the current document along with the offending error and actual document and writes the information to a separate directory in the guard.

Error Bin information is accessible from the Web interface (Figure A-4). Any severe error noted in the audit logs will automatically link to the Error Bin output. This allows an administrator to view all the Error Bin information and the message via a Web browser.

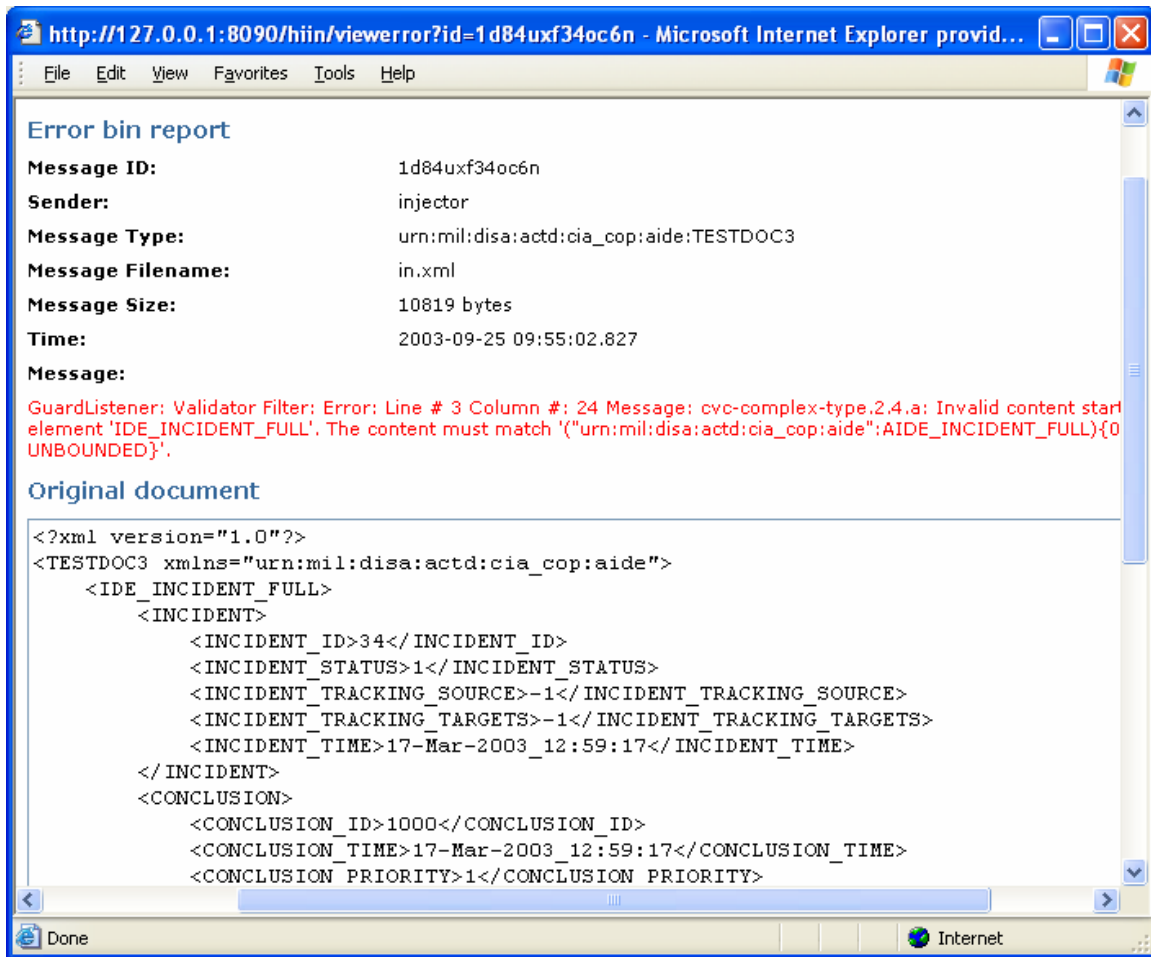


Figure A-4. Error Bin

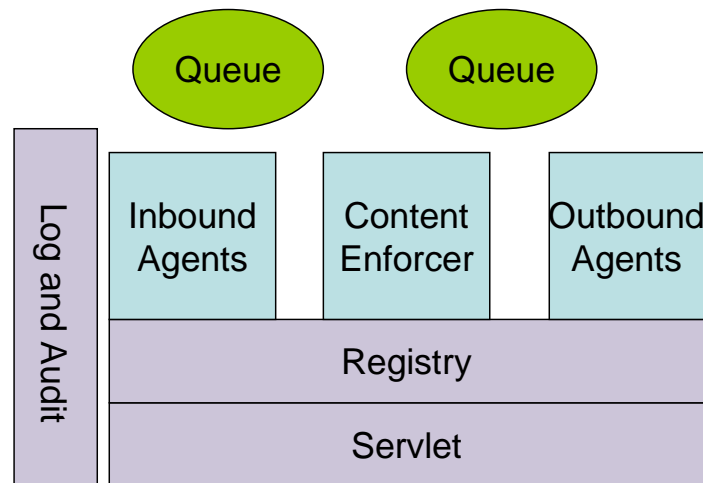
### A.1.6 Testing Messages with the Injector

To provide a simple method for testing the guard, GSIX contains a special component called the Injector. The Injector component allows the guard to process messages made available on

the local machine. A user can drop an XML message into the Injector's designated directory and the Injector will grab the message and push it through the guard. Output from processing is also placed into a designated folder on the local system making it easy to view results locally.

## A.2 Architecture

This section describes the major components of the GSIX architecture.



**Figure A-5. Key GSIX Components**

### A.2.1 Registry

The Registry provides central access to components. It exposes a standard interface to lookup and controls the lifecycle of objects registered with it. Any component that implements the LifeCyclable interface can be added to the Registry via the components.xml file. The format of the components.xml file allows a user to configure the lookup name of the object and to set arbitrary properties for the component. For example:

```
<components>
<component name="InQueueWriter" class="mitre.gsix.jms.QueueWriter">
<arg name="QueueName" type="string" value="hi-in"/>
</component>
</components>
```

Where:

- Component name is the lookup name of the component
- Component class is the full package name of the class to load into the registry
- Arg name is a property to set on the object. (NOTE: there must be a setter method on the component that matches this name. For example, `setQueueName(String v)`, etc...)
- Arg type is the Java type of the parameter to the method
- Arg value is the actual value to pass to the setter.

The Registry is initiated by the ControllerServlet. Once created, it will load the components.xml file, create any objects configured in the file, and add them to the Registry. After all objects have been added, the Registry.startAll() method is called to start all registered components.

### **A.2.2 Inbound/Outbound Managers**

Overall, the managers are responsible for pushing and pulling messages from the message queues. In addition, the inbound manager communicates with the CallbackManager to handle registered callbacks properly. The outbound manager assists in sending outbound messages.

### **A.2.3 Agents**

Inbound and Outbound agents are protocol specific objects that are registered with the Registry. Each agent is specialized at communicating with a specific protocol in order to obtain the message to pass through to the guard. For example, a system may have an SMTP agent that knows how to send a message via e-mail or a FileAgent that is expert at grabbing messages from the file system. GSIX can have many Inbound and Outbound agents at the same time. To add an agent to the guard, a user needs to implement the LifeCyclable interface and configure the agent via the components.xml file so that it will be added to the Registry.

### **A.2.4 Handlers**

Each protocol used within GSIX has a handler associated with it. The handler has the following responsibilities: 1) Extract the payload from the specific protocol and translate it into a GSIX guard message; 2) Register the request with the callback manager; and 3) Send message via the given protocol.

### **A.2.5 Content Enforcer**

The content enforcer is an agent that is configured via the components.xml file. Its primary responsibility is to collect messages, authenticate each message against the policy file, pass the message to the pipeline, and finally forward the message to an Outbound agent for shipping.

### **A.2.6 Message Queues**

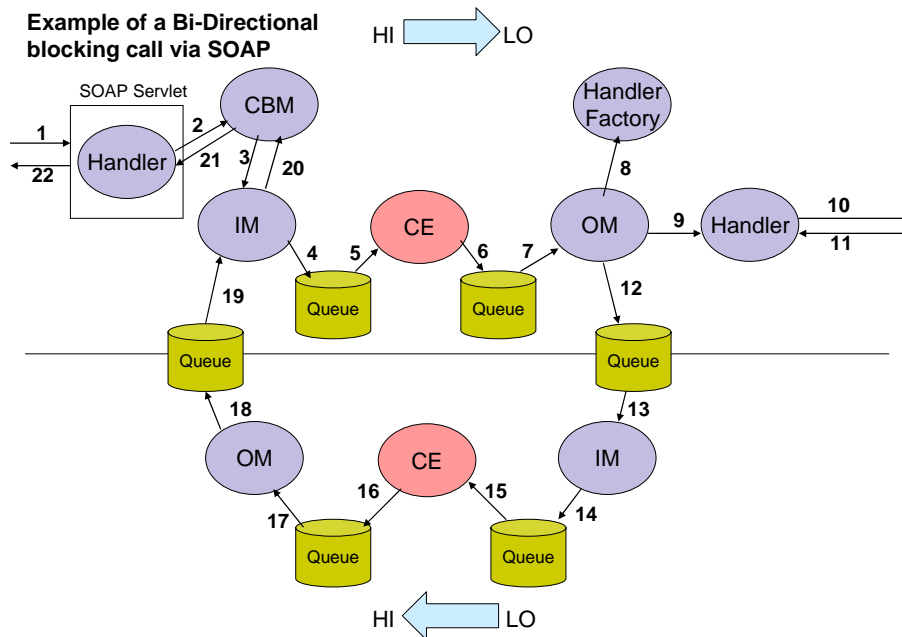
Currently GSIX uses Message Queues to pass messages through the system in an asynchronous fashion. Agents communicate to the Queues via small helpers configured with each agent in the components.xml file.

### **A.2.7 Logging and Auditing**

GSIX contains a small embedded database for storing log and audit information. The Logging utility uses the Java SDK logging API with a custom log handler that stores information in the log table. The AuditManager uses a Data Access Object (DAO) to store information in the Audit table. Logging is primarily used to capture system information and for debugging while Auditing focuses on capturing information related to processing messages in the content enforcer.

## **A.3 Message Flow**

Figures A-6 and A-7 below describe how each piece of the architecture works together to handle a message. GSIX can handle both one-way messages (HI-LOW) and Bidirectional messages (HI-LOW and LOW-HI).

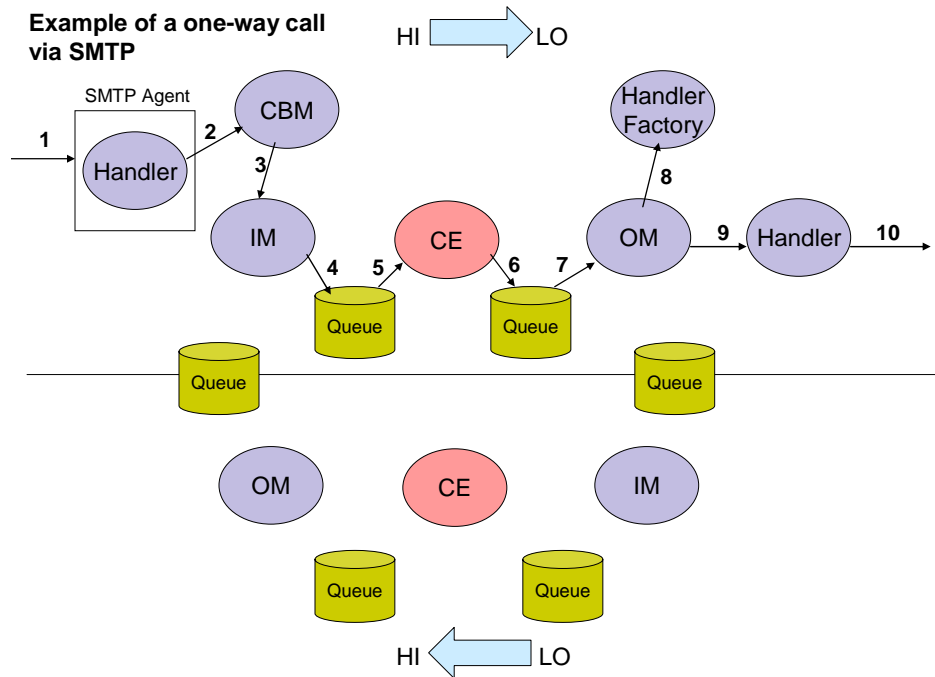


**Figure A-6. Message Flow for Bidirectional Blocking Call via SOAP**

1. Incoming SOAP request
2. Handler created in the Servlet and registered with the CallbackManager (CBM)
3. CBM passes the GuardMessage (GM) to the InboundManager (IM)
4. IM publishes the GM on the Inbound Queue
5. ContentEnforcer (CE) reads the GM from the Queue
6. CE processes the message and publishes it to the Outbound Queue
7. OutboundManager (OM) reads the message of the queue, checks the type of call and the type of protocol
8. OM creates a Protocol Handler for the message based on its protocol type
9. OM invokes the handler to send a request to an external source and blocks waiting for a response.
10. Handler sends a SOAP request to the external service
11. Handler receives a response from the external service



12. OM packages the response in a GM and places it on the boundary queue, since this is a Blocking callback
13. IM reads message off the boundary queue
14. IM checks whether it has a handler waiting for a response from this transaction. If not, IM forwards message on to the inbound queue
15. CE reads the message off the queue and processes it against policy
16. CE publishes the message to the outbound queue
17. OM reads the message off the queue. Checks the type of call and decides whether to forward or send a request. In this case it forwards
18. OM writes message to the boundary queue
19. IM reads message off the queue and checks to see if it has a handler waiting for a response from a GM with its transaction ID
20. If the IM is waiting on this response, it calls the CBM to do the callback
21. CBM provides the handler with the response
22. Handler returns the response to the blocking Servlet, which in turn sends the response back to the sender.



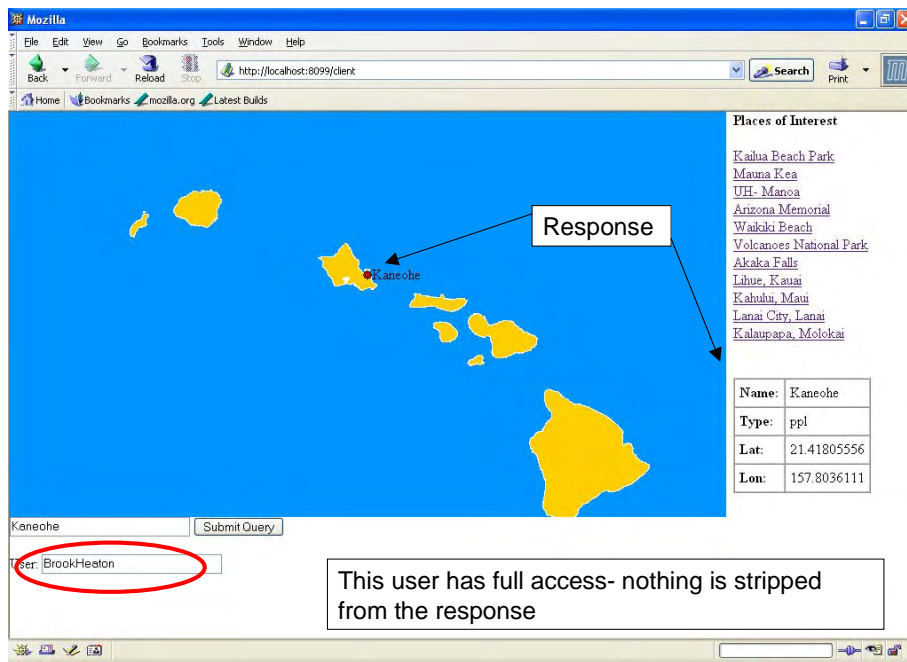
**Figure A-7. Message Flows for One-Way Call via SMTP**

1. Agent reads mail off a server
2. Handler created in the Agent and registered with the CallbackManager (CBM). Note: Since this is a one-way message no callback is registered. This is to keep the architecture consistent across call types.
3. CBM passes the GuardMessage (GM) to the InboundManager (IM)
4. IM publishes the GM on the Inbound Queue
5. ContentEnforcer (CE) reads the GM from the Queue
6. CE processes the message and publishes it to the Outbound Queue
7. OutboundManager (OM) reads the message from the queue, checks the type of call and the type of protocol
8. OM creates a Protocol Handler for the message based on its protocol type
9. OM invokes the handler to send a response to an external source
10. Handler sends an e-mail to a receiver.

## Appendix B

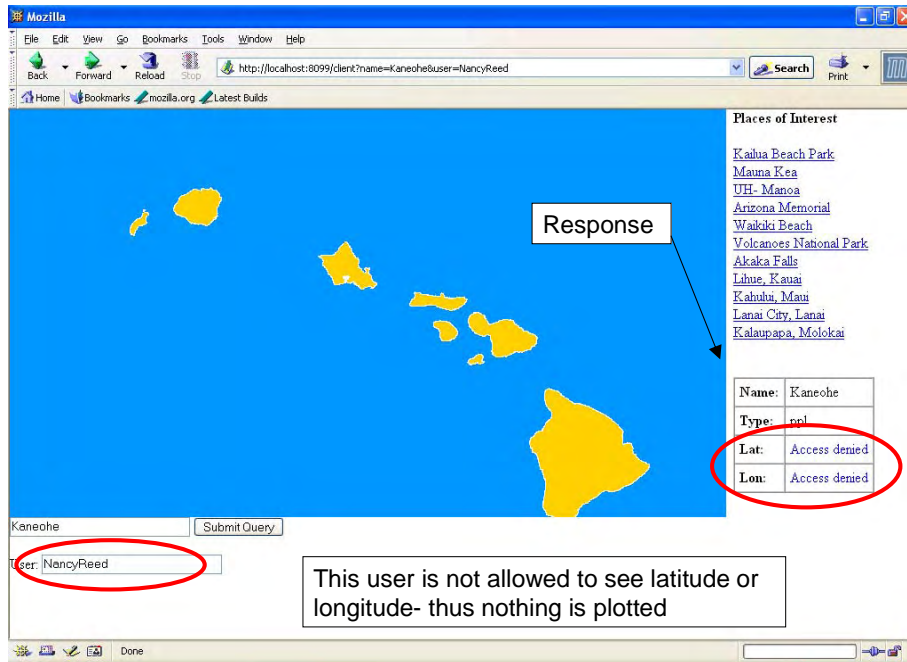
# Web Service Prototype Screenshots

The Web client allows a user to enter a search term, which in turn invokes the Web Service. As illustrated in Figure B-1, when BrookHeaton enters a term, the term is sent as a request to the gazetteer, which generates a response. Since BrookHeaton is fully authorized, the guard returns the response to the client with no changes; the web client displays the data on the map.



**Figure B-1. User with Full Access**

As illustrated in Figure B-2, NancyReed, is not authorized latitude and longitude information. The guard strips the latitude and longitude from the response. The client displays the feature type but does not display the latitude or longitude, thus the client is unable to indicate the location of the feature on the map.



**Figure B-2. User with Limited Access**

## Appendix C

# Web Service Standards

This appendix documents our examination of current and emerging XML standards to evaluate their potential impact on Web Service-capable guards. These standards are presented roughly in order of their expected timetable for usefulness. Those presented first are more likely to be useful now; those presented later are more likely to be useful at a later date.

Among the standards we examined is a related set of Web Services standards created by IBM and Microsoft. These include WS-Security, WS-Policy, WS-Trust, WS-SecureConversation, WS-Federation, WS-Privacy, and WS-Authorization.

### C.1 XML-Signature

**Author:** W3C & IETF

**Standard:** <http://www.w3.org/TR/xmlsig-core/>

**Current Release:** W3C Recommendation 12 February 2002

#### **Overview:**

XML-Signature is a specification designed to meet the special requirements for using digital signatures with XML documents [Xml04].

#### **Description:**

XML-Signature applies digital signature technologies to XML documents. It provides for multiple digital signatures at both the document and element levels. It is compatible with the traditional PKI, but is designed to account for and take advantage of the Internet and XML [Mad02]. Signature bindings include detached (the signature is in a separate document from the XML), enveloped (the signature is in the XML document), and enveloping (the XML document is in the signature).

#### **Usefulness:**

XML-Signature provides data (message) integrity, user and data (message) authentication, and support for non-repudiation.

#### **Issues and limitations:**

The PKI in one security domain does not typically intersect with the PKI in another. This would imply that digital signatures from the sending domain should be stripped in the guard, and the XML document should be re-signed using the PKI in the receiving domain. Given this restriction, the three binding options must be examined to determine which is most

suitable for a guard. As it typically deals with documents one at a time, the detached binding would not appear to be an option. The enveloping binding is probably the best choice, as the signed data is contiguous.

Stripping digital signatures may have the unintended consequence of breaking end-to-end non-repudiation in some scenarios.

If a guard re-signs XML documents using the PKI of the receiving domain, then it will have to maintain one or more keys from that domain. If the guard is bidirectional, then it may have to maintain two sets of keys. Depending upon the number of keys, key management could become a configuration management issue.

**Good Reading:**

<http://www.xml.com/pub/a/2001/08/08/xmlsig.html>

**Toolkits:**

IBM

<http://www.alphaworks.ibm.com/tech/xmlsecuritysuite>

Apache

<http://xml.apache.org/security/>

## **C.2 XML-Encryption**

**Author:** W3C

**Standard:** <http://www.w3.org/TR/xmlenc-core/>

**Current Release:** W3C Recommendation 10 December 2002

**Overview:**

XML-Encryption specifies a process for encrypting data and representing the result in XML. The data may be arbitrary data (including an XML document), an XML element, or XML element content. The result of encrypting data is an XML-Encryption element that contains or references the cipher data [CP02].

**Description:**

The basic concepts of cryptography remain much the same in XML-Encryption. The difference resides in adopting a standard format for representing and exchanging encrypted XML data [Gok02].

XML-Encryption does not replace or supersede SSL/Transport Layer Security (TLS), but provides an alternative where some data will be securely exchanged and the rest will be

exchanged as is [Map02]. It provides a mechanism for security requirements that are not covered by SSL – encrypting part of the data being exchanged and secure sessions between more than two parties.

**Usefulness:**

XML-Encryption provides confidentiality.

**Issues and limitations:**

A guard is typically required to examine the contents of a document moving from one security domain to another. To do so the guard would have to decrypt and then re-encrypt all encrypted XML documents. If there are numerous senders and receivers, then once again key management could become a configuration management issue.

**Good Reading:**

<http://www.informit.com/articles/article.asp?p=28801>

**Toolkits:**

IBM

<http://www.alphaworks.ibm.com/tech/xmlsecuritysuite>

Phaos

<http://www.phaos.com/products/xml/xml.html>

### **C.3 WS-Security**

**Author:** OASIS

**Standard:** [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=wss](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss)

**Current Release:** v1.0 – 6 April 2004

**Overview:**

WS-Security specifies how security information is added to SOAP messages [Mad03]. It is a single, flexible framework that unifies various standards-based security specifications for the purposes of securing SOAP data.

**Description:**

WS-Security describes enhancements to SOAP messaging to provide quality of protection through message integrity and message confidentiality. It defines how to attach and include security tokens within SOAP messages. A mechanism is provided for specifying binary encoded security tokens (e.g., X.509 certificates). These mechanisms can be used

independently or in combination to accommodate a wide variety of security models and encryption technologies.

WS-Security provides a general-purpose mechanism for associating security tokens with messages. It does not require a specific type of security token, and is designed to be extensible (e.g., support multiple security token formats). For example, requesters might provide proof of identity and proof that they have a particular business certification [Oas04].

Message integrity is provided by leveraging XML-Signature in conjunction with security tokens (which may contain or imply key data) to ensure that messages are transmitted without modifications. The integrity mechanisms are designed to support multiple signatures, potentially by multiple actors, and to be extensible to support additional signature formats. The signatures may reference (i.e., point to) a security token.

Similarly, message confidentiality is provided by leveraging XML-Encryption in conjunction with security tokens to keep portions of SOAP messages confidential. The encryption mechanisms are designed to support additional encryption technologies, processes, and operations by multiple actors. The encryption may also reference a security token [Atk02].

Finally, WS-Security describes a mechanism for encoding binary security tokens. Specifically, the specification describes how to encode X.509 certificates and Kerberos tickets as well as how to include opaque encrypted keys. It also includes extensibility mechanisms that can be used to further describe the characteristics of the security tokens that are included with a message [Sec02].

#### **Usefulness:**

As it uses the XML-Signature and XML-Encryption standards, it provides all the usefulness of those standards specifically for SOAP messages.

#### **Issues and limitations:**

Signature and encryption details are stored in the SOAP headers. This implies that a guard cannot delete SOAP headers without potentially impacting the functionality of the Web Service. This is true for many of the WS standards.

WS-Security does not understand the XML in the payload. To ensure the data is valid still requires a schema validator and perhaps a constraint language.

As WS-Security comprises other standards and technologies, compliance with this standard must be clearly defined.

WS-Security does not address how a SOAP client/service and a guard agree on the nature and characteristics of the security tokens that form the underpinning of WS-Security; this means that adherence to this standard does not guarantee interoperability. This has the



potential to create an interoperability barrier; WS-Trust & WS-SecurityPolicy can be used to resolve this problem.

**Toolkits:**

VeriSign

[https://www.verisign.com/corporate/news/2002/pr\\_20021210b.html](https://www.verisign.com/corporate/news/2002/pr_20021210b.html)

Microsoft

<http://www.microsoft.com/downloads/details.aspx?FamilyId=21FB9B9A-C5F6-4C95-87B7-FC7AB49B3EDD&displaylang=en>

Vordel

<http://www.vordel.com/soapbox/more.html>

## **C.4 SAML**

**Author:** OASIS

**Standard:** [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=security](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security)

**Current Release:** v1.1 – 02 Sept 2003

**Overview:**

Security Assertion Markup Language (SAML) is an XML-based security specification for exchanging authentication and authorization information [Rou02]. The primary goal of SAML is to provide standardized interoperability between security systems that supply authentication and authorization services [Sam04].

**Description:**

SAML seeks to provide a standard methodology to represent a principal's authentication and authorization information in XML format. This information can then be exchanged across intranets and the Internet [Aut02].

The OASIS group developed SAML as an XML-based framework for exchanging security information. SAML differs from other security approaches mostly because of its expression of security in the form of assertions about subjects. Other approaches use a central certificate authority to issue certificates that guarantee secure communication from one point to another within a network. With SAML, any point in the network can assert that it knows the identity of a user or piece of data. It is up to the receiving application to determine if it trusts the assertion. Any SAML-compliant software can assert its authentication of a user or data. This is important for the coming wave of business workflow Web Services standards where

secured data needs to move through several systems for a transaction to be completely processed [Coh04].

An assertion is a declaration of certain facts (statements) about a principal (subject) [Aut02]. There are three types of SAML assertions, but only one appears to be of interest to guards. The authorization decision assertion states whether a given subject has been granted specific permissions to access a particular resource. An issuing authority decides whether to grant the request by subject S for access type A to resource R given evidence E.

There are three use cases given to support the need for SAML. Of these only the authorization service appears to be useful. In it a policy enforcement point checks permissions against a policy decision point. The returned assertion can be added to the SOAP headers of the message.

SAML requests and responses can be bound to SOAP in the SOAP Body. SAML assertions in the SOAP header can provide assertions about content in the SOAP body.

Other XML standards can be used in conjunction with SAML:

- XML-Signature can be used for digitally signing and creating a canonicalized version of SAML assertions, which provides authentication, integrity, and non-repudiation.
- XML-Encryption can be used to encrypt and decrypt SAML assertions, which provides confidentiality.
- XKMS/PKI can be used to secure SAML traffic.
- XACML can be used to define access control and policy as a basis for handling SAML assertion requests. XACML and SAML are complementary technologies.

#### **Usefulness:**

SAML can be used for authentication and authorization, although there are issues with this, as described below.

SAML can also be used to provide access control for Web Services.

If a guarding solution is a multi-step process, then each step could check for proper SAML assertions from previous steps.

#### **Issues and limitations:**

For a guard to accept an externally issued SAML assertion, the guard must trust external systems. If it cannot, then it must have trusted access to a trusted SAML server to verify the assertions. This latter case somewhat defeats the non-centralized design of SAML. SAML provides protection from replay attacks by requiring the use of SSL encryption when transmitting assertions and messages specifically to prevent interception of assertions. If SAML requires SSL, then it may be limited to HTTP environments.

If each security domain provides SAML assertions, there may be SAML servers within every security domain, plus potentially another one within the guard. This increase in required infrastructure may be expensive to sustain.

**Good Reading:**

[http://www.simc-inc.org/archive0002/February02/devwed1015\\_rouault.pdf](http://www.simc-inc.org/archive0002/February02/devwed1015_rouault.pdf)

<http://java.sun.com/developer/onlineTraining/Webcasts/pdf/35plus/rpatel1.pdf>

**Toolkits:**

Phaos

<http://www.phaos.com/products/saml/saml.html>

## **C.5 XACML**

**Author:** OASIS

**Standard:** [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=xacml](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml)

**Current Release:** v1.0 - 6 Feb. 2003

**Overview:**

eXtensible Access Control Markup Language (XACML) is a general-purpose access control policy language. This means that it provides a syntax (defined in XML) for managing authorization decisions [Sun04].

**Description:**

XACML is an XML-based language for access control that has been standardized in OASIS. XACML describes both an access control policy language and a request/response language. The policy language is used to express access control policies (who can do what when). The request/response language expresses queries about whether a particular access should be allowed (requests) and describes answers to those queries (responses) [Xac04].

The typical situation is that someone wants to take some action on a resource. The user will make a request to whatever actually protects that resource (such as a file system or a Web server), which is called a Policy Enforcement Point (PEP). The PEP will form a request based on the requester's attributes, the resource in question, the action, and other information pertaining to the request. The PEP will then send this request to a Policy Decision Point (PDP), which will look at the request, find some policy that applies to the request, and come up with an answer about whether access should be granted. That answer is returned to the PEP, which can then allow or deny access to the user. Note that the PEP and PDP might both be contained within a single application, or might be distributed across several servers.

In addition to providing request/response and policy languages, XACML also provides the other pieces of this relationship, namely finding a policy that applies to a given request and evaluating the request against that policy to come up with a yes or no answer [Bri03]. Figure C-1 illustrates how XACML can be used [Com04].

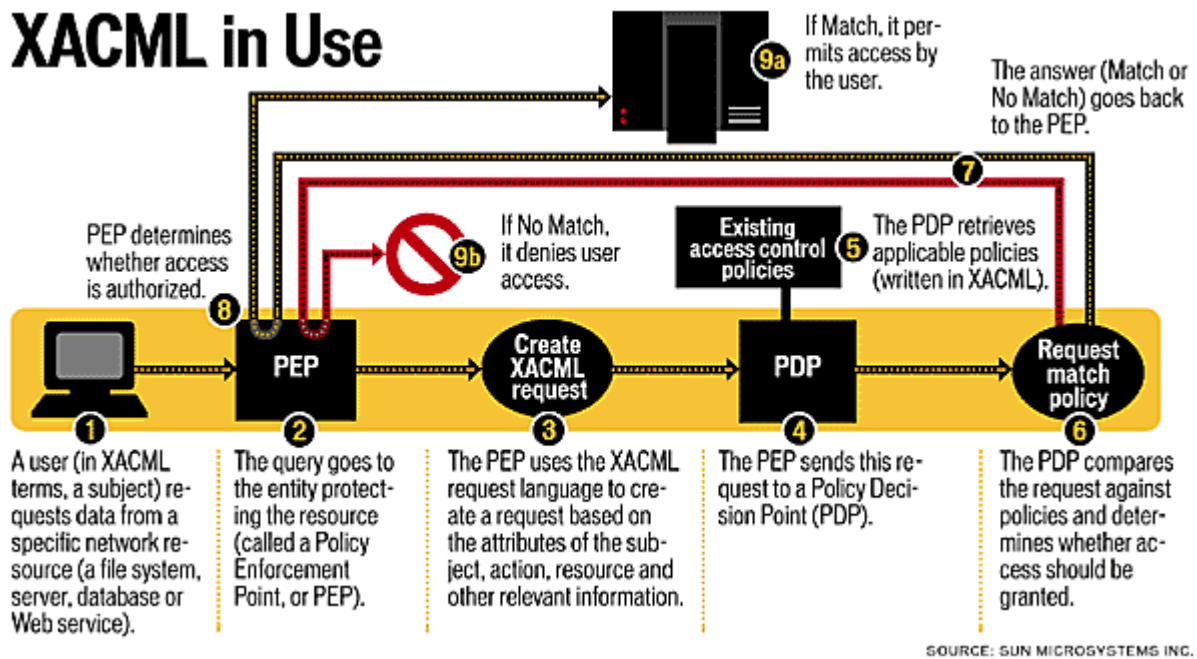


Figure C-1. XACML in Use

**Usefulness:**

One standard access control policy language could replace dozens of application-specific languages. This could be useful if a guarding solution consists of multiple systems that need to interoperate as a single unit.

This language should be explored in detail to determine if it is rich enough to support cross-domain policies. If so, it may be used to determine who is allowed to send what messages across a boundary and so on. The policy enforcer of a guard prototype, such GSIX, could be designed to use and test this.

**Issues and limitations:**

XACML is an XML technology. Although XML technologies make programming easy, they can be computationally expensive. If a user is interacting with a Web application through a

guard, repeated hits to a XACML server (or any other XML-technology for that matter) may significantly decrease application responsiveness.

Much as with SAML, if there are three security domains, then there may need to be three trusted connections to three XACML servers. This increase in required infrastructure may not always be feasible.

SAML and XACML are very flexible. This is both good (because guards can probably make them do whatever is required) and bad, because guards must constrain what they allow.

**Good Reading:**

<http://sunxacml.sourceforge.net/guide.html>

**Toolkits:**

Sun

<http://sunxacml.sourceforge.net/>

## **C.6 WS-I Basic Profile (BP)**

**Author:** Web Services Interoperability Organization

**Standard:** n/a

**Current Release:** Basic Profile Version 1.0 – 2004/04/16

<http://www.ws-i.org/Profiles/BasicProfile-1.0-2004-04-16.html>

**Overview:**

The Basic Profile consists of implementation guidelines recommending how a set of core Web Services specifications should be used together to develop interoperable Web Services.

**Description:**

BP 1.0 covers the following core Web Services standards and provides constraints and clarifications to these base specifications, along with conventions about how to use them together, with the goal of promoting interoperability:

- SOAP 1.1
- WSDL 1.1
- UDDI 2.0
- XML 1.0 (Second Edition)
- XML Schema Part 1: Structures

- XML Schema Part 2: Data types
- RFC2246: The Transport Layer Security Protocol Version 1.0
- RFC2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile
- RFC2616: HyperText Transfer Protocol 1.1
- RFC2818: HTTP over TLS Transport Layer Security
- RFC2965: HTTP State Management Mechanism
- The Secure Sockets Layer Protocol Version 3.0 [Fre04]

A majority of the specification is targeted to the audience of platform infrastructure and tool developers working on vendor-specific implementations of SOAP processors, WSDL parsers, code generators, and the like [Fer02].

**Usefulness:**

WS-I Basic Profile has the potential to increase interoperability, particularly between DoD systems that need to programmatically exchange information.

**Issues and limitations:**

This specification does not appear to be particularly relevant to guards, as guards are not responsible for enforcing interoperability standards.

The boundaries of the guard might have to be WS-I compliant to make sure they can talk to clients and services on the high and low sides. Someone needs to ensure that none of these are at cross-purposes with good security practices.

WS-I could serve as a model for a “Guard interoperability” standard, which details how Web Services should be written so that they can be allowed to cross a security boundary.

**Good Reading:**

<http://www.sys-con.com/Webservices/article.cfm?id=748>

**Toolkits:**

None identified

**C.7 XKMS**

**Author:** W3C

**Standard:** n/a

**Current Release:** W3C Note 30 March 2001

<http://www.w3.org/TR/xkms/>

**Overview:**

XML Key Management Specification (XKMS) defines a Web Services interface to a public key infrastructure.

**Description:**

With XKMS, developers can integrate authentication, digital signature, and encryption services, such as certificate processing and revocation status checking, into applications in a matter of hours – without the constraints and complications associated with proprietary PKI software toolkits [Xml04]. Most developers will only need to worry about implementing XKMS clients. XKMS server components are mostly implemented by PKI providers, such as Entrust, Baltimore, and VeriSign [Xkm04]. The client application need not concern itself with the syntax of the underlying PKI, which could be any of the following: X.509 (the most widely used), Pretty Good Privacy (PGP), Simple Public Key Infrastructure (SPKI), or Public Key Infrastructure X.509 (PKIX).

XKMS specifications are made up of two specifications, one that relates to registration of the public keys—XML Key Registration Service Specification (XKRSS) – and one that is concerned with the retrieval of information based on key information – XML Key Information Service Specification (XKISS).

Figure C-2 illustrates how XKMS works [Sal03].

XKMS provides many benefits. The benefits of XKMS include:

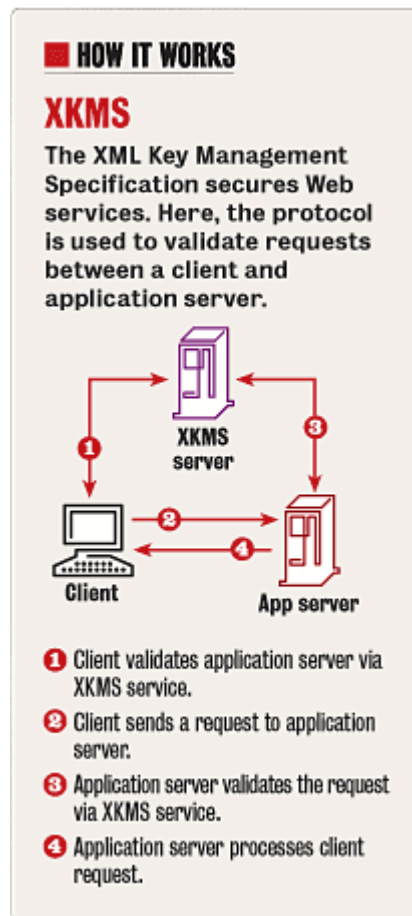
**Easy to use:** The developer-friendly syntax used in XKMS eliminates the necessity for PKI toolkits and proprietary plug-ins. The XKMS specification allows developers to rapidly implement trust features, incorporating cryptographic support for XML signatures and XML encryption using standard XML toolkits.

**Quick to deploy:** By simplifying application development, XKMS removes the need to delay PKI deployment and instead moves the complexity of PKI to server-side components.

**Open:** The common XML vocabulary used to describe authentication, authorization, and profile information in XML documents makes XKMS services completely platform-, vendor-, and transport-protocol-neutral.

**Ideal for mobile devices:** XKMS allows mobile devices to access full-featured PKI through ultra-minimal-footprint client device interfaces.

**Future-proof:** XKMS supports new and emerging PKI developments since the impact of future PKI developments is restricted to server-side components [Xkm04].



**Figure C-2. XKMS – How It Works**

XKMS messages are XML documents, as opposed to SOAP-RPC messages. They are independent of the transport or application protocol used to convey them. Further, they are defined in W3C XML Schema, and they all derive from a basic request or response abstract type [Man04].

XKMS works with the XML-Signature and Encryption standards.



**Usefulness:**

XKMS provides all the functionality that traditional PKI provides; the difference is an easier-to-use, non-proprietary interface. Guards should modernize to use this, provided the infrastructure exists.

**Issues and limitations:**

If the security guard keeps the keys internally, then this specification is not useful. Alternatively, if an XKMS server keeps the keys, then the security guard would need to trust that server.

**Good Reading:**

<http://www.nwfusion.com/news/tech/2003/0908techupdate.html>

**Toolkits:**

Phaos

<http://www.phaos.com/products/xkms/xkms.html>

**C.8 WS-Policy**

**Author:** Microsoft, IBM, BEA, SAP

**Standard:** n/a

**Current Release:** May 28, 2003 version 1.1

**Overview:**

The Web Services Policy Framework (WS-Policy) provides a general-purpose model and corresponding syntax to describe and communicate the policies of a Web Service [Met04]. WS-Policy is a building block used in conjunction with other Web Service and application-specific protocols to accommodate a wide variety of policy exchange models [Msd04].

**Description:**

WS-Policy provides a flexible and extensible grammar for expressing the capabilities, requirements, and general characteristics of entities in an XML Web Services-based system. WS-Policy defines a framework and a model for the expression of these properties as policies. Policy expressions allow for both simple declarative assertions as well as more sophisticated conditional assertions.

WS-Policy defines a policy as a collection of one or more policy assertions. Some assertions specify traditional requirements and capabilities that will ultimately manifest themselves on the wire (e.g., authentication scheme, transport protocol selection). Some assertions specify requirements and capabilities that have no wire manifestation yet are critical to proper

service selection and usage (e.g., privacy policy, QoS characteristics). WS-Policy provides a single policy grammar to allow both kinds of assertions to be reasoned about in a consistent manner.

WS-Policy stops short of specifying how policies are discovered or attached to a Web Service. Other specifications are free to define technology-specific mechanisms for associating policy with various entities and resources. Subsequent specifications will provide profiles on WS-Policy usage within other common Web Service technologies [Msd04].

#### **Usefulness:**

A guard that supports cross-domain Web Services will probably act as a proxy. If this proxy has requirements, such as WS-Security, these can be expressed to the Web Service clients using WS-Policy. For example, if an instance of a guard only uses certain security tokens (like Kerberos) or always requires an XML signature, the policy could be used to specify that.

WS-Policy could also be used as part of the advertisement of the proxy. It can be used to describe in a machine-readable way how the proxy must be called. WS-policy grammar can be embedded in an XML document, a WSDL definition, or a UDDI tModel. A Web Service client should not even bother trying to call the Web Service if it cannot meet the requirements. Further investigation is needed to determine the relationship between the requirements of the guard and the requirements of the Web Service in the other security domain. For example, if the guard only supports Kerberos security tokens and the Web Service only supports UsernameToken security tokens, can this service be accessed through this guard?

The MessagePredicate assertion can be used to develop an XPath-based constraint language [Xml03].

#### **Issues and limitations:**

WS-Policy does not define any policy assertions itself; it simply defines the framework for packaging and processing them in a standard way. To be most useful it will require the use of other related standards, such as WS-PolicyAssertions (generic policy assertion grammar), WS-SecurityPolicy (security-related policy assertion grammar), and WS-PolicyAttachment (associating policy expressions with subjects).

#### **Toolkits:**

Microsoft

<http://msdn.microsoft.com/Webservices/building/wse/default.aspx>

## C.9 WS-Trust

**Author:** IBM, Microsoft, and Verisign

**Standard:** n/a

**Current Release:** Version 1.1 (May 2004)

### Overview:

WS-Trust is a proposal that enables security token interoperability by defining a request/response protocol by which SOAP actors can request of some trusted authority that a particular security token be exchanged for another [Mad03].

### Description:

The following list identifies the key driving requirements for this specification:

- Requesting and obtaining security tokens
- Managing trusts and establishing trust relationships
- Establishing and assessing trust relationships
- Password authentication

The Web Service security model defined in WS-Trust is based on a process in which a Web Service can require that an incoming message prove a set of claims (e.g., name, key, permission, capability, etc.). If a message arrives without having the required proof of claims, the service *should* ignore or reject the message. A service can indicate its required claims and related information in its policy as described by WS-Policy and WS-PolicyAttachment specifications.

A requester can send messages that demonstrate its ability to prove a required set of claims by associating security tokens with the messages and including signatures of the message that demonstrate proof of possession of (the contents of) the tokens.

If the requester does not have the necessary token(s) to prove the claim, it contacts an appropriate authority and gets the tokens. These “authorities,” which we refer to as security token services, may in turn require their own set of claims. Security token services form the basis of trust by issuing security tokens that can be used to broker trust relationships between different trust domains.

This specification also defines a challenge response protocol. This is used by a Web Service for additional challenges to a requester regarding the freshness and proof-of-possession information provided by the requester [Web04].

**Usefulness:**

To improve interoperability, a Security Token Service (STS) could be placed on the guard (or on a sidecar).

**Issues and limitations:**

WS-Trust addresses potential interoperability issues by defining a simple request/response for security token exchange. This presents a potential difficulty: the guard is now required to trust and exchange information with an external STS. On the other hand, if a well-defined and advertised STS resided on the guard itself, it presumably would not need WS-Trust.

For the X.509 world, a proposal already exists for XML-based token issuance and token validation: namely, the X-KRSS and X-KISS components of the XKMS currently being standardized under the W3C. It remains to be seen if WS-Trust and XKMS will compete, cooperate, or coexist in this area.

**C.10 WS-SecureConversation**

**Author:** BEA, Computer Associates, IBM, Layer 7 Technologies, Microsoft, Netegrity, Oblix, OpenNetwork Technologies, Ping Identity Corporation, Reactivity, RSA Security, VeriSign, and Westbridge

**Standard:** n/a

**Current Release:** Version 1.1 – May 2004 (a revised public draft release provided for review and evaluation only)

**Overview:**

WS-SecureConversation establishes session-based security.

**Description:**

This specification defines extensions that build on WS-Security and WS-Trust to provide secure communication across one or more messages. Specifically, it defines mechanisms for establishing and sharing security contexts, and deriving keys from established security contexts (or any shared secret).

This specification introduces a security context that is defined as a new WS-Security token type, which is obtained using a binding of WS-Trust [And04].

WS-Security provides mechanisms for securing a single message in a one-way message exchange. Often interactions between a Web Service and a consumer result in multiple messages being exchanged. While each message could be secured in isolation, it is more efficient to establish some form of context that the Web Service and consumer share and use that context to reduce the burden of securing each message exchanged. WS-SecureConversation defines a Security Context Token to perform this task. A security

context provides a way to provide session-based security, rather than establishing new keys for every message [Msd04].

The key point of WS-SecureConversation is avoiding the overhead of asymmetric public/private key authentication on each request. Symmetric keys (a shared secret) are orders of magnitude faster than asymmetric keys so they are much more efficient to use. The issue is how the shared secret can be transferred securely. This is where WS-SecureConversation provides a way to use asymmetric keys to swap a shared secret that is used from then on (the same approach taken in SSL).

To use a bar analogy, the point of WS-SecureConversation is that instead of having to show your driver's license to prove your age every time you order a drink, you can prove your age at the door and get a wristband confirming that you are of legal age. It is faster for the bartender to look at a wristband than to calculate your age from the date of birth on your license every time you order a drink [Hyp04].

**Usefulness:**

This standard is only useful if a guard decides to maintain state. Although less useful for simple document exchange, it might be quite useful for Web, chat, or collaborative applications that constantly interact.

**Issues and limitations:**

Current guards are stateless, which means the WS-SecureConversation specification will not be useful unless guards modernize to handle stateful interactions. This will probably require significant design changes.

**Good Reading:**

[http://www.cs.virginia.edu/~xf4c/WS-SecureConversation\(v3\).ppt](http://www.cs.virginia.edu/~xf4c/WS-SecureConversation(v3).ppt)

**Toolkits:**

Microsoft

<http://msdn.microsoft.com/Webservices/building/wse/default.aspx>

## **C.11 WS-Federation**

**Author:** Microsoft, IBM

**Standard:**

**Current Release:** Version 1.0 (July 8 2003)

**Overview:**

WS-Federation describes a standard technology framework for creating and authenticating user identities, then using Web Services to share that identity within a company, or with customers or business partners [Com04]. It lets different security realms federate by supporting and brokering trust of identities, attributes, and authentication between participating Web Services [Baj03].

**Description:**

The WS-Federation is an effort to create a specification that will enable individuals and enterprises to authenticate each other quickly on many heterogeneous IT infrastructures [Gar04] – known as federated identity management. In the business world a company's value network spans many organizations, systems, applications, and business processes. There is no single entity or company that can purport to centrally manage or control identity information about its constituents in this end-to-end value network. Centralization would introduce significant friction in e-business collaboration, integration, and automation, resulting in high costs of identity management and reduced efficiency [Fed03].

**Usefulness:**

A guard could offload authentication responsibilities to other systems.

**Issues and limitations:**

WS-Federation requires trust among various domains. This is contrary to the current mindset, where a guard only trusts itself. Before a standard such as WS-Federation can be useful, the mindset must change. Then guard developer must devise secure ways for the guard to with devices in the low and/or high domains.

This specification overlaps some of the work being done by the Liberty Alliance, which is creating a framework for federated identity management. These standards will likely have to merge eventually.

**Toolkits:**

Ping

<http://www.sourceid.org/content.do?page=WS-Federation>

**C.12 WS-Privacy**

**Author:** Microsoft, IBM

**Standard:** n/a

**Current Release:** n/a

**Overview:**

WS-Privacy has not yet been released, so little information is available. It is a way for organizations that create, manage, and use Web Services to state their privacy policies and require that incoming requests make claims about the senders' adherence to these policies.

[Pri04] <http://www.serviceoriented.org/ws-privacy.html>

**Description:**

WS-Privacy is not yet published, but it is described in the roadmap document. It may use WS-Security (for basic security), WS-Policy (as a structured way to ask privacy questions), and WS-Trust (as a way to manage privacy across several transactions) to provide for privacy controls in Web Services networks. Systems can use WS-Privacy to make assertions about their privacy practices – for example, they can promise not to pass the data on to any third parties.

WS-Privacy will add a new set of headers for use in a SOAP message to communicating privacy policies. It may be to SOAP what the W3C's Platform for Privacy Preferences (P3P) is to Web applications [Gal04].

**Usefulness:**

More information about this standard will be required to determine its usefulness.

**C.13 WS-Authorization**

**Author:** Microsoft, IBM

**Current Release:** n/a

**Standard:** n/a

**Overview:**

WS-Authorization has not yet been released, so little information is available. It will describe how to manage authorization data and authorization policies for Web Services [Sec04].

Description:

WS-Authorization deals with authorization decisions in the context of Web Services. It will describe how access policies for a Web Service will be specified and managed [Fel04]. In particular, it will describe how claims may be specified within security tokens and how these claims will be interpreted at the endpoint [Hij03].

Its objectives overlap with those of XACML.

**Usefulness:**

More information about this standard will be required to determine its usefulness.

## C.14 WS-Addressing

**Author:** IBM, Microsoft and BEA

**Standard:** n/a

**Current Release:** March 2004

### Overview:

Web Services Addressing (WS-Addressing) is a framework to identify Web Service endpoints and ensure end-to-end endpoint identification in messages [Rel03]. It helps to define how Web Services should be invoked, routed, and replied to in a transport-neutral way. Rather than relying on HTTP to carry along the information necessary, it explicitly defines that information in the SOAP Header for Web Service calls [Eic04].

### Description:

WS-Addressing provides transport-neutral mechanisms to address Web Services and messages. Specifically, it defines XML elements to identify Web Services endpoints and to secure end-to-end endpoint identification in messages. This specification enables messaging systems to support message transmission through networks that include processing nodes such as endpoint managers, firewalls, and gateways in a transport-neutral manner.

WS-Addressing defines two interoperable constructs that convey information typically provided by transport protocols and messaging systems. These constructs normalize this underlying information into a uniform format that can be processed independently of transport or application. The two constructs are endpoint references and message information headers.

A Web Services endpoint is a referenceable entity, processor, or resource where Web Services messages can be targeted. Endpoint references convey the information needed to identify or reference a Web Services endpoint, and may be used in several different ways: they are suitable for conveying the information needed to access a Web Services endpoint, but are also used to provide addresses for individual messages sent to and from Web Services. To deal with this last usage case, this specification defines a family of message information headers that allows uniform addressing of messages independent of underlying transport. These message information headers convey end-to-end message characteristics including addressing for source and destination endpoints as well as message identity. Both of these constructs are designed to be extensible and reusable so that other specifications can build on and leverage endpoint references and message information headers [Box04].

### Good Reading:

<http://hyperthink.net/blog/PermaLink,guid,8519a1bf-e1b7-4b9b-b10c-fce8d359b83e.aspx>

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwsse/html/soapmail.asp>



**Usefulness:**

WS-Addressing allows SOAP messages to be used in environments where HTTP is not supported. It supports alternate protocols, which might provide improved performance for SOAP messages. It allows the return address to be different from the sending address, and can customize how faults are handled.

WS-Addressing can be used to support message transmission through intermediaries such as firewalls, gateways, and NAT routers. It supports fully asynchronous communication and enables responses that exceed the typical timeouts found in HTTP. Finally, WS-Addressing can allow a guard to use one network protocol on the high side and another on the low side.

**Issues and limitations:**

The effectiveness of WS-Addressing may depend upon which protocols are supported by a guard. SOAP has effectively standardized upon HTTP. It remains to be seen if adequate support will be available for effectively utilizing other transport solutions.

**C.15 XML Technologies for Further Investigation**

It may be beneficial to study the following XML technologies to evaluate their potential impact upon Web Service-capable cross-domain solutions.

- Web Services Composite Application Framework (WS-CAF)
- WS-Interoperability
- WS-ReliableMessaging
- WS-SecurityPolicy
- XLink
- XPointer
- XInclude
- XML Pipeline Definition Language
- XQuery
- XML Fragment Interchange
- Web Services Conversation Language (WSCL)
- Web Service Choreography Interface (WSCI)
- Business Process Execution Language (BPEL)
- RELAX NG

- Meta-Object Facility (MOF)
- XPipe

## C.16 List of References

- [And04] S. Anderson, “Web Services Secure Conversation Language (WS-Secure Conversation),”  
[http://ftpna2.bea.com/pub/downloads/WS\\_SecureConversation.pdf](http://ftpna2.bea.com/pub/downloads/WS_SecureConversation.pdf), May 04.
- [Atk02] B. Atkinson, “Web Services Security (WS-Security),” April 02,  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-security.asp> .
- [Aut02] Authenticating Web Services with SAML,  
[www.informit.com/articles/article.asp?p=28286&seqNum=4](http://www.informit.com/articles/article.asp?p=28286&seqNum=4), August 02.
- [Baj03] S. Bajaj, “Web Services Federation Language,” <http://www-106.ibm.com/developerworks/library/ws-fed/>, July 03.
- [Box04] D. Box, “New This Week,”  
<http://msdn.microsoft.com/WebServices/understanding/specs/default.aspx?pull=/library/en-us/dnglobspec/html/ws-addressing.asp>, August 04. [Xml04] *XML Trust Center*, <http://www.xmltrustcenter.org/xmlsig/faq.htm>, September 2004.
- [Bri03] A Brief Introduction to XACML, [http://www.oasis-open.org/committees/download.php/2713/Brief\\_Introduction\\_to\\_XACML.html](http://www.oasis-open.org/committees/download.php/2713/Brief_Introduction_to_XACML.html), March 03.
- [Coh04] F. Cohen, “Debunking SAML myths and misunderstandings,” September 04,  
<http://www-106.ibm.com/developerworks/xml/library/x-samlmyth.html?Open&ca=daw-se-news>.
- [Com04] Computer World,  
<http://www.computerworld.com/developmenttopics/development/story/0,10801,81295,00.html>, September 04.
- [Cp02] Cover Pages, <http://xml.coverpages.org/ni2002-12-10-a.html>, December 2002.
- [Eic04] Spec #1-WS-Addressing,  
<http://dotnetjunkies.com/WebLog/seichert/archive/2004/02/06/6740.aspx>, September 04.
- [Fed03] Federation of Identities in a Web Services World,  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-federation-strategy.asp>, July 03.

- [Fel04] D. Fells, "Dev Shed," <http://www.devshed.com/c/a/Security/Trust-Access-Control-and-Rights-for-Web-Services-Part-1/8/>, September 04.
- [Fer02] C. Ferris, "First Look at the WS-I Basic Profile 1.0," October 02, <http://www-106.ibm.com/developerworks/WebServices/library/ws-basicprof.html>.
- [Fre04] WS-I Frequently Asked Questions: Basic Profile 1.0, <http://www.ws-i.org/Profiles/Basic/2003-08/BasicProfile-1.0faq.pdf>, September 04.
- [Gal04] Galexia Consulting, [http://consult.galexia.com/public/research/assets/gc\\_distributed\\_identity\\_paper\\_part2\\_200311.pdf](http://consult.galexia.com/public/research/assets/gc_distributed_identity_paper_part2_200311.pdf), September 04.
- [Gar04] D. Gardner, "Interoperable Web Services Security Standard Passes Test," <http://nwc.securitypipeline.com/showArticle.jhtml?articleId=21100516&printableArticle=true>, May 04.
- [Gok02] S. Gokul, "XML Encryption," August 2002, [www.awprofessional.com/articles/article.asp?p=28801&seqNum=3](http://www.awprofessional.com/articles/article.asp?p=28801&seqNum=3).
- [Gud04] M. Gudgin, "Using WS-Trust and WS-SecureConversation," <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnWebsrv/html/ws-trustandsecureconv.asp>, May 04.
- [Hij03] A. Hijazi, "Analyzing Web Services Security against FEA TRM Enterprise Security Criteria," [http://www.giac.org/practical/GSEC/Abdulrahman\\_Hijazi\\_GSEC.pdf](http://www.giac.org/practical/GSEC/Abdulrahman_Hijazi_GSEC.pdf), December 03.
- [Hyp04] Hyperthink.net, <http://hyperthink.net/blog/CommentView.guid,FFA368D6-E728-46EC-AA94-C30751665512.aspx>, September 04.
- [Mad02] P. Madsen, C. Adams, "Privacy and XML, Part 2," May 01, 2002, [www.xml.com/pub/a/2002/05/01/privacy.html](http://www.xml.com/pub/a/2002/05/01/privacy.html).
- [Mad03] P. Madsen, "WS-Trust: Interoperable Security for Web Services," June 2003, [www.WebServices.xml.com/pub/a/ws/2003/06/24/ws-trust.html](http://www.WebServices.xml.com/pub/a/ws/2003/06/24/ws-trust.html).
- [Man04] XML Mania.com, [http://www.xmlmania.com/documents\\_article\\_202-XKMS-Messages-in-Detail.php](http://www.xmlmania.com/documents_article_202-XKMS-Messages-in-Detail.php), April 04.
- [Map02] R. Maple, "Exploring XML Encryption, Part 1," April 2002, [www.linuxsecurity.com/articles/cryptography\\_article-4729.html](http://www.linuxsecurity.com/articles/cryptography_article-4729.html).
- [Met04] Metadata Specifications Index Page, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/wsmetaspecindex.asp>, September 04.

- [Msd04] MSDN, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp>, September 04.
- [Oas04] Oasis Web Services Security (WS-Security 2004), <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>, March 04.
- [Pri04] WS-Privacy, <http://www.serviceoriented.org/ws-privacy.html>, September 04.
- [Rel03] Reliable Message Delivery in a Web Services World, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-rm-exec-summary.asp>, March 03.
- [Rou02] Introduction to SAML, [http://www.simc-inc.org/archive0002/February02/devwed1015\\_rouault.pdf](http://www.simc-inc.org/archive0002/February02/devwed1015_rouault.pdf), January 02.
- [Sal03] R. Salz, "Network World Fusion," September 03, <http://www.nwfusion.com/news/tech/2003/0908techupdate.html>.
- [Sam04] Identity-Based Security Solutions, [www.oblix.com/resources/security\\_standards/saml](http://www.oblix.com/resources/security_standards/saml), September 04.
- [Sec02] Security in a Web Services World: A Proposed Architecture and Roadmap, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>, April 02.
- [Sec04] Secure Web Services a sound business practice, [http://searchWebservices.techtarget.com/ateQuestionNResponse/0,289625,sid26\\_cid541207\\_tax294320,00.html](http://searchWebservices.techtarget.com/ateQuestionNResponse/0,289625,sid26_cid541207_tax294320,00.html), September 04.
- [Sma01] E. Simon, P. Madsen, and C. Adams, "An Introduction to XML Digital Signatures," August 2001, [www.xml.com/pub/a/2001/08/08xmldsig.html](http://www.xml.com/pub/a/2001/08/08xmldsig.html).
- [Sun04] Sun's XACML Implementation, <http://sunxacml.sourceforge.net/guide.html>, July 04.
- [Web04] Web Services Trust Language, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-trust.asp>, May 04.
- [Wsi02] WS-I Publishes Basic Work Profile Work Draft, <http://www.ws-i.org/docs/20021029wsipr.htm>, October 02.
- [Xac04] Sun's XACML Implementation Programmer's Guide for Version 1.2, <http://sunxacml.sourceforge.net/>, July 04.
- [Xkm04] About XKMS, <http://www.xmltrustcenter.org/xkms/index.htm>, September 04.

[Xml03] A. Skonnard, "The XML Files," August 03,  
<http://msdn.microsoft.com/msdnmag/issues/04/03/XMLFiles/default.aspx>.

## Appendix D

# SOAP Header Investigation

### D.1 What Are SOAP Headers?

Headers provide the ability to add new information to a SOAP message [One03]. For example, an application requiring authentication may add a header field for username and password or a service requiring transaction support may add a field for a transaction identification number. There are minimal rules governing what can go into a header field. This is intentional to allow great flexibility in extending a message. As Web Services become more sophisticated, it is likely that the headers will play an increasingly important role.

### D.2 How Is the DOD Using SOAP Headers in Web Services?

Overall, the prescribed uses of SOAP headers within the DOD seem to be limited. This is probably due to the small number of actual deployed Web Services. However, as SOAP standards mature and more organizations adopt SOAP, more guidance is likely to emerge. For now, it appears that security is the primary area of use for SOAP headers.

#### D.2.1 Network Centric Enterprise Services (NCES)

Booz Allen Hamilton has produced a Security and Service discovery Core Enterprise Services (CES) Software Developer's Kit (SDK). The SDK makes heavy use of SOAP headers. A detailed explanation of the required SOAP header elements along with the SOAP header processing rules can be found in the Security CES architecture. [Boo04]

#### D.2.2 Navy Enterprise Single-Sign On (NESSO)

NESSO is an open-source project by the Navy to achieve single-signon capabilities for Web applications and services across multiple domains. It makes heavy use of SAML and XACML and defines a custom namespace for SOAP headers. [Hut03]

### D.3 How Does SAML Use SOAP Headers?

The Security Assertion Markup Language (SAML) is an industry standard for expressing security assertions in XML [Oas04a]. It defines an XML format that allows services to exchange authentication and authorization information in a standard way. SAML consists of several small specifications. For the purposes of this paper, we are primarily concerned with the SAML protocol binding specification [Oas04b].

The protocol binding specification defines the use of SAML in communication protocols. The key requirements for SAML SOAP binding are:

SAML protocol elements *MUST* be enclosed in the SOAP *body*.

A SAML request *MAY* add arbitrary headers.

A SAML response *MUST NOT* require headers.

It is important to note that the binding specification does not define any required header information. Instead, it allows the flexibility to add header information as needed for custom protocol or vendor implementations. However, if the WS-Security standard is used to encode SAML assertions, users will find that it makes heavy use of SOAP headers.

WS-Security is a specification that defines where security information is stored in the SOAP message [Hal02]. Many security SDKs are using a combination of SAML and WS-Security to store and format security information. In this case, WS-Security specifies that information like SAML be enclosed within WS-Security tags in the SOAP header. The box shows an example of a SOAP header using SAML and WS-Security:

```
<S:Envelope xmlns:S="...">
<S:Header>
<wsse:Security xmlns:wsse="...">
<saml:Assertion
MajorVersion="1"
MinorVersion="0"
AssertionID="SecurityToken-ef375268"
Issuer="elliottwl"
IssueInstant="2002-07-23T11:32:05.6228146-07:00"
xmlns:saml="urn:oasis:names:tc:SAML:1.0:assertion">
...
</saml:Assertion>
...
</wsse:Security>
</S:Header>
<S:Body>
...
</S:Body>
</S:Envelope>
```

#### **D.4 Can SOAP Headers Be Used to Make a Guard State-aware?**

As Web Services become more sophisticated, a need will inevitably arise for a guard to be able to maintain a “conversation” with a client over a period of several requests. Following the Java Servlet model [Cow03], we present a lightweight approach for making the guard state-aware using SOAP headers.

In the Web application world, session tracking is used to enable stateful programming. A session is a sequence of requests from the same client over a set period of time. In order to enable session tracking in Web Services, we will need to define some basic requirements for the SOAP header and SOAP service:

A service wishing to be state-aware will add a header field with the following format:

```
<session Id:uuid:123456OFG43
```

```
Expires: 2004-12-14-01:23:24
```

```
Soap: mustUnderstand=1/>
```

**Id:** The session Id uniquely identifies the client for subsequent requests.

**Expires:** Identifies the maximum lifetime of the session. Additionally the server may set a time-out parameter that will void a session that's idle for a certain period of time.

**soap:mustUnderstand:** Tells the Web Service processor that it cannot ignore the header

1. When a client first contacts the guard it will be assigned a unique session id.
2. The client will be identified by the session ID for subsequent requests
3. If the client is inactive for a set period of time, the session ID will become invalid.

This should provide a basic mechanism for maintaining state regardless of the actual underlying protocol (SMTP, HTTP, etc...).

## **D.5 Other Potential Uses for SOAP Headers in the Guard**

- Use the WS-Routing protocol to route SOAP messages between clients and the guard. WS-Routing uses headers to store information.
- Each guard client could add additional guarding information to the header. Adding information such as the sender's name, organization, or machine IP could provide richer auditing and logging information.
- Headers could be used to store actual guard policies. Using a combination of digital signatures and encryption, policies could be included in the SOAP header, allowing the policy to travel with the message. This could be especially useful for highly mobile distributed guards.



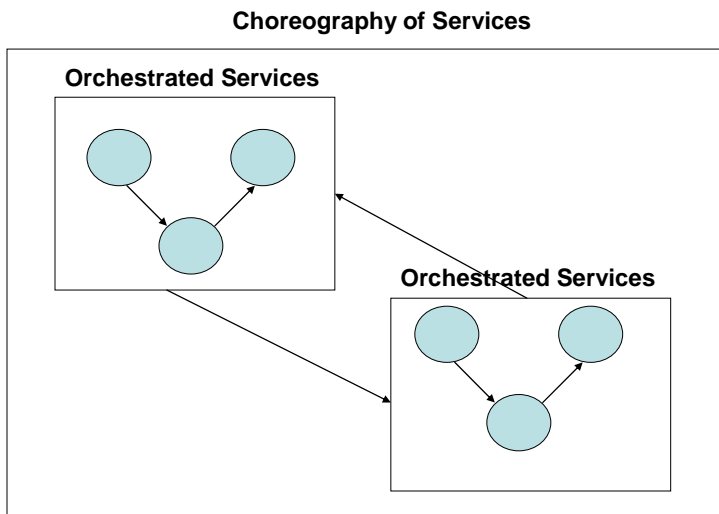
## D.6 List of References

- [Boo04] Booz Allen Hamilton, *Net-Centric Enterprise Services (NCES). Security and Service Discovery Core*, Enterprise Services (CES) pilot. *SDK User's Guide* Version 0.4, 20 May 2004.
- [Cow03] D. Coward and Y. Yoshida, *Java Servlet Specification*, version 2.4, 24 November 2003.
- [Hal02] P. Hallam-Baker et al, *WS-Security Profile for XML-based Tokens*. MSDN Library, August 28, 2002.
- [Hut03] A. Hutchinson, *Navy Enterprise Single Sign-On (NESSO) An Open-Source Project.*, NESSO Brief at TFW IA Mtg, 22 May 2003.
- [Oas04a] OASIS, *Technical Overview of the OASIS Security Assertion Markup Language*; DRAFT. Version 1.1, 4 May 2004.
- [Oas04b] OASIS, *Bindings for the OASIS SAML*. Version 2.0, SOAP Specification version 1.2., 30 May 2004 .
- [One03] M. O'Neill, *Web Services Security*, McGraw Hill, 2003

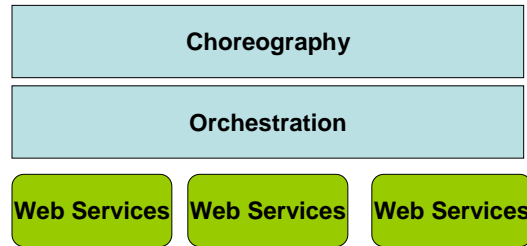
## Orchestration and Choreography Investigation

### E.1 Difference Between Orchestration and Choreography

Orchestration and Choreography standards share many similarities. They primarily differ in the scope of *who* manages the flow of messages and Web Service execution. With Orchestration, a single party defines the interaction between a set of services using an XML grammar described in the standards below [Pel03a]. Choreography is concerned with the collaboration of services across many parties [Pel03b]. It provides the capability to track a sequence of messages between services with no single party controlling the overall conversation (Figure E-1). It appears that Choreography builds upon Orchestration to provide a broader composition of heterogeneous Web Services (Figure E-2).



**Figure E-1. Choreography of Services**



**Figure E-2. Choreography and Orchestration**

## **E.2 Key Standards**

### **E.2.1 Orchestration**

Business Process Execution Language for Web Service (BPEL4WS) is a specification put forth by IBM, Microsoft, and BEA [BPE03]. It provides an XML grammar on top of WSDL that is used to model and coordinate the execution order of a set of Web Services assembled for a specific business process. BPEL4WS has a large following among key vendors and is implemented to some degree in products such as BEA Weblogic, BizTalk, and Collaxa.

Business Process Modeling Language (BPML) is another Orchestration language developed by Intalio, Sun, and others. It can be loosely compared to BPEL4WS but uses its own XML grammar and adds features such as persistence, roles, and recursive composition. BPML also includes WSCI (discussed below) to manage choreography activities.

### **E.2.2 Choreography**

Web Service Choreography Interface (WSCI) is a specification from Sun, SAP, BEA, and Intalio [W3C02]. It is an XML grammar used to describe a high-level view of message exchange between collaborating Web Services. WSCI does not define a specific executable business process as does BPEL4WS, but instead is concerned with the overall management of a set of Orchestrated services among many parties. With WSCI no single party controls the process. Each member of the collaborating group would maintain a WSCI document that describes his or her role in the overall process. WSCI leverages WSDL to describe how the services can be used to accomplish the business logic.

WS-Choreography Definition Language (WS-CDL) is a relatively new (April 2004) specification from the W3C [W3C04]. It appears to supersede WSCI and provides functionality similar to that of WSCI.

Both WSCI and WS-CDL complement the Orchestration languages. Each provides a higher level of abstraction about who controls a set of collaborating services.

### **E.3 Guarding Orchestrated and Choreographed Services**

Current guard research focuses solely on the protection of information flowing through a single Web Service. As the demand for managing an aggregating a set of services grows, so will the level of protection and abstraction needed to secure information. It seems a guard would need to move up the Web Service stack to maintain a global view of the overall process along with how the interaction between services could affect the protection of information. From a technical standpoint, it seems the guard could be more effective sitting on top of the Choreography and/or Orchestration engine to adequately manage security policies.

### **E.4 List of References**

- [BPE03] *BPEL4WS for Web Services version 1.1*. IBM, DeveloperWorks, <http://www-106.ibm.com/developerworks/library/ws-bpel/>, 05 May 2003.
- [Pel03a] C. Pelz, *Web Service Orchestration: A review of emerging technologies, standards and tools*. Hewlett-Packard Co, [http://devresource.hp.com/drc/technical\\_white\\_papers/WSOrch/WSOrchestratio.pdf](http://devresource.hp.com/drc/technical_white_papers/WSOrch/WSOrchestratio.pdf), January 2003.
- [Pel03b] C. Pelz, *Web Service Orchestration and Choreography. A look at WSCI and BPEL4WS*. Web Services Journal, Vol 03 Issue 7, <http://www.sys-con.com/Webservices/article.cfm?id=592>
- [W3C02] W3C. *Web Service Choreography Interface 1.0. W3C Note*, <http://www.w3.org/TR/wsci/>, 8 August 2002.
- [W3C04] *Web Service Choreography Description Language 1.0. WC3 Working Draft 27*, <http://www.w3.org/TR/2004/WD-ws-cdl-10-20040427/>, April 2004.

## Appendix F

# Negotiating Cross-Domain Web Service Information Exchanges

There are two possible situations where a guarding service might be implemented to enforce policies related to a cross domain Web Service. In the first situation, a Web Service provider wants to utilize the guarding service to extend access to his Web Service to clients in other domains. In this case, the guarding service will be enforcing the security policies of the Web Service provider. To cover this situation, the guarding service provider would need to ask the Web Service provider to answer the set of questions in Section F.1 below.

In the second situation, a security manager representing consumers wants to utilize the guarding service to connect his consumers to Web Services in other domains. In this case, the guarding service will be enforcing the security policies of the consumer environment. To cover this second situation, the guarding service provider would need to ask the manager for the consumer environment to answer the set of questions in Section F.2 below.

## F.1 Questions for the Web Service Provider

1. What security domains are involved in the exchange? What security domain does the Web Service reside in? What security domain(s) do the clients reside in?
2. Must web clients be identified and authenticated before using the Web Service? If so, what authentication method is required?
3. Is there any prioritization scheme that must be supported for different clients? If so, what is that scheme?
4. What information exchanges are involved in the Web Service?
  - a. Current policy dictates that automatic content review can only be accomplished for “highly structured ASCII.” Can the Web Service request(s) from clients be structured as “highly structured ASCII? If so, describe the expected structure of the request. Can the schema be sufficiently constrained to permit machine validation? If not, then a real-time cross domain Web Service cannot be implemented.
  - b. Can the Web Service response(s) be structured as “highly structured ASCII? If so, describe the expected structure of the response. If not, then describe how the guarding solution can determine that a reliable human review of the responses has been accomplished. (For example, a digital signature might be attached to the response, indicating the identity of the human reviewer; the

guarding service would simply be asked to ensure that the signature belongs to an authorized releaser.)

- c. Can the service return more than one document type? If so, can every document type be adequately described for machine validation of its content?
  - d. Will the Web Service involve a series of transactions? What is the sequence of expected transactions? Will the releasability of later messages be dependent upon information contained in earlier messages (i.e., must the guard be “state aware”?)
5. What transport protocol will be used? Can both the service provider and the client support this protocol?
  6. Are there constraints on what information can be imported within the requests from each client domain? If so, what are those constraints? Can the import rules be articulated such that a machine could interpret those rules?
  7. What metadata needs to be supported by either the service provider or the client?
  8. Is there a concern about importing malicious content from Web Service requests? If so, what mechanisms are considered to be adequate for identifying and quarantining that content? Must that protection be included within the guarding service? Or will it be taken care of by the Web Service?
  9. Is there a concern about experiencing a denial of service from excessive requests? If so, what mechanisms are considered to be adequate to protect the Web Service environment from a denial of service attack? Must that protection be included within the guarding service? Or will it be taken care of by the Web Service?
  10. Are there constraints on what information can be exported to each client domain by the Web Service? If so, what are those constraints? Can the export rules be articulated such that a machine could interpret and enforce those rules?
  11. Must Web Service activities be audited? If so, what information must be captured about those activities? Must clients be uniquely identified in that audit log?
  12. How should errors be reported? To whom? How timely must error reporting be?

## **F.2 Questions for the Security Managers of the Consumer Environment**

1. What security domains are involved in the exchange? What security domain does the Web Service reside in? What security domain(s) do the clients reside in?

2. Must web clients be identified and authenticated before using the Web Service? If so, what authentication method is required?
3. Are there any constraints on which clients can use the guarding service? If so, what are those constraints?
4. Is there any prioritization scheme that must be supported for different clients? If so, what is that scheme?
5. Must servers hosting the Web Service be authenticated to the clients before the clients use that service? If so, what authentication method is required?
6. Are there any requirements to hide or disguise the identity of the clients from the Web Services domain? If so, are there still requirements to capture information within the client domain about the client's identity (for example, for audit purposes)?
7. What information exchanges are involved in the Web Service?
  - a. Can the Web Service request(s) from clients be structured as "highly structured ASCII? (If the request is not structured enough for automatic review of the Web Service request by the guard, then a real-time cross domain Web Service cannot be implemented.) Does an XML Schema exist that describes those Web Service requests?
  - b. Can the Web Service response(s) be structured as "highly structured ASCII? Does an XML schema exist that describes the Web Service response? (If the response is not structured enough for automatic review of the Web Service response by the guard, then human review of the content of the responses must be accomplished before responses can be delivered to the clients.)
8. Are there constraints on what information can be exported within the Web Service requests? If so, what are those constraints? Can the release rules be articulated such that a machine could interpret and enforce those rules?
9. Are there constraints on what information can be imported to the client domain from the Web Service responses? If so, what are those constraints? Can the import rules be articulated such that a machine could interpret and enforce those rules?
10. What metadata needs to be supported by either the service provider or the client?
11. What transport protocol will be used? Can both the service provider and the client support this protocol?
12. Are there concerns about importing malicious content from Web Service responses? If so, what mechanisms are adequate for identifying and quarantining that content? Must that protection be included within the guarding service? Or will it be taken care of by the other components within the client environment?

13. Are there concerns about experiencing a denial of service attack from excessive responses? If so, what mechanisms are adequate to protect the client environment from a denial of service attack?
14. Must clients be held accountable for their actions? If so, what information must be captured about client activity.
15. Must client activities be audited? If so, what information must be captured about the activities? Must clients be uniquely identified in that audit log? What information must be captured about the Web Service?
16. How should errors be reported? To whom? How timely must error reporting be?



## Appendix G

# Implementation of Airspace Release Rules

## G.1 Pruning Queries

Our Semantic Web guard prototype used queries within the pruning component to query an input ontology and extract the set of information that is allowed to pass through the security guard. Any information that is not extracted using these queries is not released through the security guard and is in effect pruned. For example, looking at the queries below, notice how SpecialOperationsForces (SOFs) are not selected to pass through the guard. This is the default deny approach discussed in Section 4.

The Pruning Queries were implemented using RDQL, which is an RDF query language available as part of the Jena toolkit. Below we have included a short English description of each query and the actual query in RDQL syntax, used by our SemanticPruningFilter. A more complete description of RDQL can be found on the Jena Website at <http://jena.sourceforge.net/tutorial/RDQL/index.html>.

### 1. All AirTrafficControlAirspaces are allowed to go through

```
SELECT ?class, ?airspace, ?name, ?desc, ?shape, ?shapename, ?timeInst, ?start, ?stop
WHERE
( ?class,
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
  <http://www.w3.org/2002/07/owl#Class> ),
( ?class, rdfs:subClassOf, <http://xml.dod.mil/ontology#AirTrafficControlAirspace> ),
( ?airspace, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?class ),
( ?airspace, <http://xml.dod.mil/ontology#name>, ?name ),
( ?airspace, <http://xml.dod.mil/ontology#description>, ?desc ),
( ?airspace, <http://xml.dod.mil/ontology#hasShape>, ?shape ),
( ?shape, <http://xml.dod.mil/ontology#name>, ?shapename ),
( ?airspace, <http://xml.dod.mil/ontology#hasTimePeriod>, ?timeInst ),
( ?timeInst, <http://xml.dod.mil/ontology#startDateTime>, ?start ),
( ?timeInst, <http://xml.dod.mil/ontology#stopDateTime>, ?stop )
```

## 2. All SpecialUseAirspaces are allowed to go through

SELECT ?class, ?airspace, ?name, ?desc, ?shape, ?shapename, ?timeInst, ?start, ?stop

WHERE

( ?class,  
    <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>,  
    <<http://www.w3.org/2002/07/owl#Class>> ),  
( ?class, rdfs:subClassOf, <<http://xml.dod.mil/ontology#SpecialUseAirspace>> ),  
( ?airspace, <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>, ?class ),  
( ?airspace, <<http://xml.dod.mil/ontology#name>>, ?name ),  
( ?airspace, <<http://xml.dod.mil/ontology#description>>, ?desc ),  
( ?airspace, <<http://xml.dod.mil/ontology#hasShape>>, ?shape ),  
( ?shape, <<http://xml.dod.mil/ontology#name>>, ?shapename ),  
( ?airspace, <<http://xml.dod.mil/ontology#hasTimePeriod>>, ?timeInst ),  
( ?timeInst, <<http://xml.dod.mil/ontology#startDateTime>>, ?start ),  
( ?timeInst, <<http://xml.dod.mil/ontology#stopDateTime>>, ?stop )

## 3. All RestrictedOperationsZone airspaces that do not contain “laser” in the name field and are not SpecialOperationsForces airspaces or UnmannedAerialVehicle airspaces are allowed to go through

SELECT ?class, ?airspace, ?name, ?desc, ?shape, ?shapename, ?timeInst, ?start, ?stop

WHERE

( ?class,  
    <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>,  
    <<http://www.w3.org/2002/07/owl#Class>> ),  
( ?class, rdfs:subClassOf, <<http://xml.dod.mil/ontology#RestrictedOperationsZone>> ),  
( ?airspace, <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>, ?class ),  
( ?airspace, <<http://xml.dod.mil/ontology#name>>, ?name ),  
( ?airspace, <<http://xml.dod.mil/ontology#description>>, ?desc ),  
( ?airspace, <<http://xml.dod.mil/ontology#hasShape>>, ?shape ),  
( ?shape, <<http://xml.dod.mil/ontology#name>>, ?shapename ),

( ?airpace, <<http://xml.dod.mil/ontology#hasTimePeriod>>, ?timeInst ),

( ?timeInst, <<http://xml.dod.mil/ontology#startDateTime>>, ?start ),

( ?timeInst, <<http://xml.dod.mil/ontology#stopDateTime>>, ?stop )

AND !

( ?class eq <<http://xml.dod.mil/ontology#SpecialOperationsForces>> ||

  ?class eq <<http://xml.dod.mil/ontology#UnmannedAerialVehicle>> ||

  ?name =~ /laser/i )

**4. All RestrictedOperationsZone airspaces that contain “laser” in the name field have active time period removed**

SELECT ?class, ?airpace, ?desc, ?shape, ?shapename

WHERE

( ?class,

  <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>> ,

  <<http://www.w3.org/2002/07/owl#Class>> ),

( ?class, rdfs:subClassOf, <<http://xml.dod.mil/ontology#RestrictedOperationsZone>> ),

( ?airpace, <<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>>, ?class ),

( ?airpace, <<http://xml.dod.mil/ontology#name>>, ?name ),

( ?airpace, <<http://xml.dod.mil/ontology#description>>, ?desc ),

( ?airpace, <<http://xml.dod.mil/ontology#hasShape>>, ?shape ),

( ?shape, <<http://xml.dod.mil/ontology#name>>, ?shapename )

AND !

( ?class eq <<http://xml.dod.mil/ontology#SpecialOperationsForces>> ||

  ?class eq <<http://xml.dod.mil/ontology#UnmannedAerialVehicle>> ||

  ?name !~ /laser/i )

**5. All UAV airspaces that do not contain “laser” in the name field may be released without the name and description**

SELECT ?class, ?airpace, ?name, ?shape, ?shapename, ?timeInst, ?start, ?stop

WHERE

( ?class,

```

    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
    <http://www.w3.org/2002/07/owl#Class> ),
  ( ?class, rdfs:subClassOf, <http://xml.dod.mil/ontology#UnmannedAerialVehicle> ),
  ( ?airspace, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?class ),
  ( ?airspace, <http://xml.dod.mil/ontology#name>, ?name ),
  ( ?airspace, <http://xml.dod.mil/ontology#hasShape>, ?shape ),
  ( ?shape, <http://xml.dod.mil/ontology#name>, ?shapename ),
  ( ?airspace, <http://xml.dod.mil/ontology#hasTimePeriod>, ?timeInst ),
  ( ?timeInst, <http://xml.dod.mil/ontology#startDateTime>, ?start ),
  ( ?timeInst, <http://xml.dod.mil/ontology#stopDateTime>, ?stop )
AND ! ( ?name =~ /laser/i )

```

**6. All UAV airspaces that contain “laser” in the name field may be released without the name, description and active time period**

```
SELECT ?class, ?airspace, ?name, ?shape, ?shapename
```

```
WHERE
```

```

  ( ?class,
    <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>,
    <http://www.w3.org/2002/07/owl#Class> ),
  ( ?class, rdfs:subClassOf, <http://xml.dod.mil/ontology#UnmannedAerialVehicle> ),
  ( ?airspace, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, ?class ),
  ( ?airspace, <http://xml.dod.mil/ontology#name>, ?name ),
  ( ?airspace, <http://xml.dod.mil/ontology#hasShape>, ?shape ),
  ( ?shape, <http://xml.dod.mil/ontology#name>, ?shapename )
AND ! ( ?name !~ /laser/i )

```

## G.2 Transformation Rules

Our Semantic Web guard prototype used a rule reasoner within the transformation component to modify information before it was allowed to pass through the security guard. We used rules available in Jena2 because RDQL was not adequate to modify the content. For example, a transformation rule required “UnmannedAerialVehicles” to become “UAV,” but the query language could not provide this capability.

The transformation rules were implemented using a general purpose rule-based reasoner available as part of the Jena toolkit. Below we have included a short English description of each rule and the actual rule in Jena's rule syntax, used by our SemanticTransformationFilter. A more complete description of the general-purpose rule-based reasoner can be found on the Jena Website at <http://jena.sourceforge.net/inference/index.html#rules>.

**1. Remove name and description fields from SpecialOperationsForces airspaces**

[sofrule1a:

```
(?A rdf:type ontology:SpecialOperationsForces) (?A ontology:name ?B)
(?A ontology:description ?C) -> remove(1) remove(2) ]
```

**2. Add empty name and description fields to SpecialOperationsForces**

[sofrule1b:

```
(?A rdf:type ontology:SpecialOperationsForces)
> (?A ontology:name ' ') (?A ontology:description ' ') ]
```

**3. Add NoFireArea**

[sofrule1c:

```
(?A rdf:type ontology:SpecialOperationsForces)
> (?A rdf:type ontology:NoFireArea) ]
```

**4. Remove SpecialOperationsForces**

[sofrule1d:

```
(?A rdf:type ontology:SpecialOperationsForces) -> remove(0) ]
```

**5. Remove name and description fields from UnmannedAerialVehicles**

[sofrule2a:

```
(?A rdf:type ontology:UnmannedAerialVehicle) (?A ontology:name ?B)
(?A ontology:description ?C) -> remove(1) remove(2) ]
```

**6. Add name field with 'UAV' value and description field with empty value**

[sofrule2b:

(?A rdf:type ontology:UnmannedAerialVehicle)

> (?A ontology:name 'UAV') (?A ontology:description ' ')]

## Appendix H

# Test Cases and Results

In reviewing the following sections, readers should assume that all test cases, except the ones noted as having incomplete data, have complete instance data; that is: instances are fully populated to include name, description, shape, and time period. This is due to a limitation on our implementation using RDQL. They should also take into account that Jena reorganizes the ontology, but this has no effect on the output. Finally, one common problem with transformation to pruning pipeline consisted of SOF instances being “Restricted Operation Zones” (ROZ) and “TimePeriod” not being removed because SOF is changed to “No Fire Area” before the removal phase.

## H.1 Pruning Supported

Test cases where pruning was supported were run through the GSIX infrastructure, using a pipeline content-enforcer. Only the SemanticPruningFilter was enabled for this phase of testing.

#	Name	Description	Expected Results	Actual Results
1	Allowed Airspaces	Instances of all allowed airspaces and their properties – all ATC and SUA airspaces.	All airspace instance data gets released.	Actual results matched expected results
2	Allowed Airspaces and SOF	Same as test case #1 but with SOF airspaces.	All airspaces instance data gets released except SOF.	Actual results matched expected results
3	ROZ Airspaces without “laser” in the name field	Set of ROZ airspaces to include Drop Zone, AEW, SOF, and UAV. None with “laser” in the name field.	DZ and AEW airspace instance data gets released. SOF airspace data gets pruned. UAV airspace has name and description fields pruned.	Actual results matched expected results

#	Name	Description	Expected Results	Actual Results
4	ROZ Airspaces with "laser" in the name field	Set of ROZ airspaces to include DZ, AEW, SOF and UAV. Each category has at least one airspace with "laser" in the name field and without "laser" in the name field. Different variations on the word "laser" will be used to include "laser," "Laser," and "LASER"	DZ and AEW airspace instance data where "laser" does not appear in the name field gets released.  DZ and AEW airspace instance data with "laser" in the name field have the time period pruned.  SOF airspace data gets pruned.  UAV airspace with "laser" in the name field has only airspace and shape released.  UAV airspace without "laser" in the name field has airspace, shape, and time period released.	Actual results matched expected results
5	Allowed Airspaces with added airspace types	Add classes of airspaces that do not exist in the guard ontology	All allowed airspaces but "new" airspaces get pruned.	Actual results matched expected results
6	Allowed Airspaces with incomplete data	Add classes of airspaces that are allowed classes but with incomplete data.	Only allowed airspaces with complete data will be selected. Airspaces with incomplete data will be pruned (unfortunately).	Actual results matched expected results
7	All Airspace combinations	All potential instances of each class and even some added instances included for good measure.	Output should conform to the RDQL rules for pruning.	Actual results matched expected results



#	Name	Description	Expected Results	Actual Results
8	No allowed airspaces	Set of airspaces where none are allowed to be released. Include SOF airspaces as well as orphans (airspaces of types not defined in the guard ontology)	Empty set.	Actual results matched expected results
9	Only instance data (no inferred class data)	Only instance data was included, and there were several kinds including added instances.	Guard will work fine without class data included in file.	Actual results matched expected results
10	Entire class ontology (no import statement)	Entire class ontology (no import statement)	The guard should remove all class data and operate successfully on the instance data.	Actual results matched expected results

## H.2 Transformations Supported

Test cases where transformations were supported were run through the GSIX infrastructure, using a pipeline content-enforcer. The pipeline consisted of the SemanticTransformationFilter first with the results then fed into the SemanticPruningFilter.

#	Name	Description	Expected Results	Actual Results
11	Allowed Airspaces	Instances of all allowed airspaces and their properties – all ATC and SUA airspaces.	All airspace instance data gets released.	Actual results matched expected results
12	Allowed Airspaces and SOF	Same as test case #11 but with SOF airspaces.	All airspaces instance data, except SOF, gets released.  SOF airspaces get transformed to No Fire Areas but name and description fields are blanked out.	Actual results matched expected results

#	Name	Description	Expected Results	Actual Results
13	Allowed Airspaces with UAV	Set of allowed airspaces with UAV.	Allowed airspaces get released.  UAV airspaces have name changed to "UAV" and description field gets pruned.	Actual results matched expected results
14	ROZ Airspaces without "laser" in the name field	Set of ROZ airspaces to include Drop Zone, AEW, SOF, and UAV.  None with "laser" in the name field	DZ and AEW airspace instance data gets released.  SOF airspaces get transformed to No Fire Areas but name and description fields are blanked out.  UAV airspaces have name changed to "UAV" and description fields pruned.	Actual results matched expected results
15	ROZ Airspaces with "laser" in the name field	Set of ROZ airspaces to include DZ, AEW, SOF and UAV.  Each category has at least one airspace with "laser" in the name field and without "laser" in the name field.  Different variations on the word "laser" will be used to include "laser," "Laser," and "LASER."	DZ and AEW airspace instance data where "laser" does not appear in the name field gets released.  DZ and AEW airspace instance data with "laser" in the name field have the time period pruned.  SOF airspaces get transformed to No Fire Areas but name and description fields are blanked out and the time period gets pruned.  UAV airspaces have name changed to "UAV" and description fields pruned.	Actual results matched expected results

#	Name	Description	Expected Results	Actual Results
16	Allowed Airspaces with added airspace types	Add classes of airspaces that do not exist in the guard ontology.	All allowed airspaces but “new” airspaces get pruned.	Actual results matched expected results
17	Allowed Airspaces with incomplete data	Add classes of airspaces that are allowed classes but with incomplete data.	Only allowed airspaces with complete data will be selected.  Airspaces with incomplete data will be pruned (unfortunately).	Actual results matched expected results
18	No allowed airspaces	Set of airspaces where none are allowed to be released.  Include SOF airspaces as well as orphans (airspaces of types not defined in the guard ontology).	Empty set	<b>Potential Problem:</b> SOF Airspaces should not have been included here, because SOFs get changed to NoFireAreas and are not deleted.
19	All Airspace combinations	All potential instances of each class and even some added instances included for good measure.	Output should conform to the inference rules for transformation and the RDQL rules for pruning.	<b>Potential Problem:</b> An allowed UAV references a time period that was embedded in a removed airspace. This results in a dangling reference.
20	Only instance data (no inferred class data)	Only instance data was included, and there were several kinds including added instances.	Guard will work fine without class data included in file.	Actual results matched expected results

#	Name	Description	Expected Results	Actual Results
21	Entire class ontology (no import statement)	The entire class ontology is embedded with the instance data.	Guard will work fine without class data included in file.	Actual results matched expected results

## Acronyms

ACO	Airspace Control Order
ACTD	Advanced Concept Technical Demonstration
AEW	Airborne Early Warning
API	Application Program Interface
API RDQL	Application Program Interface RDF Query Language
ATC	Air Traffic Control
BEA	BEA Systems Inc. (Company Name)
BP	Basic Profile
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Service
BPML	Business Process Modeling Language
C2	Command and Control
CA	Certificate Authority
CAFAirspace	Combat Air Force Airspace
CBM	Callback Manager
CDS	Cross-Domain Solutions
CE	Content Enforcer
CES	Core Enterprise Services
C-IA COP	Coalition Information Assurance Common Operational Picture
COTS	Commercial off the Shelf
CPSG/NIS	Cryptologic Systems Group/Network Services Division
CRL	Certificate Revocation List
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Project Agency
DAO	Data Access Object
DISA	Defense Information Systems Agency
DNS	Domain Name Service
DOD	Department of Defense
DQL	DAML Query Language
D-SIDE	Defense Strategic Integrated Decision Environment
DTD	Document Type Definition
DZ	Drop Zone
FTP	File Transport Protocol
FY	Fiscal Year
GM	Guard Message

GSIX	Guarded Sharing of Information with XML
HTML	Hyper Text Markup Language
HTTP	Hypertext Transport Protocol
IBM	International Business Machines
IC MAP	Intelligence Community Multi-Intelligence Acquisition Program
IC-CIO	Intelligence Community Chief Information Officer
IC-MSP	Intelligence Community Metadata Standard for Publication
IETF	Internet Engineering Task Force
IM	Inbound Manager
IP	Internet Protocol
ISR	Intelligence Surveillance & Reconnaissance
ISSE	Information Support Server Environment
IT	Information Technology
JAC	Joint Analysis Center
JICPAC	Joint Intelligence Center, Pacific
JWID	Joint Warfighting Interoperability Demonstration
MAJIIC	Multi-sensor Aerospace-ground Joint ISR Interoperability Coalition
MDA	Missile Defense Agency
MIME	Multipurpose Internet Mail Extensions
MLS	Multi-level Secure
MOF	Meta-Object Facility
MOIE	Mission Oriented Investigation and Experimentation
MSL	Multiple Security Layers
NAT	Network Address Translation
NCES	Net-centric Enterprise Services
NESSO	Navy Enterprise Single-Sign On
NFA	No Fire Area
NSA	National Security Agency
OASIS	Organization for the Advancement of Structured Information Standards
OM	Outbound Manager
OMS	Ontology Management System
ONI	Office of Naval Intelligence
OWL	Web Ontology Language
OWL DL	Web Ontology Language Description Language
P3P	Platform for Privacy Preferences
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PGP	Pretty Good Privacy
PKI	Public Key Infrastructure
PKIX	Public Key Infrastructure X.509
QoS	Quality of Service

RDF	Resource Description Framework
RDF-S	Resource Description Framework Schema
RDQL	RDF Query Language
RFC	Request for Comments
ROZ	Restricted Operation Zone
RPC	Remote Procedure Call
RSA	RSA Security, Inc. (Company Name)
SAML	Security Assertion Markup Language
SAP	A company headquartered in Waldorf, Germany
SAPM	Synchronized Air Power Management
SCI	Sensitive Compartmented Information
SDK	Software Developer's Kit
SMTP	Simple Mail Transport Protocol
SNOBASE	Semantic Network Ontology Base
SOAP	Formerly Simple Object Access Protocol; with Version 2.0 of the specification, this is no longer an acronym
SOF	Special Operations Forces
SPAWAR	Navy Space and Warfare Center
SPKI	Simple Public Key Infrastructure
SSL	Secure Sockets Layer
STS	Security Token Service
SUA	Special Use Airspaces
TLS	Transport Layer Security
UAV	Unmanned Aerial Vehicle
UDDI	Universal Description, Discovery and Integration
URI	Uniform Resource Indicator, Uniform Resource Identifier
URL	Uniform Resource Locator
US	United States
VTCs	Video Teleconferences
W3C	World Wide Web Consortium
WS	Web Services
WS-CAF	Web Services Composite Application Framework
WS-CDL	WS-Choreography Definition Language
WSCl	Web Service Choreography Interface
WSCL	Web Services Conversation Language
WSDL	Web Services Description Language
WS-I	Web Services Interoperability
WWW	World Wide Web
XACML	eXtensible Access Control Markup Language
XBIS	Cross Boundary Information Sharing
XIGOR	Extended Intelligence Guard for ONI Replication

XKISS	XML Key Information Service Specification
XKMS	XML Key Management Specification
XKRSS	XML Key Registration Service Specification
XML	eXtensible Markup Language
XSLT	Extensible Stylesheet Language Transformations



# Distribution List

## Internal (Hardcopy)

### R204

Records Resources

## Internal (Softcopy)

### D010

E. Palo

### D500

T. Gannon

### G010

R. Games

### G020

D. Amos

C. Bonneau

E. Caddick

L. Chock

J. Firey

J. Guttman

J. Heaney

E. Mac Garrigle

M. Michaud

B.W. McKenney

H.W. Neugent

R. Simmons

P.S. Tasker

R. Westman

### G050

L. Costa

D.L. Martell

W.P. Nenninger

S.T. Norman

### G057

A. McClure

### W010

S. Huffman

### Project

D. Bryson

J. Garriss

S. Gosnell

B. Heaton

G. Huber

D. Jacobs

M. Pulvermacher

N. Reed

S. Semy

J. Standard

**External**

**ELECTRONIC SYSTEMS CENTER  
(ESC)**

Stephen Morrissey

Bert Hopkins

**AIR FORCE ROME LABS**

Doug Poore

Tim Farrell

Mark Gorniak

Matt Kochan

Mark Linderman

**DISA**

LCDR Tim Olson

**NSA**

Dept of Defense

9800 Savage Rd.

Attn: I124/Suite 6730

Ft. Meade, MD 20755-6730

Jeff Duerr

**SPAWAR**

Office of the Chief Engineer

Combatant Command Interoperability

Program Office

Marlen Lenk Zigler