14

Challenges of Testing Web Services and Security in SOA Implementations

Abbie Barbir[1], Chris Hobbs[1], Elisa Bertino[2], Frederick Hirsch[3] and Lorenzo Martino[4]

[1] Nortel, 3500 Carling Avenue, Ottawa, Canada {abbieb,cwlh}@nortel.com
[2] Department of Computer Science and CERIAS, Purdue University, West Lafayette, Indiana, USA bertino@cerias.purdue.edu
[3] Nokia, 5 Wayside Rd, Burlington, Mass., USA frederick.hirsch@nokia.com
[4] Department of Computer Technology and Cyber Center, Purdue University, West Lafayette, Indiana, USA lmartino@purdue.edu

Abstract. The World Wide Web is evolving into a medium providing a wide array of e-commerce, business-to-business, business-to-consumer, and other information-based services. In Service Oriented Architecture (SOA) technology, Web Services are emerging as the enabling technology that bridges decoupled systems across various platforms, programming languages, and applications.

The benefits of Web Services and SOA come at the expense of introducing new level of complexity to the environments where these services are deployed. This complexity is compounded by the freedom to compose Web Services to address requirements such as quality of service (QoS), availability, security, reliability, and cost. The complexity of composing services compounds the task of securing, testing, and managing the quality of the deployed services.

This chapter identifies the main security requirements for Web Services and describes how such security requirements are addressed by standards for Web Services security recently developed or under development by various standardizations bodies. Standards are reviewed according to a conceptual framework that groups them by the main functionalities they provide.

Testing composite services in SOA environment is a discipline at an early stage of study. The chapter provides a brief overview of testing challenges that face early implementers of composite services in SOA taking into consideration Web Services security. The importance of Web Services Management systems in Web Services deployment is discussed. A step toward a fault model for Web Services is provided. The chapter investigates the use of crash-only software development techniques for enhancing the availability of Web Services. The chapter discusses security mechanisms from the point of view of interoperability of deployed services. The work discusses the concepts and strategies as developed by the WS-I Basic Security profile for enhancing the interoperability of secure Web Services.

14.1 Introduction

The use of Web Services in IT is becoming more relevant as the key technology for enabling the foundation of a loosely coupled, language-neutral, platform-independent way to link applications across multiple organizations. Despite the heterogeneity of the underlying platforms, Web Services enhance interoperability and are thus able to support business applications composed of chains of Web Services. Interoperability among heterogeneous systems is a key promise of Web Service technology and therefore notions such as Web Service composition and technologies like workflow systems are being investigated and developed. Interoperability and security play an important role in positioning Web Services as the industry choice for realizing Service Oriented Architectures (SOAs) [27].

The use of Web Services in loosely coupled service environments enables enterprises quickly to adapt to changing business demands. However, the benefits of Web Services and SOA come at the expense of introducing new complexity to the environments where these services are deployed (see, for instance, [31] and [47]). In this new programming paradigm, services are used as part of a business process. The solution implements a workflow composed of many services that are combined to achieve the required business objective. The complexity is compounded with the ability to compose services whereby services can be interchangeable based on various factors such as QoS, availability, security, reliability, and cost. This complexity compounds the task of securing, testing, and managing [35] the quality of the deployed services.

In the SOA environment, testing Web Services requires an end-to-end approach including service integration points and the connected systems themselves. The set of connected systems includes services, web applications, security gateways, legacy systems, and back-end systems. Web Service(s) consumers and providers should be able to measure the service levels of the invoked services [35].

Service Level Agreement (SLA) is a term widely used in the industry to express contracts between service consumers and service providers [35]. A Service Level Agreement defines the quality of service, how quality is measured, and what happens if the service quality is not met. In today's environment, SLAs are implicit and in most cases negotiated in advance. However, the same concept can be extended and used as means to express the performance of a Web Service throughout the whole integration chain. Consumers of Web Services can generate tests against the exposed services and compare the results with the SLAs. Service providers can publish test suits for their services that can be used by the consumers to test the compliance to the published SLA. However, service management and diagnosis require the knowledge and view of the end-to-end service. This constraint entails the sharing of management across the administrative domain in order to provide an end-to-end view [14, 35].

Web Service must protect its own resources against unauthorized access. This in turn requires suitable means for identification, whereby the recipient of a message must be able to identify the sender; authentication, whereby the recipient of a message needs to verify the claimed identity of the sender; authorization, whereby the recipient of a message applies access control policies to determine whether the sender has the right to use the required Web Services and the protected resources.

In a Web Service environment it is, however, not enough to protect the service providers, it is also important to protect the parties requiring services. Because a key component of the Web Service architectures is represented by the discovery of services, it is crucial to ensure that all information used by parties to this purpose be authentic and correct. Also, we need approaches by which a service provider can prove its identity to the party requiring the service in order to avoid attacks, such as phishing attacks. Within this context, the goal of securing Web Services can be decomposed into three broad subsidiary goals:

1. Providing mechanisms and tools for securing the integrity and confidentiality of messages as well as the guarantee of message delivery.
2. Ensuring that the service acts only on message requests that comply with the policies associated with the services.
3. Ensuring that all information required by a party in order to discover and use services is correct and authentic.

The overall goal of Web Services security standards is to make interoperable different security infrastructures and to reduce the cost of security management. To achieve this goal, Web Services security standards have to provide a common framework, and common protocols, for the exchange of security information between/among Web Services that, once implemented and deployed,

- can accommodate such existing heterogeneous mechanisms, i.e. different encryption algorithms, different access control mechanisms, etc.
- can be extended so as to cope with new requirements and/or available security technologies.

Ensuring the integrity, confidentiality, and security of Web Services through the application of a complete security model is essential for the wide adoption of this technology in SOAs. Security should be a testable property of a published service SLA. Testing Web Services for vulnerabilities is a difficult task [31] complicated by current trends in service collaboration that includes federation. The OASIS WS-Security versions (WSS) 1.0 and 1.1 [12, 13, 14, 15, 16, 17, 18] use SOAP extensibility facilities to provide security functions for SOAP messages. The WS-Security specifications secure the SOAP foundation layer by leveraging core technologies such as XML Signature [20], XML Encryption [21], XML Canonicalization [28], and SSL/TLS [30] as well as standards for conveying key and other information, such as SAML [16] and X509 [15] certificates. The WS-Security (WSS) specification provides a flexible framework for securing SOAP messages. For this reason, it is difficult for various implementations of WS-Security to interoperate unless similar choices have been made. Lack of interoperability at the SOAP and SOAP message security layer adds complexity for security testing of Web Services implementations.

This work is organized as follows. Section 14.2 discusses Web Services security challenges. Section 14.3 develops a Web Services security standards framework. Section 14.4 discusses the complexities of testing Web Services. This section provides a brief overview of current testing strategies and discusses the role of Web Services middleware in real-life deployments. Additionally, this section identifies Web Services interaction stake holders, their testing perspectives, and their testing levels.

Section 14.5 addresses Web Services security interoperability as an enhancement for testability. The section discusses the Basic Security Profile (BSP) usage scenarios, BSP strength of requirements, BSP conformance and BSP testability. This section provides an example of BSP profiling and takes a close look at BSP security considerations.

Section 14.6 considers strategies for testing Web Services security with a focus on developing a Web Services security fault model. The section provides an overview of testing strategies for Web Services security that includes general testability guidelines. A case study for securing an application is also provided. Section 14.7 investigates the use of crash-only software for enhancing the availability of and reducing testing complexities for composite Web Services. Section 14.8 provides research proposals and discusses open research issues. Section 14.9 concludes the discussion of this work and provides guidelines for future research.

14.2 Web Services Security Challenges

The discussion in the previous sections highlights the difficulties of testing Web Services. The complexities increase when security is taken into consideration, especially when application security applied at the server is not enough. A multilayer service and resource protection strategy is required, including the application of security techniques for Web Services description, discovery, and messaging.

The main features that make Web Services attractive to enterprises, such as accessibility to data, dynamic application connections, platform independence, and open run-time environment, are at odds with traditional security approaches. Security can be a key inhibitor to the wide adoption of Web Services [22]; there is need to develop a Web security vulnerabilities framework reflecting the service deployment environment. Understanding these vulnerabilities would help developers choose the right testing tools to detect faults in the services.

14.2.1 Web Services Threats

This section provides a brief description of the most common threats facing the security of Web Services. For a more complete analysis of these challenges, the reader is referred to [10], where Web Services threats and possible counter measures are discussed in more detail.

1. Message alteration: In this threat, message information is altered by inserting, removing, or modifying information created by the originator of the information and mistaken by the receiver for the originator's intention. This type of attack is easily performed due to the use of intermediaries and transformation mechanisms in Web Services.
2. Confidentiality: This type of threat makes information within the message visible to unauthorized participants.
3. Falsified messages: This threat occurs when an attacker constructs counterfeit messages and sends them to a receiver who believes them to have originated from a party other than the sender.
4. Man in the middle: The term "man in the middle" is applied to a wide variety of attacks that have little in common except for their topology. In our context, this type of attack occurs when a third party poses as the other participant to the real sender and receiver in order to fool both participants (e.g. the attacker is able to downgrade the level of cryptography used to secure the message). Designers have to examine their developed applications on a case-by-case basis for susceptibility to anything a third party might do.
5. Principal spoofing: A message is sent which appears to be from another principal (e.g., Alice sends a message which appears as if it is from Bob).
6. Forged claims: A message is sent in which the security claims are forged in an effort to gain access to otherwise unauthorized information (e.g., a security token which was not really issued by the specified authority).
7. Replay of message parts: A message is sent which includes portions of another message in an effort to gain access to unauthorized information or to cause the receiver to take some action (e.g., a security token from another message is added). This technique can be applied in a wide variety of situations. All designs must be carefully inspected from the perspective of what could an attacker do by replaying messages or parts of messages.
8. Replay: A whole message is resent by an attacker.
9. Denial of service: This is a form of an amplifier attack where the attacker does a small amount of work forcing the system under attack to do a large amount of work. This can cause the attacked system to provide a degraded service or even fail completely.

14.2.2 End-to-End Security Requirements for Web Services

For Web Services Security, the objective is to create an environment where message-level transactions and business processes can be conducted securely in an end-to-end fashion. The requirements for providing end-to-end security for Web Services are as follows:

- Mutual authentication: Mutual authentication of a service provider and a service invoker to verify their identities enables them to interact with confidence. This also includes data origin authentication whereby the receiver can be sure that the data came from the sender without modification.
- Authorization to access resources: Authorization mechanisms control invoker access to appropriate system resources, controlling access to systems and their components. Authentication may be necessary to perform authorization.
- Data integrity and confidentiality: Ensure that information has not been modified during transmission and is only accessible by the intended parties. Encryption technology and digital signature techniques can be used for this purpose.
- End-to-end integrity and confidentiality of messages: Ensure the integrity and confidentiality of messages even in the presence of intermediaries.
- Integrity of transactions and communications: Ensure that the business process was done properly and the flow of operations was executed in a correct manner.
- Audit records and mechanisms: Enable dispute resolution and system verification. Records are necessary to enable a resolution if a party to a transaction denies the occurrence of the transaction, or if other disputes arise; also to trace user access, behavior, and to enable system integrity verification.
- Distributed enforcement of security policy: Enable implementers to define a security policy and enforce it across various platforms with varying privileges.

14.2.3 Role of Cryptography

Cryptography [10] can play an important role in mitigating some of the threats to Web Services. For example, symmetric cryptography in the form of encryption can be used to provide confidentiality for messages. Asymmetric cryptography, on the other hand, can be used to enable authentication, confidentiality, and dispute resolution. This can be achieved through the appropriate use of public and private keys. Here, it is assumed the original sender and the receiver (including intermediaries if they are on the trust path) have access to the appropriate public keys through some mechanism. In this regard, dispute resolution can be enabled when the sender encrypts a message using its private key since it is harder for the sender to deny the sending of the message.

Confidentiality can be achieved when the original sender encrypts a message using the receiver public key, requiring the receiver private key for decryption. In a typical deployment, asymmetric cryptography is used as a mechanism for exchanging session keys to be subsequently used in symmetric cryptography. This technique is more efficient since symmetric cryptography is usually faster to execute than asymmetric cryptography.

Asymmetric cryptography can provide authentication. In this case the sender digitally signs a message using its private key. The receiver then verifies the signature and the authenticity of the sender certificate to confirm that the sender is the party that actually sent the message.

Asymmetric cryptography can be used to provide message integrity whereby a message is first digitally signed by the original requestor private key and is then encrypted using the ultimate receiver public key. Upon receiving the message, the receiver first decrypts the message with its corresponding private key and then decrypts the message with the sender public key. Integrity is verified when the process executes without any errors.

14.2.4 Transport Layer Security

This subsection addresses using Transport layer [10] to secure SOAP messages that are sent from a sender to a receiver. This approach is limited when there are intermediaries; since termination of transport layer security at an endpoint may allow that intermediary to modify or examine messages. For HTTP-based bindings of SOAP, TLS/SSL provides point-to-point security (Fig. 14.1). For Web Services, however, there is a need for end-to-end security, which becomes an important distinction when one or more intermediaries exist between the original service requester and the service provider. In this case the use of Transport layer TLS/SSL has significant limitations. Transport layer security mechanisms may be used to secure messages between two adjacent SOAP nodes and message layer security mechanisms should be used (possibly in conjunction with TLS/SSL) in the presence of intermediaries or when data origin authentication is required.

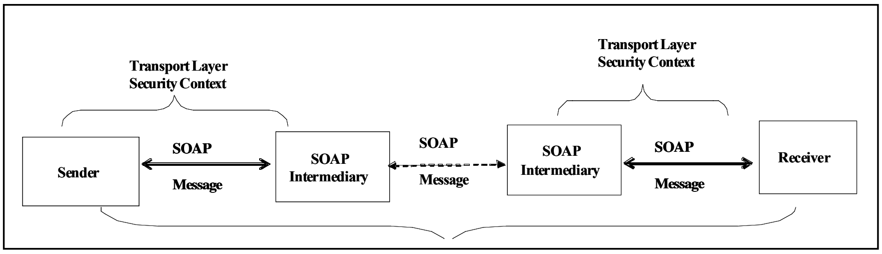

Fig. 14.1. Transport and message level Security

14.2.5 Message Level Security

The SOAP specifications [23, 24] do not specify how to deal with security-related issues such as authentication, integrity, and confidentiality. However, they provide an extensibility model that can be used to build extensions to the original SOAP standard. The OASIS WS-Security versions (WSS) 1.0 and 1.1[12, 13, 14, 15, 16, 17, 18] use these extensibility facilities to add security functions to SOAP. WS-Security specifications secure the SOAP foundation layer by leveraging core technologies such as XML Signature[20], XML Encryption [21], XML Canonicalization [28], and SSL/TLS. The WS-Security specification adds security to SOAP messages by specifying how the header part of the message can carry security information in conjunction with rules on how to apply security technologies.

The OASIS WSS 1.0 standard [12] provides the underlying foundation for SOAP message level security. It defines mechanisms for identifying the origin of a message and verifying tampering through the use of signatures. It provides mechanisms for message confidentiality by ensuring that only the intended recipient is able to see the message through the use of encryption. WSS 1.0 introduces a security header in a SOAP message and three key elements:

1. Tokens: SOAP messages can contain security tokens with authentication information. The standard defines Username tokens, X.509 Tokens, and SAML tokens, among others. These tokens can be part of security headers and can vouch for security claims to a recipient.
2. Signature elements: Security headers can contain Signature elements that contain an XML Signature used to sign any part of the message. The recipient can use the signature to verify that the request of the sender has not been changed and that the message really originated from the sender.
3. Encryption elements: Some parts of the SOAP message can be encrypted to protect sensitive information from unauthorized entities.

WS-Security defines a security header for SOAP messages as a mechanism for conveying security information with and about a SOAP message. This header is, by design, extensible to support many types of security information. The security header may contain security tokens, references to security tokens found elsewhere, timestamps, nonce, signatures, encrypted keys, and encrypted data. Each security header is targeted to a specific SOAP actor. A SOAP message may contain multiple security headers; however, each must be targeted to a different SOAP actor. Each security header may contain multiple security tokens, security token references, nonce, signatures, encrypted keys, and encrypted data; however, the BSP recommends that there may be at most one timestamp in a message.

The WS-Security standard describes the processing rules for using and processing XML Signature [20] and XML Encryption [21] in the context of a SOAP message; however, these rules do not apply to using these standards directly in application data. These WS-Security rules must be followed when using any type of security token. The specification does not dictate if and how claim confirmation must be done; however, it does define how signatures may be used and associated with security tokens (by referencing the security tokens from the signature) as a form of claim confirmation.

WS-Security 1.1 enhances WSS 1.0 with additional mechanisms to convey token information (e.g., sending Thumbprint of an X.509 certificate, or a SHA1 hash of an Encrypted key). WSS1.1 also introduces the concept of SignatureConfirmation that enables a communication sender to confirm that the received message was generated in response to a message it initiated in its unaltered form. In this technique, the recipient sends back the signature values received from sender in SignatureConfirmation element. This technique helps to prevent certain forms of reply and message alteration attacks. WS-Security 1.1 has become an OASIS standard as of February 1, 2006. WSS1.1 also introduces a mechanism to encrypt SOAP headers.

WS-Security provides mechanisms to send security tokens as part of a message, message integrity, and message confidentiality. Developers can use the specification in conjunction with other Web Service extensions and higher-level application specific protocols to accommodate a wide variety of security models and security technologies. It does not specify how a security context or authentication mechanisms are established. Furthermore, key derivation, advertisement and exchange of security policy, trust establishment, and non-repudiation are out of scope of the specification.

14.3 Web Services Security Standard Framework

In this section we present first the different notions of standards. We then present the conceptual framework for Web Services security standards, and, for each level of this framework, we survey existing and proposed standards, their specific purpose, and their current status.

14.3.1 The Concept of Standard

The concept of “standard” covers different notions, ranging from a public specification issued by a set of companies, to a de jure standard issued by a recognized standardization body. These different notions can provide to the potential users useful indications about the maturity, the stability, and the level of endorsement of a given standard. A de facto standard is a technology that is used by a vast majority of the users of a function. Such function may, e.g., be provided in a product from a single supplier that dominates the market; or it may be a patented technology that is used in a range of products under license. A de facto standard may be endorsed by a standardization initiative, and eventually become a consortium recommendation, or a de jure standard. The relevant requirements are that it is widely used, meets the needs for functionality, and supports interoperability.

A de jure standard is usually defined by entities with a legal status in international or national law such the International Organization for Standardization (ISO). Consortium recommendations on the other hand are a technology agreed on and recommended by groups of companies in order to fulfill some functionality. The Organization for the Advancement of Structured Information Standards (OASIS), the World Wide Web Consortium (W3C), and the Internet Engineering Task Force (IETF) are examples of examples of such consortia.

De facto standards, eventually promoted to the de jure standard by a subsequent endorsement by a standardization body, offer a higher guarantee of support for interoperability. Conversely, de jure standards or consortia recommendations do not guarantee per se that a standard will be widely endorsed nor the market availability of really interoperable implementations by multiple vendors. Moreover, the definition of a standard and its issuance by a standardization body or by a consortium is a long-lasting process, subject to formalized organizational procedures.

14.3.2 Framework for Web Services Security Standards

Web Services security standards address a variety of aspects, ranging from the message-level security to the identity management. In order to provide a structured and engineered approach to the development of the standards, an overall conceptual reference framework was needed. Such a reference framework is crucial in organizing the standards according to layers and in promoting the reuse of already developed specification.

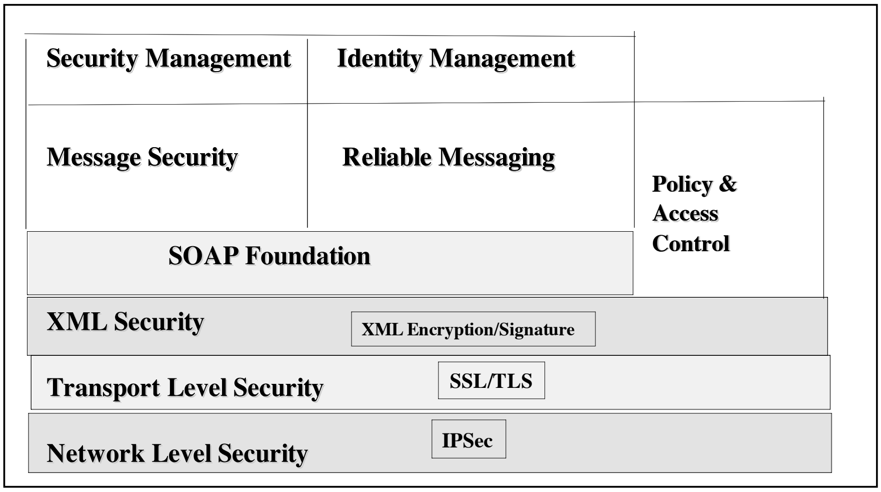

Fig. 14.2. Refined classification of standards

In this work, we adopt the following classification, as shown by Fig. 14.2. This classification has been adopted in order to take into account in the discussion the standards below the SOAP layer and most importantly, to group the standards by their main intended purpose rather than adopting a "stack" view that emphasizes mainly how each specification is built on top of the other ones. In particular, we deemed useful to separate message-level security specifications (the two groups labeled Message Security and Reliable Messaging) from the specifications addressing Policy and Access Control, Security Management, and Identity Management issues.

14.4 Complexities of Testing Web Services

Currently, Web Services are generally managed using tools supplied by platform vendors [34]. This makes testing a vendor-dependent activity. Service management includes configuration, accounting, QoS, policy and fault identification, containment, and repair. Passive or active testing mechanisms are widely used as tools for fault detection. Active testing techniques require the generation and the application of test cases in order to detect faults [31, 32, 33, 34, 35]. Passive testing techniques use observers to track the interaction between the entities being tested. Observers can be inserted directly on-line in the data flow or can be off-line and with access to log files.

Current proposed Web Services strategies are either based on active testing techniques [34] or require Web Services to participate in their management through the support of a management interface to active testers [34]. These solutions assume that a Web Service will participate in its management by providing specific interfaces that are based on active testers. Requiring Web Services to provide their own management interfaces adds complexity to Web Services architecture and may impact performance. In addition, there are security risks associated with these interfaces.

14.4.1 Brief Overview of Current Testing Strategies

The advent of Web Services and their role in realizing SOA are changing the Internet to a platform of application collaboration and integration. This will change the traditional design, build, test, launch, and retire software life cycle. The change will be more profound once companies start to realize the importance of orchestrating loosely coupled services into coarse-grained business services as a way of quickly developing business solutions.

As enterprises adopt SOA principles, the traditional test after development approach to software testing will no longer work. Instead, software projects in enterprises will be based on agile approaches [48]. Accordingly, software development will require developers to work closely with their clients to identify their needs. Developers will produce code that is tested and evaluated by the customers and the process is repeated until the project is done. This approach requires a change in the way test code is developed and will result in developing the test code as part of the software development process [48].

In the Web Services world, dynamic binding allows developers to define service centric systems as a workflow of invocations to abstract interfaces. The interfaces are then bound to concrete services before or during workflow execution. Testing techniques that require the pre-identification of system components cannot be used to test these workflows [47]. The ability to use dynamic bindings in a workflow raises the need to test a composite service partner link for all possible endpoints [47]. The problem can be very complex since the endpoint can be dynamically generated or unknown at testing time [47].

Currently, Web Services testing is a discipline at its early stages of study by the academic and industrial communities [38]. Some approaches in the R&D community [42, 43] have suggested the possibility of augmenting the functionality of a Universal Description, Discovery and Integration (UDDI) service broker with logic to permit a testing step before a service is registered. The aim of the testing step is to ensure that the logic of the registered service is error free. This approach focuses on requiring Web Services to include well-defined test suites that can be run by the enhanced UDDI, or the inclusion of Graph Transformation Rules that allow the automatic derivation of meaningful test cases that can be used to evaluate the behavior of the service when invoked. This approach require that Web Services providers implement interfaces that increase the service testability by bringing the service into a known state from which a specified sequence of tests can be performed.

A modification of this approach is presented in [38]. In their work, the authors propose a UDDI extension mechanism to verify that a Web Service can correctly cooperate with other services. This is done by checking that a correct sequence of invocations to the service results in a correct interaction of the service with services from other providers. The proposed framework extends the UDDI registry role from a passive service directory to an accredited testing entity that performs service validation before registering a service.

Mei et al. in [46] propose a framework to automate the testing process of Web Services. This framework is designed to generate test data according to the description of Web Services in an extended version of Web Services Description Language (WSDL). The work extends WSDL with contract information, including pre-conditions and post-conditions. From the basic information and the contract information, test data for a Web Service can be generated. Relational expressions appearing in the pre-conditions are used to partition the range of each input parameter into several sub-ranges. For each parameter, the technique randomly selects a value from a sub-range together with the boundary values between sub-ranges. The different combinations of the values for the parameters become the initial generated test data which is used for the automatic execution of the Web Service under test. For composed Web Services, the framework can intercept and record the inputs from each

Web Service to be used for future regression test. The framework specifies two ways to acquire test data. The first way is to use a test data generator; the second way is to record the runtime information while executing an application that invokes the service under test.

Benharref et al. [34] proposes architecture based on passive testing techniques (using observers) for detecting faults in Web Services. The observers are designed as Web Services that makes them platform independent. Their architecture enables the testing of deployed Web Services by independent third parties.

Tsai et al. [38] proposes an XML-based, object-oriented framework to test Web Services. The framework supports test execution and test scenario management, consisting of a test master and a test engine. The test master enables developers to identify test scenarios, perform dependency, completeness and consistency analysis. The test engine interacts with the Web Services under test and provides tracing information. XML perturbation testing techniques, such as discussed in [43, 44, 45], can also be conducted in the framework of Tsai as given in [38].

Testing strategies are even more complex when Web Services security is also taken into consideration. Security challenges when testing Web Services and the need for interoperability at the SOAP message security level are addressed in a subsequent section.

14.4.2 Web Services Middleware

Web Services Middleware [37], also known as Web Services Management (WSM), is a distributed infrastructure that acts like enforcement points. WSM can be either a gateway that handles traffic for multiple Web Services or agents co-resident on systems running a given Web Service.

The presence of the WSM infrastructure [37] is often transparent to a given Web Service and to any software invoking that service. In actual deployment scenarios, a WSM would appear like a standard service consumer to a Web Service and a Web Service to the consuming application. The WSM uses standard Web Services technology to communicate with the Web Service and the software consuming that service.

WSM infrastructure addresses key areas that are related to Web Services; in particular, security, system management, and service virtualization. Interoperability of Web Services at the WSDL, SOAP, and SOAP message–level security can also be addressed in the WSM.

Figure 14.3 provides an overview of Web Services Management framework's functional components. Components can communicate with each other. For clarity, the communication lines are omitted from Fig. 14.3. Not all components need to be present in WSM infrastructure. The following components are included in Fig. 14.3:

- Access control component enforces access control policies that may include the capability to authenticate and authorize Web Services' clients.

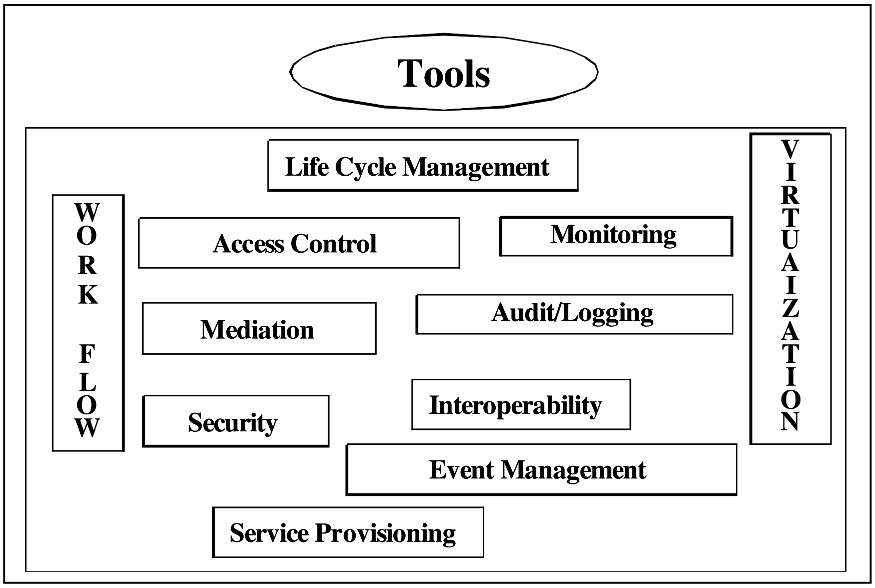

Fig. 14.3. WSM architecture

- Audit/logging logs requests, responses, various events, and session information.
- Event management handles events that are related to Web Services. For example, alerts can be sent based on pre-set conditions.
- Interoperability component is responsible for insuring interoperability that may include many layers such as the WSDL, SOAP, and SOAP message security layer.
- Life cycle manger manages the development, deployment, registering, and testing stages of services.
- Mediation component enables Web Services federation through the enforcement of federation policies.
- Monitoring component monitors events from all deployed Web Services.
- Security component is a Policy Enforcement Point (PEP). It addresses security-related issues across services that may include secure communication channels, authentication, authorization, privacy, trust, integrity, confidentiality, federation, delegation, and auditing.
- Workflow manager creates, tests, and manages the logical flows of Web Services.
- Service provisioning defines system behavior policies and the interactions of the Web Services. It can be used to specify how Web Services clients can subscribe to a given service. It can also be used to specify the rules for client authentication and authorization before they can access Web Services.

- Virtualization component creates and manages virtual endpoints for Web Services. These endpoints can be dynamically associated with physical endpoints to manage fail-over, provide load balancing, and concurrently manage multiple versions or invocations of a Web Service.

The tools are presented as separate components to emphasize the need not to be locked into vendor-specific tool set that can lead to limited testing functionality of the WSM infrastructure. Selecting the right tools is critical for the task of testing Web Services. It is important to note that Web Services impose specialized testing challenges for test tools. These tools need to be able to emulate realistic usage scenarios. They should enable developers to create the ability to rapidly test Web Services for functionality, performance, reliability, scalability, and security. Since service re-use and service availability are essential to achieving robust SOA implementations, automated regression testing is necessary in order to guarantee secure, reliable, and complaint services.

The use of an agile software development methodology requires the testing process to be capable of detecting errors early in the development cycle. This requires the flexibility to address known usage scenarios as well as unanticipated use cases. Most errors are caused when a system component is used in an unexpected manner. Improperly tested code, executed in a way that the developer did not intend, is often the primary culprit for security vulnerability.

The WSM framework allows developers to perform fault management, configuration, accounting, performance, and security aspects of service management. Fault management includes fault detection, localization, and repair. Passive or active testing techniques can be used for fault detection. Active testing is based on the generation and the application of specific test cases while passive testers just observe the interaction between a tested system and its clients. The introduction of WSM into a corporation's infrastructure allows developers to concentrate on developing the services while letting the WSM handle non-application, context-specific security needs, manageability, and other aspects of the service. Developers need to note that Web Services' gateway solutions usually do not have access to application context. Developers still need to perform tests that check the content of XML messages since attackers can embed malicious content in the XML documents that pass straight through the WSM software to the service interface of the application.

The use of a WSM framework allows practical implementations of Web Services where providers can develop Web Services Service Level Agreements (WSLAs) that the clients can use as contracts when invoking the services. In traditional terminology, SLAs represent a formal contract between a service provider and a client guaranteeing quantifiable network performance at defined levels. These types of SLAs are network centric and generally deal with packet flows across a network. At the Web Services level, a WSLA is more concerned with message flows that span the end-to-end business transaction. These are both depicted in Fig. 14.4. WSLAs can be used to provide QoS that is based on the contract they have agreed upon when they subscribed

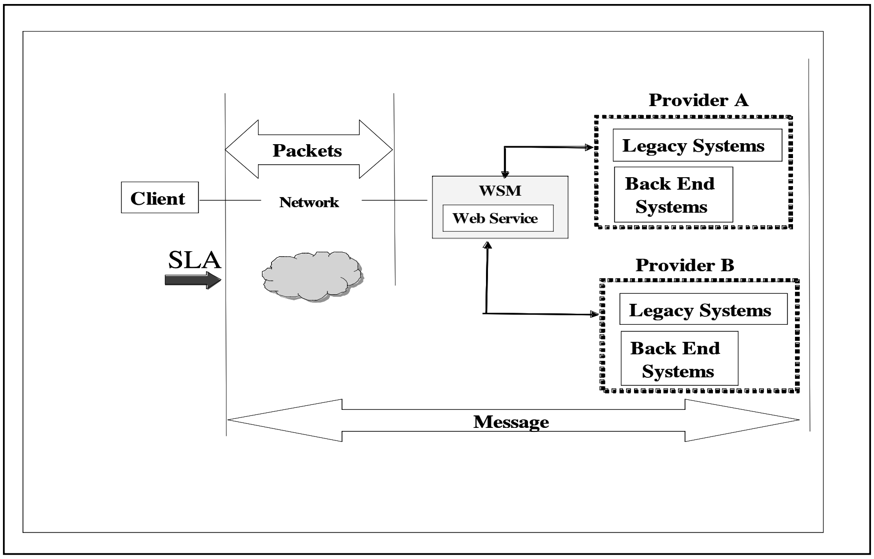

Fig. 14.4. Client's view of SLA testing for Web Services

to services. Clients can develop testing strategies that stress the WSLA to ensure that the service provider has met the contracted QoS commitment.

Stress testing WSLA requires interoperability of the Web Services at the WSDL, UDDI, SOAP, and SOAP message–level security. The Basic Profile (BP) [11] from the Web Services Interoperability Organization (WS-I) provides a profile for enhancing interoperability at the SOAP level. In addition, WS-I has developed the Basic Security Profile (BSP) [11] for enhancing interoperability at the SOAP message–level security. In a subsequent section, we will take a closer look at the BSP.

14.4.3 Stake Holders Testing Perspectives and Levels

Many players can get involved when a Web Services consumer invokes a Web Service. The stakeholders are the end-user or client, service developer, service provider, service integrator, and service broker (certifier) [47]. Testing scope, strategies, techniques, and perspective will vary based on the stakeholder. Each stakeholder must deal with different requirements and issues [47].

The client or end-user expects any application to perform in a satisfactory manner. An important aim of the service provider is to provide reliable services; the service provider will focus on functional testing in order to ensure the minimum number of failures. The service provider cannot anticipate the details of how the service will be combined with all other services. Hence, although the service developer can perform some non-functional testing of the developed services, these tests are limited since the service developer lacks exposure to the infrastructure of the end-to-end message flow. In general, the service developer will focus on performing service functional testing that can be based on common techniques that are used in component or subsystem testing [47]. The provider will need to perform tests on the WSDL, UDDI, and SOAP layers. Tests based on mutation strategies [44, 45, 46, 47] are important for the service developer in order to detect faults. The developer will need to perform regression testing if any of the components of the service change. The developer may need to provide an interface to the service to allow the service provider, integrator, or certifier to test the service in an SLA scenario.

The main focus of the service provider is to ensure that the service meets the claims as stated by the service developer. The aim is to be sure it can meet the requirements of a WSLA. The service provider can use the same testing techniques as the service developer. However, load testing the service may also be an option in order to gain confidence in its WSLA conformance. From the service provider point of view, non-functional testing of the service has limited value since it does not include the end customer infrastructure.

The role of service integrator is to test its services that can be used in a composite fashion by consumers in order to ensure that the original design (functional and nonfunctional) objectives are met at invocation time [47]. Dynamic binding adds complexity to the service integrator testing scope, strategies, and capabilities. At runtime, dynamic binding adds uncertainty since the bound service can be one of many possible services. From the point of view of a service workflow, the service integrator has no control over the service in use, since it can change over its deployed lifetime. To increase confidence in the testing process, the service integrator will need to invoke the service in order to gain insight on how it will behave in the real world. Testing from this perspective will result in additional costs to the service integrator. The use of service emulators and stubs can reduce this cost, but do not fully replace the need for the actual invocation of the services under test. The service integrator may invoke more sophisticated test generation strategies. Pre-conditions, post-conditions, and genetic algorithm testing ([43, 44, 45, 46, 47]) can be used to create test oracles. Stress testing WSLA (at least with focus on the infrastructure components or sub-systems that the integrator can control) should be performed in order to get better understanding of whether the service will meet the QoS requirements of the contract.

The service certifier can be used by the service developer, provider, or integrator to help test and find faults. The service certifier can also be the service broker. The service certifier can play an important role in reducing the number of players involved in testing a service and as a result can reduce the overall cost of testing. However, the service certifier still lacks visibility of how the service will be composed with other services and lacks access to the infrastructure of the end-to-end message flow. The service certifier may invoke more sophisticated test generation strategies on the service. Pre-conditions, post-conditions, and genetic algorithm [47] testing can be used to create test oracles for a given service. Due to dynamic binding and the lack of visibility of network and infrastructure factors that can affect the performance of a service, the service certifier may not be able to guarantee the QoS claims of a WSLA.

14.5 Interoperability as an Enhancement for Testability

The framework for security standard development postulates a layered approach, such that every upper layer standard can re-use and extend the specification of lower-layer standards. However, the specifications of the standard at a given layer (e.g., WS-Policy) are sometimes developed by a standardization body different from that specifying the standard at the other layer (e.g., SAML). Thus, the two involved standard specifications are not always synchronized. Such situation requires an activity of verification and alignment of the specifications, which involves further iterations within each standardization body.

Due to the role played by SOAP messages and by SOAP message security, interoperability of different WS-Security implementation is crucial. For this reason, WS-I has developed the Basic Security Profile (BSP) [11] to provide clarifications and constraints in order to enhance the interoperability of WS-Security implementations. The BSP extends the profiles created by the WS-I SOAP Basic Profile (BP) [11] by adding interoperability guidelines for security. BSP 1.0 profiles WSS 1.0 and is available on the WS-I public site. BSP 1.1 profiles WSS 1.1 and should be available to the public in the near future. In this chapter, we use the term BSP to mean both BSP 1.0 and 1.1

SOAP messages are composed of XML elements. Using WS-Security techniques, these elements may be signed, encrypted, or signed and encrypted. The elements can be referenced from other elements. Each element within a SOAP message may be processed by an intermediary that can add more data and sign and encrypt the incremental data and/or the combined data. For example, in an order processing chain of events, one intermediary can assign an order number and sign the associated element. Another intermediary can check credit worthiness of the consumer and either signs only the credit data or the whole order data, and so forth.

14.5.1 BSP Usage Scenarios

The BSP adds security to the following three basic Message Exchange Patterns (MEPs) that were adapted from the scenarios defined for the Basic Profile [11]:

1. One-way: A SOAP message is sent, potentially through intermediaries, to a SOAP receiver. No response message is returned (Fig. 14.5).

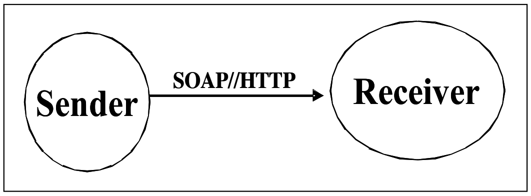

Fig. 14.5. One-way SOAP message

2. Synchronous request/response: A SOAP message (the request) is sent, potentially through intermediaries, to an ultimate SOAP receiver. A SOAP message (the response) is sent by the request's ultimate SOAP receiver through the reverse path followed by the request to the request's initial SOAP sender (Fig. 14.6).
3. Basic callback: A SOAP message (the request) is sent, potentially through intermediaries, to an ultimate SOAP receiver, and an acknowledgment message is returned in the manner of synchronous request/response. The request contains information that indicates an endpoint for a SOAP node, where the response should be sent. The request's ultimate SOAP receiver sends the response to that SOAP node, which returns an acknowledgment message in the manner of synchronous request/response (Fig. 14.7).

14.5.2 BSP Strength of Requirements

The BSP focuses on improving interoperability by strengthening requirements when possible and constraining flexibility and extensibility when appropriate. The BSP limits the set of common functionality that vendors must implement and thus enhances the chances for interoperability. This in return reduces the complexities for the testing of Web Services security.

The guiding principles enumerated in the BSP declare that there is no guarantee interoperability, that the profile should "do no harm," that it makes testable statements when possible, and focus on interoperability. The BSP committee worked so that enhancing interoperability does not create new security threats.

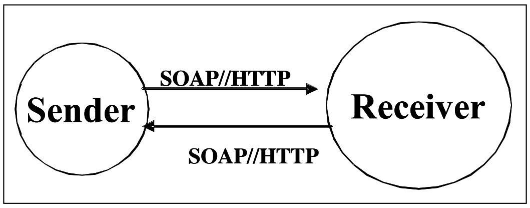

Fig. 14.6. Synchronous request/response

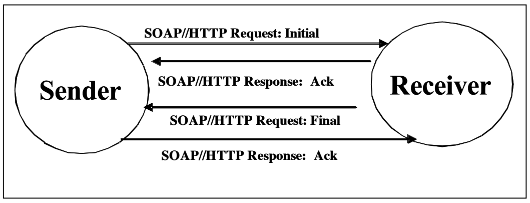

Fig. 14.7. Basic callback

It is not the intent of the profile to define security best practices. However, when multiple options exist, the profile considers known security weaknesses and makes choices that reduce the risks and reduces choice thus enhancing interoperability. The Profile speaks to interoperability at the Web Services layer only; it assumes that interoperability of lower-layer protocols (e.g., TCP, HTTP) and technologies (e.g., encryption and signature algorithms) are adequate and well understood. The Basic Security Profile restates selected requirements from the WS-Security Errata rather than including the entire Errata by reference, preferring interoperability over strict conformance.

The profile includes requirement statements about two kinds of artifacts: SECURE_ENVELOPE and SECURE_MESSAGE. A SECURE_ENVELOPE is a SOAP envelope that has been subjected to integrity and/or confidentiality protection. A SECURE_MESSAGE expands the scope of the SECURE_ENVELOPE to include protocol elements transmitted with the SECURE_ENVELOPE that have been subjected to integrity and/or confidentiality protection (an example is SOAP messages with attachments).

14.5.3 BSP Conformance

In order to conform to the BSP, any artifact that contains a construct that is addressed in the profile must conform to any statements that constrain its use. Conformant receivers are not required to accept all possible conformant messages. Conformance applies to deployed instances of services. Since major portions of the BSP may or may not apply in certain circumstances, individual URIs may be used to indicate conformance to parts of the BSP including the core profile or additional sections of the BSP for Username token, X.509 token, and SOAP messages with attachments.

The BSP includes statements that are interoperability requirements as well as statements that are security considerations. The normative requirement statements are identified by numbers prefixed with the letter ‘R’, e.g., Raaaa where aaaa is the statement number. These statements contain one requirement level keyword (i.e., “MUST”) and one conformance target. Examples of BSP conformance targets include the following:

SECURE_ENVELOPE: A SOAP envelope that contains sub-elements that have been subjected to integrity and/or confidentiality protection.

A message is considered conformant when all of its contained artifacts are conformant with all statements in the BSP that are related to them. Use of artifacts for which there are no statements in the Basic Security Profile does not affect conformance.

SECURE_MESSAGE: Protocol elements that have WS-Security applied to them. Protocol elements include a primary SOAP envelope and optionally associated SOAP attachments.

SENDER: Software that generates a message according to the protocol(s) associated with it. A sender is considered conformant when all of the messages it produces are conformant and its behavior is conformant with all statements related to SENDER in BSP.

RECEIVER: Software that consumes a message according to the protocol(s) associated with it. A receiver is considered conformant when it is capable of consuming conformant messages containing the artifacts that it supports and its behavior is conformant with all statements related to RECEIVER in the BSP.

In BSP, certain statements are considered clarifying statements. The intent of these statements is to eliminate confusion about the intended interpretation of a requirement from an underlying specification. Clarifying requirements are identified by adding a suffix of a superscript letter 'C', i.e. `RxxxxC`, where `xxxx` is the requirement number. Additional consideration statements are also identified by numbers prefixed by the letter 'C', i.e. `Cyyyy`, where `yyyy` is the statement number. These statements are non-normative and are used to provide clarification in order to eliminate confusion.

14.5.4 BSP Testability

The security consideration statements provide guidance that is not strictly interoperability related but are testable best practices for security. It was considered valuable to include these statements so that testing tool designers can have the option of flagging potentially insecure practices. It is not feasible to provide a comprehensive list of security considerations and not all security considerations can easily be converted into testable statements. A complete security analysis must be conducted on specific solutions based on the BSP and underlying standards, based on a risk analysis of the application using BSP technologies.

Even a fully standard compliant application may not interoperate with another when the set of functionality supported is disjoint. For example, while a sender may encrypt using one of three specific algorithms prescribed by the BSP, a receiver may expect a different one of the three. Certain agreements must be made using mechanisms currently out of scope for the profile.

14.5.5 Example of BSP Profiling

This section provides an example of BSP profiling with respect to SOAP Message Security. BSP allows limited flexibility and extensibility in the application of security to messages. Since no security policy description language or negotiation mechanism is within the scope of the profile, BSP expects that the sender and receiver can agree out of band over which mechanisms and choices should be used for message exchanges including which security tokens can be used. The next sections provide selected examples of the profiling in the BSP. The reader can check [11] for the complete profile.

WSS 1.1 allows a Binary Security Token for the option of specifying its Value Type, but requires that it specifies its encoding type. Base64Binary is the only acceptable value. The Profile restricts the Value Type to one of those specified by a security token profile and requires its specification. Note that this token profile need not be one of the OASIS WSS profiles, although that is preferred when possible. The listing below provides an example of the profiled usage of Binary Security Token.

```
Correct:

<wsse:BinarySecurityToken wsu:Id='SomeCert'
      ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-
      200401-wss-x509-token-profile-1.0#X509v3"
      EncodingType="http://docs.oasis-open.org/wss/2004/01/
      oasis-200401-wss-soap-message-security-1.0#Base64
      Binary">
lui+Jy4WYKGJW5xM3aHnLxOpGVIpzSg4V486hHFe7sHET/uxxVBovT7JV
1A2RnWSWkXm9jAEdsm/...
</wsse:BinarySecurityToken>
```

14.5.6 BSP Security Considerations

This section lists a number of security considerations as specified by the BSP that should be taken into account when using one or more of the technologies discussed in this section.

Use of the `SOAPAction` in protected messages can result in security risks. The `SOAPAction` header can expose sensitive information about a SOAP message such as the URI of the service, or the context of the transaction. If the `SOAPAction` header is used for routing messages, there is a possibility that an attacker can modify the header value to direct the message to a different receiver. This can defeat a replay detection mechanism based on the assumption that the message would always be routed to the same place.

Additional risks can occur if multiple intermediaries are present. For example, one intermediary can be designed to select the set of its processing steps based on the value of SOAPAction or application/soap+xml. A second intermediary (such as a security gateway) can base its processing on the message content (which could be secured through XML signatures). An attacker can manage to trick the security gateway by allowing illegal operations by modifications in HTTP headers. To remedy the situation, the BSP requires that SOAPAction attribute of a `soapbind:operation` element to either be omitted or have as its value an empty string.

BSP uses time-based mechanisms to prevent replay attacks. These mechanisms will be ineffective unless the system clocks of the various network nodes are synchronized. The BSP assumes that time synchronization is performed.

For messages that are signed using a Security Token that binds a public verification key with other claims, and if specific processing is performed based on those claims, the BSP requires that the Security Token itself be part of the signature computation. This can be achieved by putting child `ds:Reference` element whose URI attribute contains a shorthand XPointer reference to the `wsse:SecurityTokenReference` that specifies the Security Token into the `ds:SignedInfo` element of a signature. If a `ds:SignedInfo` contains one or more `ds:Reference` children whose URI attribute contains a shorthand XPointer reference to a `wsse:SecurityTokenReference` that uses a potentially ambiguous mechanism to refer to the Security Token (e.g., KeyIdentifier) then it is recommended that the content of the Security Token be signed either directly or using the Security Token Dereferencing Transform. This approach can help to protect against post-signature substitution of the Security Token with one that binds the same key to different claims.

When a key is provided in band within a Security Token or for the purpose of specifying a key to be used by another node for encrypting information to be sent in a subsequent message, the profile recommends that the sender of the key cryptographically bind the key to the message in which it is transmitted. This can be done either by using the key to perform a Signature or HMAC over critical elements of the message body or by including the key under a signature covering critical elements of the message body that uses some other key. If a key is sent in a message that the receiver is expected to encrypt data in some future message, there is a risk that an attacker could substitute some other key and thereby be able to read unauthorized data. This is true even if the key is contained in a signed certificate, but is not bound to the current message in some way. If the future encryption key is used to sign the initial request, the receiver can determine by verifying the signature that the key is the one intended. Readers can consult [11] for more detailed security risk analysis.

14.6 Strategies for Testing Web Services Security

Testing developed and deployed secure Web Services applications is a challenge. Security is an ongoing process as opposed to a one-time development task. Developers should start with the security of the application in mind from the origin concept of the service and during the development, deployment, and maintenance phases.

The major concerns in testing the security of Web Services are the lack of security testing standards and specifications. For a given service at the functional level, input manipulation, information disclosure, and DoS constitute the most common vulnerabilities against a service [45]. Testing strategies should emphasize testing for these vulnerabilities. Common defense techniques involve the use of strategies based on integrity and confidentiality to counter these threats.

The WSM framework allows developers to perform fault detection, configuration testing of security aspects of Web Service. Passive or active testing techniques can be used for fault detection. As stated before, the incorporation of WSM into Web Services infrastructure allows developers to concentrate on developing the services while letting the WSM handle the non-application specific context security tasks, manageability, and other aspects of the services.

The following sections provide an approach for testing security for Web Services. In this approach, the WSM plays an integral part, where layer separation between the service and its security is established.

14.6.1 Web Services Security Fault Model

Effective testing of Web Services security requires the development of a fault model covering all interaction aspects of the service and spanning all the Web Services layers that include UDDI, WSDL, and SOAP. The fault model should encompass the entire group of stakeholders as discussed in Section 14.2. The fault model is assumed to be part of the security component of the WSM. In effect, the model builds on the generic fault model for Web Services security as proposed in [45].

The fault model [45] must address threats to the UDDI that includes information disclosure, availability, DoS attacks, and unauthorized access. Attackers could use UDDI's published WSDL to obtain information about Web Services and use the information to carry out the attack [45]. Attackers could use scanning and parameter mutation techniques to search for unpublished backdoor capabilities of the services in order to gain unauthorized access to resources and data. Buffer overflow and other types of attacks can also be used.

The fault model should consider the effect of intermediaries on a client's messages. The presence of intermediaries introduces threats to these messages. For example, an attacker may take over an intermediary and launch man-in-the-middle attacks. The attacker may redirect the messages to a different destination causing the equivalent of a DoS attack on the service [45].

At the service provider, integrator or certifier level, the fault model takes into consideration that Web Services could spread over multiple tiers [45]. Services on these tiers could be exposed as Web Services. Exposed Web Services could interact with infrastructure components that include mail servers, application servers, file systems, and various databases [45]. Web Services security may be affected if any of these infrastructure components are compromised. The advent of Web Services has the effect of forcing developers to re-think which components of a system should be trusted or which components should be considered vulnerable.

In this work, XML firewalls are viewed as an integral component of the Web Services fault model. The XML firewall can perform deep inspection of the messages with the ability to inspect data for XML conformance and exploits. XML firewalls can protect against attacks that do not require application context. In Fig. 14.8, the extended model from [45] is depicted.

14.6.2 General Testability Guidelines

An application or service may consist of a single functional component or multiple sets of local or networked components. Security is a multifaceted process consisting of mechanisms that cover network security elements, application level security systems, authentication systems, and cryptography systems. In a layered security approach, these mechanisms are developed independently at different OSI layers and are expected to be combined together to secure the deployed services in a useful manner.

For all phases of the application development life-cycle, it is important to identify security-related threats and vulnerabilities. This requires that developers embrace the use of systematic security design methodologies with well thought-out implementation processes.

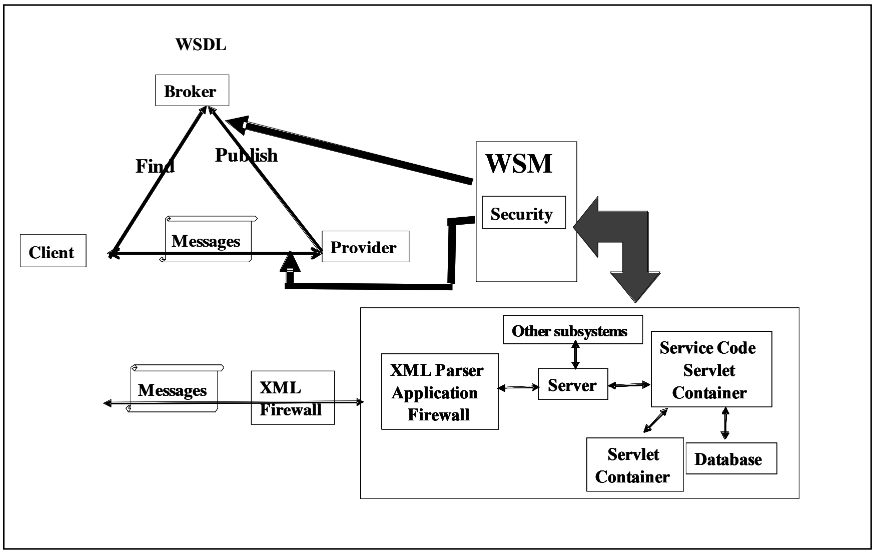

Fig. 14.8. Fault model

Architects should ensure that all aspects of application security are considered at early design stage in a structured manner. Best practices for applying security should be put in place before starting an application design process. It is important to be proactive in checking and verifying the security design for risks, tradeoffs, security policies, and defensive strategies ahead of the completion of the application design phase. During the deployment and production phases, it is good practice to adopt reactive security measures to ensure service stability and recovery in the event of a security breach.

Organizations can minimize the effort of testing for security of the developed applications by following strategies that reduce the factors that need to be tested. A crude but effective approach for minimizing the scope of testing is to pursue some of the following steps:

1. Create a set of use case scenarios that can accommodate the majority of services to be exposed as Web Services.
2. Determine the security boundaries of these services. Identify which services are internal and which are external.
3. Determine the overall security requirements of the service, including threats, risks, and vulnerabilities (internal and external).
4. Determine the set of messages to be exchanged by each service.
5. Determine the security requirements of each message. This can vary, depending on whether the message is internal or external.
6. Determine the resources that are required or can be accessed by each service and the type of access mechanism allowed.
7. Take a close look at the organization current network infrastructure and determine what is currently available to support the security requirements for these services. It is preferable to try to re-use existing security infrastructure (such as LDAP directories or PKI systems [10]) to support the security requirements of the services.
8. Determine if any specific applications must be either developed in house, out-sourced, or purchased to fulfill the security needs or other functionality of the new Web Services.
9. Determine the impact of the new services on the management, auditing, and logging facilities in the network and the applications.
10. Take a close look at the organization's security policy and integrate the new requirements to it.
11. Build the new services using secure code practices and standard-based technologies. Developers need to be conservative in the use of features in this step. Developers need to identify the minimum set of capabilities that would be specified to meet the security requirements so far identified. Minimizing the extent of supported services from SOAP message security reduces the testing scope and reduces overall vulnerabilities.

Developers can generate test suits for testing the new security mechanisms for the developed services. If the use case scenarios were broadly chosen then they should be able to incorporate new services where developers can re-use the test patterns. However, developers need to understand that using regular Web testing tools is not appropriate for testing Web Services. Web Services testing requires understanding of the unique security issues related to them, including XML, SOAP, WSDLs, and other WS standards.

14.6.3 Testing Strategies for Web Services Security

Testing strategies should conduct forcing errors tests to ensure that the error messages that are returned by the service do not reveal information about the service [45]. Testing for man-in-the-middle attacks should be used in the event that intermediaries are expected to be in the data path. Data origin authentication techniques can be used to remedy this threat. Authentication bypass tests should be conducted to ensure that only authorized requests are processed by the service.

At the UDDI, testing should be constructed that includes WSDL scanning and parameter tampering to detect vulnerabilities in the exposed service. Mutation tests techniques can be used to test for parameter tampering. Buffer overflow tests can also be used in this step. Tests should also include sanity checks on the UDDI to ensure that hackers cannot access services that should not be made public. These tests are similar to file or directory traversal attacks in web applications [45]. WSDL scanning tests must be conducted to ensure that hackers cannot access unpublished transactional methods by playing on variations of the published ones. It is really a bad practice when developers provide unpublished methods as a backdoor technique for invoking the service by insiders. This practice leaves the service vulnerable to the persistent attacker.

The above-mentioned testing steps need to be repeated if configuration of the system or the security mechanism is changed. For this reason, tests should be repeated if the system configuration is changed [45]. Testing cannot guarantee an error-prone implementation. Testing is used to increase the level of confidence that the service will operate according to its design objectives. Test oracles should be saved and used in regression testing for the modified service.

14.6.4 Testing Strategies for Web Services Application Data

This section addresses functional testing strategies for Web Services security from the developer's point of view, with a focus on testing security related to message data passed to applications. The testing strategy aims at addressing the vulnerabilities as specified by the fault model of the previous section. The aim is to perform tests that involve application context–type attacks. Some examples of these types of attacks are given next, from [45].

1. Cross-site scripting: In this type of attack, the hacker embeds a script into an XML document. The aim here is that the script will be stored (for example, in a database) and then served to an unsuspecting client

Web browser. The script can then execute in the client's browser and can perform tasks on the behalf of the attacker. For example, the script can steal sensitive information such as credit card numbers or passwords from the unsuspecting client. Variation on this type of attack occurs when a hacker embeds in an XML document a script, such as a shell script. The attacker hopes that the script will be executed on the targeted system. If the code executes, the attacker can perform unauthorized operation on the compromised system. Possible counter measures include proper data parsing and validation and to scan for all possible attack patterns and the use of application layer countermeasures, such as Web application firewalls.

2. XPath exploits: This is a form of XML injection attack. In this type of attack, a hacker aiming for illegally accessing data from a database injects malicious input into an XML document. The attacker aims to get the data to be part of a dynamically created XPath query against an XML document in a native XML database. An example of malicious input for XPath exploits is OR 1=1. This expression, when executed in the content of an XML document will always be true and can return data to the attacker. Possible counter measures include proper data parsing and validation and to scan for all possible attack patterns and the use of application layer countermeasures, such as Web application firewalls.
3. SQL injection exploits: In this attack, a hacker injects malicious input disguised as data into an application via an XML document or Web form, with the hope that the input will end up in a WHERE clause of a SQL query that is executed against a backend database. The hackers hope to gain access to data in an unauthorized fashion. The main vulnerability that enables Web Service enabled databases to be attacked in this fashion is the insecure practice of configuring the backend systems to accept and execute valid SQL queries received from any user with the necessary access privileges. Possible counter measures include proper data parsing and validation and to scan for all possible attack patterns and the use of application layer countermeasures, such as Web application firewalls.
4. Buffer overflow exploits: These exploits are targeted atWeb Service components that store input data in memory. These attacks succeed when the Web Service component does not adequately check the size to ensure that it is not larger than the allocated memory buffer that will receive it. Possible counter measures include the use of programming languages that perform input validation such as JAVA. The employment of appropriate memory management techniques that protect memory segments that are allocated for code form data overwrite can also be used as a counter measure to this threat.

Developing remedies for the above threats requires the practice of safe coding technique and the training of the developers in safe code practice. In some cases, there will be a need to have the code inspected by independent security professionals to ensure that the code can pass tests performed to detect these threats. Buffer overflow attacks usually target endpoints [45]. Tests must be conducted to ensure that the endpoint is capable of filtering out large data loads. Hence, testing with large data load must be conducted by developers to gain confidence that the service will survive such attacks.

Mutation-based test techniques can play an important role in detecting vulnerabilities in the code for threats 1 to 3. Mutation test strategies change, or mutate, inputs to the Web Service under test. By applying these changes to input messages, testers can check whether these mutations produce observable effects on the service outputs. Based on the observed behavior, faults in the service can be detected and the offending code can be fixed. The test oracles must be saved and then used to perform regression testing when any modifications have been performed on the service. Testing for script injection exploits may require the identification of the operating system commands in the language that is used to implement the system. These commands can then be imbedded in the validation tests.

14.6.5 Securing an Application: Case Study

To illustrate some of the points of the above-stated approach, consider a fictitious book selling company that has stores nation wide. The company has two warehouses for storing book supplies. For a given book, the company will contact the supplier to re-order copies if a minimum threshold is reached. For the purpose of this example,the company deals with only one supplier. The company needs to develop a web application based on a Web Service to be used by the store employees to query the warehouses for the availability of a given book. The company will use a Web Service to re-order a book once the threshold is met. This example is based on the same concepts of use-case scenarios developed in WS-I to illustrate usage of the BSP profile [11].

Case Study Functional Overview

An employee uses a web-based application that invokes a Web Service to interact with the retailer application. For simplicity, the retailer service manages stock in the two warehouses (Fig. 14.9). If Warehouse A cannot fulfill an order, the retailer service checks Warehouse B. When a warehouse's inventory of a book falls below a defined threshold, the retailer service orders more books from the supplier. The example consists of the following:

1. Client Web Service: A web-based application that provides a user interface. The web client application invokes the client Web Service to get catalogue information and submit an order for a book to the retail service. It also sends to the Retail service a one-way store update statement frequently.

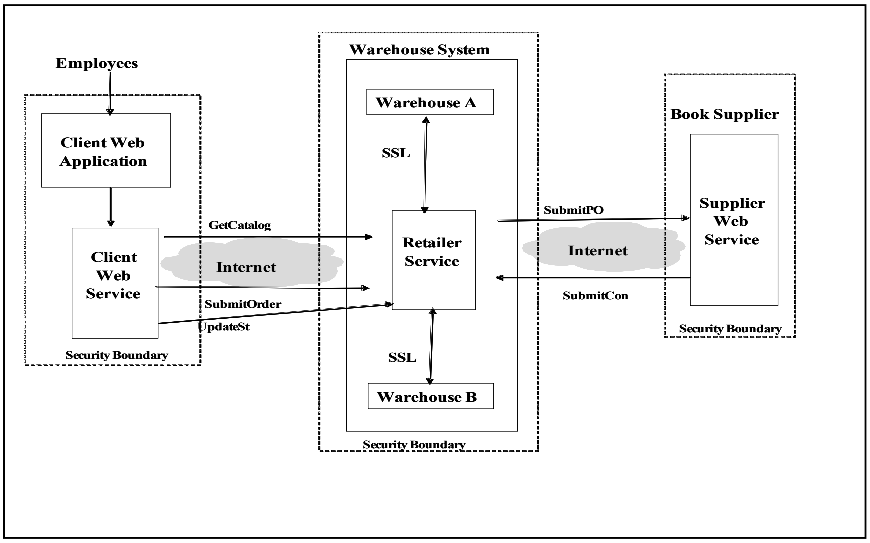

Fig. 14.9. Book service functional overview

2. Retailer service: A service invoking a Web Service that interacts with the warehouse to determine the availability of book and the time to order based on a given threshold.
3. Supplier service: An application that invokes a Web Service for accepting purchase orders and provides callback functionality when the order is fulfilled or an error occurs.
4. At a store, employees use the Web Client Application to view and place orders for available books. A standard web browser that supports SSL is used. Employees are authenticated using a user ID and password. The system does not have certificates that could be used for authentication.

The company has existing X.509 certificate security capabilities. The company uses the Internet for connecting the stores to the retail application and to communicate with the supplier. The company has a dedicated communication service with the warehouses and uses SSL for extra security. The company would like to use SOAP message layer security for securing the interactions.

SOAP Messages Usage Patterns

The use case scenario employs three usage patterns as follows:

1. One-way: Request messages are sent to a Web Service that does not issue a corresponding response. For example, the store update message that is sent to the retail service is a one-way exchange.

2. Synchronous request/response: A SOAP request elicits a SOAP response.
3. Basic callback: A set of paired request/response messages to enable asynchronous operation. The interchange between a retail service and the supplier requires a callback pattern since the supplier cannot instantly respond to the retail service request. The conventions used for callbacks can vary. In this fictitious example, the following sequence of events takes place:

- In an initial synchronous exchange, the retail service sends a purchase order. The supplier validates the order and sends back an acknowledgment.
- In a follow-up exchange between the supplier and the retail callback service, the supplier ships the goods and sends a shipping notice to the retail service. The retail service then sends back an acknowledgment.

Security Requirements

For each of the systems and operations of the use case scenario, the security requirements are specified for message integrity, authentication, and confidentiality.

Message integrity. Message integrity is needed to ensure that messages have not been altered in transit. For simplicity, attachments are not considered. In order to support verification of message integrity, messages are signed. In order to improve on processing speed, digest values are first calculated, and then these values are signed. Developers need to determine which elements of the messages need a signature. For the case under consideration, some or all of the following parts may need to be signed:

- UsernameToken: The `wsse:UsernameToken` element in the WS Security header containing the identity of the user who originally made the purchase request.
- Timestamp: The `wsu:Timestamp` element added to the message when it was created as defined in [12].
- Any custom SOAP headers such as a Start header that contains a conversation ID element and a callback location element. The conversation ID is provided by the Retailer to the Supplier so that the Supplier can include it in the Callback header responses asynchronously.
- The Callback custom header which keeps the conversation ID apart from the Start header.
- SOAP Body: The part of the SOAP message (e.g., soap:Body) that contains the exchanged document (such as a purchase order).

Message integrity is implemented by creating a digital signature using the sender's private key over the elements that need to be signed. To protect against dictionary attacks on plain text signature, the signature is encrypted, meaning that a `xenc:EncryptedData` element replaces this `ds:Signature` element in the message. Note that only the children of each element are used by the signing algorithm. The element itself is not signed.

Authentication. Authentication is performed to allow the receiver to establish the message origin. It is a good practice for the recipient of a message to authenticate the sender of a message. This is done by first checking that the signed data in the message has been signed using the public certificate whose private key was used to sign the message for message integrity purposes and then checking the credentials in that public certificate to determine the identity of the sender. If the sender includes a `wsse:BinarySecurityToken` in the `wsse:Security` header, the token contains the X.509 signing certificate.

The recipient should verify that it can trust the certificate issuer, and may also need to compare the data in the content of the message that identifies the sender, either in the SOAP header or in the payload, with the identity as stated in the public certificate.

The identity of the original user may also be included, in a UsernameToken. If the username token is not used for authentication, a password is not required.

Confidentiality. Confidentiality is required to conceal sensitive information in messages. Not all parts of messages are necessarily sensitive, and in some cases a message may not be considered sensitive at all, and thus there may be no need for confidentiality. In this example, parts of the message that are considered sensitive include the following:

- The SOAP Body since it could contain information such as order data, which could aid competitors.
- The Signature element since in some cases the body of the message can contain predictable variations, making it subject to guessing or dictionary-based attacks. Encrypting the signature can prevent such attacks.
- Custom headers such as the Start Header since it include the location of the callback service.

Confidentiality is implemented by first deriving the `xenc:EncryptedData` elements with the appropriate encryption algorithm and using the appropriate public key. The `xenc:EncryptedKey` element is encrypted using a chosen encryption algorithm. The `xenc:EncryptedKey` element will contain a security token reference to the public key information of the X.509 certificate used for encrypting, along with DataReferences to the `xenc:EncryptedData` elements. In this scenario the certificate itself is not included since it is assumed to be already public. For the Soap Body and the Start Header elements, only the children of the elements are encrypted. For the Signature element, the whole element is encrypted.

14.7 Crash-Only Web Services Design

In an SOA [53] environment, new Web Services are typically built by orchestrating underlying services. In mission-critical applications, it is necessary for both the underlying services and the composite service to advertise their failure models. In general, failure models are complex and difficult to combine but this section argues that the “crash-only software” architecture [49, 50, 51] provides not only a simplifying coherence for designers, but also a paradigm whose characteristics align particularly well with those of Web Services.

Within a Service-Oriented Architecture (SOA), new services are typically created by orchestrating underlying services. Figure. 14.10 illustrates a particularly simple case when two underlying services, X and Y, are orchestrated in some way to produce a new service, Z. In the general case, X and Y will not be owned by the developer of Z, being services exposed by other service providers.

To determine Z’s characteristics, so that WSLA guarantees can be offered to customers, it is necessary to combine the characteristics of X and Y with those of the additional logic provided by Z. To enable this, the characteristics of X and Y must be known and although the SOA specifications make provision for X and Y to advertise their interface syntax, their behavior and their contracts, no formal method has been proposed for defining many of the behavioral characteristics. For example, the performance, scaling, management, security, privacy, availability, reliability, and many other models of X and Y need to be known by the developer of Z in order to determine the service-level agreements for the corresponding characteristics of Z. As a simple example, consider privacy: X, Y, and Z may be implemented in different countries with differing laws regarding privacy and security of data. For the developer of Z, to ensure that the service complies with the local regulations and to be able to offer reassurances to customers of Z about the privacy of their data, the privacy policies of X and Y need to be available.

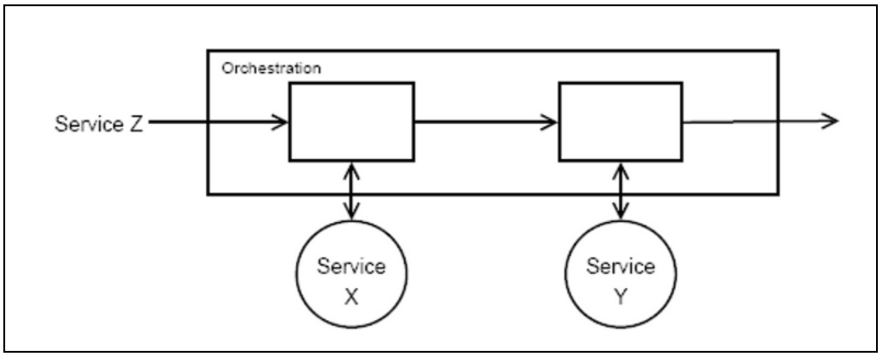

Fig. 14.10. Service orchestration

Each of the models listed above, and others, are needed but this section addresses one particular model: the failure model. If the failure model of Z is to be understood, the failure modes of X and Y have to be known. It may be, e.g., that X supports a transactional model and rolls back its input following a failure, guaranteeing that it returns to a sane state, putting the responsibility for re-submission of inputs onto the consumer (Z). Y, on the other hand, may buffer information and the precise state of an interaction may be difficult to determine when a failure occurs.

To permit Z to determine necessary actions following the failure of X or Y and to allow it to make claims about its own failure modes, a failure ontology is required that could capture X's and Y's (and Z's) failure semantics. This section argues that the technique of "crash-only software" [50] is particularly suited to the loosely coupled environment of SOAs, providing particularly simple behavior that can be described and advertised in a formal manner. It is unrealistic to expect all services to comply with this failure paradigm but this work proposes that it forms the basis of the failure semantics for Web Services.

14.7.1 Crash-Only Software

Studies (dating back to 1986 see [52]) support the view that failures in deployed software are mainly caused by Heisenbugs [52], bugs caused by subtle timing interactions between threads and tasks which are impervious to conventional debugging, being non-reproducible and sensitive to tracing and other observation. Reproducible bugs, the so-called "Bohrbugs," are easier to detect and fix during development and can largely be removed before shipment of a final product.

It must be accepted that, in any software-based system, if Heisenbugs exist then failures will occur. When they do, telecommunications and other high-availability systems have been built to detect the failure, save state, shut down the offending task and any other affected components (defined by some form of failure tree) down gracefully, take whatever recovery action is required and then restart the affected components.

The technique of crash-only argues that this is not only unnecessary but, in many cases, counter-productive. Consumers of the services, it is argued, must anticipate that their provider will, from time to time, crash cleanly without the opportunity for sophisticated failure handling (perhaps because of a power failure to the computer running the provider). Consumers must therefore already have the capability of handling such a crash. If this is the case, then consumers can always crash the component whenever any failure occurs.

This crash-only semantic has several advantages:

- It defines simpler macroscopic behavior with fewer externally visible states.
- It reduces the outage time of the provider by removing all shutting-down time.

- It simplifies the failure model significantly by reducing the size of the recovery state table. In particular, crashing can be invoked from outside the software of the provider. Recovery from a failed state is notoriously difficult and the crash-only paradigm coerces the system into a known state without attempting recovery, reducing substantially the complexity of the provider code.
- It simplifies testing by reducing the failure combinations that have to be verified.

If software is to crash cleanly more often, then it should also be written in such a way as to reload quickly [51].

14.7.2 Crash-Only Software and Web Services

Candea in [50] lists the properties required of a crash-only system and these can be abstracted remarkably well to match those of Web Services as described in [53]:

- Components have externally enforced boundaries. This is an implementation recommendation but one supported by the virtual machine concept used on many Web Service systems.
- All interactions between components have a timeout. This is implicit in any loosely coupled Web Services interaction.
- All resources are leased to the service rather than being permanently allocated. This is an implementation recommendation but clearly one which it is useful to follow in any implementation, particularly a Web Services one.
- Requests are entirely self-describing. For crash-only services, this requires that the request carries information about idempotency and time-to-live. The work in [50] maps this request to a REST[1]-like [54] environment but the comments are equally applicable to a true SOAP-defined Web Service.
- All important non-volatile state is managed by dedicated state stores.

In Section 14.2 of this chapter, in effect, the WSM performs runtime governance. The WSM is enabled in an SOA environment by having access to the service description of the invoked service and being in a position to intercept and decode all incoming requests.

Deliberately induced crashes are a useful technique for software rejuvenation (see [55]) and this requires detection of periods of low usage of the service. Runtime governance is an obvious candidate for recognizing such periods and causing the restarts.

The description of crash-only software in [50] assumes, when recast using SOA terminology, that the providers (X and Y in Fig.14.10) will exhibit crash-only failure behavior but that consumers, having failed to obtain timely

[1] Representational State Transfer.

or correct service, can initiate the crash. This is acceptable only when the consumer and provider, although loosely coupled, are within one trust domain. This is clearly not generally the situation with Web Services.

One common function of the WSM software layer is the monitoring of response times from the service to ensure that the consumer is getting the level of service paid for. This provides the perfect location for invocation of the power-off switch provided by crash-only software that switch is external to the service, relying in no way on continued correct operation of the service code, and its operation is idempotent, ensuring that the decision to kill the server does not require knowledge of internal state.

Crash-only design principles can be used as a starting point in the design of Web Services (we term them crash-only Web Services). In this aspect, the service can be designed in such a fashion that the state of the service that identifies critical information is always stored in the system even in the event of a crash. The same crash-only design principles are extended at the service level whereby, e.g., in business process interaction, information such as the status of an order is stored in a non-volatile state [50]. Tree techniques as defined in [49] and [50] can be used to identify the data components from a service that should be stored and be available when the service is crashed.

The use of crash-only systems combined with crash-only Web Services in combination with a reliable SOAP stack can enhance on the availability of a Web Service and reduces the complexity of its testing. Crash-only Web Services can be re-started quickly and with a known state. In [56] a SOAP reliable transport protocol is described (WSRM). The protocol allows a reliability agent to acknowledge the receipt of SOAP messages to the Web Services consumer. Reliability in WSRM is used to ensure that the messages are delivered to the targeted server (application server). The reliability agent can be implemented at the Web Services end point in Fig. 14.11.

For systems with hardware redundancy, by using crash-only techniques, SOAP WSRM can be extended in order to produce an always available Web Service (although at reduced WSLA if and when a service is forced to crash) from the provider's and consumer's point of view. The architecture is depicted in Fig. 14.11. Here, the components of the system are designed using crash-only, which means that re-booting is fast and reliable. The Web Services end point is used as the gateway between the Web Services consumer and provider. At runtime, the system stores all of its important data innonvolatile states. The WSRM agent acknowledges the receipt of the SOAP messages to the consumer only after a confirmation is received from the system that the important data is safely stored in the system. The recovery agent monitors the operation of the Web Services. If the agent determines that the Web Service is misbehaving (due to fault in the code or any other reason, actually the cause need not be known or investigated), then the agent will instruct the stall proxy to delay the acknowledgment of the SOAP messages to the consumer. The stall proxy will basically ensure that the session is

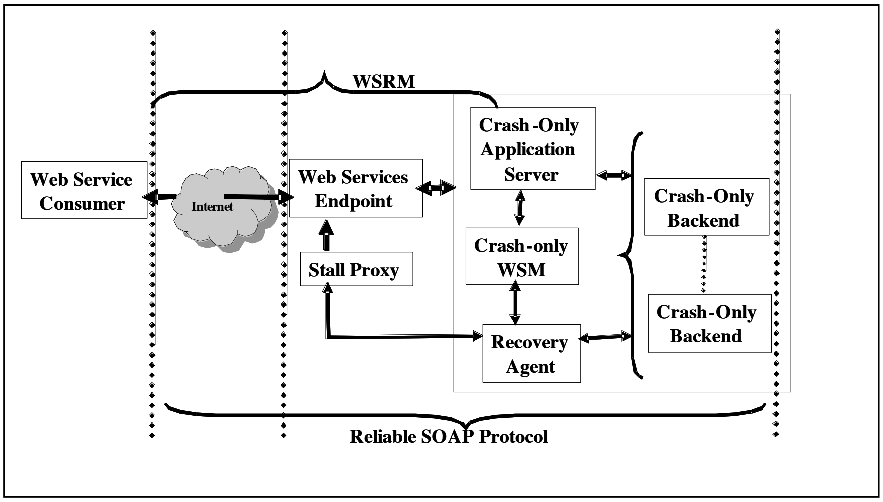

Fig. 14.11. Architecture of crash-only reliable Web Services

kept alive. The system is re-started (multiple sub-component reboots may be needed see [49] and [50] for details). When the system is back up again, the WSRM agent can request the transmission of the last set of lost messages from the consumer.

The above approach provides for the capability of extending the SOAP WSRM protocol to enhance on the availability and reliability at the service level. Testing scope is also minimized.

14.8 Research Proposals and Open Research Issues

Despite such intense research and development efforts, current Web Service technology does not yet provide the flexibility needed to "tailor" a Web Service according to preferences of the requesting clients. At the same time, Web Service providers demand enhanced adaptation capabilities in order to adapt the provisioning of a Web Service to dynamic changes of the Web Service environment according to their own policies. Altogether, these two requirements call for policy-driven access controls model and mechanisms, extended with negotiation capabilities. Models and languages to specify access and management control policies have been widely investigated in the area of distributed systems [3].

Standardization bodies have also proposed policy-driven standard access control models [1]. The main goals of such models are to separate the access control mechanism from the applications and to make the access control mechanism itself easily configurable according to different, easily deployable access control policies. The characteristics of open web environments, in which the interacting parties are mostly unknown to each other, have lead to the development of the trust negotiation approach as a suitable access control model for this environment [4, 5].

Trust negotiation itself has been extended with adaptive access control, in order to adapt the system to dynamically changing security conditions. Automated negotiation is also being actively investigated in different application domains, such as e-business. However, a common key requirement that has been highlighted is the need of a flexible negotiation approach that enables the system to dynamically adapt to changing conditions. In addition, the integration of trust negotiation techniques with Semantic Web technologies, such as semantic annotations and rule-oriented access control policies, has been proposed. In such approaches, the resource under the control of the access control policy is an item on the Semantic Web, with its salient properties represented as RDF properties. RDF metadata, managed as facts in logic programming, are associated with a resource and are used to determine which policies are applicable to the resource. When extending a Web Service with negotiation capabilities, the invocation of a Web Service has to be managed as the last step of a conversation between the client and the Web Service itself. The rules for such a conversation are defined by the negotiation protocol itself. Such a negotiation protocol should be described and made publicly available in a similar way as a Web Service operation is publicly described through WSDL declarations. An XML-based, machine-processable negotiation protocol description allows an electronic agent to automatically generate the messages needed to interact with the Web Service.

The client and the Web Service must be equipped with a negotiation engine that evaluates the incoming messages, takes decisions, and generates the outgoing messages according to the agreed upon protocol. The models already proposed for peer-to-peer negotiations assume that both parties are equipped with the same negotiation engine that implements the mutually understood negotiation protocol. This assumption might not, however, be realistic and may prevent the wide adoption of negotiation-enhanced, access-control model and mechanisms.

In the remainder of this section, we present a short overview of a system, addressing those requirements, and then we discuss open research issues.

14.8.1 Ws-AC1: An Adaptive Access Control System for Web Services

In order to address the adaptation and negotiation requirements, we propose the use of a system that supports Web Service access control model and an associated negotiation protocol as given in [6]. The proposed model, referred to as Web Service Access Control Version 1 (Ws-AC1, for short) is based on a declarative and highly expressive access control policy language.

Such language allows one to specify authorizations containing conditions and constraints not only against the Web Service parameters but also against the identity attributes of the party requesting the service and context parameters that can be bound, e.g., to a monitor of the Web Service operating environment. An additional distinguishing feature of Ws-AC1 is the range of object-protection granularity it supports. Under Ws-AC1 the Web Service security administrator can specify several access control policies for the same service, each one characterized by different constraints for the service parameters, or can specify a single policy that applies to all the services in a set. In order to support such granularity, we introduce the notion of service class to group Web Services. To the best of our knowledge, Ws-AC1 is the first access-control model developed specifically for Web Services characterized by articulated negotiation capabilities. A model like Ws-AC1 has important applications, especially when dealing with privacy of identity information of users and with dynamic application environments. In order to represent the negotiation protocol, an extension to the Web Services Description Language standard has also been developed.

The main reason of that choice is that, although the Web Services Choreography Description Language (WS-CDL) is the emerging standard for representing Web Services interactions, WS-CDL is oriented to support a more complex composition of Web Services in the context of a business process involving multiple parties.

Ws-AC1 is an implementation-independent, attribute-based, access-control model for Web Services, providing mechanisms for negotiation of service parameters. InWs-AC1 the requesting agents (also referred to as subjects) are entities (human being or software agents). Subjects are identified by means of identity attributes qualifying them, such as name, birth date, credit card number, and passport number.Identity attributes are disclosed within access requests invoking the desired service. Access requests to a Web Service (also referred to as provider agent) are evaluated with respect to access control policies. Note that in its initial definition,Ws-AC1 does not distinguish between the Web Service and the different operations it provides, i.e., it assumes that a Web Service provides just a single operation. Such a model can be applied to the various operations provided by a Web Service without any extension. Access control policies are defined in terms of the identity attributes of the requesting agent and the set of allowed service parameters values. Both identity attributes and service parameters are further differentiated in mandatory and optional ones. For privacy and security purposes, access control policies are not published along with the service description but are internal to the Ws-AC1 system. Ws-AC1 also allows one to specify multiple policies at different levels of granularity. It is possible to associate fine-grained policies with a specific service as well with several services. To this end, it is possible to group different services in one or more classes and to specify policies referring to a specific service class, thus reducing the number of policies that need to be specified by a policy administrator. A policy for a class of services is then applied to all the services of that class, unless policies associated with the specific service(s) are defined.

Moreover, in order to adapt the provision of the service to dynamically changing conditions, the Ws-AC1 policy language allows one to specify constraints, dynamically evaluated, over a set of environment variables, referred to as context, as well as over service parameters. The context is associated with a specific service implementation and it might consist of monitored system variables, such as the system load. The access control process of Ws-AC1 is organized into two main sequential phases. The first phase deals with the identification of the subject requesting the service. The second phase, executed only if the identification succeeds, verifies the service parameters specified by the requesting agent against the authorized service parameters.

The identification phase is adaptive, in that the provider agent might eventually require the requesting agent to submit additional identity attributes in addition to those originally submitted. Such an approach allows the provider agent to adapt the service provisioning to dynamic situations;for example, after a security attack, the provider agent might require additional identity attributes to the requesting agents. In addition, to enhance the flexibility of access control, the service parameter verification phase can trigger a negotiation process. The purpose of this process is to drive the requesting agent toward the specification of an access request compliant with the service specification and policies. The negotiation consists in an exchange of messages between the two negotiating entities in order to limit, fix, or propose the authorized parameters the requesting agent may use. The negotiation of service parameters allows the provider agent to tailor the service provisioning to the requesting agent preferences or, at the opposite, to enforce its own preferred service provisioning conditions.

14.8.2 Open Research Issues

Even though Ws-Ac1 provides an initial solution to the problem of adaptive access control mechanisms for Web Services, many issues need to be investigated. A first issue is related to the development of models and mechanisms supporting a comprehensive characterization of Web Services that we refer to as Web Service profiles. Such a characterization should be far more expressive than conventional approaches, like those based on UDDI registries or OWL. The use of such profiles would allow one to specify more expressive policies, taking into account various features on Web Services, and to better support adaptation.

The second issue is related to taking into account the conversational nature of Web Services, according to which interacting with real world Web Services involves generally a sequence of invocations of several of their operations, referred to as conversation. Most proposed approaches, like Ws-AC1, assume a single-operation model where operations are independent from each other.

Access control is either enforced at the level of the entire Web Service or at the level of single operations. In the first approach, the Web Service could ask, in advance, the client to provide all the credentials associated with all operations of that Web Service. This approach guarantees that a subject will always arrive at the end of whichever conversation. However, it has the drawback that the subject will become aware of all policies on the base of which access control is enforced. In several cases, policies need to be maintained confidentially and disclosed only upon some initial verification of the identity of the client has been made. Another drawback is that the client may have to submit more credentials than needed. An alternative strategy is to require only the credentials associated with the next operation that the client wants to perform. This strategy has the advantage of asking from the subject only the credentials necessary to gain access to the requested operation. However, the subject is continuously solicited to provide credentials for each transition. In addition, after several steps, the client may reach a state where it cannot progress because of the lack of credentials. It is thus important to devise strategies to balance the confidentiality of the policies with the maximization of the service completion. A preliminary approach to such strategies has been recently developed [2]; the approach is based on the notion of k-trustworthiness where k can be seen as the level of trust that a Web Service has on a client at any point of their interaction. The greater the level of trust associated with a client, the greater is the amount of information about access control policies that can be disclosed to this client, thus allowing the client to determine early in the conversation process if it has all necessary credentials to satisfy the access control policies. Such approach needs, however, to be extended by integrating it with an exception-based mechanism tailored to support access control enforcement. In particular, in a step-by-step approach, whenever a client cannot complete a conversation because of the lack of authorization, some alternative actions and operations are taken by the Web Service.

A typical action would be to suspend the execution of the conversation, ask the user to acquire the missing credentials, and then resume the execution of the conversation; such a process would require investigating a number of issues, such as determining the state information that need to be maintained, and whether revalidation of previous authorizations is needed when resuming the execution.

A different action would be to determine whether alternative operations can be performed to replace the operation that the user cannot execute because of the missing authorization. We would like to develop a language according to which one can express the proper handling of error situations arising from the lack of authorization.

The third issue is related to security in the context of composite services; in such a case, a client may be willing to share its credentials or other sensitive information with a service but not with other services that can be invoked by the initial service. To handle such requirement, different solutions may be adopted, such as declaring the possible services that may be invoked by the initial service or associating privacy policies with the service description, so that a client can specify its own privacy preferences. Other relevant issues concern workflow systems. Such systems represent an important technology supporting the deployment of business processes consisting of several Web Services and their security is thus crucial. Even though some initial solutions have been proposed, such as the extension of the WS-BPEL [9] standards with role-based access control [7], more comprehensive solutions are required, supporting adaptive access control and sophisticated access-control constraints.

Finally, the problem of secure access to all information needed to use services, such as information stored by UDDI registries, needs to be addressed. To date, solutions have been developed to address the problem of integrity through the use of authenticated data structures [8]. However, work is needed to address the problem of suitable access control techniques to assure the confidentiality and privacy of such information in order to support its selective sharing among multiple parties.

14.9 Conclusion

Testing Web Services and security in an SOA environment is a discipline that is still in its infancy. Experience gained from Web Development can be used as a guiding principle for the development of testing strategies in the SOA world at large. There are still many open areas that still need to be worked on. For example, there are no standard mechanisms to share management information between the various service providers. Faults in the Web Services stack are more centered toward the SOAP message level. Current standards are not designed with fault management in mind. Regression tests need enhancement, coverage, and speed improvement to be able to cope with the testing scope of composite services.

References

1. OASIS eXtensible Access Control Markup Language 2 (XACML) Version 2.0 OASIS Standard, 1 Feb 2005.
2. M. Mecella, M. Ouzzani, F. Paci, E. Bertino. Access Control Enforcement for Conversational-based Services. *Proceedings of 2006 WWW Conference*, Edimburgh, Scotland, May 23-26, 2006.
3. N. Damianou ,N. Dulay, E. Lupu and M. Sloman. The Ponder Policy Specification Language. *Proceedings of the 2nd IEEE International Workshop on Policies for Distributed Systems and Networks*, 2001.
4. T. Yu, M. Winslett, K. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. *ACM Transactions on Information and System Security*, Vol. 6, No. 1, February 2003.
5. E. Bertino, E. Ferrari, A.C. Squicciarini. X -TNL: An XML-based Language for Trust Negotiations. *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, 2003.

6. E.Bertino, A.C. Squicciarini, L.Martino, F. Paci. An Adaptive Access Control Model for Web Services. *International Journal of Web Service Research*, (3), 27-60 July-September 2006.
7. E.Bertino, B.Carminati, E.Ferrari. Merkle Tree Authentication in UDDI Registries. *International Journal of Web Service Research*, 1(2): 37-57(2004).
8. E.Bertino, J.Crampton, F.Paci. Access Control and Authorization Constraints for WS-BPEL. Submitted for publication.
9. OASIS Web Services Business Process Execution Language Version 2.0. Committee Specification, 31 January 2007
10. Schwarz, J, Bret Hartman B., Nadalin A, Kaler C., F. Hirsch, and Morrison S, , Security Challenges, Threats and Countermeasures Version 1.0, WS-I, May, 2005, http://www.ws-i.org/Profiles/BasicSecurity/SecurityChallenges-1.0.pdf
11. Barbir, A. Gudgin M and McIntosh M., , Basic Security Profile Version 1.0, WS-I, May 2005, http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0-2004-05-12.html
12. Nadalin, A., Kaler C., Hallam-Naker, P., Monzillo R., Web Services Security: SOAP Message Security 1.0, (WS-Security 2004), OASIS, March 2004, http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf
13. Web Services Security: SOAP Message Security 1.1, (WS-Security 2004), OASIS, February 2006, http://www.oasis-open.org/committees/download.php /16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf
14. Nadalin, A., Kaler C., Hallam-Naker, P., Monzillo R.,, Web Services Security: UsernameToken Profile 1.1,OASIS, February 2006, http://www.oasis-open.org/committees/download.php/16782/wss-v1.1-spec-os-Username Token-Profile.pdf
15. Nadalin, A., Kaler C., Hallam-Naker, P., Monzillo R., Web Services Security: X.509 Certificate Token Profile 1.1, OASIS, February 2006, http://www.oasis-open. org/committees/download. php/16785/wss-v1.1-spec-os-x509TokenProfile.pdf
16. Monzillo R., Kaler C., Nadalin A., Hallam-Naker, P.,., Web Services Security: SAML Token Profile 1.1, OASIS, February 2006, http://www.oasis-open. org/committees/download.php/16768/wss-v1.1-spec-os-SAMLTokenProfile.pdf
17. Nadalin, A., Kaler C., Hallam-Naker, P., Monzillo R.,, Web Services Security: Kerberos Token Profile 1.1, OASIS, February 2006, http://www.oasis-open.org/committees/download.php/16788/wss-v1.1-spec-os-KerberosTokenProfile.pdf
18. Monzillo R., Kaler C., Nadalin A., Hallam-Naker, P., Web Services Security: Rights Expression Language (REL) Token Profile 1.1, OASIS, February 2006, http://www.oasis-open.org/committees/download.php/16687/oasis-wss-rel-token-profile-1.1.pdf
19. Hirsch, F., Web Services Security: SOAP Messages with Attachments (SwA) Profile 1.1, OASIS, February 2006, http://www.oasis-open.org/committees/download.php/16672/wss-v1.1-spec-os-SwAProfile.pdf
20. Signature Syntax and Processing, W3C Recommendation February 2002, http://www.w3.org/TR/xmldsig-core/
21. XML Encryption Syntax and Processing, W3C Recommendation December 2002, http://www.w3.org/TR/xmlenc-core/
22. Nortel Unified Security Framework for corporate and government security, Nortel, http://www.nortel.com/solutions/security/collateral/nn104120-051705.pdf

23. SOAP Version 1.2 Part 1: Messaging Framework, W3C, June 2003, http://www.w3.org/TR/soap12-part1/
24. Simple Object Access Protocol (SOAP) 1.1, W3C Note, May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
25. Rescorla E., HTTP Over TLS, RFC 2818, May 2000.
26. Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl
27. Bloomberg, J., Schmelzer, R, Service Orient or Be Doomed!: How Service Orientation Will Change Your Business, SBN: 0-471-79224-1, Wiley, May 2006.
28. Boyer, J., Exclusive XML Canonicalization Version 1.0, W3C, July 2002, http://www.w3.org/TR/xml-exc-c14n/
29. Bray, T., Extensible Markup Language (XML) 1.0 (Third Edition), W3C, February 2004, http://www.w3.org/TR/REC-xml/
30. The Transport Layer Security (TLS) Protocol,Version 1.1, RFC 4346, April 2006.
31. Demchenko, Y.,, Web Services and Grid Security Vulnerabilities and Threats Analysis and Model, Grid Computing Workshop, 2005.
32. Nakamura, Y., Model-Driven Security Based on a Web Services Security Architecture, Proceedings of the 2005 IEEE International Conference on Services Computing (SCC'05), 2005.
33. Tarhini et al., Regression Testing Web Services-based Applications, Computer Systems and Applications, March 8, Page(s):163 - 170, 2006.
34. Benharref A. et al, A Web Service Based-Architecture for Detecting Faults in Web Services, IFIP/IEEE International Symposium on Integrated Network Management 2005.
35. Bhoj, P. , Management of new Federated Services, Integrated Network Management V., 1997.
36. Weiping He, Recovery in Web Service Applications, Proceedings of the 2004 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'04), 2004.
37. Papazoglou, M. and Heuvel, W., Web Services Management: A Survey, IEEE Internet Computing, November 2005.
38. Bertolino A. and Polini A., The Audition Framework for Testing Web Services Interoperability, Proceedings of the 2005 31st EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO-SEAA'05), 2005. 30. Karjoth, G., Service-oriented Assurance: Comprehensive Security by Explicit Assurances, Publications of the Network Security and Cryptography Group, 2005.
39. Tsai, W., Ray Paul R., Weiwei S. and Cao Z.,, Coyote: An XML-Based Framework for Web Services Testing, 7th IEEE International Symposium on High Assurance Systems Engineering (HASE'02), 2002.
40. Yuan Rao, Y., Feng, O, Han, J., and Li, Z.,, SX-RSRPM: A Security Integrated Model For Web Services, Proceedings of the Third International Conference on Machine Learning and Cybernetics, Shanghai, 26-29 August 2004.
41. Bruno, M., Gerardo, C., and Di Penta, M., Using Test Cases as Contract to Ensure Service Compliance across Releases, Proc. 3rd Int'l Conf. Service Oriented Computing (ICSOC 2005), LNCS 3826, Springer, 2005, pp. 87-100.
42. Tsai, W., Paul, R, Cao, Z., L. Yu, L., A. Saimi, A. and B. Xiao, B., . Verification of Web Services using an enhanced UDDI server. In Proc. of WORDS 2003, pages 131-138, Jan., 15-17 2003. Guadalajara, Mexico.

43. Tsai, W., Paul R., Wei S. and Cao Z. Scenario-based Web Service testing with distributed agents. IEICE Transaction on Information and System, E86-D(10):2130-2144, 2003.
44. Xu, W., Offutt, J., Juan Luo, J., Testing Web Services by XML Perturbation, Proceedings of the 16th IEEE International Symposium on Software Reliability Engineering (ISSRE'05), 2005.
45. Yu, W., Supthaweesuk, P., and Aravind, D. Trustworthy Web Services Based on Testing, Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05), 2005.
46. Mei H. and Zhang L., A Framework for Testing Web Services and Its Supporting Tool, Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE'05), 2005.
47. Canfora G. and Di Penta M., Testing Services and Service-Centric Systems: Challenges and Opportunities, IT Pro Published by the IEEE Computer Society, April 2006.
48. Zapthink, www.zapthink.org
49. Fox A. and D. Patterson D., When does fast recovery trump high reliability? In 2nd Workshop on Evaluating and Architecting Systems for Dependability (EASY), 2002.
50. Candea G. and A. Fox A., Crash-only software. In 9th Workshop on Hot Topics in Operating Systems, 2003.
51. Candea G. Et, Microreboot-a technique for cheap recovery. In Proceedings of the 6th Symposium on Operating Systems Design and Implementation, 2004.
52. Gray J., Why do computers stop and what can be done about it? In 5th Symposium on Reliability in Distributed Systems, 1986.
53. OASIS SOA Reference Model TC. Reference model for service-oriented architecture 1.0. Technical report, OASIS, 2006.
54. Fielding, R., Architectural Styles and the Design of Network-based Software Architectures. Ph.D. Dissertation. University Of California, Irvine, 2000.
55. K. Vaidyanathan, K. et al., Analysis and implementation of software rejuvenation in cluster systems. In SIGMETRICS '01: Proceedings of the 2001 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 62-71, New York, NY, USA, 2001. ACM Press.
56. OASIS (www.oasis-open.org) Web Services Reliable Exchange Technical Committe (WS-SX). 49. W3C (www.w3.org) WS-Policy WG.
57. IBM; The Enterprise Privacy Authorization Language (EPAL 1.1) - Reader's Guide to the Documentation.
58. OASIS eXtensible Access Control Markup Language 2 (XACML) Version 2.0 OASIS Standard, 1 Feb 2005.
59. T. Yu, M. Winslett, K. Seamons. Supporting Structured Credentials and Sensitive Policies through Interoperable Strategies for Automated Trust Negotiation. ACM Transactions on Information and System Security, Vol. 6, No. 1, February 2003.
60. OASIS (www.oasis-open.org) WS-BPEL TC.
61. Mecella, M., Ouzzani, M., Paci, F., Bertino, E. Access Control Enforcement for Conversation-based Web Services. Proceedings of the 2006 WWW Conference, Edinburgh, Scotland, May 23-26, 2006.
62. Bertino, E., Crampton J.,, and Paci F. Access Control and Authorization Constraints for WS-BPEL. Submitted for publication.

63. Bertino, E., B. Carminat, and E. Ferrari, E. Merkle Tree Authentication in UDDI Registries. International Journal of Web Service Research, 1(2): 37-57 (2004).
64. Liberty Alliance Project - Introduction to the Liberty Alliance Identity Architecture Revision 1.0 March, 2003