FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

SOA Security Policy Validation and Authoring

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur/in

im Rahmen des Studiums

Medizinische Informatik

eingereicht von

Stefan Prennschütz-Schützenau
Matrikelnummer 9917140

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer/in: Univ.Prof. Dr. Schahram Dustdar
Mitwirkung: Univ.-Ass. Mag. Philipp Leitner

Wien, February 1, 2011 _____ _____

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Abstract

This thesis examines the issues around security policy compliance validation and policy refinement in SOA environments, focusing on message-level security in the context of WS-Security, WS-Policy and WS-SecurityPolicy. Conformance against WS-I's *Basic Security Profile* (**BSP**) is explored as comprehensive case study of a SOA requirement whose adoption is a must for an expedient SOA. We engineer this target requirement using a novel model-driven security policy authoring approach that utilizes the Schematron validation standard. We leverage the *Schematron Validation Pipeline* (**SVP**) mechanism in a prototype for policy definition and selection, which is implemented as plug-in on the Eclipse Platform. The tool provides an interactive interface for *build-time policy validation* along a newly elaborated authoring model. We motivate and verify our methods using the BSP conformance requirement and its implementation, that is by its own useful, as WS-I compliance is a prerequisite for a secure and interoperable SOA. Additionally, we implemented *runtime BSP validation* of WS-Security SOAP message exchanges for cross-checking and evaluating our method of static policy validation against the dynamic message requirements of the BSP interoperability profile. The authoring model concentrates on finding different user roles for the tool and the right arrangement of the authoring process. It focuses on how to express complex architecture requirements (such as WS-I compliance) in the form of schematron validation rules on policy languages, such as WS-SecurityPolicy, and how this mapping can best be specified by the user. The matchmaking is realized as a *policy mediation* based on a prototypical architecture policy model and a subsequent selection of IT-level policy instances, escorted by an interactive SVP validation against that model. Schematrons are rule-based *syntactic* constraint schemata and lend themselves to an expressive means for aiding (semi-)automated services selection. We use it to support the search and validation of Web Services according to their non-functional properties, i.e. policies. Validation results produced by the SVP are fed back to the policy author, who may in turn make potential corrections to the policy in question or select different policy instances, that more accurately meet her requirements. Abstract schematron policies may be stored for reuse which allows also developers that are not necessarily domain experts to configure and use available Web Services security infrastructure by means of abstract policy templates that map to sets of concrete operational policies. Despite of the method using a custom abstract model, the schematron mediation is model-agnostic as well as system-independent, since Schematron is a standardized schema definition language.

Kurzfassung

Diese Arbeit erforscht Fragen der Validierung von Security Policy Compliance und der Verfeinerung von Policies in SOA Umgebungen, mit einem Fokus auf message-level Security im Konext von WS-Security, WS-Policy and WS-SecurityPolicy. Konformanz gegen WS-Is *Basic Security Profile* (**BSP**) wird als Fallstudie einer SOA Anforderung herangezogen, deren Durchsetzung in brauchbaren SOAs unabdingbar ist. Wir implementieren diese Anforderung mit Hilfe einer neuen modell-basierten Security Policy Entwicklungsmethode, bei der der Schematron Standard zum Einsatz kommt. Wir nutzen den *Schematron Validation Pipeline* (**SVP**) Mechanismus in einem Prototyp zur Auswahl und Definition von Policies, realisiert als Plugin auf der Eclipse Platform. Das Policy-Werkzeug stellt eine interaktive Schnittstelle für die Validierung von *build-time* Policies bereit. Das Modell und die Methode werden mit der BSP-Konformanz Anforderung und deren Implementierung motiviert und verifiziert, wobei die Implementierung nützlich an sich ist, da WS-I Konformanz eine Vorbedingung für ein sicheres und interoperables SOA darstellt. Zusäzlich implementierten wir eine *runtime* Validierung der BSP Konformanz in WS-Security SOAP Nachrichtenaustauch, zur Überprüfung und Evaluierung dieser statischen Policy Validierungsmethode für die dynamischen Anforderungen des BSP Interoperabilitäts-Profile. Das Entwicklungswerkzeug unterstützt verschiedene Benutzerrollen und legt den Schwerpunkt auf die richtige Ausrichtung des Prozesses des Policy Authoring. Im Fokus steht die Frage, wie komplexe Architekturanforderungen (so wie WS-I Konformanz) mit Schematron Regeln für die Validierung von Policies (wie WS-SecurityPolicy) ausgedrückt werden können, und wie der Benutzer diese Zuordungen am besten anzugegeben hat. Die Anpassung der Regeln erfolgt über eine *Policy Mediation* basierend auf einem prototypischen Policy Modell auf Architekturebene und einer Auswahl von IT Policies, die von einer interaktiven SVP Validierung gegen dieses Modell begleitet wird. Schematrons sind regelbasierte *syntaktische* Bedingungsschemata und bieten sich an als ausdrucksstarkes Mittel für die (semi-)automatische Auswahl von Services. Wir verwenden sie zur Unterstützung der Suche und Validierung von Web Services entsprechend ihrer nicht-funktionalen Eigenschaften, i.e. Policies. Validierungsergebnisse, die von der SVP produziert werden, gehen zurück zum Policy Autor, der potentielle Korrekturen an der in Frage kommenden Policy machen kann, oder aber eine andere Policy auswählt, die den Anforderungen genauer entspricht. Abstrakte Schematron Policies können zur Wiederverwendung gespeichert werden, was es auch Entwicklern die nicht unbedingt Experten in diesem Gebiet sind, erlaubt, die verfügbare Web Services Security Infrastruktur zu konfigurieren und zu benutzten: die abstrakten Policy Vorlagen können wiederverwendet werden und werden auf eine Menge von Policies auf Operationsebene abgebildet. Obwohl die Methode ein eigenes abstraktes Modell benützt, ist die Schematron Mediation modellagnostisch und systemunabhängig, da Schematron eine standardisierte Schemadefinitionssprache darstellt.

Acknowledgement

Special thanks are due to my family at this point, in particular to my mother and my father, whose support enabled me to take these studies. Thank you both for holding out so bravely until the end. Most notably I also thank you, Ines, that you have always stood by me and believed in me. In general, I thank many companions, who have accompanied me on my way at some occasion or the other - the ones addressed, know they are. I thank Prof. Schahram Dustar and Philipp Leitner for their supervision of this thesis and for their support in the course of the internship. Exceptionally, I would like to thank my manager at IBM, Paco, as well as my mentor, Nirmal, for their warm-hearted supervision during the internship. I have learned a lot, had great fun and gained a weighty experience. Thanks also to you for that, Alek, Axel, Dough and Math!

Danksagung

Besonderem Dank gebührt an dieser Stelle meiner Familie und ganz besonders meiner Mutter und meinem Vater, deren Unterstützung mir dieses Studium erst ermöglicht hat. Danke Euch beiden, dass Ihr bis zum Schluss so tapfer ausgeharrt habt. Vorallem Danke ich auch Dir, Ines, dass Du immer zu mir gehalten und an mich geglaubt hast. Generell danke ich vielen Weggefährten, die mich an der einen oder anderen Stelle auf diesem Weg begleitet haben - diejenigen die gemeint sind, wissen bescheid. Ich danke Prof. Schahram Dustdar und Philipp Leitner für die Betreung dieser Arbeit und für ihre Unterstützung im Zuge des Internships. Ganz besonders möchte ich meinem Manager bei IBM, Paco, danken, sowie meinem Mentor, Nirmal, für die warmherzige Betreuung während des Internships. Ich habe viel gelernt, hatte grossen Spass und bin um eine gewichtige Erfahrung reicher. Danke auch Euch dafür, Alek, Axel, Dough und Math!

Projects Background and Context

The projects outlined in this thesis were conducted during the author's six month internship at the IBM Thomas J. Watson Research Center. A generic validation engine already existed [135], which had been in use with the IBM WebSphere Services Registry and Repository (**WSRR**) for *Basic Profile* [103] (**BP**) conformance validation. The initial goal and the bases for the approach presented in this thesis, was to extend the engine to encompass also Basic Security Profile (**BSP**) conformance validation. The existent validation interface for the WSRR was created by a team from the IBM China Research Laboratories, and designed to work with static artifacts; Web Services Description Language (**WSDL**) documents in case of BP conformance. BSP however, speaks about very specific runtime (dynamic) WS-Security message requirements and thus is not innately amenable to static validation (in contrast to WSDL validation against BP). This is where our approach of static WS-SecurityPolicy validation emerged. In order to be compatible with the existent validation technology we decided to re-express BSP as static schematrons on WS-SecurityPolicy policies. We summarize our methods and prototype implementations in [201] and leverage this paper for the BSP conformance case study in this thesis.

China Labs aimed at a static approach for the validation of artifacts from the very beginning. This is due to the fact that WSRR is designed to store static artifacts on the one, and in consequence of the fact that static assurances on conformance is a preferable approach on the other hand, since knowable before deployment. This increases the value of WebSphere to the client, i.e., it enhances consumability. A policy tool [6] developed at the IBM Zürich Research Laboratories, that enables the definition, assignment and composition of Roll Based Access Control (**RBAC**) rights to static platform artifacts, like Databases or WSDL descriptions, inspired us to the idea of a policy authoring tool that utilizes the schematron validation to guide practitioners in the development of policies. Leveraging the expressiveness of Schematron, we decided to elaborate a methodology and a subsequent implementation of a prototype tool that is able to map high-level requirements (like the adherence to the BSP profile, to some legislative requirement or to an in-house specification for example) to operational Extensible Markup Language (**XML**) machine policies.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

"Declare the past, diagnose the present, foretell the future."

Hippocrates of Cos [141]

Initially, the web's major goal [57] "was to be a shared information space through which people and machines could communicate". In the digital world of today, this notion perpetuates but is also being excelled: services like power grids, health care, traffic control, water supplies, food and energy, insurance, life sciences, media and entertainment, retail, telecommunications, travel and transportation, along with most of the world's financial transactions, just to name a few, all now depend on the information technology. Major parts of business are executed semi-automatically in "cyberspace", across cultural and geographic boundaries and involving a manifold of diversified technologies. The bases for this evolution are computer networks that allow to share workload via *communication protocols*. Such distributed systems [226] involve a plethora of different participating processing units and encompass many technologies. The standard model for distributed communication is the OSI Reference Model [252], whereby the Internet Protocol Suite [68, 66, 67] (**TCP/IP**) is used in the WWW and constitutes a subset of this model. Both abstract communication between heterogeneous entities as a layered protocol stack that contains the application-layer at the top-most level of abstraction, that may be furnished with or built upon by additional (agreed-on) protocols. The protocol stack homogenizes the network on a higher level, with underlying protocols transparently encapsulated (i.e. it "flattens cyberspace"). This allows for extensions without having to take care about core communications.

1.1 SOA Security

Today's predominant paradigm for large-scale Enterprise Application Integration [115, 156, 153] (**EAI**) is the *Service Oriented Architecture* [160, 100, 192, 108] (**SOA**). SOA is a model and method that architectures the aspects of modern cross-organizational businesses by exposing application logic (i.e. business functionality) as loosely-coupled, system-independent, autonomous, reusable and composable IT services. Extensions of the service-oriented approach

with issues related to composition, conversation, monitoring, and management of services [191] is subsumed by the notion of *Service-oriented Computing* (**SoC**) [193, 194]. The cross-disciplinary *Services Computing* [250] technology family tries to effectively create and leverage computing and information technology to model, create, operate and manage business services for higher flexibility facing future business dynamics. These developments promise to eventually give rise to a (world-wide) services ecosystem [251] for automated agile and on-demand collaboration, further flattening the world [112] and globalizing businesses. Eventually, Cloud Computing [149] is taking shape now, also catch-phrased as *"Everything as a Service"* (**EaaS**, **XaaS**, ***aaS**), superseding Grid Computing [138, 111] in the ability to support automated, intelligent and efficient resource sharing besides the sharing of computing power and storage. Extended SOA presses ahead with the utility computing vision and bears the prospect to become the next step forward in evolving the internet.

Web Services [119] have shaped up as the foundational building blocks commonly used to realize SOA, as stipulating sufficient autonomy to enable service reuseability and the management of individual services according to changing business strategies. Web Services specifications yield an extendable platform and modular superstructure of universally supported interoperability protocols and standards on top of a lower level distributed communication stack, such as TCP/IP. With the emergence of advanced and widely utilized Web Services technologies, more and more companies are able to expose their business applications as IT-enabled services into the services ecosystem. As SOA further blurs intra- and inter-organizational boundaries across heterogeneous infrastructures, disparate operating systems and programing languages and above all, across different ownership realms, a main challenge in perpetuating this development is *security* [23, 101, 204, 144]. Secure Web Services communications for the interoperability across heterogeneous infrastructures and multiple trust realms are a major concern for SOA practitioners. However, security has its dark sides in a flat world [242, 99, 122], which this thesis is committed to shed some light on.

At the application level, SOA security essentially is *XML [237] security*. The high degree of abstraction and the overt nature of the set of security standards in the Web Services domain manage to subsume a wide variety of security mechanisms and technologies across different ownership domains, achieved through generic security specifications based on XML document model and grammar extensions. Hence, security in conjunction with Web Services technologies shifts from "security through obscurity" to *open* XML security standards accessible to everyone. Indeed, security methods have not always been provided in an open, standardized way: protection used to be (and sometimes still is) accomplished through proprietary methods, which is considered a very out-dated approach in modern cryptography. This is true at latest since Kerckhoffs' second principal had been widely adopted which originates from lessons learned by France during warfare in the 19th century: "*The system must not require secrecy and can be stolen by the enemy without causing trouble*" [145], reformulated by Shannon in a modern version as "*The enemy knows the system*" [216]. One reason for the principle's wide acceptance is caused by the simple fact that modern cryptography is predominantly public, especially in the civil sector (and innately public with respect to SOA Security). Today almost everybody uses (often not knowingly) open cryptographic methods, rendering these open, standardized methods magnitudes better tested than any (poorly tested) proprietary method will ever be. Open security

contracts are the prerequisite for secure interoperability in cross-boundary SOA collaboration.

On the other hand, the open specification of the standards and the various security technologies that SOA security incorporates to provide protection at the XML level, bring about a multitude of (novel) security vulnerabilities. Each monolithic application brings into play new potential vulnerabilities; additionally, new vulnerabilities emerge because of the permanent change inherent in SOAs. Business process automation can make security intransparent, resulting from unclear associations of responsibilities to users and services: composition/orchestration makes tracking single consumers hard. In fact, more than 13000 vulnerabilities had already been identified up until 2003 with respect to XML processing and messaging [76]. A major part of these vulnerabilities arise due to the SOA security specifications being complicated for developers that are not security experts. This results in the release of additional standardized deliverables, such as WS-I [8] guidelines, to clarify and amplify the standards and there usage. In order to enable runtime interoperability between multiple implementations that reside in heterogeneous environments, these deliverables support SOA practitioners in creating compatible artifacts as well as in handling those in a standard conformant manner. On the other hand, they procure guidance for implementors of SOA platforms in the development of tools that produce and generate interoperable artifacts. In addition, non-standard custom contracts and specifications are often part of an organization's contracting collection, that mandate the (in-house best-practice) way SOA security standards are to be applied within the very organization by its services or middle-ware implementors.

1.1.1 SOA Policy and WS-Security

Quality of Service (**QoS**) are meta-data defining non-functional requirements and capabilities of services. From the business perspective it is crucial to deliver services in such a way that a satisfying level of QoS can be supported. Customers must be able to assess and control the risk and benefit of utilizing services, in order to compensate the lack of control over the underlying IT infrastructure naturally inherent in SOA. Therefor, fostering trust in the governance and security of the infrastructure is important and can be addressed by (security) transparency. WS-Policy [25] *policies* are meant to make this class of meta-data visible in a well-defined and machine-processable fashion, with the prospect of *semantic interoperability* of services and their interactions across multiple systems at real-time. Interoperability as such is supplied by standardized interfaces and communication protocols and can be defined as the capability for two (or more) systems to exchange information [10] and to reciprocally access and utilize each other's application components and data. Constraints, qualities and goals (i.e. the QoS) laid down in the policies are the semantics that must be met by (non-resident) services for their business activity to be interoperable and to turn out the intended way. Generically, policies define "rules governing the choices in the behavior of a system" [220] and lend themselves to obtaining assurances that customers' infrastructure and services needs are accounted for, given a particular solution: by *policy compliance*.

It is widely recognized that security and its policies must be considered in all phases of the SOA development process, from the analysis of the organizational context to the final implementation of the system. This is however not the case in most solutions in use today. Systems are in the worst case designed without addressing security at all and employed protocols not

supporting accurate (interoperable) protection. Thus, security is often introduced by hindsight, resulting in the need for myriads of security specifications on a high level of abstraction that are able to enclose all involved (horizontally distributed) security mechanisms and technologies. These specifications are often evolved in parallel by different standards bodies or industry organizations, while sometimes addressing similar or overlapping issues. Additionally, the semantics of the standards are not always unambiguous by nature. Even if an individual runtime platform is compliant to the interpretations that its developers derived from security specifications the platform is supposed to support - which would guarantee interoperability among this very type of system - interoperability among different types of SOA runtimes is by far not guaranteed, as interpretations at the time of implementation are likely to deviate between different interpreters. All these circumstances inhibit the advancement of secure SOA collaboration.

The WS-Security [21] sets of specifications is the standard Web Services framework for end-to-end protection of Simple Object Access Protocol (**SOAP**) message exchanges. WS-Security provides security at both the transport- and XML-message-level in an open fashion and is therefor vulnerable to a wide range of security attacks [99, 161, 246]. It is a complex set of specifications, resulting from the mere complexity of the subject of security in general and the fact that WS-Security in a SOA is a "horizontal" framework, encompassing many technologies and mechanisms. Without descent knowledge of SOA security and an understanding of all relevant specifications, WS-Security meta-data will not be fail-safe; wrong and/or insecure/uninteroperable generation of SOAP messages and their security headers is likely to occur. Additionally, WS-Security is a specifications and is therefor inherently ambiguous and open to interpretation to a certain degree. These facts can diminish or even cancel out interoperability and/or security countermeasures incorporated in the SOAP message headers. Thus, care has to be taken by both, developers of WS-Security messaging middle-ware and authors of WS-SecurityPolicy [26] security policies. Deliverables for guidance in the usage of and compliance to these standards are therefor very valuable.

An example of such normed deliverables are the mentioned WS-I best-practices: WS-I releases standardized guidelines for selected groups of Web Services standards in the form of *profiles*, devoted to enable and enhance interoperability among different vendors of Web Services middle-ware. An extension and clarification of WS-I's Basic Profile [103] (**BP**), the *Basic Security Profile* [50] (**BSP**), constitutes such best-practice guidelines in the context of SOA security. It defines a set of requirements, considerations and recommendations - predominantly for WS-Security-augmented SOAP messages - to both, maximize interoperability and minimize security vulnerabilities. In contrast to other Web Services standards (i.e. the WS-* standards), WS-I profiles are semi-structured natural language documents that lack a formal model and are designed to enhance *runtime interoperability*, as opposed to interoperability between developer tools. BSP conformance is an absolute necessity for interoperable and secure Web Services communications between disparate SOA systems.

1.2 SOA Security Policy Validation and Authoring using Schematron

A common goal of policy-based/policy-driven management approaches (such as [35, 93, 196, 219, 28, 220, 167, 37, 104, 155, 197], to name a few) is to enable practitioners to define high level directives for the enforcement of business policies and objectives, instead of manually configuring and auditing services. As SOA security is a particularly complicated field for the developer that is not a domain expert, such a function is predestinated for security policies. We call the configuring of policies in terms of high-level directives policy *authoring* and (semi-)automated auditing - that is, the verification of the high-level directives - policy *validation*. Model-Driven Security (**MDS**) and Model-Driven Development (**MDD**) methods - such as meta modeling and model transformations of abstract (potentially graphical) security specifications (e.g. [52, 154, 166, 72, 137, 213, 36, 58]) - are increasingly used to relief practitioners from parts of the complex implications of security in SOA, incorporating security as an integral part of the development life-cycle and aiming at the protection of the system as a whole. Especially useful are MDD/MDS security policy validation and authoring tools that enhance standard conformance and that improve the ability to engage security behavior statically at design time.

1.2.1 Problem Background

We take a model-driven approach to policy authoring and grasp it as a process of policy refinement by means of an escorted policy compliance validation that is undertaken against the background of a two-dimensional *policy hierarchy* [164, 29]. In this notion, policies are decomposed along each of the hierarchy's dimensions: the *vertical dimension* representing levels of abstraction of policies and the *horizontal dimension* that stands for different policy domains and types. The simplest form of a policy is a single abstract requirement. Depending on the maturity of the SOA infrastructure, policies must be expressed with varying degrees of detail at each vertical level, ranging from natural language policy documents in the limit, to various detailed machine specifications at the micro-level of analysis. The nature and taxonomy of policies of different types and domains at each abstraction level is determined by the policy models/languages in use at that level, representing the horizontal policy management dimension.

The vertical policy management dimension can be categorized into the *Business*, *Architecture* and *Operation* levels of abstraction, as seen on Figure 1.1. The diagram gives a schematic overview of the policy hierarchy and how our approach mediates policies from the Architecture level down to the Operation level: example macro-level business requirements are given, such as the adherence to the Health Insurance Portability and Accountability Act (**HIPAA**) of 1996 or to a simple Services Component Architecture (**SCA**) [47] intent. The business requirements imply (sets of) architectural requirements, like "WS-I compliance" or "encryption of social security numbers", for instance. Architecture requirements must further be decomposed into sets of operational policies. No matter if these architecture requirements are stated in natural language or defined with the help of an intermediate formal architectural policy model, in any way these requirements map to sets of micro-level operational policies that are likely to diversify horizontally, i.e. use different policy domains/types (policy languages/models) at the Operation

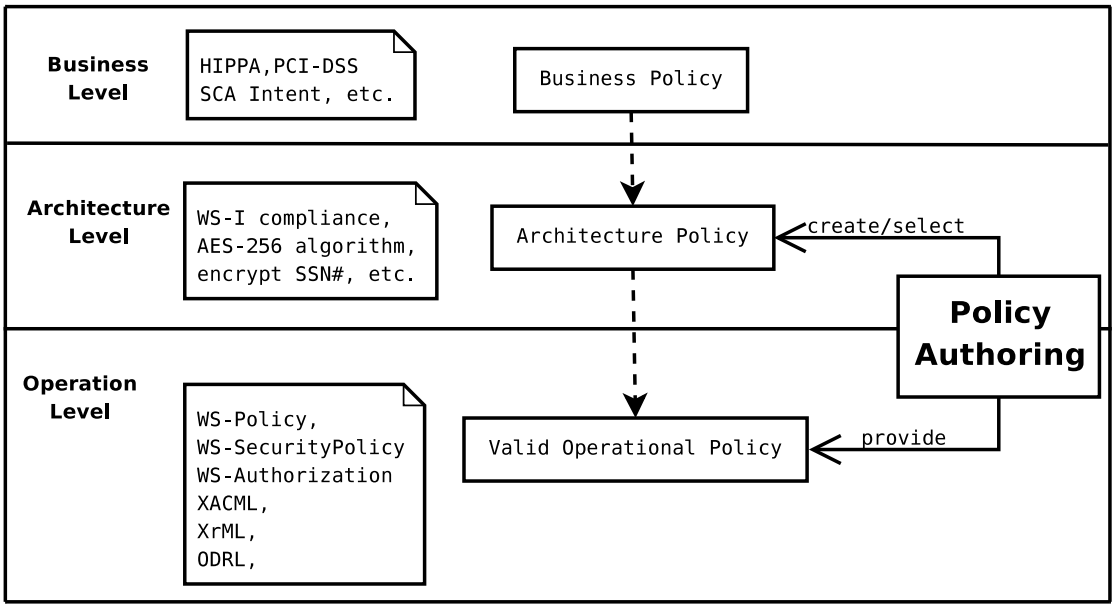


Figure 1.1: Policy authoring against the background of the policy hierarchy.

level, such as WS-Policy, Extensible Access Control Markup Language [18](**XACML**), and so on. Thus, both vertical and horizontal policy compliance validation must be performed, in order to correctly map policies throughout the hierarchy.

1.2.2 Solution

This thesis proposes a guided model-driven policy authoring method that leverages the Schematron [92] standard for an *XML Mediation* between architectural and operational policies on the one, and, to different models, domains, and types in operational policies on the other hand: we use the *Schematron Validation Pipeline* (**SVP**) for evaluating schematron rules expressing the policy mediation within the architectural policy model, against instances of (potentially multiple) operational policy models. This allows to infer compliances and exceptions of assertions in the concrete machine policies with respect to assertions representing abstract directives. Schematrons achieve that by incorporating rule-based validation patterns that define (additional) *syntactic* (schema-)constraints on (yet XML Schema-constrained [228, 61]) XML. These syntactic rules enable us to provide abstract composable policy templates capable of capturing natural-language non-functional macro-level requirements, that automatically map to sets of concrete micro-level machine policies that correctly reflect the directives laid down in the templates. Following, a brief overview of a WS-I conformance case study is given that was conducted along the development of the authoring model. In the course of the implementation in Chapter 3 we will examine the policy authoring approach in conjunction with this target security requirement in detail and discuss the implementation of a prototype, as well as the integration with the SVP.

1.3 A Motivating Example: Basic Security Profile Conformance

In this thesis, we explore BSP conformance as an extensive case study for a security/interoperability requirement that is indispensable for an expedient SOA. We engineer this requirements in an architecture policy with the help of our authoring methodology. Adherence to the best-practices issued by WS-I implies the need to solve the following two tasks:

(a) formalize WS-I standards into a machine processable representation

(b) validate compliance against WS-I standards (using the representation)

We provide solutions in this thesis for both problem (a) and (b) stated above with respect to the BSP profile. Compare the "WS-I Compliance" sample requirement in Figure 1.1 that is stated at the architectural level. "BSP Conformance" is subsumed by "WS-I Compliance" and can in parts be accounted for using WS-SecurityPolicy's model and policy types/domain at the Operation level. We discuss the formalization of the BSP conformance requirement with the help of schematrons on WS-SecurityPolicy and how the SVP serves us as a flexible mechanism for (semi-)automated validation along this process. We therefor infer the families of WS-Security augmented SOAP message from WS-SecurityPolicy, if one exists, and thereby provide assurances on WS-I conformance statically, prior to deployment. For this build-time approach, we leverage Schematron's system independent reference implementation.

Second, we summarize our implementation of an interceptor module for runtime WS-Security validation against BSP - the implied method of the profile by itself for checking conformance against its requirements. This dynamic validator engine mimics a subset of the Schematron standard only, in support for a better WS-Security message processing performance, however, is sufficient to cross-check the correctness of the static BSP formalization. For both BSP validation approaches we discuss the nature of appropriate schematron rules and give a comparison of the coverage (i.e. % of BSP requirements we are able to validate) of each of them, while further examining the overhead introduced through dynamic validation on the wire. The case study validates the authoring model with respect to complex security requirements and underlines the benefits of Schematron for the management of (the security life-cycle of) policies.

1.4 Structural Organization of the Thesis

The thesis is organized as follows. Chapter 2 starts with considerations concerning the SOA component model and XML contracts, followed by a brief outline of the state of the art in SOA security and policies. In particular an introduction to the Web Services platform as an instantiation of a SOA is given, with a focus on the WS-Security and WS-Policy (resp. WS-SecurityPolicy) frameworks. BSP compliance and authoring tool implementations are discussed in Chapter 3. In the evaluation in Chapter 4 we summarize our BSP conformance validation experiments, review the authoring approach and discuss advantages of XML mediation with the help of schematron. Finally, related work concerning policy compliance validation and security policy authoring is presented in Chapter 5, putting the emphasis on XML schema(-constraint) languages in the context of meta-models. The thesis concludes in Chapter 6 with a summation of the results obtained, lessons for SOA practitioners and an outline of future research directions.

CHAPTER 2

State of the Art

"Every statement about complexes can be analyzed into a statement about their constituent parts, and into those propositions which completely describe the complexes."

Ludwig Wittgenstein[1] ***[243]***

Open standardization is the prerequisite for cross-platform collaboration among different SOA runtimes of multiple vendors. Web Services [80] technologies constitute a widely standardized collection of open specifications for designing and building loosely coupled software solutions that exhibit business functions as programmatically accessible application logic wrapped as Web Services, meant to be used by other components and applications through publishable and discoverable XML interfaces and protocols. This entirely contract-based nature of the SOA component model differentiates SOA from other distributed application-to-application approaches (e.g. CORBA [1], COM [2], J2EE [4], etc.) in such a way, that functionality can be provided at a level of granularity and abstraction much closer to what is meaningful at the business modeling level [81]. Services interactions are disburdened from the dependence on static shared types which renders SOA components *virtual*, that is, completely system-independent and programing language-agnostic, as opposed to OMG's CORBA and COM IDL, for instance, which prefer the C type space for interface description, whereas J2EE concentrates on Java. Virtualization is conferred by separating structure and behavior, while the explicit, machine verifiable description of these aspects in *external* XML/XML Schema specifications constitutes a means for services interactions to be defined in a self-descriptive fashion, encapsulated from a SOA component's implementation and execution environment, i.e. from its *internal* specification [79]. The sum of a virtual SOA component's external specifications is called the component's *contract*, containing all *meta* information for using a service. The contract defines the XML API into the functionality offered by a service.

[1] English translation of statement *2.0201* from `http://www.gutenberg.org/files/5740/5740-h/5740-h.htm`

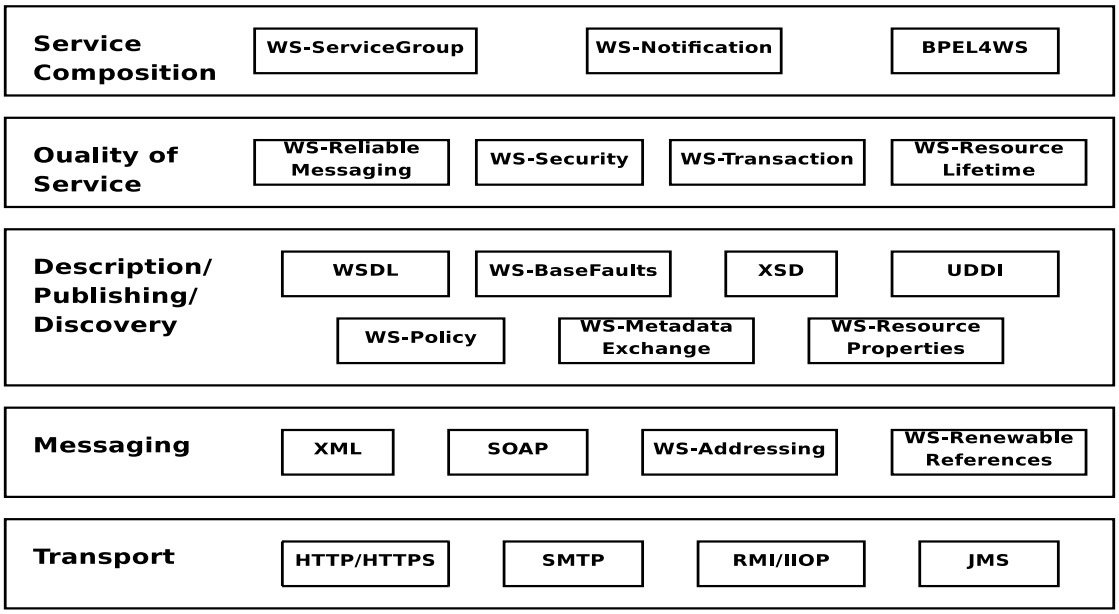


Figure 2.1: The Web Services Standards Stack - a proposed topology of WS-* specifications

Standards bodies - foremost OASIS [5], W3C [9], and WS-I [8], but also IETF [136] and UN/CEFACT [230] - formalize various available contemporary Web Services specifications and contract languages into open, composite - and vendor-neutral - standards, catering business processes realized as Web Services in such a way, that they become interoperable across diverse SOA solutions and platforms. The goal of these standardization efforts is to provide an overt interoperable XML platform such that the market will not tolerate proprietary (system-dependent) solutions, ultimately establishing the mentioned services ecosystem in a manner that accurately mirrors modern collaborative business. There are many options to structure and visualize Web Services contracts, as there are many means to make use of Web Services. The Web Services Standards Stack [7, 222, 251] (**WSSS**) is a common logical structuring of diverse WS-* specifications evolved by different standards bodies that are available for implementing contemporary SOA. Due to W3C XML Schema, the WSSS holds the possibility to gradually solidify service-orientation and confers (modular) extension by leveraging name-space based composition. See Figure 2.1 for a visualization of this model, introducing a protocol stack of five layers on top of a lower level distributed communication stack, such as TCP/IP. The diagram is meant to serve as background for the subsequent discussion on SOA Security and WS-Security and shall give an impression of how security standards relate to other contemporary SOA specifications, in terms of hierarchy and terminology.

W3C's XML Schema is generally utilized to describe the XML models of WSSS' WS-* specifications - including those of WSDL [73], SOAP, WS-Security, WS-Policy and WS-SecurityPolicy -, but is also frequently used to describe the message payload, that is, the SOAP header and body child elements, respectively. The latter is done within the services' WSDL, that constitutes the basic *functional* contracting language and frame for the XML API of Web

Services, determining the input, the function and the output of services. WSDLs describe a service's endpoints, available operations and message types. Additionally, WSDLs contain binding information, determining transportation protocol bindings for the services endpoints and the input/output message format for available operations. Strictly speaking, communication protocol details and the data format used during interaction belong to the class of *non-functional* meta-data, underlined by the fact that WSDL allows multiple bindings to the same business interface. Non-functional metadata express preferences in the function of services, rather than defining specific behavior; i.e. real-time/runtime system constraints [72]. The main container for contracts that express non-functional metadata beyond binding information are provided by WS-Policy policies, that may be defined isolated from WSDL's services model. WS-Policy policies may be associated with WSDLs at multiple levels of granularity and pave the way to management of the life-cycle of services with respect to QoS. The QoS layer of the WSSS contains specifications defining XML model extensions on top of the SOAP foundation for different QoS domains, that can be interpreted as the *visible* "QoS runtime space" of domain-specific vocabulary extensions defined on top of the non-functional Web Services contract model, that is, on top of WS-Policy's grammar. These domain policy languages mandate the nature and XML syntax of WS-*'s QoS SOAP add-ons used to establish fundamental QoS capabilities, such as security, reliability and transaction. In the domain of security, WS-Security provides the object model and syntax for valid runtime SOAP security (header) syntax, while WS-SecurityPolicy defines security-specific assertions meant to be adopted on top of WS-Policy's policy aggregation model for governing runtime security behavior, including the production of (visible) WS-Security at runtime. Besides visible behavior, security policies may target *invisible* behavior not resulting in any wire activity, i.e. policy assertions whose enforcement do not manifest in the form of WS-Security-enriched SOAP message exchanges. For example, a provider may not wish to interact unless a client can accept an assertion describing provider behavior, such as an assertion that describes the privacy notice information of a provider. In this case, the interacting participants may require some sort of additional mechanism to indicate policy compliance and to enable dispute resolution.

2.1 SOA Security

The term security is often used to cover a multitude of requirements [105, 38, 106, 39], essentially *Confidentiality* (**C**), *Integrity* (**I**) and *Availability* (**A**); that is, the so called *CIA-Triad*. Confidentiality protects (secret) data from disclosure, while integrity tries to obviate data modifications and indicates if such modifications have occurred. Availability helps the resilience to various vulnerabilities (e.g. Denial of Service (**DoS**) attacks) and is typically concerned with restricting access to information and services using an access control model, such as RBAC [202]. An access control model is maintained in a timely and dependable manner for authorizing, authenticating and auditing agents and involves *security policies* that define when access should be allowed or denied. Policies are generally used to specify (non-functional) rules, regulations and requirements of services and their interactions. Thus, besides restricting access according to control models, policies are adopted for mediations of *security transactions* that determine the nature and usage of the security context information specific to the very transaction. Such poli-

cies include the expression of the security binding(s) as well as applicable tokens and their usage during the transaction, compulsory message elements, encryption and signature algorithms leveraged to provide data/message confidentiality and integrity as well as the canonicalization (**C14N**) method, supported key derivation methods, et cetera.

Cross-domain and cross-infrastructure SOA collaboration in a secure fashion requires that all services used for conducting business are properly protected, including routing, management, publication and discovery services. Numerous threats, such as DoS, man-in-the-middle or spoofing attacks, XML reply attacks as well as additional threats to data and message content must by countermeasured by security mechanisms. SOA security must shelter SOA solution from such vulnerabilities by establishing *security services* that provide appropriate mechanisms and that implementors can use for securing their Web services implementations, in particular building upon mediation mechanisms for the enforcement of security policies. Security policies define who may use what information within the system, while protection mechanisms are built into the SOA platform to enforce security policies. In allusion to what was said above, the security services implement the mechanisms for the enforcement of WS-SecurityPolicy policies using WS-Security at runtime. A refined view of necessary security mechanisms [183, 16] can be inferred from the CIA-Triad:

- Identification/Authentication
- Authorization
- Data Integrity and Confidentiality
- End-to-End message Integrity and Confidentiality
- Non-Repudiation
- (Distributed enforcement of) Policies

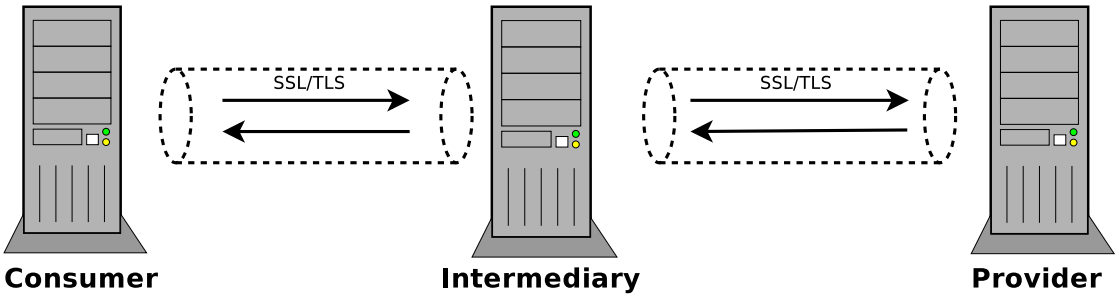
Confidentiality and integrity algorithms are based on the science of cryptography [163], utilized to derive access control (that is, authorization, identification/authentication and auditing) as well as non-repudiation mechanisms: *identification* is provided through identity tokens, like user name tokens or signed identity certificate tokens which are signed by a trusted third party (Certification Authority (**CA**)), for instance. Proof of a claim of identity takes into account additional credentials (like passwords or signatures). Tokens providing such credentials in addition to an identity claim are called (singed or unsigned) security tokens, and are said to provide *authentication* credentials that are used by authentication services to evaluate that the credentials provided are in line with the identity claim, i.e. authentication [162] corresponds to the verification of (that is, proof of) identity of service consumers every time a Web Service is accessed. *Non-repudiation* implies the same mechanisms as the authentication of data, i.e. proof of integrity and origin through digital signatures and certificates, respectively, in order to prevent both consumers and providers from false-denial that data has been sent or received. *Authorization* is the process of verifying that an authenticated identity is allowed to have its request fulfilled and may be granted access to a requisitioned resource or not, based on the authorization decision. The domain of *auditing* is to record, analyze and report security-related events, and,

besides providing the bases for non-repudiation, auditing is used to verify security *compliance*, meant to check whether security-events comply with intended outcomes. Compliance is closely related to the management of security, where security policies play a major role.

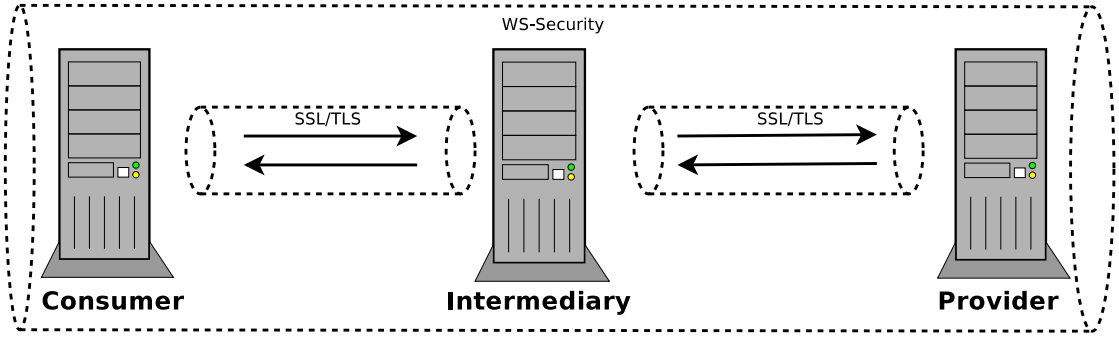
The security services enforce mechanisms according to security policies. Distributed enforcement of policies (security policies associated with requester entity, service and discovery mechanism) may be used to manage cross-domain identities (centrally). The secure discovery mechanism associated to a distributed policy can prevent unauthorized user to discover restricted Web Services. As an example, take into account a sensitive health care service that only users associated with certain privileges are allowed to discover. Such a service might be considered secure only, if visible to doctors of a certain institution while being indiscoverable for all others. In order to establish advanced delegation and federation relationships, i.e. *identity federation*, trust mechanisms can be leveraged. For example, a (distributed) trust policy attached to the service contract could define how to resolve and process identity tokens embedded by a sender into messages. Hence, distributed security policies together with federate identity management enable controlled secure interactions between Web Services across organizational boundaries in a distributed fashion.

Specifically, RPC- and document-style message models (such as SOAP) in conjunction with static service descriptions that abstract the physical network and application infrastructure to end-points, frequently incorporate a multi-hop topology with intermediate actors. When data is received and forwarded on by an intermediary beyond the transport layer, both, the integrity of messages and any security context information that flows with it is always at risk to be lost. This forces any upstream message processors to rely on the security evaluations made by previous intermediaries and to completely trust their handling of the message content. See Figure 2.2 (a) for an illustration of transport-level security, that introduces multiple security contexts for messages that pass an intermediary, one between each pair of adjacent actors. The solution to this problem is the provision of mechanisms for *end-to-end security* - that is, end-to-end confidentiality and integrity, respectively in contrast to point-to-point protection configurations. This implies a single security context spanning all the way from originator to target, instead of several contexts for each transport channel between communication points. Observe on Figure 2.2 (b) that messages stay protected over the entire communication path. Message exchanges secured this way are capable of maintaining a single security context between all actors beyond the transport level, and exclude intermediaries as possible sources of vulnerability. Successful security solutions will be able to leverage both transport and application layer security mechanisms, to provide a comprehensive suite of security capabilities.

The WS-Security composite standard [206, 214, 20] is a high-level XML platform for implementing the security mechanisms defined above and facilitates both, protection at transport-level as well as end-to-end confidentiality and integrity of (portions of) messages at the XML level. Confidentiality is still often provided at transport-level, because encrypting the whole communication channel at the lower level is much faster. This however neglects potential intermediaries where messages could pass unprotected, and ignores the benefit of end-to-end message protection: the channel is deciphered at intermediaries and XML message content is at risk. Hence, communication is insecure when not protected beyond the transport-level. Integrity must naturally be afforded at the XML level in order to provide security on an end-to-end bases, also



(a) Transport-Level point-to-point Security using SSL/TLS



(b) Message-Level end-to-end Security using WS-Security

Figure 2.2: Transport-Level VS Message-Level Security

because tokens specified in the messages themselves might need to be signed. WS-Security's security tokens are meant to be included in the SOAP message header to confer per-message authentication and protection at either of the two protocol levels, so also at the transport-level (compare TLS/SSL context in the diagram (b)). Consequently, WS-SecurityPolicy defines both, transport binding assertions and respectively, symmetric and asymmetric XML level binding assertions, the latter instructing the security middle-ware to secure message traffic throughout the whole communication path (compare WS-Security context in the diagram).

As adumbrated in the last Section, WS-Security is responsible for providing the syntax for visible security policy behavior (i.e. message exchanges) at runtime whose production follows the WS-SecurityPolicy specification. To put it the other way around, WS-SecurityPolicy describes how messages are to be secured on a communication path (including invisible behavior) by defining domain-specific security assertions, extending WS-Policy's grammar to represent security characteristics of WS-Security: SOAP Message Security [21], WS-Trust [172] and WS-SecureConversation [171]. Hence, it mandates protocol-level security, in contrast to XACML [18] or SAML [109] for instance, which are used to specify requirements for accessing Web Services interfaces and their associated resources at runtime. One example based on WS-Policy are WS-ReliableMessaging assertions that can be incorporated into the SOAP header dynamically, along with many other available WS-Policy vocabulary expansions, to support features like indicating message sequencing, for instance. WS-SecurityPolicy policies are therefor fun-

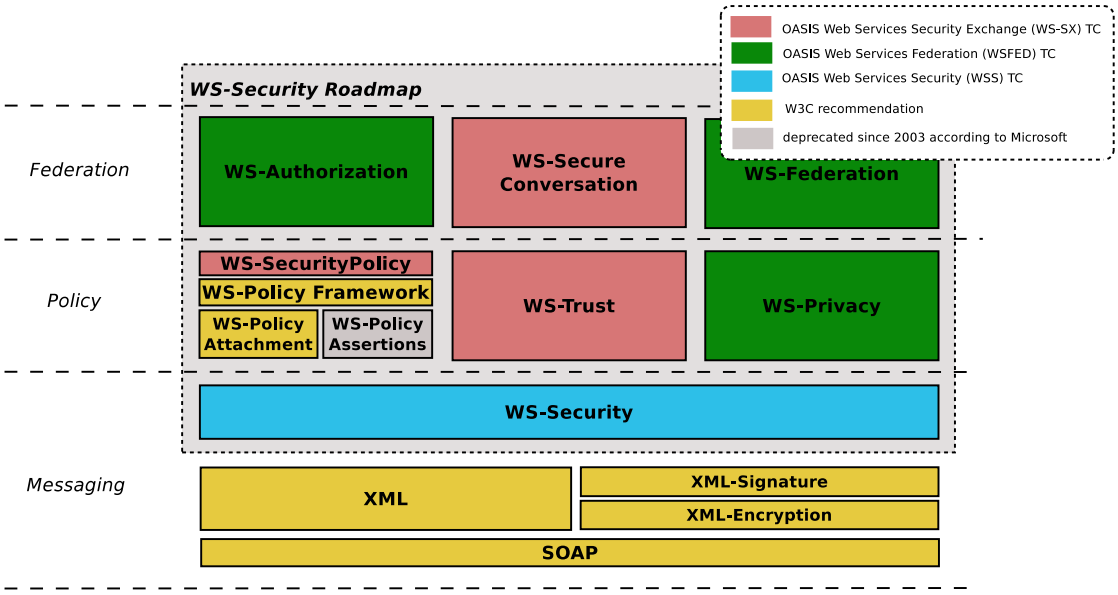


Figure 2.3: The Web Services Security Road-map

damentally different compared to such dynamic policies: though WS-SecutrityPolicy policies are meant to instruct the middle-ware in the processing of WS-Security at runtime, there is little sense in enriching runtime messages with such policies, since protocol-level security requirements usually address protection of an interaction as a whole, i.e. a security transaction. In other words, WS-SecurityPolicy policies will predominantly be consulted *before* an interaction takes place, whereas XML policy tokens are often contained in the request messages themselves, when relevant for calling an operation.

2.1.1 SOA Security Contracts

WS-Security supports a variety of security tokens that can be included in the SOAP message header, such as unsigned user name tokens, singed binary security tokens and XML tokens. Moreover, WS-Security defines multiple encryption and signature algorithms and several trust domains. Reconsider the WS-Security building block at the QoS-layer of the WSSS in Figure 2.1: the WSSS hides the fact that WS-Security is a collection of - i.e. *composite* - standards. In April 2002, the WS-Security road map [14] was published that structures WS-Security specifications into messaging, policy and federation layer. See Figure 2.3 for a visualization of the road map, which the WSSS' coarse-grained view of WS-Security masks. As seen, beneath the surface, WS-Security consists of a number of individual specifications that are being developed by different standards bodies and technical committees (**TC**), indicated by color. This integrated collection of security specifications yields a flexible toolkit of security technologies, as all based on XML, which renders them extensible while only a subset of the specifications may be (interimly) implemented.

Federation Layer WS-SecureConversation provides formal models for the definition and secure exchange of security context information and associated session keys between Web Services. WS-Authorization is a standard for managing authorization and access control policies and defines how to integrate claims with security tokens. However, it has not been published, which is one reason for alternative implementations and standards for authorization and access control, as will be detailed shortly. WS-Federation [117] specifies security models that can be used to establish a level of trust between disparate trust domains, i.e. a federation of identity, account, attribute, authentication and authorization across different trust realms, using a series of other standards including lower level specification like WS-Policy, WS-Trust and WS-Security.

Policy Layer The foundation of the policy layer is the mentioned WS-Policy framework leveraging WS-PolicyAttachment and WS-PolicyAssertions, the latter defining a set of common message policy assertions that can be contained within a policy (e.g. reliable messaging assertions). According to Microsoft, WS-PolicyAssertions has been deprecated as of 2003; domain-specific assertion vocabulary is usually developed in separated specifications (analogous to WS-SecurityPolicy, such as WS-ReliableMessaging-Policy). WS-Trust tries to unite and abstract underlying disparate trust models and defines methods to request and issue security tokens - potentially involving a third-party CA (called *Security Token Service* (**STS**) within WS-Trust) or a manned trust authority for certifications - in order to establish trust relationships. WS-Privacy works in conjunction with WS-Policy and WS-Trust and may be used to communicate privacy claims with the help of WS-Policy's policy aggregation model and associate such claims with messages. WS-Privacy is another building block of the WS-Security road-map for which no publication exists and whose functionality can be provided by an alternative means. WS-Policy(-Attachment) and WS-SecurityPolicy will be examined separately in detail in the next Sections.

Messaging Layer Core WS-Security (precisely, *WS-Security: SOAP Message Security* [21]) facilitates confidentiality and integrity protection as well as authentication of SOAP message exchanges at the XML level, throughout the communication path between two Web Services endpoints. End-to-end security can be provided by the introduction of a security header into the SOAP messages that contains the tokens needed for decryption, validation of the integrity/origin of messages and verification of identity, respectively. This way, security no longer relies on point-to-point security which would neglect possible intermediaries, as discussed above. WS-Security's schema definition is used to extend the SOAP specification with the grammar for valid visible WS-SecurityPolicy behavior, that is, valid WS-Security security headers. WS-Security therefor leverages XML-Signature [94] and XML-Encryption [95], which define a model and respective algorithms and XML processing rules to sign and encrypt XML content.

2.1.1.1 WS-Security Composite Standards

WS-Security belongs to the messaging layer, which the diagram shows in the context of lower level technologies, in order to convey its dependence on XML-Signature and XML-Encryption and its typical usage on top of SOAP. The same way WS-Security uses XML-Signature and

XML-Encryption syntax in the SOAP messages to indicate confidentiality and integrity protected message content, it uses other specifications to include both, binary and XML tokens within SOAP security headers. Hence, the WS-Security building block of the road-map, besides defining WS-Security: SOAP Message Security for end-to-end confidentiality and integrity protection, additionally references so called *token profiles* that clarify how WS-Security is extended and used with tokens of a certain technology. The following profiles are incorporated by reference into version 1.1 of WS-Security:

- Username Token Profile 1.1 [176]
- X.509 Token Profile 1.1 [175]
- SAML Token profile 1.1 [174]
- Kerberos Token Profile 1.1 [173]
- Rights Expression Language (**REL**) Token Profile 1.1 [82]
- SOAP with Attachments [113] (**SwA**) Profile 1.1 [132]

This refined picture of WS-Security reveals that we are again dealing with a composite standard. Essentially, the profiles define how to integrate the listed technologies into and with WS-Security in terms of XML namespace inclusion. All involved specifications make use of the inherent modularity and extensibility of the composite security standards.

As the specifications that are part of the WS-Security road map are under more or less avid development (and partially deprecated - see WS-PolicyAssertions), and, as the Web Services world is changing rapidly, many - in parts competing - security standards emerged in the community that may be used in individual contemporary SOA solutions to aid/supplement/extend/replace parts of the standards that are proposed by the WS-Security road-map. These specifications can be useful stand-alone on their own, but also lend themselves to furnish different aspects of the road-map's security stack (messaging, policy, federation layers). Though these security standards are used in conjunction with Web Services, strictly speaking, they do not belong to the WS-Security core standards. Rather, they represent the underpinning of WS-Security, which in turn can be thought of as acting on top those specifications [177] (the same way WS-Security leverages XML-Encryption and XML-Signature).

See Figure 2.4 for a set of specifications being referenced by WS-Security. All these standards are modular and composable, rendering WS-Security and related standards to complex sets of specifications. Observe the security categories on the left side of the figure, compare the security mechanisms mentioned earlier and recover standards that WS-Security references through profiles. In terms of XML Schema, a reference (arrow) corresponds to a namespace import of the specification's respective schema definition into the WS-Security document context. As seen, SOAP may be secured at the transport level using SSL and TLS [87, 88, 86], respectively, as well as on an end-to-end basis using WS-Security and XML-Encryption/XML-Signature, respectively. The Security Assertion Markup Language [109] (**SAML**) is used by WS-Security for authentication assertions. For binary security tokens, the Kerberos ticket [181]

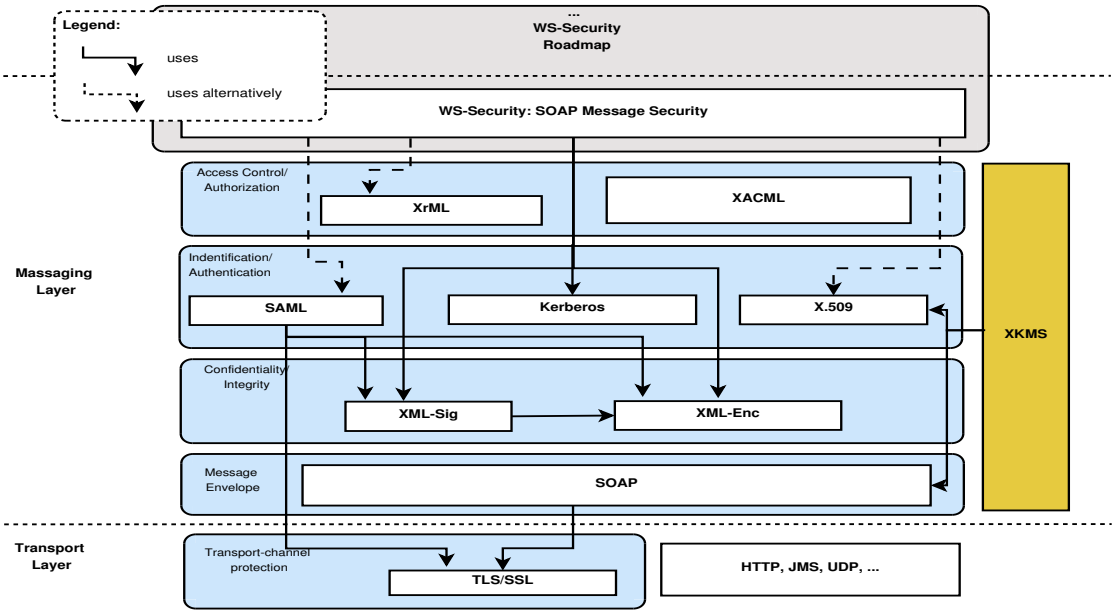


Figure 2.4: XML Security in the context of Web Services Security Standards: interdependencies against the background of the WS-Security Road-map

and the X.509 [77] certificate is available. XACML [18] is proposed for access control and authorization policies, together with the Extensible Rights Markup Language [13] (**XrML**). Note that XrML is the XML version of the Rights Expression Language (compare REL Token profile above). A Public Key Infrastructure (**PKI**) uses public-key cryptography [90] for encrypting, signing, authorizing and verifying the authenticity of information based on (binary) tokens and can be managed using the Extensible Key Management Specification (**XKMS**). Still many other standards exist that are not shown and that are related to SOA security and/or WS-Security in some way. One could mention Identity Web Services Framework (**ID-WSF**) of the Liberty Alliance for basic federate identity management, for instance, that exists in parallel to the specifications recommended by OASIS' WS-SX TC.

2.1.1.2 WS-I Security Standards

Since Web Services advocate the exchange of information across heterogeneous networks using open standards, SOA security standards are complicated while SOA solutions are vulnerable to a wide range of security attacks, as outlined. Web Service protocols like WS-Security composite standards define a rich but also complex framework in terms of additional SOAP header elements and processing rules to secure SOA solutions. Even though the core specifications and accompanying bindings try to be as accurate as possible, much effort is needed to achieve a common understanding among different implementations - and thus interpretations - of a standard in order to provide runtime interoperability. For WS-Security, this not only applies to the SOAP layer, but also to the exchange of user credentials which means that each specified dynamic

token must provide an interoperable format and semantic - ideally procured through *standard conformance*. However, having in mind the complexity of Web Services security specifications and SOA security in general, it is not surprising that additional clarification on the security specifications are needed to further promote WS-Security interoperability across platforms, operating systems and programming languages. WS-I standards fulfill this role by offering clarifications on yet standardized specifications in the form of *profiles* that represent consensus among major vendors (such as J2EE, WebSphere, .NET, etc.) on the interoperable and secure usage of the targeted specifications. Hence, profiles establish a best practice reference for improved service interoperability across multiple Web Services runtime platforms in a secure fashion, consisting of the mentioned BP profile, Attachments Profile 1.0, Simple SOAP Binding Profile 1.0 and the BSP at its basis. Compliance against the guidelines defined in WS-I profiles is a must, in order to make Web Services securely interoperable beyond unary organizations and singular SOA infrastructures.

2.2 A Policy Perspective: WS-Security Revisited

Keeping in mind the legally separated nature of services collaboration in a SOA, it is crucial for policies to meet security objectives. WS-Policy policies promote transparency by describing capabilities, requirements and general characteristics of Web Services and their interactions encapsulated from the functional WSDL contract, conferring the management of services throughout their life-cycles. Policy expressions registered in a UDDI, for instance, provide a means for non-functional requirements to be independently defined, selected, updated, validated, etc., and, lend themselves to assessing compliance against (QoS) business objectives and customer interests. Requirements and capabilities defined within the WS-Policy policies are called *policy assertions*. Sets of policy assertions are pieces of non-functional XML metadata that describe requirements in a specific WS-Policy domain [24], and are typically defined in dedicated specifications stating semantics, applicability, scoping and data type definition in that domain using XML Schema. Like XML Schema libraries, policy assertions are a growing collection. WS-SecurityPolicy, WS-ReliableMessaging Policy, WS-AtomicTransaction, WS-BusinessActivity Framework and the Devices Profile for Web Services are each examples of WS-* specifications defining domain-specific policy assertions that may be incorporated at either design-time into the non-functional contract or at runtime into the SOAP header. WS-Policy vocabulary extensions are expected to follow a set of best practices including to define both, syntax and semantics of assertions, to provide clarifications concerning nested and parameterized assertions and to describe to what extend policy assertions apply to the services model.

2.2.1 WS-Policy Revisited

WS-PolicyAttachment defines attachment mechanisms that allow to enrich WSDLs with WS-Policy policies at multiple extension points in the services model, to support adaptable QoS, such as secure, reliable and transactional messaging. Policies apply to different components of the services model, conferred by WS-PolicyAttachment's mechanisms for associating policies to specific WSDL object types in a multi-granular manner. The set of applicable attachment

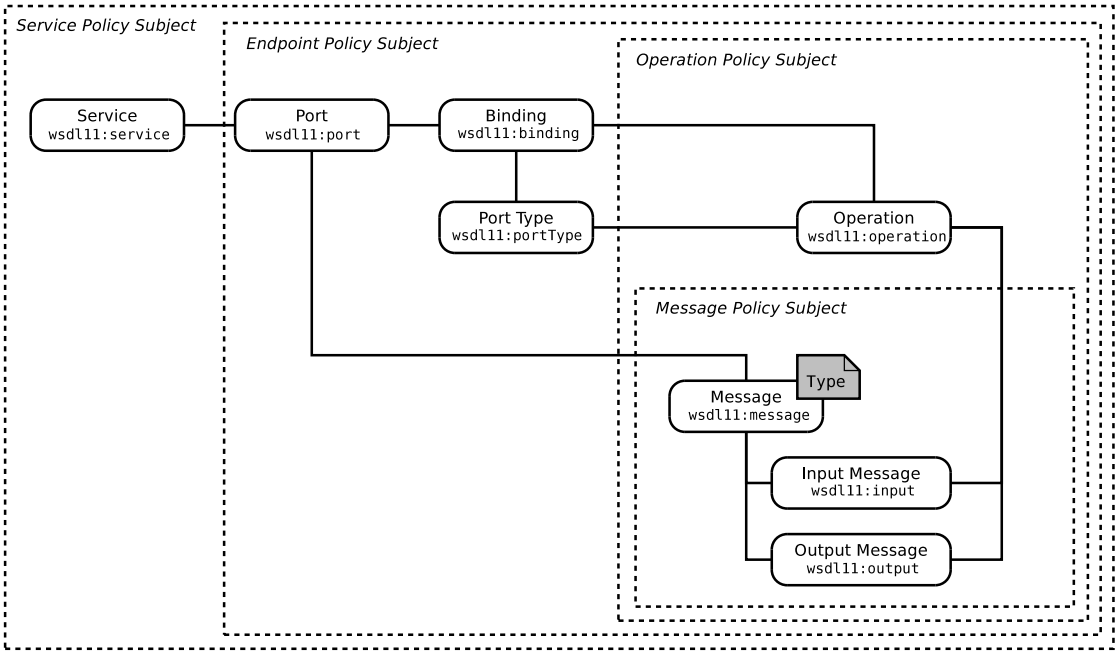


Figure 2.5: WSDL constructs and corresponding policy subjects

points for individual assertions must be declared by the specifications of the individual WS-Policy domain extensions. Hence, WS-Policy's grammar constitutes the base language for policy specification and aggregation, meant to be extended with assertions that concern the nature and usage of the QoS specifications (QoS layer of the WSSS). When enforced, the visible aspects of these assertions result in the production of runtime metadata that follows the XML model of the targeted QoS specifications, in such a way, that services interactions accord with the rules, regulations and principles (i.e. semantics) described by the assertions.

Consider the diagram in Figure 2.5 that shows main constructs of the WSDL services model in the context of policy attachment. The dashed frames represent so called policy *subjects*, each enclosing a subset of the other subjects, except the message subject which is self-contained. Subjects map to WSDL types and policies scoped at a specific subject are associated to the respective object type(s). A policy's scope implicitly determines the set of sub-subjects it applies to. Hence, policies scoped at a more specific subject, i.e. attached to a sub-type of a type that is associated with a (set of) "parent" policy have to be combined with that very (set of) policy. For example, a policy scoped at the operation subject and the `wsdl:binding/wsdl:operation` element, subsumes also the message subject and the `wsdl:binding/wsdl:operation/wsdl:input` respectively `./wsdl:output` (as well as `./wsdl:fault`, not depicted in the figure), resulting in the policy to be apply to both subjects. Thus, the overall policy at the scope of an operation is the aggregate of both, the sum of policies scoped at the operation subject and at the message subject it encloses. Therefor the policies over all scopes and subjects have to be combined in order to obtain an *effective* policy (at the scope of the Services Policy Subject).

Listing 2.1: WS-Policy's policy references in WSDL

```
1  <wsdl:definitions targetNamespace="ns.example.com" xmlns:tns="ns.example.com"
       xmlns:wsdl=".." xmlns:wsp="..." xmlns:wsu="..." xmlns:domns="...">
2    ...
3    <wsp:Policy wsu:Id="MyPolicy">
4      <domns:DomainSpecificAssertion>
5        <wsp:Policy>
6          ...
7        </wsp:Policy>
8      </domns:DomainSpecificAssertion>
9      ...
10   </wsp:Policy>
11   <wsdl:binding name="MyBinding" type="tns:MyPortType" >
12     <wsp:PolicyReference URI="#MyPolicy" />
13     ...
14   </wsdl:binding>
15 </wsdl:definitions>
```

`wsp:Policy` is the root element of WS-Policy policies whose value (i.e. sub-content representing the policy) may be reused by specifying a `wsp:PolicyReference` element when outside the current XML context. See Listing 2.1, illustrating how a policy is attached to a WSDL binding by using a policy reference. Note the inclusion of the `domns` namespace in line (1) that is assumed to define a domain-specific assertion vocabulary. In this example, the accumulation of attached policies to an effective services policy naturally requires the composition of the specified binding policy and policies potentially associated with the operation and messages that the very binding incorporates, as well as the composition with broader scoped policies.

WS-Policy mimics *boolean logic*, while allowing policies to remain extensible, by making use of the `xsd:any` schema type for assertion vocabulary. This permits to retain independence and broaden the range of available metadata. A policy may be formulated in either its *compact* form or in its *normal* form, while the compact form is transformable into the normal form by a deterministic algorithm defined in WS-Policy. Figure 2.6 shows a conceptional view of WS-Policy's policy model and how it relates to the corresponding XML syntax in compact form. As indicated, WS-Policy's model specifies a policy as a (unordered) collection of *policy alternatives* which are grouped by *policy operators*. Policy alternatives in turn specify a (unordered) collection of policy assertions that constrain a service's capabilities, requirements and general characteristics.

The schema outline of WS-Policy's normal form is depicted in Listing 2.2. In this notation, the asterisk ("$*$") is to be interpreted as the short notation for a XML Schema type definition of the corresponding prepended element, in this case indicating an element type with attribute $minOccurs = 0$ and $maxOccurs = unbounded$. Hence both, an `Assertion` element within an `wsp:All` operator element and a `wsp:All` operator element within a `wsp:ExactlyOne` operator element, must each be contained zero or more times. Custom or domain-specific `<Assertion>` elements as descendant of `wsp:Policy` policies, may contain *nested* policy expressions themselves, or may incorporate custom child attributes, both for *assertion parameterization*. This allows for recursive policy alternatives, as seen on line (6) ("?" stands for zero or one occurrence of the prepended element).

Listing 2.2: WS-Policy schema outline.

```
1  <wsp:Policy ...>
```

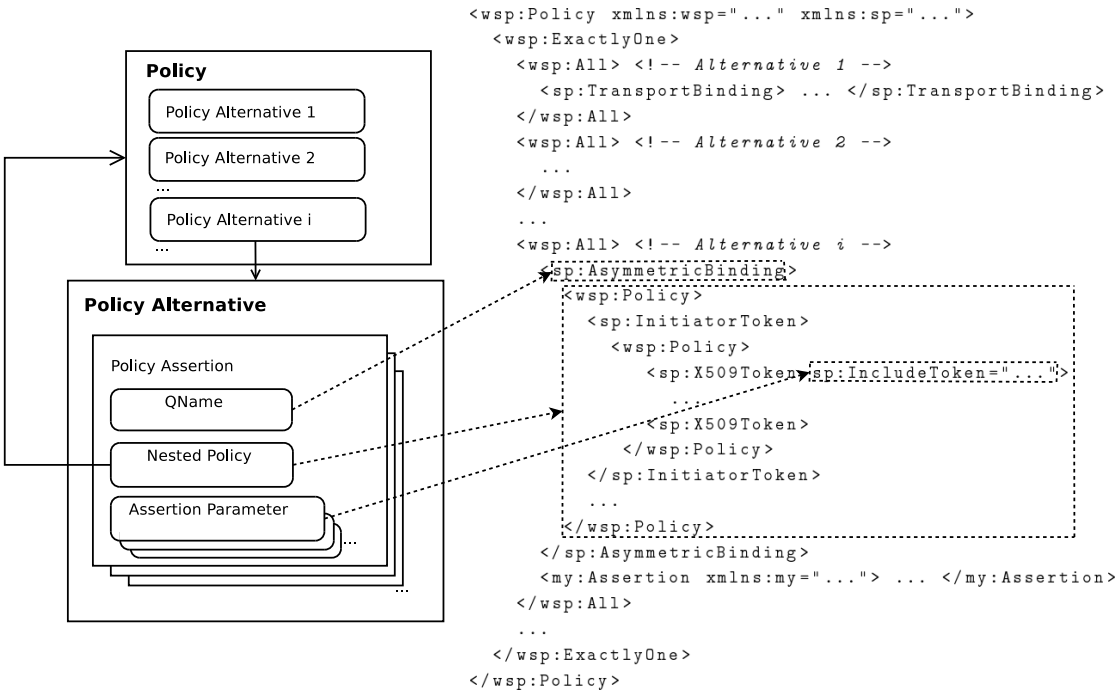


Figure 2.6: WS-Policy's policy model

```
2   <wsp:ExactlyOne>
3     ( <wsp:All>
4         ( <Assertion ...>
5             ...
6             ( <wsp:Policy ...> ... </wsp:Policy> )?
7             ...
8           </Assertion> )*
9       </wsp:All> )*
10  </wsp:ExactlyOne>
11 </wsp:Policy>
```

WS-Policy's normal form inherently exhibits *commutativity*, *associativity*, *idempotency* and *distributivity*. These algebraic properties of WS-Policy's flexible grammar and recursively extensible object model, permit comprehensive policy aggregation and intersection between different parties in an act of communication, leveraging the (transformation algorithm to the) normalized form together with some XML processing constraints, as described in the WS-Policy specification. Considering the grammar, the boolean basis of WS-Policy is not hard to recognize: policy operators `wsp:Policy`, `wsp:ExatlyOne` and `wsp:All` are used to group policy assertions into disjunct and conjunct policy alternatives, respectively. Inclusion of children within the `wsp:ExatlyOne` element corresponds to a logical exclusive disjunction of these assertions (logical *XOR*), whereas children of the `wsp:All` element correspond to a logical conjunction of the same (logical *AND*). Note that `wsp:Policy` is equivalent to the `wsp:All` operator. In the normal form, each nested policy expression contains at most one policy alternative, for the purpose of assertion matching. Version 1.1 of the WS-Policy Framework additionally defined

the `wsp:OneOrMore` operator (logical *OR*), was dismissed however in later versions, since expressible in terms of the other operators. Hence, the *Disjunctive Normal Form* (**DNF**) can be derived from any WS-Policy policy, explicitly identifying the set of all admissible configurations. The DNF lends itself to the logical aggregation, intersection and/or composition of policy alternatives to obtain accumulated overall services policies or to check if policies are compatible.

2.2.2 WS-SecurityPolicy

WS-SecurityPolicy defines security-specific requirements and capabilities of Web Services in the form of an XML Schema-constrained vocabulary on top of WS-Policy's model for the logical aggregation of policies. If the assertions of a WS-SecurityPolicy policy (i.e. within `<wsp:Policy/>` elements) represent visible behavior, then they are accounted for inside the WS-Security message headers dynamically at runtime, dependent on their attachment scope. This means that the set of WS-SecurityPolicy assertions scoped at corresponding applicable policy subjects, mandate at design-time (before deployment) the structure of the WS-Security metadata that messages have to include at runtime. Besides the visible behavior, assertions may statically qualify behavior that is accounted for internally, resulting in invisible dynamic behavior. From a conceptional perspective, WS-SecurityPolicy is concerned with the technical mediation of a security transaction between an initiating entity and a recipient entity, called the *Initiator* and the *Recipient* respectively [203]. The security policy specifies the context information and its usage during the transaction, using one or more of the various technologies that are usable with WS-Security (compare Section 2.2). Typical WS-SecurityPolicy use cases additionally involve the concept of an approving entity known as the *IdentityProvider* or *Authority*. Authorities can be either a *ValidatingAuthority* or an *IssuingAuthority*, and are in general either a WS-Trust STS which unites *ValidatingAuthority* and *IssuingAuthority* within one entity, or any other *Authority*.

2.2.2.1 WS-SecurityPolicy Security Transaction

When moving the perspective from conceptual to specific, the conceptual initiator "entity" becomes an "actor" that requests a security transaction by sending a message that contains the details of what is being requested, triggering interaction involving the *Recipient* and *Authority* "actors". See Figure 2.7 for a diagram that shows resulting actors in a typical security policy scenario [203]. Observe *Requester* and *Initiator* as separate actors that constitute the "client-side" actors, whereas *Recipient* and *RelyingParty* can be regarded as to form the "server-side" actors. The *Requester* acts as the supplier of credentials that ultimately get passed to the *Recipient*, whereas the *RelyingParty* is responsible for making the business interface of the requested service available. The *IssuingAuthority* typically provides authentication credentials to the *Requester*, while the *ValidatingAuthority* is requested by the *Recipient* and in turn validates *Requester* credentials for *RelyingParty*s. A *ValidatingAuthority* in general maintain credentials in an ongoing reliable manner. In other words, the technical resources for an end-to-end interaction between *Requester* and *RelyingParty* are provided by the *Initiator* on the client end and by the *Recipient* on the server end of a security transaction's message exchanges. A *Requester* hands over requests to the *Initiator*, who in turn issues a query to the *Recipient* and eventu-

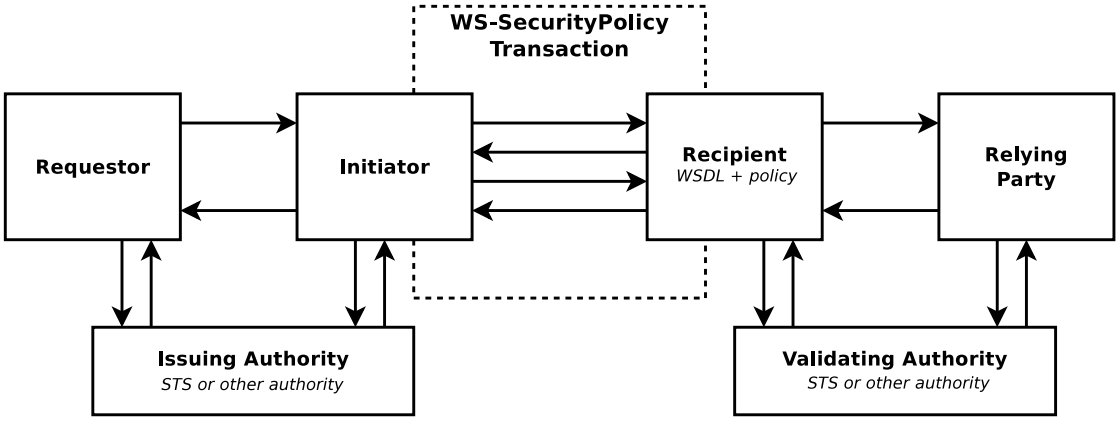


Figure 2.7: WS-SecurityPolicy's specific actors in a typical use case scenario

ally receives a WSDL describing the Web Service. The WSDL may contain WS-SecurityPolicy assertions defined and attached with respectively, WS-Policy and WS-PolicyAttachment as discussed, in order to describe the security policies supported by the *Recipient*. Alternatively, WS-SecurityPolicy policies may also be stored locally based on out of band agreements between the *Requester* and *RelyingParty*.

After receipt of a WS-SecurityPolicy policy, the *Initiator* interacts with the *Requester* and the *IssuingAuthority*, meeting requirements specified by the policy's assertions. Credentials and tokens are obtained from the *Requester* and *IssuingAuthority* and assembled into a new WS-Security request message that is to be issued to the *Recipient*. The *Initiator* then includes necessary tokens, applies message protection mechanisms as required by the considered policy and finally sends the message to the *Recipient*. *Requester*s in general do not know in advance what policies each Web Service provider expects, which makes front end *Initiator*s practical: these may resolve the policy and coordinate whatever actions are required to exchange *Requester* tokens with tokens required by the service. The necessary requests and process responses from an *IssuingAuthority* may be conducted using WS-Trust and its STS. The details of how the *Recipient* processes the message and uses a *ValidatingAuthority* to validate tokens on the "server-side", and the method by which a *RelyingParty* accept or reject a request, are system-specific. In any way though, the identified actors on the *Recipient* side will be involved to accept and process security transaction requests. In other words, the primary focus of the three server-side actors is to provide a WS-SecurityPolicy policy for accessing the *RelyingParty*'s Web Service and to validate credentials and messages received. *Requester*, *Initiator* and *IssuingAuthority* in contrast, must each account for different aspects of the preparation of the security transaction request message.

2.2.2.2 Assertion Types

The WS-SecurityPolicy specification structures its security assertions into the following (coarse-grained) families:

- Binding assertions
- Token Assertions
- Protection assertions
- SupportingTokens assertions
- WS-Security 1.0 and 1.1 assertions
- WS-Trust 1.3 assertions

Each assertion category defines a set of assertions used to characterize - that is specify, assert and constrain - different aspects of the *syntax and semantics* of the SOAP security header (visible policy behavior) and security mechanisms applied (invisible policy behavior), respectively, while performing a security transaction. Assertions may be nested (see Section 2.2.1 above), hence may be used to further qualify specific aspects of other assertions. This means, that individual assertions are meant to be used in multiple combinations. Each of WS-SecurityPolicy's assertion types defines nested assertions for assertion customization. The assertion concept - and in particular the notion of nested assertions - have the virtue to shift parameterization from XML element values to the simple occurrence of (XML Schema-compliant) XML elements as such and combinations thereof (i.e. not necessarily having any value). Instead of further qualifying XML elements with content that they include (i.e. their value), nested XML elements by themselves (that is, assertions) are used to parameterize ascending assertions, which allows to automatically validate parameter syntax against the WS-SecurityPolicy schema definition more effectively: where this makes sense, assertions in WS-SecurityPolicy try to rely on as little parameters as possible and policies are configured in terms of nested assertions, chosen from predefined collections. This enables first-order QName-based assertion matching to compute a composite collection of policy alternatives of two parties that try to establish a secure communication path.

For the association of security policies with WSDL service descriptions, WS-SecurityPolicy uses a subset of three of the four extension points that WS-PolicyAttachment defines: the *End-Point*, *Operation* and *Message* subjects. The policy subjects that assertions are meant to be scoped at, are defined in the appendix of WS-SecurityPolicy specification. Assertions scoped at the end-point subject indicate context information (that is, tokens and their usage) needed to secure the entire set of messages described for an end-point inside the WSDL, such as binding patterns utilized (i.e. binding assertions) for mediating the security transaction, accepted supporting tokens (i.e. supporting token assertions containing enclosed token assertions) as well as WS-Security SOAP-header and WS-Trust specific qualifications. Assertions (and enclosing security policy) subjected to the operation extension point, designate token usage on a per-operation basis for operations defined inside the WSDL, including (exclusively) XML-level binding assertions and assertions on supporting tokens necessary for usage of the operations. Finally, assertions scoped at the level of a message mark characteristics on the message type inside the WSDL, such as which parts of the message need to be encrypted and which portions should be signed (i.e. protection assertions). Note, that the set of nested assertions defined in WS-SecurityPolicy are not specific to any WS-PolicyAttachment extension subject (leaving a

policy's scope unaltered) and thus may be used as part of (scoped) ancestor assertions. WS-SecurityPolicy exactly specifies, which WSDL element types are valid to be used for policy attachment as well as to what extend they are valid to be used, based on the RFC-2119 best-practices.

2.2.2.3 Policy Properties

To account for policy parameters, the specification introduces *Properties* that are generally flagged by brackets (i.e. "[<Property>]") and that represent a "particular aspect of securing an exchange of messages". Also the policy subject is a particular aspect while securing an exchange of messages, i.e. [End-point Subject], [Operation Subject] and [Message Subject] are each represented as properties.In other words, properties contain the security context information of the security transaction a policy is supposed to mediate. The nature of a security transaction is determined by the security binding (scoped at the end-point subject), representing a common security usage pattern that yields a set of properties specific to that very binding. In other words, the set of properties specific to a particular **binding assertion** mandate how security is performed and what to expect in the `wsse:Security` header. `wssp:TransportBinding` assertions define solely a [Transport Token] property, as only a single token is needed for point-point protection, whereas a `wssp:SymmetricBinding` assertion is to be further qualified using the [Encryption Token] and [Signature Token] properties. `wssp:AsymmetricBinding` assertions in contrast have four binding specific properties: [Initiator Signature Token], [Initiator Encryption Token], [Recipient Signature Token] and [Recipient Encryption Token].

Thus, different sets of assertions map to different sets of properties needed during the security transaction. The same way bindings map to binding specific-properties, particular token assertions provide (assertion-)specific sets of properties for parameterization. Nested assertions determine the value of properties that are specific to parent or ancestor assertions, ranging from collections of pre-defined enumeration type-like constants, such as particular token types or available encryption/signature operations, to primitive typed values, such as booleans and strings representing URIs. The sum of properties specific to a policy (i.e. specific to its set of security binding alternatives) fully determines the context information required for the (binding-specific) security transaction. Thus, the aggregation of all assertions of a policy completely configures WS-SecurityPolicy's properties in such a way, that the visible WS-Security runtime behavior is derivable: the minimum number of tokens required and how they are bound to messages, key transport mechanisms (if any), required message elements and the ordering and content of the resultant message security header, to name a few.

Nested `wssp:AlgorithmSuite` and `wssp:Layout` assertions together with the family of **token assertions** form conditional assertions for general aspects or pre-conditions of the security(-binding), the latter mainly qualifying the token types used during the transaction. WS-SecurityPolicy defines extension mechanisms, allowing to introduce custom XML and binary tokens that may be used in combination with existent assertion types. **Protection assertions** specify mandatory message elements (e.g., timestamps), header to be integrity protected, as well as which message parts or elements have to be signed/encrypted, including SwA attachments. In other words, they determine what is being protected and the level of protection provided. Since security bindings use the tokens specified by the binding policy to create a message signature,

and because WS-Security supports protection also at transport-level, transport bindings will result in the transport protocol signing the message outside the message XML. Hence the signature itself will not be part of the message. Therefore, additional supporting tokens may be specified by a security binding to augment the claims provided by the token that is associated with the message signature. **SupportingTokens assertions** may define token usage patterns and may be differently scoped compared to the binding that references them, such as supporting tokens defined at the scope of a message with corresponding binding defined at the scope of an end-point, for instance. Overlapping scopes are being merged, as defined in the WS-Policy framework and clarified in the WS-SecurityPolicy specification.

WS-Security 1.0 and 1.1 assertions mandate optional aspects of WS-Security: SOAP Message Security that are independent of the trust and token taxonomies, and MUST be scoped at the End-Point policy subject. These assertions target token reference capabilities of initiator and recipient and therefor indicate which WS-Security token reference options are supported by a particular service end-point. Finally, **WS-Trust 1.3 assertions** represent options in the exchange of security context information involving WS-Trust's STS authorities, including supported WS-Trust-specific header syntax, as well as statements of client and server challenges and an indication of the entropy behavior, that is, determining if the server/client's entropy is used as key material for a requested proof token versus not using the entropy for key derivation.

CHAPTER 3

Implementation

"That which is below is like that which is above that which is above is like that which is below to do the miracles of one only thing."

Hermes Trismegistos[1] ***[89]***

According to the SOA Reference Architecture [160], the "governance of SOA-based systems requires an ability for decision makers to be able to set policies about participants, services, and their relationships. It requires an ability to ensure that policies are effectively described and enforced." Thus, SOA governance implies the accumulation and management of services policies and a way to select IT-level policies according to business requirements. The management of the QoS of services requires machine processable policies that instruct a middle-ware in a way, that the solution's (visible and invisible) runtime behavior complies with the constraints defined by requirements that the policy is supposed to cover. A policy compliance validation has to be performed. In particular in the context of security, requirements are frequently issued as natural language specifications. In absence of a formal model, ambiguity is inherently introduced and renders automatic validation difficult. Examples for natural language requirements are "protect credit card information", "use strong encryption" or "BSP Compliance". In this Chapter we distill a policy authoring model and validation approach that enables the reuse of policies(-expressions) by composing abstract requirement policies and performing a subsequent validation of IT policy candidates against schematron rules expressing these abstract requirements at the more concrete operation level. If the results of a validation indicate validity against the rules, the operational policies can safely be selected.

Besides of assurances that policies are effectively described and enforced, a policy validation can support the policy authoring process and improve the ability to set policies about services in SOA. By analyzing the validation results of the SVP, we establish a policy mediation mechanism between Operation and Architecture level (compare Chapter 1 once more), that eases security

[1] Isaac Newton's translation of the Tabula Smaragdina in his alchemical papers (c. 1680)

policy development and enables also users that are not necessarily security experts to manage security with the help of policies. The SVP confers horizontal and vertical policy hierarchy compliance validation and enables the selection and/or development of operational XML policies escorted by a semi-automatic mediation through architecture policy schematrons. In order to explain this guided XML-policy mediation approach in conjunction with the BSP conformance case study, this Chapter starts with some background on the WS-I profiles and Schematron validation. Afterwards, the actual implementation of the BSP architecture requirement is explained. Finally, the policy authoring methodology as well as the SVP mediation are being discussed.

3.1 Background

3.1.1 WS-I Conformance

While functional services contracts (WSDL) support interoperability between developer tools, runtime interoperability is conferred by non-functional contracts (WS-Policy) and WS-I profiles conformance. Profiles are devoted to establish and enhance dynamic interoperability between different vendors of Web Services middle-ware and define "clarifications to and amplifications of" [50] standardized specifications. A (standardized) specification is generally bound to a specific technology and defines a set of *requirements* on this technology. Profiles are therefor specifications on specifications, i.e. meta-specifications. A technology evolves, so do the corresponding specifications. The *scope* of a specification may encompass one or more extension points of other specifications, meaning that these are being referenced within the very specification. Specification references are usually categorized by RFC-2119's [69] best-practices when referencing other specifications, indicating the 'strength of reference'. In other words, references among specifications bind them to each other at different magnitudes, called *requirement levels*. In addition, specifications categorize requirements that they define into these requirement levels, so indicating the 'strength of requirement'. This is how BSP uses the levels for its requirements. These facts result in both, different strengths for specific requirements within a specification and different strengths of references to other specifications, that in turn define requirements on different levels, and so on. On the one hand, this allows self-containedness of individual specifications and a flexible means for specification extension, while just a subset of the specifications may be chosen that is needed to establish the desired SOA capabilities. On the other hand, exact adherence to these standards is rendered complex.

To convey the scheme of profile requirements, consider the following sample, that defines a conditional requirement on a profile's *conformance target* with exactly one respective RFC-2119 requirement level:

R9999 Any **WIDGET** SHOULD be round in shape.

Conformance targets, such as the "**WIDGET**" target of requirement **"R9999"**, reference SOA artifacts and are printed in bold and capital letters within WS-I profiles, while the requirement level is specified with capital letters solely. Concrete profiles list all conformance targets that they reference in non-bold letters in a dedicated section, identifying the applicable SOA artifacts for the individual requirements. BSP (mainly) targets runtime SOAP artifacts, and conditions

concern the *syntax* of these targets, i.e., grammar of these XML elements and their content. Mostly, concomitant text illuminates the requirements, but must not be considered for validation purposes. Missing to meet this particular fictitious sample requirement might in some situations still maintain profile conformance, indicated by the "SHOULD" level. Claims of conformance against WS-I profiles can be asserted using the Conformance Claim Attachment Mechanisms [15], by specifying the appropriate conformance claim URIs. Version 1.0 of the BSP profile incorporates more than 20 external specifications by reference and clarifies their usage with Web Services, referred to as the BSP profile's *conformance scope*. Thus, a conformance scope - in accordance with the notion of specification scope from above - is not essentially "solid", due to different levels of references and requirements.

3.1.1.1 Profile Conformance

The BSP profile predominantly references and clarifies WS-Security, providing guidance for SOA practitioners in implementing and configuring their service-oriented solution more interoperably and securely. This is necessary because inconsistencies between multiple Web Services engines can arise due to wrong or assailable content and structuring of the WS-Security portion of messages headers, despite and on top of conformance of artifacts to the WS-Security specification and its respective XSD. Evoked by divergent interpretation of the security standards, interoperability is inhibited or even canceled out and the door is opened to a variety of security vulnerabilities. Hence, BSP clarifies how to correctly apply WS-Security, aimed at diminishing these risks, by imposing further constrains on the syntax of (yet schema-constrained) WS-Security, besides the constraints already defined by WS-Security's XML grammar and object model. BSP's WS-Security requirements are concerned with the requirement levels "MUST", "SHOULD", "MUST NOT", and "SHOULD NOT", while the former two are conveniently expressible in terms of the ladder two, through simple logical negation. Besides requirements, BSP defines so called considerations (Cxxxx) that equally follow the requirements scheme from above. The remainder of the specification states natural language recommendations in addition to profile requirements and considerations that follow no particular scheme and are not explicitly assigned a profile code.

Besides promoting interoperability, adhering to the best-practices of BSP helps the prevention [50] from (some forms of) security token substitution attacks, security token replay attacks, leaking user name token passwords and plain text guessing attacks. One of BSP's guiding principals states that BSP conformance provides no guarantees of interoperability. With respect to secure Web Services communications it is coevally true that BSP conformance is in no way enough to guarantee for security when applying WS-Security for a security transaction, as evidenced by e.g. [122, 239]. Nevertheless, valid runtime WS-Security syntax with respect to BSP is an absolute necessity for more secure interoperability and has to be assured by a *conformance validation*, the same way as a validation of functional and non-functional XML contract artifacts should ideally be performed implicitly by the messaging middle-ware, using the schema definitions of the respective Web Services specifications. Therefor, first of all, problem (a) stated in the introduction (compare Section 1.3) has to be solved: the BSP must be formalized into a machine processable representation.

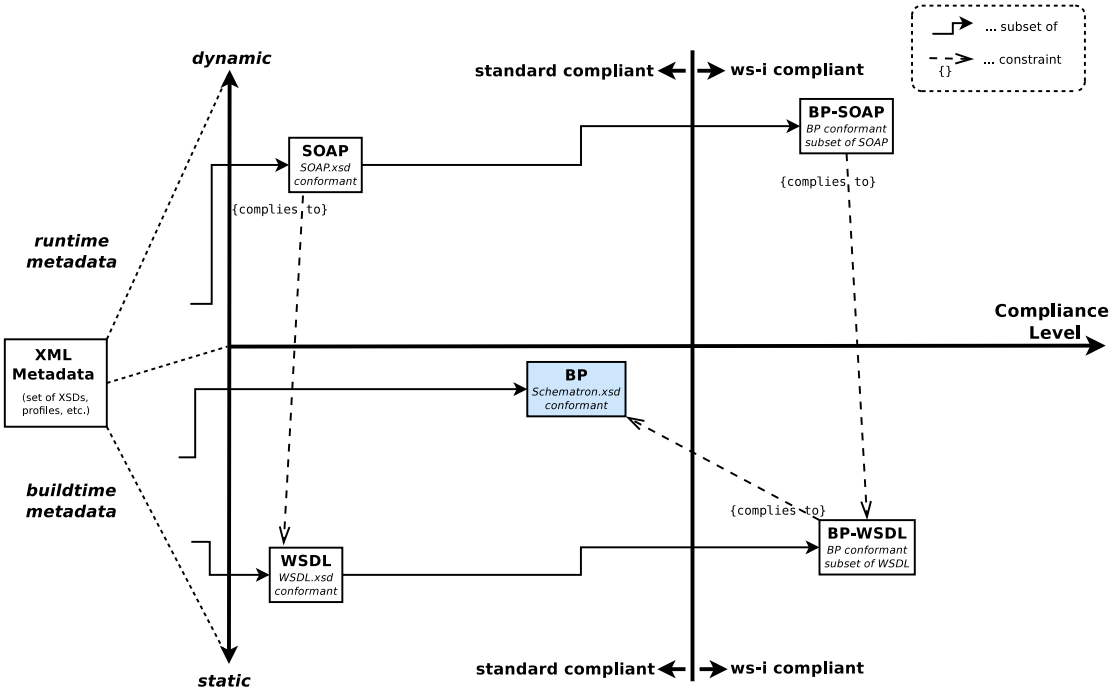


Figure 3.1: WS-I Compliance in the context of BP Validation

BP Conformance Validation As mentioned in the *Projects Background and Context* preamble, we can fall back on a validator engine in use with the WSSR, which also supplies a formalization of the BP in Schematron. In other words, the existent implementation provides us with a solution to problem (a) with respect to the BP specification. We therefor consider the BP profile first, to devise the methods for BSP validation. Take a look at Figure 3.1 that reveals the design-time and runtime aspects of BP validation in the context of Web Services standards compliance. The existent BP schematron formalization corresponds to one instance of the exemplified `BP` artifact class. The dashed arrows representing constraints on meta-data are meant to convey the interrelationships XML artifacts, whereby the "x-axis" stands for the meta-data's level of compliance with respect to schema conformance (standard compliant versus WS-I compliant). The diagram distinguishes between build-time (lower half-plane) and run-time (upper half-plane) meta-data artifact classes, of which each is a subset of the class of all possible XML metadata that is conformant against the respective XML Schema definitions, as indicated by the "subset of" relations. WS-I compliant artifacts reside in the profile compliant "bin" of the compliance level dimension and are likewise subsets of the respective standard compliant parent classes. The boundary between static and dynamic meta-data is to be interpreted as the transition from build-time to runtime, which can be thought of as the act of service deployment. Dynamically conformant meta-data is shown on the half-open plane in upper part of the diagram, which is produced as a result of deployment of the respective static meta-data depicted at the bottom (observe the `complies to` constraints).

Obviously, `BP` is standard compliant, in the sense that it complies to the Schematron stan-

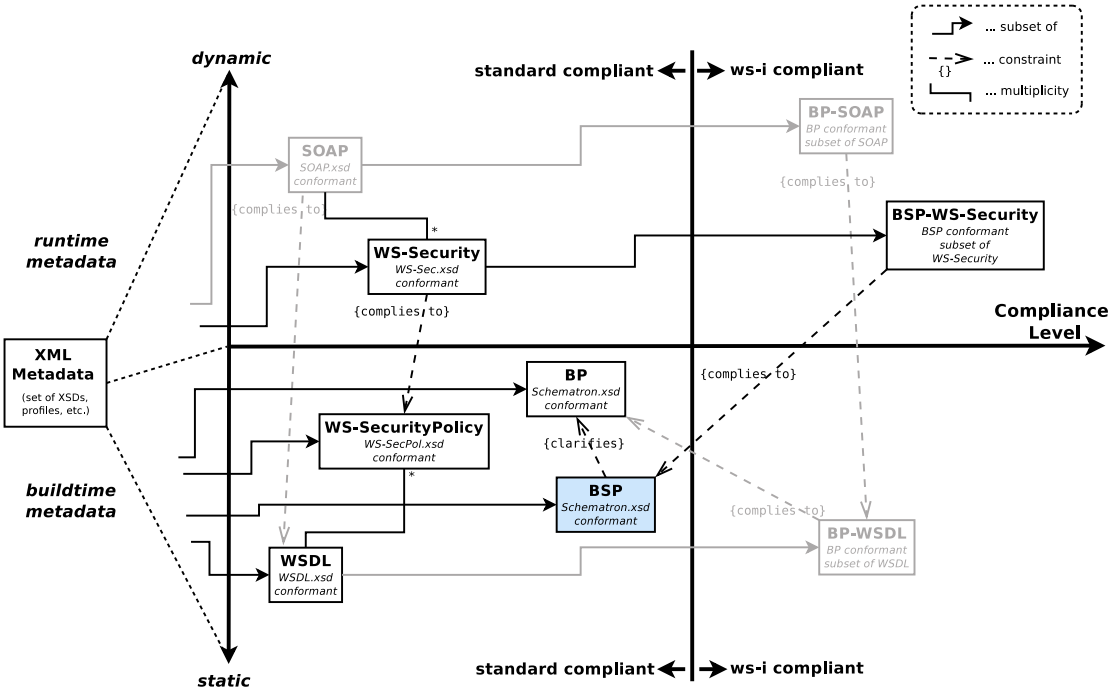


Figure 3.2: WS-I Compliance in the context of *dynamic* BSP Validation

dard (i.e. *Schematron.xsd compliant*). Static WSDLs and dynamic SOAP messages must each adhere to their respective XML models, i.e. must be *WSDL.xsd compliant* and *SOAP.xsd compliant*, respectively, in order to exhibit standard compliance. The class of WS-I compliant WSDL descriptions (i.e. `BP-WSDL`) on the other hand, must additionally comply to the Schematron formalization of the BP (again note the *complies to* constraint). The BP artifact class means Schematron instances that contain BP's requirements in the form of schema-constraint rules on WSDL, such as the one for the WSRR. Thus, the BP schematron provides rules that go beyond the schema constraints defined by the WSDL standard itself. Therefor, as a result of deployment of a `BP-WSDL` instance, the SOA runtime will produce SOAP messages that are BP conformant (compare `BP-SOAP` artifact class and `complies to` constraint). This amounts to a subset of the `SOAP` artifact class complying with `BP-WSDL`. Thus, by nature, it is possible to validate BP solely within the WSDL services descriptions.

Dynamic Validation of BSP Conformance Though both BP and BSP are devoted to runtime interoperability across (multiple) platforms, in terms of how validation has to occur, they in fact significantly differ: while BP conformance requires an endpoint to use *statically* BP-compliant WSDL contracts that when deployed produce *dynamically* BP-compliant SOAP messages, as just outlined, BSP in contrast speaks about very specific - and *exclusively dynamic* - WS-Security SOAP header syntax requirements. See Figure 3.2 that illustrates relevant constraints and artifact classes in the context of runtime BSP validation. SOAP may incorporate WS-Security conformant XML at runtime, given a WSDL provides build-time WS-SecurityPolicy policies determin-

ing the runtime behavior (both indicated by the "*" multiplicity of the respective associations). For BSP conformant SOAP message exchanges it is necessary that each message complies to all of BSP's requirements. A Schematron instance of the `BSP` artifact class must be found that constrains the universe of all possible WS-Security syntax that can be included in a SOAP message, to a BSP conformant subset of the same. This subset is exemplified as `BSP-WS-Security` in the diagram. Observe the `complies to` relation between `BSP-WS-Security` and `BSP`.

Figure 3.2 conveys that from the BSP specification only, it is impossible to test conformance to BSP's requirements before deployment of a service, while on the other hand, BP's requirements can be checked in the static WSDL contract. Recall that in Figure 3.1, BP only constrains metadata that resides on the static half-plane of the diagram. On top of that, the BSP profiles the BP specification by itself, clarifying some of its requirements (see "clarifies" relation). All this renders validation of the BSP far more complex, and above all more costly, since it is unavoidable to check message traffic at runtime, whereas BP has no relations to the runtime space. Due to BSP constraining - in majority - runtime WS-Security metadata, validation of conformance must be performed directly on the wire, checking if its requirements are fulfilled for every message exchanged over communication channels. WS-I issues additional deliverables for testing purposes, including a BSP test assertions document [244] that defines machine tractable rules for BSP's requirements that are verifiable from the SOAP envelope at runtime. Thus, with the restatement of these rules as Schematron on dynamic WS-Security-augmented message exchanges, we can provide the implied solution for problem (a) with respect to the BSP. In Section 3.2.1 we supply the details on how we intercept the message traffic for dynamic validation against BSP.

Static Validation of BSP Conformance Keeping in mind that policies are meant to mandate the behavior of the Web Services mediation architecture (visibly or invisibly), and thus how corresponding runtime QoS specifications are being used, it gets apparent that BSP does quiet the same thing as a WS-SecurityPolicy policy does: both shrink the valid set of SOAP messages with respect to the legal universe of WS-Security header metadata. This is where our static policy compliance approach comes into play: take a look at Figure 3.3, revealing the partial congruency of BSP with WS-SecurityPolicy. The picture conveys that WS-SecurityPolicy holds the possibility to validate BSP requirements indirectly within the policy, rather than checking message traffic. Though most of BSP's WS-Security requirements for runtime interoperability appear too specific to be captured by WS-SecurityPolicy's static security assertions, we were able to identify a subset of the requirements that we can validated solely in the policy, if one exists. This however implies (unambiguous) inference of the resultant (visible) runtime behavior from policies. With such a revision of the BSP, the BSP as meta-specification of the WS-Security standard becomes a policy about WS-SecurityPolicy policies, i.e. a meta-policy.

Despite of the affinities of profiles and policies, a major difference between a policy based on WS-Policy's model and the BSP runtime profile is the degree of ambiguity introduced by their respective specifications. In the first place, profiles completely lack in a formal model (i.e. do not ship with a schema) and state requirement conditions generally in natural language, while applicable artifacts are presented in a semi-structured manner, as seen above. This naturally introduces ambiguity and results in the need for BSP requirements to be unambiguously

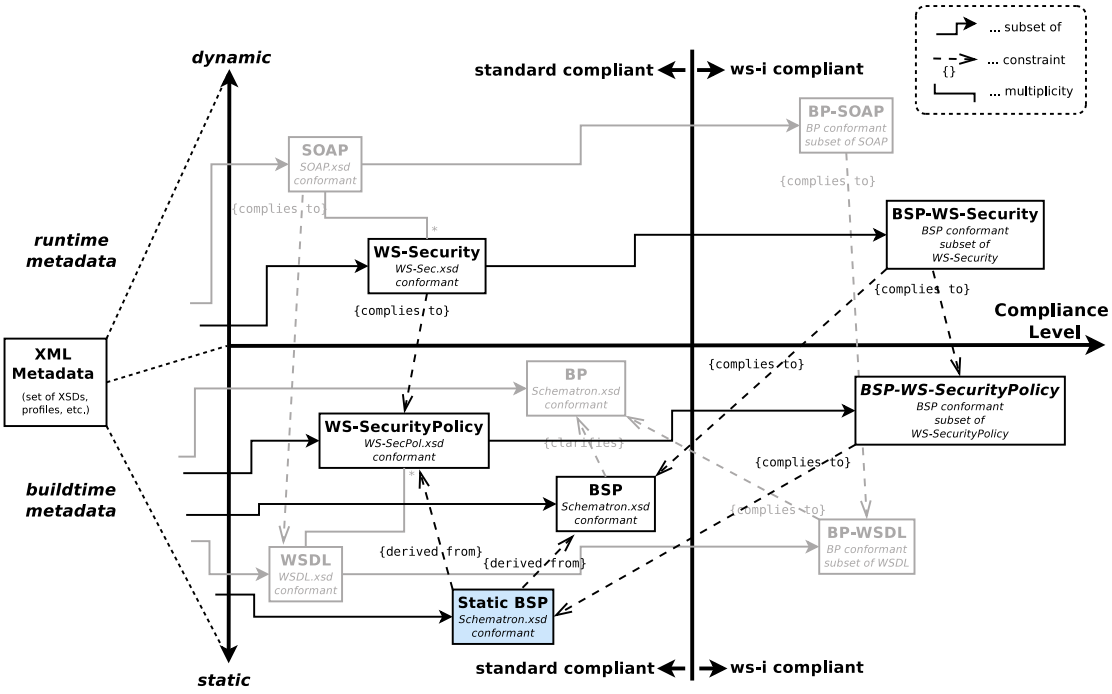


Figure 3.3: WS-I Compliance in the context of *static* BSP Validation

formalized (i.e. translated into a machine-processable representation); WS-I delivers a major part of this work with its test assertions document for runtime message exchanges. However, the validation of BSP within a policy requires to re-express BSP's dynamic WS-Security requirements as static requirements on WS-SecurityPolicy syntax. This complicates the situation: consider the "derived from" constraints of the "Static BSP" artifact class of Figure 3.3, that inherits from both, the BSP profile and WS-SecurityPolicy by translating BSP's runtime requirements to schema-constraints on the WS-SecurityPolicy's grammar and object model. This is contrasted by dynamically examining each message originating form or targeting an endpoint, the method of checking BSP that is implied by the profile itself.

In the diagram, an instance of `Static BSP` contains build-time schematron policy validation rules with no relations to dynamic artifact classes: observe that there is only the relation to the (static) WS-SecurityPolicy class of metadata. In Section 3.2.2 we explain details on how we are able to provide this more sophisticated solution to problem (a), implementing a static revision of a subset of the BSP in Schematron on WS-SecurityPolicy policies, i.e. a BSP meta-policy expressed in Schematron. We can validate a subset of BSP's dynamic requirements in an entirely static fashion, given we are able to eliminate ambiguity in the translation of BSP to policy grammar. In fact, through this method the relationship of BSP to XML artifacts is being conformed to the relationship the BP exhibits with its corresponding SOA artifacts, i.e. WS-SecurityPolicy takes the role for the BSP that WSDL takes for the BP.

3.1.1.2 Standardized Profiles vs Application Profiles

In addition to the necessity to restrict the usage of standardized (and machine-processable) SOA security standards, it makes sense that profiles like BSP are not defined within or along the lines of the standards and the schema definitions that they constrain (WS-Security in case of BSP), but as a separate specification. One reason for this is the fact that activated schema validation is a trade-off, in particular runtime validation: in many middleware solutions XSD schema validation is indeed often simply turned off by default, to avoid overhead in support for a better message processing performance. Even if activated, it is unlikely that schemata for WS-I conformance can be accurately stated using the XML Schema language. One the one hand, this is because it is unintuitive and difficult - if not impossible - to use grammar-based languages, such as XML Schema, to formalize most profile requirements (e.g. co-occurrence constraints). On the other hand, even if expression in XSD is possible, most default parser configurations, if at all configured to perform XSD validation, won't support processing XSD constraints by default that are able to capture (some of) the very specific requirements of the WS-I recommendations. Given the parser does perform accurate XSD validation of those requirements which are (theoretically) expressible in XSD - and thereby ensuring conformance of the WS-* XSD in question against (runtime) profile requirements - we end up with processing overhead that is far too costly. XML artifact constraints required for cross-platform runtime interoperability must therefor be taken care of by a more economic and expressive/flexible means.

Consequently, one has to strike a balance between overly hard runtime schema constraints on the one hand, and the processing performance on the other hand: strict schema validation by default versus freedom with respect to (supposedly valid) WS-Security syntax at runtime, the latter increasing with loosening the restrictiveness of the schema constraints. Thus, it is desirable and obviously intended by WS-I to provide the interoperability profiles as best-practice references independent form the basic compatability standards. The profiles' (automated) validation may be conducted or not, or conducted at different levels of compliance, contingent upon the context and services demands. If a service is critical, one will want to have hard assurances of WS-I compliance, however in other cases this might not be needed and validation overhead can be spared. In any case it is true that build-time assurances on SOA artifacts compliance (and thereby assurances on the compliance of underlying tools) does not suffer from the drawbacks of the schema validation pit-fall, in terms of runtime performance overhead. Static validation occurs only once (or a very limited number of time before "real-time"), while the runtime artifacts do not have to be checked, given guarantees can be procured, that the inferences of the behavioral dynamics from available static structural descriptions (i.e. contracts) are in fact correct, and, given the respective profile admits such inferences. These assumptions can be checked off-line, validating compliance of each SOA runtime platform individually, i.e. ensuring that the middle-ware's interpretation of the standard to be validated is consistent with the derived validation rules.

Moreover, the demand for some user interaction and a semi-automated process of compliance validation gets apparent: a developer will want to choose the level of compliance individually for each of her solutions, by means of some guided schema validation, that allows her to adjust validation to her needs. This does not only apply to standardized best-practices such as the WS-I profiles, but also to requirements beyond standard conformance as such: specific

(best-practice) *application profiles* [127, 131, 134, 150] could mandate conformance against individual legislative requirements or collaborative objectives that can be accounted for in terms of constraints in the usage of SOA security standards. For example, a medical institution is usually obligated to anonymize patient data, or may be committed to encrypt/sign specific portions of the data. An application profile in this context provides the best-practice (in-house or custom) reference for securing services within the institution's SOA environment, the same way WS-I profiles provide such a reference for cross-platform WS-Security interoperability. In fact, WS-I profile compliance is likely to be part of an organization's SOA security standards application profile. Thus, the methods for (meta-data) application profile compliance validation largely align with methods capable of procuring standard compliance validation against WS-I's profiles. In allusion to the considerations regarding meta-policy, BSP is turning into a standardized (meta-)policy application profile, so to speak, whose conformance can (in parts) be accounted for by a policy validation. In Section 3.3, we show how to use the SVP mediation and authoring approach to integrate multiple abstract (security) policy (application-)profiles apart from and on top of the static policy reformulation of the BSP.

3.1.2 Schematron Validation

For the solution to problem (b) stated in the Introduction (compare Section 1.3) we leverage the powerful ISO standard language Schematron [92]. Schematron is part of DSDLs and facilitates XML validation beyond XML Schema conformance, being rich enough in expressiveness to meet our validation needs. Schematron is a rule-based language and allows the definition of conditional assertions on the *syntax* of XML documents, that produce explanatory output in case of assertion *failure* and/or *success*. At many capabilities Schematron is superior compared to other schema(-constraint) languages due to its ease of use, while supporting complex schema constraints that other such languages lack, such as Relax NG [107]. Amongst others, schematron allows for conditional element-attribute checks and supports validation of identity constrains. To give an example, consider an XML document that defines a price and rating attribute as children of a product element, that is descendant of an order element that contains a date and payment method attribute. With schematron it is easy to define a schema-constraint that checks for products with a small price and a high rating, or vice versa, that have been payed with credit card and ordered before a certain date. It is unintuitive and overly complex (if not impossible) to express such constraints using grammar-based schema language like XSD/RNG. Schematron capabilities however go far beyond such simple examples and we will go into details of the language in Section 5.

3.1.2.1 Schematron's Object Model

The document root of a `.sch` document - the widely accepted suffix for Schematron documents - is the `schema` element that is part of the schematron namespace. The schematron object model further incorporates `pattern` elements, that consist of `rule` elements which in turn consist of `assert` and/or `report` elements. `assert` and `report` elements both contain natural-language assertions as its XML element value and both constitute a formal representation of the respective natural language assertion that is amenable to automated *assertion tests*. Each

assertion test can either "succeed" or "fail". Optionally, schematron incorporates `diagnostic` elements that are meant to encourage a more clear statement of natural-language assertions, i.e. the element content of `assert` and `report` elements. A rule is evaluated within a specified XML document context, that is defined with the help of the rule's `context` attribute - an XPath [74] expression selecting the correct XML context from the document model for validation. Both, `assert` and `report` elements define a `test` attribute used to perform checks within the rule's `context`. Also the `test` attribute is of type XPath and is queried against the contextual document selection that has previously been extracted using the rule, while being subsequently evaluated to a boolean: the query result is either empty (*false*) or not (*true*).

To put it differently, `assert` and `report` elements make rules fire in case of *exceptions* and *compliances* of the test checks, respectively. In other words, they are inverse to each other: if an `assert`'s test fails, the same test run as a `report` would succeed and vice versa. An `assert` element produces its natural-language assertion messages if its `test` attribute evaluates to *false* and `report` elements fire if their `test` attribute evaluates to *true*, resulting in both of the respective assertion tests to succeed in this case. Rule exceptions and compliances of patterns are made available in a schematron output document, including a (potentially human readable) reason for the failure in Schematron Validation Report Language (**SVRL**) format (part of ISO standard) or any other customized report language (by a Schematron transformation chain extension, as detailed later). See Listing 3.1 for a minimal schematron on schematrons that validates (part of) its object model by itself. The example is self-explanatory when able to handle XPath, and obviously primitive, since reporting only what elements are missing in an empty schematron schema (compare the `context` attribute in line (5)). Nevertheless, this schematron conveys the fact that there are many valid families of schematron schemata that conform to this very schematron schema. This means, we can express $1 : n$ relationships adequately, leveraging the rich expressiveness of (the subset of) XPath that schematron supports. In order to enable namespace aware validation, Schematron may define *namespace* elements, responsible for inclusion of external schemata (compare line (3) in the Listing).

Listing 3.1: A limited schematron that validates Schematron's object model by itself

```
1  <schema xmlns="http://purl.oclc.org/dsdl/schematron">
2    <title>A Schematron Mini-Schema for Schematron</title>
3    <ns prefix="sch" uri="http://purl.oclc.org/dsdl/schematron">
4    <pattern>
5      <rule context="sch:schema">
6        <assert test="sch:pattern">
7          A schematron schema contains patterns.
8        </assert>
9        <assert test="sch:pattern/sch:rule[@context]">
10         A pattern is composed of rules. These rules should have context attributes.
11       </assert>
12       <assert test="sch:pattern/sch:rule/sch:assert[@test] or sch:pattern/sch:rule/
             sch:report[@test]">
13         A rule is composed of assert and report statements. These rules should have
               a test attribute.
14       </assert>
15     </rule>
16   </pattern>
17 </schema>
```

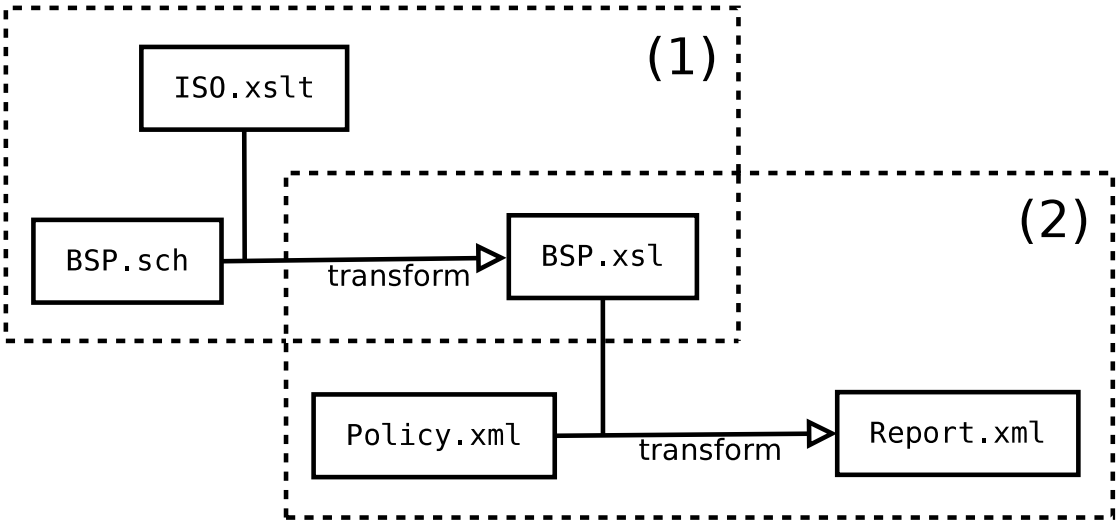


Figure 3.4: The Schematron Validation Pipeline.

3.1.2.2 Schematron Validation Pipeline

Figure 3.4 shows a conceptional view of the SVP, revealing that the SVP expects two inputs (schematron and policy) and produces a single output (report). Though a compacted view, the picture conveys the principals of the schematron validation process, resulting in a SVRL report document. As indicated by the two dashed frames on the diagram, the validation can coarsely be divided into two stages: the SVP gets an input schematron document, depicted as `BSP.sch`, which is (in the simplest case) translated into an XSLT style sheet - called `BSP.xsl` - by an execution of a chain of document transformations in stage (1). The diagram depicts this standardized base-line transformation chain as a single transformation, for readability reasons (`ISO.xslt`). In stage (2) this style sheet is itself used to transform a given policy (`Policy.xml`) into a report document (`Report.xml`) that contains assertion test failure (resp. success) statements for each violating (resp. compliant) schematron pattern defined previously inside `BSP.sch`. The resulting report contains SVRL compliant XML that may finally be parsed and feedback on exceptions/compliances may be propagated to an arbitrary destination.

A schematron schema may be interpreted as an Extensible Stylesheet Language Transformation (**XSLT**) [11, 27] abstraction and was designed with XSLT in mind. In fact, as seen above, Schematron's reference implementation is realized using XSLT style sheets, while no XSLT knowledge is needed to use the Schematron API itself. As exemplified in the SVP diagram, a schematron is itself compilable into a concrete XSLT style sheet through the application of a chain of normed XSLT transformations that are available from the schematron homepage (including the one that implements SVRL). Note that XSLT is only one option that naturally lends itself to implementing the transformation chain in a system-independent fashion. For the implementation of the dynamic BSP conformance validation we provide a minimal implementation in Java using AXIOM [44]. Quite recently, the report transformation of Schematron's ISO XSLT skeleton API now supports both version 1.0 and 2.0 of XSLT. We utilize XSLT 1.0 for both the

custom dynamic and the standardized static Schematron validation of the BSP, as this will often be the only available version in security infrastructures, due to vulnerabilities risks introduced by advanced XSLT 2.0 functions.

The reference implementation utilizes the XPath expressions that are specified within the schematron schema's `rule` and `assert`/`report` elements to extract relevant portions from the document to validate. It is designed in a way that allows for extensions of both, the validation behavior and the output of the SVP, by plugging custom XSLTs in between the transformation chain, utilizing the "super" or "parent" XSLT transformations of the skeleton API. See Listing 3.2 that shows how to invoke the base-line pipeline, assuming the `xslt` command to call an appropriate XSLT processor that supports the XSLT version used by the schematron. Stage (1) apparently spans the first three lines of the Listing while stage (2) corresponds to the last transformation invocation. The third transformation of pipeline stage (1) can be used to alter the XSLT version used for generating the SVRL report, indicated by the N in `iso_svrl_for_xslt`**N**`.xsl`.

Listing 3.2: Invocation of base-line SVP

```
1 xslt -stylesheet iso_dsdl_include.xsl BSP.sch > BSP1.sch
2 xslt -stylesheet iso_abstract_expand.xsl BSP1.sch > BSP2.sch
3 xslt -stylesheet iso_svrl_for_xsltN.xsl BSP2.sch > BSP.xsl
4 xslt -stylesheet BSP.xsl Policy.xml > Report.xml
```

3.2 BSP Conformance Validation using Schematron

The machine processable representation WS-I provides with its test assertions document naturally deals with dynamic security header syntax in SOAP message exchanges, as BSP defines compliant WS-Security syntax. Hence, BSP is not amenable to static policy management innately (and therefor not amenable to our authoring approach). Therefor, runtime policy validation is necessary in these cases, where no build-time policy is available that instructs the runtime platform as required or where build-time policies are impossible to validate. In other words, if static policies and dynamic security metadata are both defined formally in XML, adherence to requirements of BSP comes down to a static versus a dynamic syntax validation of XML policies versus SOAP message exchanges. We can provide a powerful means for both approaches using Schematron. Static validation is generally the preferable approach, not the least because it is avoiding significant runtime overhead, as evidenced by the evaluation in Chapter 4. We use Xalan as the XSLT processor of our choice and leverage Schematron's ISO reference implementation of the SVP for static policy validation/mediation. Additionally, we reimplemented a subset of the Schematron standard using Jaxen layered on AXIOM [44] for the dynamic BSP conformance validation, in order to gain XML processing performance when checking the WS-I's validation rules on the wire. In the course of the next Sections, we will deal with the following two simple BSP requirements for demonstration purposes:

R5404 Any **CANONICALIZATION_METHOD** Algorithm attribute MUST have a value of "xml-exc-c14n" indicating that it uses Exclusive C14N without comments for canonicalization.

R3033 Any **X509_TOKEN** ValueType attribute MUST have a value of "X509v3".

Note, that we utilize more of a compact pseudo syntax in examples with several abbreviations, omitting irrelevant syntax.

3.2.1 Dynamic WS-Security Validation of BSP Conformance

Consider BSP requirement R5404, which demands that a runtime WS-Security SOAP header includes a `ds:CanonicalisationMethod` element with an `Algorithm` attribute of `"xml-exc-c14n"`. This corresponds to the usage of exclusive C14N to obtain a normalized form of the XML model without comments. See Listing 3.3, line (6) for a valid WS-Security header fragment with respect to R5404. Note that parts of the header that are not relevant for the example have been omitted.

Listing 3.3: Valid WS-Security example header with respect to R5404

```
1  <wsse:Security xmlns:wsse="..." soap:mustUnderstand="1">
2    ...
3    <wsse:BinarySecurityToken ValueType="X509v1" wsu:Id="CertId-148082">MIICT...</
         wsse:BinarySecurityToken>
4    <ds:Signature xmlns:ds="..." Id="...">
5      <ds:SignedInfo>
6        <ds:CanonicalizationMethod Algorithm="xml-exc-c14n" />
7        <ds:SignatureMethod Algorithm="..." />
8        ...
9      </ds:SignedInfo>
10     ...
11   </ds:Signature>
12 </wsse:Security>
```

In order to verify that the right C14N method according to R5404 is specified, a simple comparison of the `Algorithm` attribute value has to be performed, which is straight forward for this security header (i.e. its value has to be `"xml-exc-c14n"`). As seen in the schematron on Listing 3.4, this is not the only schema-conformant location the `ds:CanonicalisationMethod` element (line(4)) is allowed to be placed, implying the need to check R5404 in different contexts. This is recognizable by the relative XPath the `pattern` uses as value of the `context` attribute (line (5)). Nonetheless, this requirement stays relatively artless to validate, even if verified in different contexts. The `rule`'s `context` query selects line (5-9) of Listing 3.3, while the `assert`'s `test` attribute checks the appropriate `Algorithm` attribute value (i.e. "xml-exc-c14n") of the `ds:CanonicalisationMethod` element (line(4) in Listing 3.3). Observe the inclusion of the XML-Signature, WS-SecurityPolicy and SOAP envelope namespaces in Listing 3.4 (line (2-4)).

Listing 3.4: Schematron implementation for dynamic R5404 compliance

```
1  <schema id="WS-I BSP" xmlns="...">
2    <ns prefix="ds" uri="..." />
3    <ns prefix="wsse" uri="..." />
4    <ns prefix="soapenv" uri="..." />
5    <pattern id="dynamic R5404 compliance">
6      <rule id="ExclusiveC14N" context="soapenv:Envelope/soapenv:Header/wsse:Security/
           ds:Signature/ds:SignedInfo">
```

```
7       <assert id="exclusive C14N" test="./self::ds:CanonicalizationMethod[@Algorithm
            ='xml-enc-c14n'] or "./self::ds:Reference/ds:Transforms/ds:Transform/
            wsse:TransformationParameters/ds:CanonicalizationMethod[@Algorithm='xml-
            enc-c14n']">
8         Exclusive C14N must be used.
9       </assert>
10    </rule>
11  </pattern>
12  ...
13 </schema>
```

Note that a `context` attribute with a (relative XPath) value of `ds:CanonicalizationMethod` alone would select any node-set that contains the element at the top level. Thus, a test attribute with a value of `@Algorithm='xml-exc-c14n'` alone would do the job in that case, sufficient to check the `Algorithm` attribute at the two schema conformant locations of the `ds:CanonicalizationMethod` element. However, since we know the exact occurrences of R5404 related WS-Security syntax, we take the approach to check for a disjunction in the test xpaths. When the SVP is invoked with the SOAP envelope and schematron depicted, no assertion failures are produced (as C14N method is valid in Listing 3.3), whereas SOAP envelopes defining no algorithm attribute at all or an algorithm other than exclusive C14N, would result in a violation of R5404 and the production of a corresponding SVRL error report.

When taking a look at R3033, things turn out to be more complicated. This is due to the nature of WS-Security's security token type definition. Within the WS-Security namespace, there only exists the `wsse:BinarySecurityToken` element (line(3)) facilitating the declaration of (binary) tokens. There is no way to know from the message solely, that this in fact corresponds to a X.509 token. In the example of Listing 3.3, the `ValueType` specified within the security token is `"X509v1"` which violates R3033, so in this specific case runtime violation can be ascertained (i.e. rule exception) and therefore the non-compliance with BSP. However, to assess validity is not possible (i.e. rule compliance). Even if the `ValueType` attribute would specify a value of `"X509v3"`, we could not safely declare the example header to be compliant against R3033. Simply comparing the attribute value in the same way as was suggested for R5404, will not guarantee that the element value of `wsse:BinarySecurityToken` is indeed a binary X.509 certificate. The element could specify any binary value and there is no way to verify from the visible behavior solely, that the value complies with the format of X.509 tokens and that it in fact is a valid certificate in the current security transaction. The SOAP envelope alone is not enough, as this would require decryption of the token. In the subsequent Section we will show how this requirement can in fact be validated by checking the corresponding WS-SecurityPolicy policy, if one exists.

3.2.2 Static WS-SecurityPolicy Validation of BSP Conformance

To validate WS-SecurityPolicy against BSP, i.e., in order to find static policy validation rules for dynamic BSP requirements and use them in our authoring approach, the projection from WS-SecurityPolicy to runtime WS-Security artifacts has to be analyzed. This mapping is likewise open to interpretation, and not homomorphic by nature, as being partially dependent on parameters that are unknowable until real-time, and, on top of that contingent upon the execut-

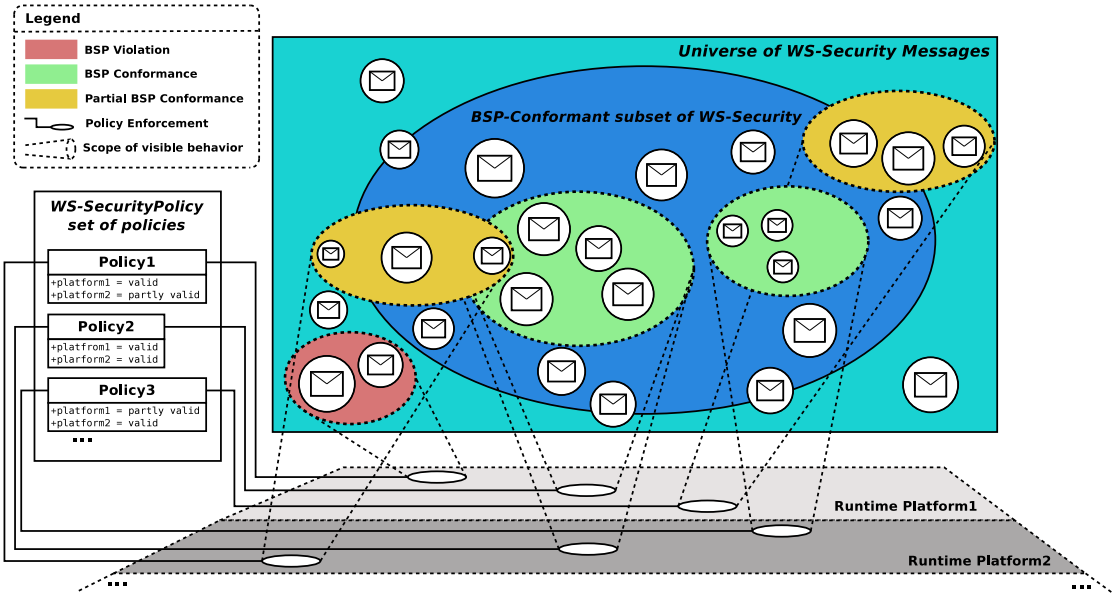


Figure 3.5: Mapping of design-time WS-SecurityPolicy policies to runtime SOAP messages.

ing runtime platform's implementation of the policy standard. Figure 3.5 tries to capture this issue, showing how BSP requirements define a secure and interoperable subset of the universe of all WS-Security-enriched SOAP messages. The diagram shows three sample policies of the set of all possible WS-SecurityPolicy-conformant policies, whose enforcement on two exemplified runtime platforms results in divergent projections to the WS-Security space of messages. WS-SecurityPolicy policies project to families of SOAP messages, in the sense that multiple `wsse:Security` headers conformant to the policy can be produced. If the collection of messages resulting from the deployment of a policy lies within the BSP-compliant subset of messages, i.e. if the WS-SecurityPolicy's scope of visible behavior lies within BSP's conformance scope, the policy can be declared valid against BSP. If all policies attached to a service are BSP-compliant, the service can be stated to be BSP-compliant as well. In order to identify every BSP requirement for which this is possible, we need to examine each requirement individually, which can be difficult and time consuming.

Policy1 in the diagram results in partially valid visible behavior on *Runtime Platfrom1*, and in an invalid scope of visible behavior on *Runtime Platfrom2*. *Policy3* is partially BSP compliant on *Runtime Platfrom1*, while being totally conformant on *Runtime Platfrom2*. The scopes of visible behavior of policies in reality often overlap, as signified for *Policy1* and *Policy2* on *Runtime Platform2*. *Policy2* is an example policy that is valid on both runtime platforms, while being projected to the exact same subset of WS-Security-enriched messages as a result of deployment on either of the two platforms. From these considerations follows, that all available runtime platforms have to be tested individually for compliance against (our interpretation of) the BSP-relevant portion of the WS-SecurityPolicy standard, i.e. against our policy validation Schematron rules for the BSP profile. In principal, identification of invalid policies on individual

platforms can be broken down to finding those WS-SecurityPolicy assertions (or a combination of these), whose occurrence or non-occurrence within a policy violates/validates a specific BSP requirement, i.e. the counterparts to (policy) exceptions and compliances, respectively. We distinguish the following cases where we can perform static BSP validation of WS-SecurityPolicy:

- One or more WS-SecurityPolicy assertions in the policy violate a BSP Requirement when specified (*type 1* rule)
- One or more WS-SecurityPolicy assertions violate a BSP Requirement when *not* specified (*type 2* rule)

Thus, a security policy can be statically validated by ensuring that none of the *type 1* assertions are contained and that all of the *type 2* assertions are part of a WS-SecurityPolicy policy. This is done by specifying a set of schematron rules that every security policy has to comply with. If the presence or absence of any assertion is violating one of these rules, non-compliance can be assessed. Of course, rules of *type 1* can encompass rules of *type 2* and vice versa. If all policies associated with a service comply with the BSP requirements that are amenable to static validation, we can declare the service itself - before wiring - as being BSP-compliant for that subset of requirements. Note that we found four BSP requirements to be innately valid, meaning that there is no instance of a WS-SecurityPolicy policy that can violate these requirements. Those can be thought of as being assigned a rule type of *type 0* - to keep consistency. Strictly speaking, these requirements cannot be covered by any rule and are not separately explored here, due to their trivial nature.

Note, that assertions can be part of an exclusive group of assertions, meaning that only one member of the group can be present inside a `wsp:Policy` element at one time. These assertions are grouped by a `xsd:choice` element in the underlying WS-SecurityPolicy XSD definition and mutually exclude each other within the same policy, i.e. WS-SecurityPolicy defines inclusion of exactly one of these policy alternatives. If such a group contains members of *type 1* assertions, one must check for (a) every of these assertion to be absent in a policy, in order to comply to the related requirement, that would otherwise be violated. This equals to a logical conjunction of these *type 1* assertions (logical AND). On the other hand, when validating a `wsp:Policy` element, it is enough for a group of assertions to contain just one member of a *type 2* assertion, in order to be able to (b) neglect all other assertions that are part of the same group (no matter of which *type*). This is because *type 2* assertions must be present in order to make a policy compliant to a respective requirement. So assertions of *type 2* that are part of the same group are disjunct and must be checked for by a disjunction of these very assertions (logical OR). This implies that, if a group of assertions contains both, *type 1* and *type 2*, there are two ways to validate: either check for the conjunction of the *type 1* assertions to be absent (clause (a) above), or check for the disjunction of the *type 2* assertions to be present inside the analyzed `wsp:Policy` element (as in clause (b)). Whenever possible, we take the (b) approach in our implementation.

BSP contains requirements that sometimes translate to validation rules of *type 0*, i.e. comply with BSP innately, in consequence of the definition of default values for properties specified in the WS-SecurityPolicy specification or specifications referenced by WS-SecurityPolicy. We list them in Appendix A.4.1 for completeness, together with the requirements we can validate within

WS-SecurityPolicy policies through inference. For each requirement corresponding assertion(s) to check are given, their rule type, the requirement level, as well as the document sections in the WS-SecurityPolicy specification from which the rules have been inferred. Clarification on the rules derived, as well as potential prerequisites and assumptions are given subsequent to each mentioned requirement. Note that we extracted two requirements from natural-language recommendation that the BSP defines, abbreviated by SEC_17_9 and SEC_17_13. We provide these rules in their composed form (i.e. rules that encompass other rules), as opposed to the "normalized" style we use for the ordinary requirements. The rule type determines if the respective assertion(s) must be present or absent, respectively, while assertions are to be grouped line by line using logical OR. For a minimal schematron validating all requirements we can cover statically, consult Appendix A.4.2.

Referring to Figure 3.5, let us see if we are able to find a mapping from static WS-SecurityPolicy policies to runtime WS-Security headers for the specific (rather simple) requirements R5404 and R3033 in such a way, that all WS-Security enriched SOAP messages producible through endpoints that are constrained by such policies, lie within the subset of R5404 and R3033 conformant messages. R5404 is a *type 1* rule, since checking for the *absence* of the `wssp:InclusiveC14N` element, whereas R3033 is of *type 2*, requiring the *presence* of one of the version 3 assertions defined for adequate X.509 certificate signature verification, as indicated in Appendix A.4.1. The two assertions used for R3033 are additionally an example of members of an disjunctive assertion group that we check for by logical OR. To assure that corresponding WS-SecurityPolicy assertions used for the validation of these requirements are interpreted uniformly among multiple vendor platforms, the runtime behavior resulting from the enforcement of that particular (set of) policy assertions has to be tested individually for each platform. Our BSP implementation was prototypically tested solely with the Axis2 [45] Web Services engine.

See Listing 3.5 for an invalid policy with respect to R5404, making use of the `sp:InclusiveC14N` assertion (line (13)). This assertion will result in production of a SOAP message that uses *inclusive* C14N, violating R5404. The WS-Security specification defines the default C14N method to be exclusive, which is overridden by this example policy, as seen on line (13). Static validation of requirement R5404 can thus be accomplished by making sure that the `sp:InclusiveC14N` assertion is *absent* in an endpoint's policy, which can be accomplished at deployment time. This is contrasted by dynamic inquiry, where each runtime message reaching or originating from an endpoint has to be checked, ensuring the C14N algorithm attribute has a satisfactory value (as discussed in the previous Section).

Let us reconsider R3033 which requires version 3 for X.509 certificates in the WS-Security header and turned out to be impossible to validate at runtime. As before, the presence or absence of a (set of) WS-SecurityPolicy assertion(s) that guarantees the usage of the right `ValueType` attribute value for X.509 certificate tokens has to be identified. WS-SecurityPolicy specification defines such assertions as descendant of the `sp:X509Token` assertion. Since the example policy in Listing 3.5 does not specify any child element for the X.509 token assertion (line (6)), this policy is invalid against BSP requirement R3033. In order to render this policy compliant, line (6) must be replaced by one of the following two policy expressions:

1. `<sp:X509Token>`

Listing 3.5: Invalid WS-SecurityPolicy policy with respect to R5404.

```
1  <wsp:Policy xmlns:wsp="..." xmlns:sp="...">
2    <sp:AsymmetricBinding>
3      <wsp:Policy>
4        <sp:InitiatorToken>
5          <wsp:Policy>
6            <sp:X509Token />
7          </wsp:Policy>
8        </sp:InitiatorToken>
9        <sp:RecipientToken> ... </sp:RecipientToken>
10       <sp:AlgorithmSuite>
11         <wsp:Policy>
12           <sp:Basic192 />
13           <sp:InclusiveC14N />
14         </wsp:Policy>
15       </sp:AlgorithmSuite>
16       ...
17     </wsp:Policy>
18   </sp:AsymmetricBinding>
19   <sp:SupportingTokens> ... </sp:SupportingTokens>
20   ...
21 </wsp:Policy>
```

```
      <wsp:Policy>
        <sp:WssX509V3Token10/>
      <\wsp:Policy>
    </sp:X509Token>
```

2.
```
<sp:X509Token>
  <wsp:Policy>
    <sp:WssX509V3Token11/>
  <wsp:Policy>
</sp:X509Token>
```

Either of these expressions will force a X.509 binary security token to have a `ValueType` of '`X509v3`' at runtime (Compare Listing 3.3 once more). Through definition of the X.509 security policy assertion we know that the `ValueType` will have the desired value at runtime. Additionally, we can extend coverage of BSP beyond those requirements that can be dynamically validated [50]: we can definitely conclude that the `wsse:BinarySecurityToken` will in fact contain such a binary token, given the invisible behavior of the runtime platform has been sufficiently tested. Remember that we were unable to test R3033 in the runtime message exchanges. These assumptions rely on the correct interpretation of the security policy by the web services middle-ware (can be checked off-line, as outlined), for which also additional research exits (e.g. [124]). In the evaluation in Section 4, we will exactly indicate for which BSP requirements we were able to extend runtime validation coverage.

See Listing 3.6 for the schematron that statically validates R5404 and R3033. For our policy validation, the relevant namespaces are WS-Policy and WS-SecurityPolicy and are being introduced as shown in Listing 3.6, lines (2-3). The pattern implementation for R5404 is depicted in lines (4-10), while R3033 is represented by lines (11-17). As a translation from dynamic runtime requirement to static policy requirement, R5404 could be intuitively read as "R5404

requires that a policy MUST NOT specify an wssp:InclusiveC14N assertion" and R3033 as "R3033 requires that a X509Token assertion MUST include a wssp:WssX509V3Token10 or wssp:WssX509V3Token11 assertion".

Listing 3.6: Schematron implementation for R5404 and R3033.

```
1  <schema id="WS-I BSP" xmlns="...">
2    <ns prefix="wsp" uri="..." />
3    <ns prefix="wssp" uri="..." />
4    <pattern id="R5404">
5      <rule id="Rule for R5404" context="wssp:AlgorithmSuite/wsp:Policy">
6        <assert id="Assert exclusive C14N" role="MUST" test="not(wssp:InclusiveC14N)">
7          Inclusive Canonicalization algorithm must not be used.
8        </assert>
9      </rule>
10   </pattern>
11   <pattern id="R3033">
12     <rule id="Rule for R3033" context="wssp:X509Token">
13       <assert id="Assert for R3033" role="MUST" test="wsp:Policy/
             wssp:WssX509V3Token10 or wsp:Policy/wssp:WssX509V3Token11">
14         Any X509_TOKEN MUST use version 3 for signature verification as the
               ValueType.
15       </assert>
16     </rule>
17   </pattern>
18   ...
19 </schema>
```

When validating R5404 against our sample security policy in Listing 3.5, the context attribute of the rule element (line (5)) selects line (10-15) of the sample, while the test attribute of the assert element (line (6)) checks for the absence of the wssp:InclusiveC14N element within that context, using a negated XPath expression. If attribute test turns out to be false for a policy (which it does for our example - compare line (13) in Listing 3.5), an assert failure is produced, allowing to conclude non-compliance of the policy against requirement R5404. The rule for R3033 must be evaluated within the wssp:X509Token element context (Listing 3.6, line (12)), checking for the presence of either wssp:WssX509V3Token10 or wssp:WssX509V3Token11 assertion (Compare the correction we made to line (6) of the sample policy of Listing 3.5). Here it is notable that R3033 omits the wsp:Policy xpath in the context attribute (in contrast to R5404) and adds it to the test attribute. This is because the rule would otherwise fail to fire when needed. With a context attribute of value wssp:X509Token/wsp:Policy, the relevant context in our sample policy would not be selected. The assertion in line (6) of Listing 3.5 would remain unaffected, since does not contain any sub-content (and thereby no wsp:Policy child element).

3.3 Policy Authoring

MDD/MDS, and, in particular policy-driven security management, has to build upon both, the adherence to security standards for the interoperability between developer tools as well as policy hierarchy compliance, in order to ensure validity of policies against (current) business rules. Both cases are addressed by a compliance validation, as discussed. Standards compliance is ideally inherently conferred by XSD validation, while more complex requirements - such as

BSP Conformance and/or individual application profiles - require more expressive schemata that coevally have to be customizable to current services needs. Schema(-constraints) definitions of (security) best-practices on static policies can be accommodated by a validation of the policies against these schemata in dependence of what set of requirements the developer chooses to be part of the schemata. The SVP is a powerful mechanism for providing such a functionality and lends itself to implementing policy validation and authoring: schematrons are an accurate - and system-independent - means for the evaluation of operational policy schema(-constraint) rules, while producing meaningful validation responses.

Our authoring approach concentrates on the selection and validation of operational XML policies against instances of a custom generic - and prototypical - *Architectural Policy Model* (**APM**). The APM enables practitioners to create a so called *Architectural Policy Template* (**APT**), defining the vocabulary and logical grouping for architecture requirements. Architectural policies selected from the APT are called *Architectural Policy Selection* (**APS**). The mapping for requirements defined in the APT that have correspondences in the operational policy model (**OPM**) is expressed as Schematron rules, which must be defined by a domain expert architect. The rules express syntactic constraints on OPM instances. Note that both requirement templates and requirement selections (i.e. APTs and APSs) are instances of the same APM in our approach. The APM was solely invented for testing the SVP mediation and authoring model and is not meant for provide a comprehensive Architecture-Level policy representation. It is kept generic, in support of being applicable to a wide range of policy requirements, while exposing enough information to conduct the mediation to the lower level.

With the help of the APM, a practitioner can reuse and compose architectural requirements (i.e. architecture policy expressions) that in turn map to valid (sets of) operational policies. She obtains guarantees of validity through the validation of instances of the OPM against the schematron rules defined in the APS. Since we use the SVP in conjunction with the APM, we can validate the policy hierarchy both vertically and horizontally, as schematrons are capable of including multiple namespaces. The validation results prevent the policy author to choose policies that do not fully map the APS, which would result in unintended outcomes and will in the worst case scenarios produce an insecure and/or uninteroperable solution. Naturally, this process is *static* and takes place prior to deployment of services. We implemented a proof-of-concept policy tool on the Eclipse Platform [97] that integrates this authoring methodology with the SVP. As based upon the Eclipse Modeling Framework [96] (**EMF**), major parts of the tool may be regenerated for different APMs. This allows to experiment with different models at the architecture level of abstraction, as long as ways are maintained to extract the correct set of Schematron mediation rules from the model in use.

3.3.1 Authoring Steps and Policy Reuse

Our policy authoring prototype tool brings together the policy hierarchy concept and the compliance validation in a process of policy mediation, dividing the authoring of high level (architecture or business) target requirements into three steps:

1. Define an APT as instance of the APM
2. From the APT make a selection, i.e. define a (set of) APS

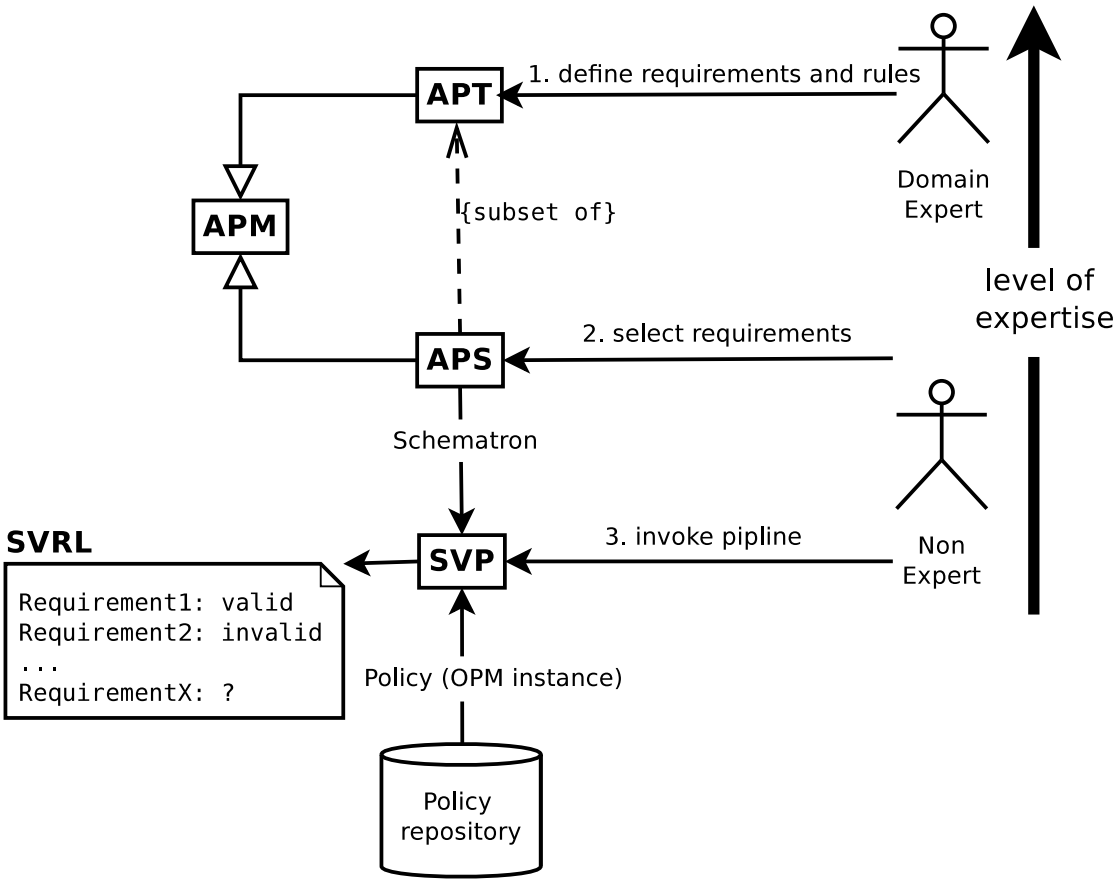


Figure 3.6: Policy authoring: 3 steps for APM to OPM mediation

3. Invoke the SVP to validate the APS(s) against OPM candidates (e.g. from a repository)

This authoring approach is an incremental process, where the authoring steps are not necessarily executed by the same policy author, as exemplified on Figure 3.6. Step (1) is executed independently from step (2) and step (2) independently from step (3), while each step may be executed repeatedly. As step (1) encapsulates the validation rules as ATP, step (2-3) may be repeatedly executed also by a non expert user, using the APT. Likewise, step (2) encapsulates valid architecture policy configurations and solely step (3) may be executed repeatedly using APSs created in step (2). For step (3) there is no need for domain specific knowledge at all, if the user can trust in accuracy and validity of available APSs and given she has instructions on which APSs to choose for her current solution. In step (2), still some domain knowledge may be needed to assemble valid APSs from the APT, resulting from the fact that some combinations of requirements may be invalid and therefor produce an incorrect policy mapping. As a consequence of these considerations, observe that, although the figure depicts just two kinds

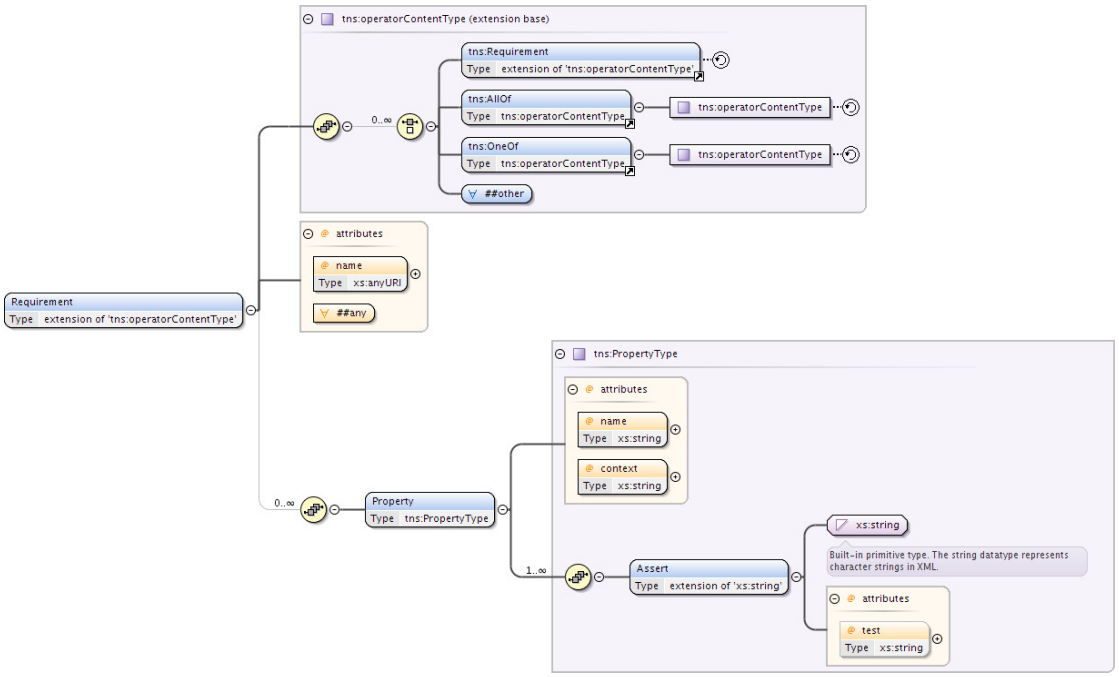


Figure 3.7: Schema visualization of the Architectural Policy Model.

of policy authors, the prototype supports policy development on three levels of expertise: all authoring steps may be performed by separate users, a collection of steps (e.g. steps (2-3) as in the diagram) may be conducted by the same developer, or all steps could be carried out by a single expert author.

Step (1) involves to store the ATP for reuse in some location and any user may load it to start the authoring process entering at step (2). Also APSs may be stored and another user may join the authoring process not until step (3), by loading an appropriate (set of) APS produced by a more experienced user in step (2). In step (3), the user has to select both, an APS and an (set of) operational policy from the policy repository to invoke the SVP. Before policies can be fed into the pipeline, the properties contained in the APS are transformed to a valid schematron schema (such as the one termed `BSP.sch` in Figure 3.4) for pipeline input. The SVRL results produced indicate compliance or non-compliance of the operational policy. Of course, a new operational policy may also be created right in place and APSs may be used to validate it along the development process, until the policy reaches compliance to the desired outcomes (i.e. APS used). Note that both APT and APS inherit from the APM object model while APS instances are subsets of specific ATP instances, in terms of the collection of requirements they contain, as indicated respectively, by the generalization and constraint relations in Figure 3.6.

3.3.2 Architectural Policy Model

See Figure 3.7 for a visualization of the XSD we use for defining the APM: an architecture policy consists of a root `Requirement`, that may contain sub requirements or the `AllOf`

(logical AND) and `OneOf` (logical XOR) operators. Requirement elements may be qualified by zero or more `Property` elements and Property elements must contain at least one `Assert` element. We adopt exceptions only, since we can express `Report` elements in terms of `Assert` elements. Note that requirements may be extended with any attribute or element, which is useful for backwards compatability. Observe on the schema outline of the APM in Listing 3.7 its similarities to both, the schematron and the WS-Policy object model: Requirements that do define one or more properties match the `pattern` elements of the schematron, whereas properties match `rules` defined within the pattern. `Requirement` elements that define no properties have no correspondence in the schematron (i.e., do not map to a pattern). Observe the optional `ValidationNS` attribute that defines the XML namespace that rules of patterns apply to. `Property` elements define the schematron rule's `context`, while each of its child `Assert` elements incorporates a `test` attribute. The APM further defines operators, analogous to those of WS-Policy, for the logical grouping of requirements. `Requirement` operator equals to `AllOf` operator in the APM the same way `wsp:Policy` equals to `wsp:All`. For the complete XSD of the APM refer to Appendix A.2.

Listing 3.7: Schema outline of the APM

```
1 <Requirement name="..." (validationNS="...")? >
2   <Property name="..." context="...">
3     <Assert test="..."/> +
4   </Property> *
5   (  <Requirement name="..."/> |
6      <AllOf/> |
7      <OneOf/>   ) *
8 </Requirement>
```

APT The APT is an instance of the APM that defines the vocabulary to express architectural objectives in the form of requirements and their properties with a choice of assertion tests. Listing 3.8 shows a sample ATP that defines a vocabulary for basic architectural security requirements, created in step (1) of the authoring process. Note that we omit namespace declarations. In this example, the `Confidentiality` requirement assertion in line (2) has a choice of two sub requirements, i.e. an exclusive disjunction of the `"HighConfidentiality"` and `"LowConfidentiality"` requirement using the `OneOf` operator. The former further decomposes the `"PasswordProtection"` and `"Encryption"` requirements as seen on lines (5-9) and lines (10-14), respectively, this time conjuncted using the `Requirement` operator (i.e. the parent requirement element). The requirements that define `Property` elements can be translated to schematron `pattern` elements, and their parent requirement to a schematron `schema` element, as discussed. We will call such requirements *pattern requirements* and *schema requirements* in further discussions. In the sample, the `"HighConfidentiality"` schematron requirement validates that user name tokens use digest passwords and that encryption is provided using a TripleDes algorithm. This is done with the help of the `"PasswordProtection"` and `"Encryption"` pattern requirements, respectively. In lines (21-28), recover (parts of) the schematron from Listing 3.6, implemented as `"BSPConformance"` requirement in the sample.

Listing 3.8: Example of an APT

```
<Requirement name="Security">
  <Requirement name="Confidentiality">
    <OneOf>
      <Requirement name="HighConfidentiality">
        <Requirement name="PasswordProtection">
          <Property name="UsernamePassword" context="wssp:UsernameToken">
            <Assert test="wsp:Policy/wssp:HashPassword">
              A password digest must be used
            </Assert>
          </Property>
        </Requirement>
        <Requirement name="Encryption">
          <Property name="TripleDes" context="wssp:AlgorithmSuite/wsp:Policy">
            <Assert test="wssp:TripleDes or TripleDesRsa15 or wssp:TripleDesSha256
                sp:TripleDesSha256Rsa15">
              A TripleDes Algorithm must be used
            </Assert>
          </Property>
        </Requirement>
      <Requirement name="LowConfidentiality">
        ...
      </Requirement>
    </OneOf>
  </Requirement>
  <Requirement name="Integrity"> ... </Requirement>
  <Requirement name="BSPConformance">
    <Requirement name="R5404Conformance">
      <Property name="ExclusiveC14N" context="wssp:AlgorithmSuite/wsp:Policy">
        <Assert test="not(wssp:InclusiveC14N)">
          Exclusive C14N must be used
        </Assert>
      </Property>
    </Requirement>
    ...
  </Requirement>
  ...
</Requirement>
```

APS Before we can validate the pattern requirements of APTs, operators other than conjunctions have to be eliminated. A (potentially non-expert) policy developer has to perform step (2) of the authoring process and select an APS that serves best her needs; she may store it for reuse, if desired. There will be situations where she chooses a security configuration that keeps information highly confidential, with just minor integrity needs. At another occasion she defines a different APS for a second service, for which she prefers, say, highly reliable traffic without any confidentiality/integrity needs. Using the APT in Listing 3.8, it is easy to assemble an APS by means of the requirement vocabulary. The only restriction is that exclusively disjuncted requirements cannot be selected at the same time. In other words, APSs contain only requirement conjunctions. See Listing 3.9 for an APS that asserts high confidentiality as well as BSP conformance. Internally, the APS contains the schematron rules in the form of `Property` elements, while the leaf nodes in the APS tree depicted, correspond to the selected pattern requirements. Thus, we can conveniently use APSs to validate OPM instances, such as WS-SecurityPolicy instances in the example given.

Listing 3.9: Example of an APS

```
1  <Requirement name="Security">
2    <Requirement name="Confidentiality">
3      <Requirement name="HighConfidentiality">
4        <Requirement name="PasswordProtection">
5        <Requirement name="Encryption">
6      </Requirement>
7    </Requirement>
8    <Requirement name="BSPConformance">
9      <Requirement name="R5404Conformance"/>
10   </Requirement>
11 </Requirement>
```

Reconsider Listing 3.8 again and consider the abstract natural-language requirement "High Confidentiality", that could be the condition for a services search that a user undertakes. Imagine a security expert decided that the enforcement of this abstract policy requires a TrileDes algorithm suite, and she implements the validation rule accordingly, as seen on lines (11-13). Another policy author recognizes, that the message payload has to be encrypted as well in order to gain compliance of her specific policy hierarchy instance. She comes up with the rules necessary for validating this policy and integrates it with the original "HighConfidentiality" architecture requirement. She may reuse the ATP defined previously by her colleague and compose the "MyConfidentialityRequirements" policy, incorporating the new abstract requirement "EncryptStockSymbol".

See Listing 3.10 that integrates "HighConfidentiality" with the additional requirement that `watns:StockSymbol` elements within message payloads have to be encrypted, as seen on lines (9-13) (apparently using one of the algorithms captured by "HighConfidentiality" - compare Listing 3.8 once more). The "StockSymbol" requirement child of the "EncryptStockSymbol" pattern requirement assures that at least one nested `wssp:XPath` assertion is included within a WS-SecurityPolicy that further qualifies the `wssp:EncryptedElements` assertion. The corresponding schematron rule asserts that the element value of the `XPath` element has to identify the `watns:StockSymbol` SOAP body descendant, resulting in the message payload to be encrypted within those SOAP messages that are produced as a consequence of enforcement of a particular WS-SecurityPolicy policy that is compliant to the "MyConfidentialiyRequirements" requirement.

Listing 3.10: Example of an APT

```
1  <Requirement name="MyConfidentialityRequirements">
2    <AllOf>
3      <Requirement name="HighConfidentiality">
4        <Requirement name="Encryption">
5          ...
6        </Requirement name="Encryption">
7      </Requirement>
8      <Requirement name="EncryptStockSymbol">
9        <Property name="StockSymbol" context="wssp:EnyryptedElements">
10         <Assert test="sp:XPath='/soapenv:Envelope/soapenv:Body/watns:StockSymbol'">
11           The watns:StockSymbol element in the message payload MUST be encrypted.
12         </Assert>
13       </Property>
14     </Requirement>
15     ...
16   </AllOf>
```

```
17 </Requirement>
```

As mentioned, protection assertions, such as the `wssp:EncryptedElements` assertion, are used by the messaging middle-ware to enforce encryption and signature mechanisms. We use them in this sample to conclude an operational policy's compliance to requirement "EncryptStockSymbol" - at built-time. Observe that this architecture policy expresses xpath-based schematron schema-constraints on xpath-based schema-constraints: the nested `wssp:XPath` assertion's element content (i.e. the xpath string identifying the element to be encrypted within messages) that the WS-SecurityPolicy specifications defines with the help of the `wssp:EncryptedElements` assertion for confidentiality, is itself validated using the `Assert` test xpath expression. This way, we can guarantee encryption of dynamic message payload statically. Note that, instead of including this requirement as `Property` child of the "HighConfidentiality" policy, the author could have equally defined a new sub-requirement "EncryptStockSymbol" as descendant of "HighConfidentiality", for instance. Optionally, she could have integrated other requirements and used the operators to group these requirements in con- and disjunctive alternatives. Any user that is not a specialist can now (re-)use the ATP in Listing 3.10 to search for services that have the property of either of "HighConfidentiality", "EncryptStockSymbol" and "MyConfidenialityRequirements", without any specific security knowledge. This highlights the value of the SVP policy mediation approach that (at least in parts) replaces *customization* by *configuration*.

Since the schematrons for the matching of APTs to operation policies are provided encapsulated from the architecture model, the SVP permits any XML document apart from WS-SecurityPolicy policies to be provided as second input to the pipeline (in addition to the schematron schema). In other words, APTs are reusable with different schematron mappings. Therefor, extensions to other operational policy models and languages can be delivered with minor effort, because only the rules of the APT will have to be customized, while the ATP's vocabulary is not necessarily required to change. For this purpose, take a look at the APT in Listing 3.11, which combines the constraint on WS-SecurityPolicy introduced above with an abstract WS-ReliableMessaging policy. First, the sample asserts a reliable messaging integrity constraint, as seen on lines (5-11): pattern requirement "WS-ReliableMessagingIntegrityCheck" simply checks if a SOAP header - as a result of enforcement of a compliant operational policy - will contain a signed body and a signed header at runtime. This constraint can be checked within the WS-SecurityPolicy namespace by ensuring headers named by the WS-ReliableMessaging namespace get singed (compare `@Namespace` in the test attribute in line (7)). Hence, this integrity requirement reveals no new insights, as it must be validated within the same policy model (i.e. WS-SecurityPolicy's model); we however mention it here, as the sample illustrates the flexibility of the SVP approach. The last pattern in the Listing in contrast is validated in a different name-space, defining a primitive WS-ReliableMessaging-Policy check that ensures a WS-ReliableMessaging policy is present within the overall policy, i.e. that a `wsrmp:RMAssertion` is present, as seen on lines (13-17). Hence, it is easy to validate the policy hierarchy also horizontally, by simply including the (domain-specific) namespace in Schematron (such as the WS-ReliableMessaging-Policy namespace in the sample). In other words, with Schematron we are able to account for both, the vertical and horizontal policy management dimensions.

Listing 3.11: Example of an APT

```
<Requirement name="MyRequirements">
  <AllOf>
    <Requirement name="MyConfidentialityRequirements" />
    ...
    <Requirement name="WS-ReliableMessagingIntegrityCheck">
      <Property name="SignedBodyAndWS-RM-Header" context="/">
        <Assert test="//wssp:SignedParts[wssp:Body and wssp:Header[@Namespace='http:
            //docs.oasis-open.org/ws-rx/wsrm/200702' and not(@Name)]]">
          Any policy must conatin a realiability policy.
        </Assert>
      </Property>
    </Requirement>
    <Requirement name="Reliability">
      <Property name="WS-RMP" context="wsp:Policy">
        <Assert test="wsrmp:RMAssertion">
          Any policy must conatin a realiability policy.
        </Assert>
      </Property>
    </Requirement>
    ...
  </AllOf>
</Requirement>
```

CHAPTER 4

Evaluation

"What we get by abstraction from something can be returned."

Raymond L. Wilder [241]

4.1 BSP Compliance

To test the static validation we take a simple approach: for each requirement identified in Chapter 3 (and listed in Appendix A.4.1) we developed valid as well as invalid policies, fed as input into the pipeline. With the help of JUnit tests, we verify that the pipeline reports invalid policies accordingly, indicating the reason for assertion failures. So the evaluation of the static rules' accuracy is straight forward, in particular since we only experimented with one runtime platform: the open source Web Services engine Apache Axis2 [45]. Using our dynamic WS-Security validation implementation, we are able to cover the test assertions document [244] of WS-I. This document contains rules for the validation of exactly 81 requirements defined in the BSP specification that are classified as testable within this document. This amounts to approximately 40% of the BSP's 184 requirements and considerations. For cross-checking the static WS-SecurityPolicy validation rules that we inferred from BSP, we solely experimented with 8 requirements however. This corresponds to the intersection of those BSP requirements we are able to validate both at runtime and statically within the policies at the same time. The static policy validation rules we inferred form the BSP specification cover 20 requirements in sum, of which 8 are testable according to the test assertions document. Thus, for 12 BSP requirements we are able to extends coverage of dynamic validation using the static Schematron implementation. The subset of 8 requirements for the runtime validation experiments is sufficient to measure the performance overhead introduced through checking message exchanges on the wire. Hence, we can conclude a lower boundary for the overhead that our static policy validation method allows us to save. In addition, as the time needed for validation increases linearly with the number of requirements considered, we can accurately estimate the overall overhead introduced by dynamic WS-Security validation against BSP.

Requirement	RFC-2119 Level	Rule Type	Runtime Validation
$R3212$	MUST	0	
$R3227$	MUST	0	$\surd$
$R5421$	SHOULD	0	$\surd$
$R5621$	MUST	0	
$R3002$	MUST	1	
$R5423$	MUST	2	$\surd$
$R5412$	MUST	1	$\surd$
$R5404$	MUST	1	$\surd$
$R5420$	SHOULD	2	$\surd$
$R5620$	MUST	2	$\surd$
$R5625$	MUST	2	
$R5626$	MUST	2	$\surd$
$R3033$	MUST	2	
$R6302$	MUST	2	
$R6902$	MUST	2	
$R6903$	MUST	2	
$R6904$	MUST	2	
$R6905$	MUST	2	
$SEC17.9$	MUST	2	
$SEC17.13$	SHOULD	2	

Table 4.1: Comparison of Static and Dynamic Validation Coverage

See the Table 4.1 for a compacted view of the requirements we identified to be amenable to static policy validation (compare Appendix A.4.1 for a detailed version). The last column indicates whether the individual requirements are verifiable from the SOAP envelope also, as defined by WS-I. Requirement level and rule type are displayed again for convenience. As seen, we are able to validate 20 requirements statically within policies, that is, around 11% of BSP's overall requirements/considerations/recommendations. The power of static validation is highlighted by the fact that we are able to extend coverage in 12 cases, for which no runtime WS-Security validation rules exist (compare last column). We use schematron's `role` attribute to indicate the requirement level during pipeline invocation and this way customize the behavior of the SVP for the BSP validation on different compliance levels. The dynamic validation experiments conducted for the 8 requirements have equally been undertaken using Axis2, running as Web Application within a Tomcat Servlet container instance. A WS-SecurityPolicy policy demanding integrity protection for the entire header and body as well as confidentiality for the body of exchanged messages, utilizing an asymmetric binding with TripleDesRsa15 algorithm suite, has been applied to a simple echo service deployed within that Axis2 Web Application for experimentation. On the client-side the same policy is imposed to govern request messages, which are likewise produced by employing Axis2 as the message generation API, calling the

service's echo operation. See appendix A.3.1 for the basic policy used in the experiments. Appendix A.3.2 shows a message produced by Axis2 after enforcement of this example policy.

To enable message-level security support for SOAP exchanges, we use an Axis2 module called Rampart [46] that utilizes interceptors to provide WS-Security features. Our implementation leverages the same plug-in mechanism to realize runtime BSP validation, providing an interceptor wrapped as a WS-I compliance module. This interceptor validates the BSP requirements in each phase (in-flow, out-flow, fault-in-flow or fault-out-flow phase) within which it is activated, executing XPath queries (representing the requirements) on the exchanged SOAP envelopes. These queries make up the Schematron patterns for validation and therefor allow for conclusions on BSP compliance. In principal, the experimentation scenario could be extended to other WS-I profiles than BSP, by simply providing a handler (the synonym for interceptor in axis2 terminology) for the desired profile. The BSP handler intercepts the SOAP envelope while passing either the requester or provider side communication boundary.

For the message traffic interception we use the pull-parser based XML meta-model and processing framework AXIOM [44] which is also from Apache and implemented in Java. AXIOM extracts relevant XML fragments, places them directly into the memory and exposes them via a pull interface. This stream-based approach allows for faster XML processing. Since AXIOM is compliant with the XML infoset, we can layer Jaxen, our XPath engine of choice for these experiments, on top of it. AXIOM/Jaxen provide a convenient way to execute the xpath queries that represent our dynamic validation rules on the SOAP envelope in an efficient manner. A two-step method is used, where the first step uses xpath to select the relevant document context and the second step involves another xpath selection evaluated to a `boolean`, indicating compliance (*true*) or non-compliance (*false*) against the analyzed BSP requirement. This only imitates a subset of the standard schematron behavior, as unnecessary functions for the BSP validation are left out this way (phases, diagnostics, etc.), however improves performance by sparing validation processing. We chose to re-implement this subset and abstained using the reference implementation in this case, to be as efficient as possible with regards to message processing overhead.

With the focus on the system's runtime performance and the service provider and service requester deployed on two separate machines, we concentrated on different ways of deploying our WS-I compliance interceptors which were programmed to validate the eight requirements at runtime that we are able to cover also statically at build-time within policies. The server system compromised a VMWare Virtual Machine with Windows XP Professional (Version 5.1.2600) running on an Intel Xeon 2.33 GHZ machine with 1.5 GB of memory and gigabit ethernet. The client was executed via an IBM-Ubuntu Linux (Version 8.04.2) OS installed on a Thinkpad T41p notebook with Intel Pentium M 1.7 GHZ Processor, 1 GB of RAM and a gigabit ethernet adapter. The latency between these, as measured experimentally, was approximately 1ms, and the bandwidth approximately 77MBits/s. We ran two test cycles, one with all four interception options enabled and another cycle with the WS-I compliance module completely deactivated (i.e. no message interception with respect to BSP validation taking place at all). In that way, we measure the maximum time that a dynamic validation of these eight requirements takes and thereby the time the static validation allows us to save. In order to procure meaningful results, an iterative execution of the client, sending requests ranging from 100 to 10000 messages was

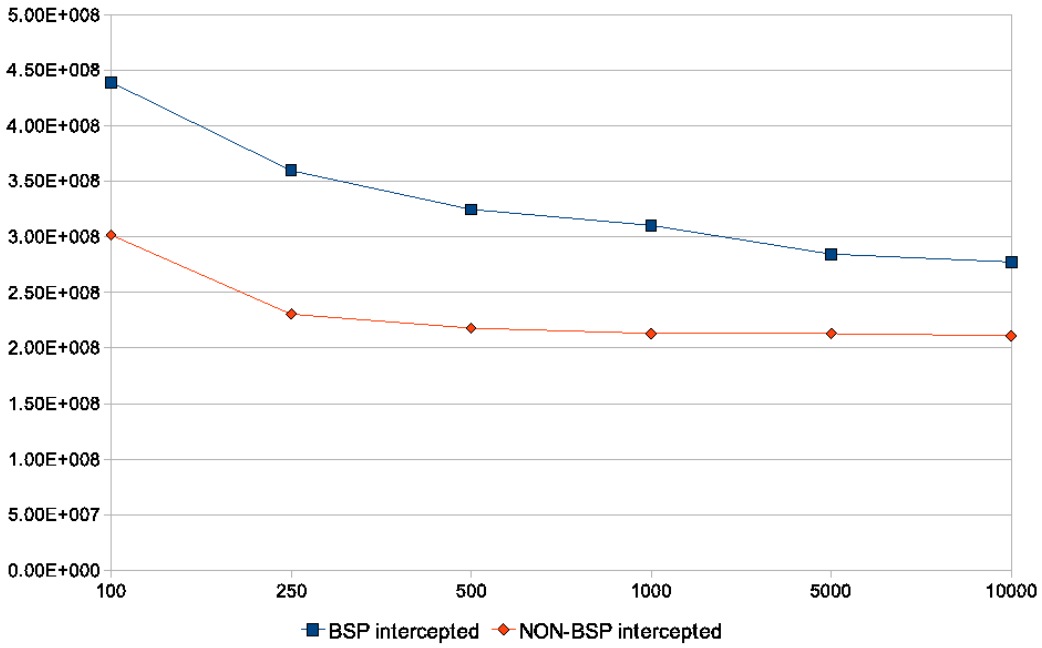


Figure 4.1: Comparison of runtime performance BSP-intercepted versus without BSP interception

conducted, facilitating the derivation of an average interception duration.

Figure 4.1 shows the resulting average time per web service call in nanoseconds. On the graph we compare the case where we have introduced an interceptor for dynamic BSP validation versus the absence of such an interceptor. We observe a gradually decreasing response time with an increased number of calls due to inefficiencies (such as class loading) that mostly occur at the start of the test run. In the limiting case, we see that for 10000 message exchanges, checking BSP compliance at every entry/exit point into/from the web services infrastructure on both the service user and provider, results in a roughly 30% to 50% penalty. Within each of the four interceptor runs we have the time for the actual xpath execution itself and the remainder, which can be considered the overhead of the Java runtime and web services middle-ware to activate the interceptor. This estimate would be more accurate if one accounted for testing on multiple platforms and implementations, introduced SOAP processing optimization [224, 31, 236] and accounted for WS-Security performance issues [157, 218].

As we experimented with Axis2 solely, and, as results will severely deviate for different Web Services engine on different machines deployed, we desist from a detailed evaluation of the runtime performance of this specific services constellation on these specific machines. We instead additionally conducted a measurement of XPath processing alone, which yielded a duration of 13 milliseconds on average to check our 8 BSP requirements. We can assume that this result holds for at least the same machines, when using disparate message generation APIs,

which supports the result obtained solely for Axis2 and allows to conclude implications that are independent from the SOA runtime platforms in use: extrapolating to all 81 requirements that WS-I's test assertions deliverables specify as testable, we can say that checking BSP compliance through runtime validation alone would add a penalty in the order of a tenth of a second to message processing each time this is done. This result lends itself to a strong argument for our static schematron approach, since even partial savings here would be valuable. The result of 13ms provides a lower bound on the performance gain harvested by utilizing static validation, since it allows us - at minimum - to avoid validating eight BSP requirements at runtime.

4.2 Policy Authoring

The APM is a very basic, however generic model that is limited in its scope and meant to serve as a prototype model only. Nevertheless, due to its genericity, new requirements are easy to be introduced, as one may provide sub-requirements for any requirement within APTs (potentially using the APM operators) and since one can include `Property` elements for validation of these requirements where ever desired. Our prototype policy authoring tool is designed in a way that allows to arbitrarily change the APM to a completely different model, as long as we maintain a means to extract a valid schematron object model that maps the requirements. One could utilize WS-Policy itself instead of the (intermediate) APM, defining a vocabulary of custom schematron assertions with the help of XSD, which would follow the recommendations that the WS-Policy standard states for policy model extension. Also semantically rich description, such as security policy ontologies are thinkable, in which schematrons could be embedded. This way more complex service interrelationships can be modeled, while delivering in addition the respective validation rules for the concepts in the form of syntactic constraints on policies. This is contrasted by a semantic validation that is concerned with the semantic (conceptual) correctness of the policy instances. The WS-SecurityPolicy and BSP case study from above underlines schematron's capability to represent incomplete knowledge.

The SVP is able - besides lending itself to check complex runtime policies (such as the adherence to the BSP profile) - to provide assurances that architectural requirements (APSs) are satisfied by a (set of) operational policy through a syntactic validation. As validation rules are encapsulated as schematrons, validation can be provided totally transparent to the user, after the APT has been defined. This means, that also non-experts will be able to reuse APTs, i.e. reuse corresponding mapping rules defined by an expert to assemble a valid APS that matches her objectives. The author may subsequently validate operational SOA policies against the APS, which triggers the SVP in the background. Of course a policy author may also develop new OPM instances and verify them using the approach. The architecture abstraction layer introduced into the policy hierarchy eases the policy selection process which is very valuable in particular in the domain of security, as being complex for developers that are not specialists. In our prototype, we use the EMF and the *Ecore* meta-modeling facilities to generate a skeleton to programmatically interface the APM. Hence, we can easily experiment with different APMs, while major parts of the tool require no adaption at all. The prototype is realized as an Eclipse Plug-in [98] and supports the OSGi [190] architecture's plug-in and extension mechanisms. The tool uses built-in UI code generation mechanisms extensively and extends several eclipse UI widgets. In addition

to leveraging EMF's adapter and generator mechanisms for the Ecore model, we use Eclipse's built-in capabilities for managing new projects and for binding distinct file formats to different editors. The tool mimics a policy repository (essentially, a directory containing policy files) and confers the execution of the three authoring steps discussed in Chapter 3.

For conducting the authoring steps, the tool provides two main UI components. First, a tree view of the APM is supplied that allows the creation of APTs in terms of newly introduced `Requirement` elements (that may in turn be grouped by operators). For the definition of the Schematron mediation rules, a simple dialog is provided that the experts can use to define appropriate Schematron patterns for each `Requirement` for which this is desired. Second, a tree view for the selection of requirements form the APT is supplied, i.e. the UI is employed in the second authoring step, when defining an APS. Therefor, requirements in the tree are implemented as selectable tree nodes that react in dependency of their parent operator. If a requirement is part of a conjunction (i.e. child of `Requirement` or `AllOf`), also the selection of the requirement's siblings is permitted, whereas requirements that are part of disjunctions (i.e. child of `OneOf`) must be the only requirement selected. Using this UI, it is straight forward to select requirements from multiple APTs; in fact, the "Confidentiality" and "BSPConformance" architecture policy examples of Listing 3.8 have been combined this way. Finally, to actually mediate a policy selection, a simple Button invokes the SVP and validates the (set of) APS against policies selected from the policy repository.

It is notable that the prototype implementation treats exclusively conjuncted policy alternative in a specific way. For example, when checking BSP's requirements in the policy, we check if any nested `wssp:InclusiveC14N` assertion is present as descendant of an `wssp:AlgorithmSuite` assertion (compare Section 3.2.2), which means a policy providing two alternatives grouped by the `wsp:ExactlyOne` operator of which only one asserts inclusive C14N will be considered invalid. If this is desired depends on the context. The approach we take considers any service invalid that admits *any* policy alternative that results in interaction that is not BSP conformant (which is the case in the example given). However, there are scenarios where it might be useful to check if a service is capable of supporting BSP conformance, which would require a different way of treating *XOR*. In that case, we would need to consider a policy invalid, if it specified an `wssp:InclusiveC14N` assertion in *all* of its policy alternatives that define a nested `wssp:AlgorithmSuite` assertion (i.e. in all `wsp:Policy` elements containing such a nested assertion while being grouped by `wsp:ExactlyOne`). Using our approach, it is convenient to modify this behavior, by defining more specific `context` attributes in the schematron rules that account for the disjunctions introduced through the *XOR* operator.

CHAPTER 5

Related Work

"It is rare that a great assembly is reasonable: it is too readily passionate."

Napolean Bonapartes*[1] *[64]

In this Chapter we will compare Schematron's capabilities with other popular schema languages and tools. We give a detailed discussion on the kinds of constraints Schematron supports. This is contrasted with established grammar-based document types and their coverage of the introduced constraint capabilities. Second, a selection of policy validation and authoring approaches is outlined. In particular rule-based XML mediation mechanisms are treated, as well as other (graphical) abstractions for policy management. We put the emphasis on security policies and application profiles and only briefly touch research on runtime message validation.

5.1 Schematron and Schema Validation

5.1.1 XML Processing

XML [237] is an extensible language that is able to include a formal definition of the structure of the content that is written with it, i.e. it is self-describing in a machine processable way. XML is a restrictive application profile (i.e., a subset of) of SGML [215], that is, the SGML grammar's syntax is expressed by an XML declaration. SGML is derived from IBM's GML [116] and thus a generalized mark-up language, meaning it strictly isolates document type definition (**DTD**) from document instance definition. DTDs make documents subordinate to a *schema*, that defines constraints on the structure and content of documents of that type, above and beyond the basic syntactical constraints imposed by the XML grammar itself. As opposed to the original SGML specification, the XML declaration of SGML delimits schema validation and document parsing. This eliminates the drawback of having to run a complex, validating parser each time one wants to assess structural correctness of a document. In other words, XML introduced the concept of *well-formedness*, cleanly separating out validation from parsing.

XML as self-descriptive context-free regular grammar can be processed in a variety of ways. XMLPipe [158] differentiates XML processing to belong to one of the following categories:

- Augmenting processes
- Constructive processes
- Extraction processes
- Packaging processes
- Inspection processes

W3C XML Schema [228, 61] implies the production of a new information set or the addition of information items or property values as defined in the XML Infoset [78] specification. For example, XSD augments XML by adding datatype information. An XSD which defines default values, is - besides augmenting - also constructive, as are XInclude [159], XQuery [63] and Extensible Stylesheet Language Transformations (XSLTs) [11, 27]. Processes that copy or remove parts of the information set are extractive, by the ability to address specific parts of a document, e.g. by the use of XPointer [84, 121]. An example for a packaging process is SwA, allowing to package resources within SOAP messages. XMLPipe characterizes Schematron validation as an inspection process, inspecting but not modifying an XML document. This definition however misses the fact that Schematron extensively uses XPath queries to extract document fractions for validation, in addition to creating a validation report - i.e., the SVP renders schematron to an extraction process while being explanatory. Cast in this light, Schematron is without doubt also constructive, whereas being augmenting innately, since adding constraints on the XML infoset that are unintuitive or impossible to define with XSD. In a nutshell, Schematron takes the approach of validation by transformation, outputting some rich information constructed from looking at the document set (compare Section 3.1.2 once more).

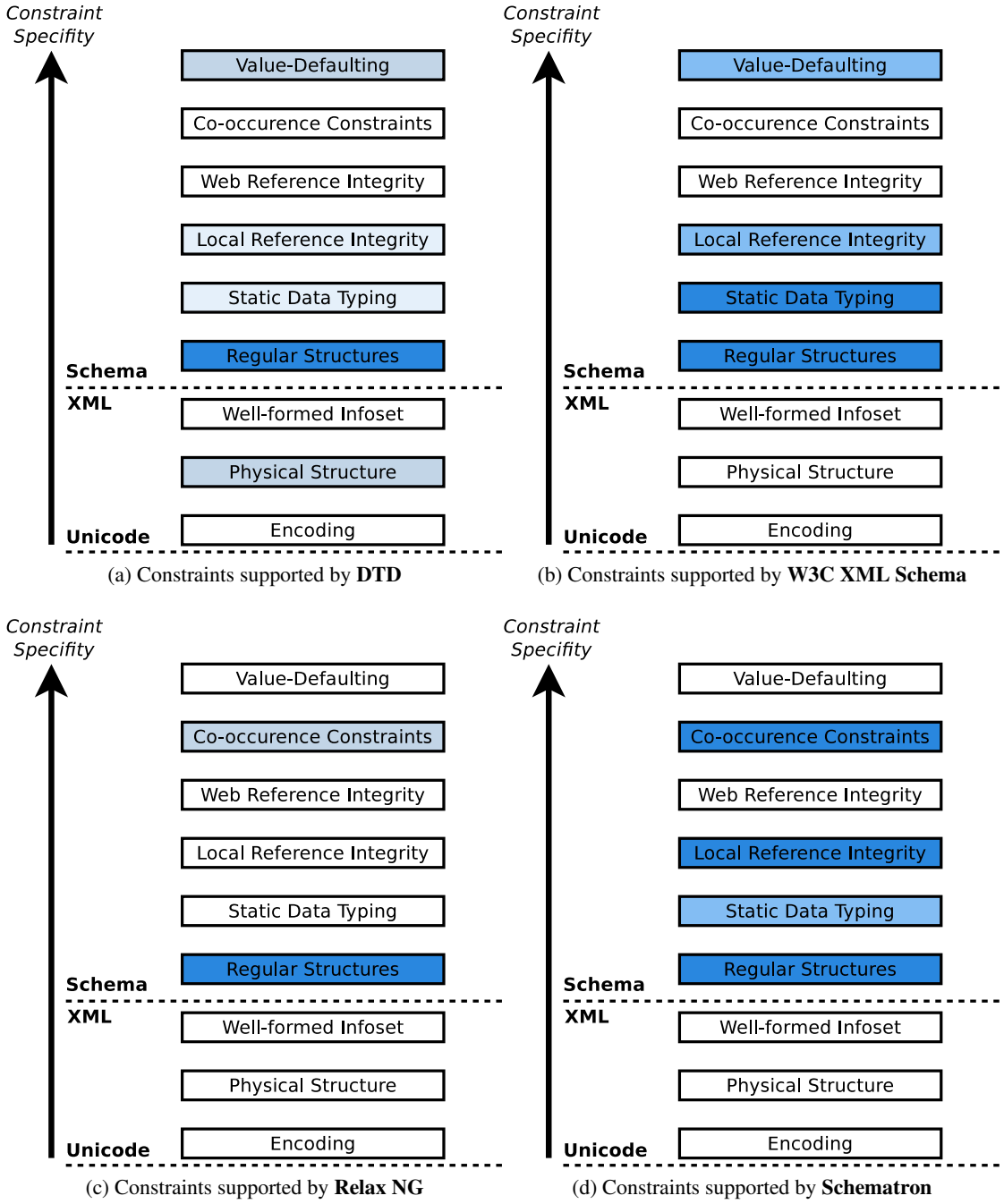
5.1.2 Schema Languages

Up-to-date, newer XML Namespace-aware [19] schema languages have superseded the initial DTD specification, resulting form their richer expressiveness (i.e. capable of expressing richer type-systems) and ability to (namespace-based) schema composition. In contrast to DTD, which was developed along the lines of the core XML specification itself, these (superior) schema(-constraint) languages are specified isolated in separate specifications. One of the most established successors of DTD is W3C XML Schema, which adds additional primitive datatypes, the possibility to define custom complex data types, it supports object-oriented concepts such as polymorphism and inheritance and last but not least, it uses XML syntax to express schemata, as opposed to DTD's ad hoc approach. Essentially, the XML Schema specification supplies a comprehensive *component model* for recursive typing that allows to define types in terms of other types. The XML Schema specification therefor uses *properties* similar to those discussed in Section 2.2.2, each taking certain values and being assigned a significance. It further defines the XML representations and the mapping to properties as well as constraints on the representations. It is surprising that experiments in 2005 [59] yielded a fraction of only 15% of XML

schema instances which really leverage XML Schema's expressive power beyond that of DTDs, as opposed to 85% that could have coevally been defined as DTD. Thus, comprehensive schema definition and editing fails through complexity of the language, rather than lack of expressiveness. The W3C XML Schema specification is hard to read and to most users the effects of typing and validation remains unclear and hard to understand.

The process of schema *assessment* refers to the task of local validation as well as the synthesis of an overall validation outcome for each information item and its potential descendants. While local XML Schema validation refers to assessing whether an element or attribute information item satisfies the constraints embodied in the relevant components of the schema, overall schema-validity implies the assessment of the local well-formedness of all element and attribute information items combined with the results of schema-validity assessments of its descendants, if any. Constraints on the components themselves deal with type definitions as well as element and attribute declarations, whereas additional validation rules govern white space normalization during validation. XML Schema introduces the Post-Schema-Validation-Infoset (**PSVI**) for infoset augmentations that are generated after schema compilation of an XML Schema instance, i.e. as a result of *conformance processing*. The PSVI captures the augmentations to (locally) well-formed information items and their descendants, making explicit the type information as well as normalized and/or default values of these attribute and element information items. The PSVI is therefor exposed as read-only properties, as opposed to the read-write properties used for the Pre-Schema-Validation-Infoset that is built during editing of schemata. Rick Jelliffe [139] describes the PSVI as a 'disruptive force', underlined by the fact that this entails having data accessible in a form for which there is no straight-forward XML representation. Though for some users this is the desired functionality, in majority this will increase complexity of XML systems and requires a regeneration of many XML standards and specifications. The author argues that much efficiency can be gained by casting the type of information items in queries rather than relying on a PSVI. Proponents of XML Schema - such as architects of data-interchange systems - will bring forward the argument that schema instances can be compiled into efficient code, which obviates the need in Web-based environments to download the schema for each document validated.

The equivalent to the PSVI in terms of Schematron [92] is the Schematron report produced after calling the Schematron transformation chain, however, with the advantage that the report is supplied in standardized XML. Indeed, schematrons are actually different types of schemata compared to W3C XML Schema: schematrons are *rule-based* tree patterns. While *grammar-based* schema languages are defined by a formalism and respective top-down production rules that yield a context-free grammar, rule-based schema languages are characterized by a set of rules that are defined either in an *open* or *closed* manner, i.e., all that is not forbidden is allowed versus all that is not allowed is forbidden. Due to Schematron's context-dependent rules being defined as document paths that are based upon XPath, Schematron easily lends itself to be implemented using XSLT (and XPath). The advantage of the ISO XSLT skeleton API is its modularity and extensibility, such that custom implementations are simple to be realized. For using the Schematron API by itself, no XSLT knowledge as such is needed, rendering Schematron an easy yet powerful schema(-constraint) language, while the implementation in XSLT is trivial, as the subset of XPath that XSLT supports is sufficient to capture Schematron's fixed four-layer



(a) Constraints supported by **DTD**

(b) Constraints supported by **W3C XML Schema**

(c) Constraints supported by **Relax NG**

(d) Constraints supported by **Schematron**

Figure 5.1: Different Schemata are suited for different constraints

hierarchy: phases (top-level), patterns (defines XML context), rules (defines xpath test that is subsequently evaluated to a boolean), and assertions. In Chapter 3 we have gone into details of

only three of these layers (patterns, rules and assertions), as these are most important for using embedded Schematron rules within other (grammar-based) schema definitions (such as XML Schema, RelaxNG or our custom APM). Considered as a document type, a Schematron schema contains natural-language assertions concerning a set of documents, marked up with various elements and attributes for testing these natural-language assertions, and for labeling, simplifying and grouping assertions.

Xlinkit [180] is an example of another rule-based language, whereas - besides W3C XML Schema - , RELAX NG [107] and DSD [147, 3] are common choices for grammar-based schema languages. Take a look at Figure 5.1 that compares Schematron with three major grammar-based schema languages. This layered model is proposed by [139] and tries to find important criteria to characterize schema languages: Figure 5.1a, 5.1b, 5.1c and 5.1d show the resulting schema languages stack for DTD, XML Schema, RelaxNG and Schematron, respectively. The boundary between XML and Schemata is given by the ability of a language to define regular structures. XML as such lacks this capability, as it is a stripped version of SGML. The author also treats the mentioned DSD and Xlinkit schemata as well as so called Examplotron [233]; he likewise assigns the shown constraints categories to these languages. We focus on DTD, W3C XML Schema and RelaxNG as popular candidates for grammar-based schema languages, to elaborate advantages of Schematron's rule-based tree pattern approach. The diagram highlights the power of Schematron, as it supports *regular structures*, *static data typing* to a certain degree, *local reference integrity* and the mentioned *co-(occurrence) constraints* (see Figure 5.1d). The saturation of the blue color indicates the strength in the support of a certain kind of constraint. At the time this comparison was released, there has been no strong support for static data typing, however, the ISO standard [92] supplies appropriate typing options. As apparent, there is no winner: different schema languages - and, tools leveraging the languages - are suited to different tasks. Schematron is especially simple yet powerful, but has the drawback of requiring storage or access interaction during validation.

5.1.3 Detailed Comparison of Schematron with DTD, W3C XML Schema and RelaxNG

A more detailed comparison of DTD, W3C XML Schema, RelaxNG and Schematron can be given: consider Table 5.1 that is largely in line with [238, 85], comparing the three with Schematron's rule-based approach. The support ($++$), lack ($--$) and the limited support ($+-$) of important properties of schema languages are shown, each being assigned to one of five categories:

- Content Models and Datatyping
- Modularity
- Namespaces
- Linking
- Co-(occurrence)-constraints

Table 5.1: Comparison of common schema languages with Schematron

	XML/DTD	**XML Schema**	**RelaxNG**	**Schematron**
	Content Models and Datatyping			
Unordered content	++	++	++	++
Simple datatypes	+−	++	++	++[2]
Enumerations	+−	++	++	++
Default values	+−	++	+−	−−
Pattern matching	−−	++	++	++[3]
Exclusions	++	+−	+−	++
Pernicious mixed content	++	−−	++	++
Empty mixed content	−−	−−	−−	++
	Modularity			
Multiple top-level elements	++	++	++	++
Schema modularity	+−	++	++	++
Content model extensibility	+−	++	++	++
Subtype/equivalence relationships	−−	++	−−	++
Context-dependency	−−	−−	++	++
	Namespaces			
Namespace support	+−	++	++	++
Explicit namespaces	+−	++	++	++
Any namespace	−−	++	++	++
Any namespace (XSL)	−−	−−	++	++
Any namespace (nesting)	−−	++	++	++
	Linking			
Simple links	++	++	+−	++
Typed links	−−	+−	−−	++
ID-type element content	−−	++	−−	++
	Co-constraints			
Sibling content	−−	−−	++	++
Sibling attribute values	−−	−−	−−	++
Mutual exclusion	−−	−−	++	++
Element type from attribute presence	−−	−−	++	++
Element type from attribute content	−−	−−	++	++
Attribute type from element content	−−	−−	++	++
Attribute value exclusion	−−	−−	−−	++

[2] XPath datatypes natively and others by expressions on the lexical and value spaces.
[3] If provided by the datatype library and respective URI: e.g., XML Schema Part 2: Datatypes.

The view highlights the usefulness and broad applicability of Schematron with respect to these properties. Schematron supports *unordered content*, i.e. required content without a required order, which approximates SGML's "&" connector as well as *simple datatypes* like integers, dates, currency. Moreover, *enumerations* of both attributes and elements are possible, however coevally, no *default values* of attributes and elements are supported, which the grammar-based languages do confer. Central to Schematron is *pattern matching*, allowing to evaluate the validity of element and attribute content by the usage of regular expression (and/or XPath), for instance, while exceptions can easily be introduced for those elements and attributes for which the pattern validation fails. Furthermore, Schematron is capable of handling *pernicious mixed content*, allowing to formulate restrictions on mixed content. This also includes *empty mixed content*, meaning elements that have no required content but may not be completely empty.

Schematron instances are highly modular, supporting *multiple top-level elements*, as they may be rooted at more than one element. *Schema modularity* is provided by composing schemata of distinct schema modules, while *content model extensibility* is supplied by the capabilities of such modules to extend or modify the content of an element declared in another module. Schematron schemata procure *subtype/Equivalence relationships* to indicate that two elements are (in some of their properties) the same kind of object, and, they allow for *context-dependency* in terms of the content model, by the ability to declare local (context-dependent) elements. Moreover, vocabularies with or without namespace binding may be used with Schematron, that is, general *namespace support*, *explicit namespaces* for inclusion of elements in their explicit form as well as the *any namespace* for both, elements without and with URI namespace restriction (like XSL top-level elements). Finally, also the *nesting* of restrictions on elements declared within the any namespace are supported, which essentially is any namespace not containing elements or attributes from some explicit namespace at any depth. Schematron also turns out to be best suited for linking, supporting *simple links* for internal identities and references with the help of ID/IDREF-style links. In addition, *typed links* that target a specific element type as well as *ID-type element content* are supported, the ladder allowing content to be linked by IDs.

5.1.4 Co(-Occurrence-)Constraint Schemata

A whole section is devoted to one of Schematron's most important assets: the support of rule-based constraints going beyond structural rules, termed *co-constraints*, which are in its simplest form so called *co-occurrence-constraints*. In its simplest form, a co-occurrence-constraint asserts the presence (or absence) of an attribute `A` as child of element E, *iff* attribute `B` is present (or is not present) in the same element E or any element E' part of the information set on hands. Referring to Table 5.1, Schematron supports *sibling content* (i.e. the restriction that sibling elements must be equivalent or not), *sibling attribute values* demanding unequal values of a certain attribute of sibling elements, and, *mutual exclusion* which assert that an attribute or a child element must be present, but not both. On top of that, a schematron schema may restrict the element data type dependent on the presence of absence of an attribute, the element value (i.e. content) dependent on an attribute value, as well as attribute values in dependence of element content. Last but not least, schematrons may constrain two or more attributes such that these attributes must be different for each attribute specified.

There are many options of how to use Schematron in combination with other (grammar-

based) languages. An outline on embedding Schematron within W3C XML Schema instances is given by [22]. This approach uses XML Schema's optional `xs:appinfo` element to annotate the data model with schematron validation patterns, using the `xs:annotation` documentation feature. See Listing 5.1 that shows a sample of a co-occurrence-constraint on a `xs:complexType` using this mechanism. Compare Listing 5.2 for a valid instance of this schema. The embedded schematron pattern ensures that a `Person` with a `Title` attribute of value `'Mr'` defines a `Sex` sub-element with the value of `'Male'`.

Listing 5.1: An XML Schema with embedded Schematron co-occurrence constraint on a `xs:complexType`

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Person">
      <xs:annotation>
         <xs:appinfo>
            <sch:pattern name="Co-occurrence constraint on attribute Title"
                xmlns:sch="http://www.ascc.net/xml/schematron">
               <sch:rule context="Person[@Title='Mr']">
                  <sch:assert test="Sex = 'Male'">
                    If the Title is "Mr" then the sex of the person must be "Male".
                  </sch:assert>
               </sch:rule>
            </sch:pattern>
         </xs:appinfo>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Sex">
               <xs:simpleType>
                  <xs:restriction base="xs:string">
                     <xs:enumeration value="Male"/>
                     <xs:enumeration value="Female"/>
                  </xs:restriction>
               </xs:simpleType>
            </xs:element>
         </xs:sequence>
         <xs:attribute name="Title" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Listing 5.2: Valid document instance with respect to the schema from Listing 5.1

```
<Person Title="Mr">
   <Name>Eddie</Name>
   <Sex>Male</Sex>
</Person>
```

[205] gives a good overview of the how to apply Schematron in various ways, covering most of the constraint capabilities outlined in Table 5.1 above. Inter alia, the authors also cover the embedding of multiple schematrons into distinct XML Schema instances: see Listing 5.3 and Listing 5.4 for examples of two schema modules representing a `Person` and `Car` object, respectively, of which one holds a schematron that ensures that car owners are a person.

Listing 5.3: The `Car` schema with embedded Schematron co-constraint and reference to an external document

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Car">
      <xs:annotation>
         <xs:appinfo>
            <sch:pattern name="Car owner must link to a person" xmlns:sch="http://
                www.ascc.net/xml/schematron">
               <sch:rule context="Car">
                  <sch:assert test="document('Person.xml')/Person/Name = @Owner">
                    The owner of the car must match the name of the person in Person
                        .xml.
                  </sch:assert>
               </sch:rule>
            </sch:pattern>
         </xs:appinfo>
      </xs:annotation>
      <xs:complexType>
         <xs:attribute name="Owner" type="xs:string" use="required"/>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

Listing 5.4: The `Person` schema that documents imported into the schema in Listing 5.3 should conform to

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:element name="Person">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
         </xs:sequence>
      </xs:complexType>
   </xs:element>
</xs:schema>
```

By using XSLT's `document()` function the schematron is capable of validation across multiple document instances that adhere to different schemata. Referring to Listing 5.5 and under the assumption that the document instance of Listing 5.2 resides in the same directory as file called `Person.xml`, the pattern of Listing 5.3 takes care - if validated against the car instance of Listing 5.5 - that a `Car` has an `Owner` attribute that matches the content of the `Name` element, which in turn is child of the `Person` element.

Listing 5.5: A simple `Car` instance owned by `"Eddie"`

```
<?xml version="1.0" encoding="UTF-8"?>
<Car Owner="Eddie"/>
```

The authors of [205] also treat the embedding of Schematron within RelaxNG, resulting in a composite schema language that combines benefits of both RelaxNG's grammar-based approach with Schematron's tree patterns. This way, all types of co-occurrence constraints from Table 5.1 can be supported within RelaxNG schemata. We forgo to give a detailed example of how this works.

The authors of SchemaPath [75] claim to have found a superior method for the task of delivering validation code that goes beyond XML Schema constraints within the schema itself. They introduce two new elements - the `xs:alt` and `xs:error` elements to combine co-constraints with XML Schema. This results in a more specific PSVI after compilation of the schema compared to the Schematron above, as validation rules are bound to specific XML Schema types (i.e. elements that are of a certain type). According to the authors, this is a great advantage of Schemapath. In our mind, the benefit of processing or using XML Schema's PSVI for checking the correctness of a XSD instance in dependence of co-constraints is down to the context and intended usage of the schema and foremost up to the accuracy of the (custom) parser. In particular if validation rules are on hands in the form of natural language assertions, an individual validation report - such as the schematron transformation chain result - appears more accurate, providing the user of meaningful suggestions/errors in a standardized way. Schemapath could provide such a functionality as a post-processing step.

See Listing 5.6 that depicts a XML Schema fragment that defines a simple co-occurrence-constraint with the help of SchemaPath.

Listing 5.6: SchemaPath co-occurrence-constraint for the `xsl:template` element

```
1  <xsd:element name="template">
2    <xsd:alt cond="not(@match) and not(@name)" type="xsd:error" />
3    <xsd:alt priority="0" type="xsl:templateType"/>
4  </xsd:element>
5  <xsd:complexType name="templateType">
6    <xsd:sequence>
7      <xsd:group ref="xsl:templateContent"/>
8    </xsd:sequence>
9    <xsd:attribute name="match" type="xsl:patternType"/>
10   <xsd:attribute name="name" type="xsd:NCName"/>
11 </xsd:complexType>
```

The sample expresses that an `xsl:template` element may contain both a `match` and a `name` attribute. More precisely, *the absence of the name attribute implies the presence of the match attribute* [75]. The `priority` attribute mimics XSLT's template priorities. Compare Listing 5.7 for a minimal Schematron that produces equal outcomes.

Listing 5.7: Schematron co-occurrence-constraint that equals to the SchemaPath check in Listing 5.6

```
1  <xs:element name="template">
2    <xs:annotation>
3      <xs:appinfo>
4        <sch:rule context="xsl:template">
5          <sch:report test="not(@name) and not(@match)">Error</report>
6          <sch:assert test="@type='xsl:templateType'">Error</assert>
7        </sch:rule>
8      </xs:appinfo>
9    </xs:annotation>
10 </xs:element>
11 <xsd:complexType name="templateType">
12   <xsd:sequence>
13     <xsd:group ref="xsl:templateContent"/>
14   </xsd:sequence>
15   <xsd:attribute name="match" type="xsl:patternType"/>
16   <xsd:attribute name="name" type="xsd:NCName"/>
```

```
17 </xsd:complexType>
```

As apparent, schematrons can be almost as compact as SchemaPath annotations, while no additional elements have to be defined within the XML Schema namespace, which SchemaPath requires. Schematron in contrast leverages already existent documentation elements to embed schematron patterns - one intended usage of these elements. Moreover, Schematron could include additional syntactic sugar, such as `diagnostics` elements, meant to make a richer and more readable statement of the assertion test messages. In either that way or by utilizing one of Schematron's attributes, such as the optional `flag` attribute of `rules`, priorities can be modeled in a schematron, the same way SchemaPath's and XSLT's priorities work. Besides that, Schematron is able to reference content within the XML Schema to validate, which helps in supplying meaningful validation feedback (compare Chapter 3). From these considerations Schematron appears superior to Schemapath in both expressiveness and flexibility, as schematrons may be used in a variety of ways, while SchemaPath scope is fairly specific. The schematron object model is able to expose more information to the (custom implementation of) parser than a SchemaPath instance is providing. On top of that, SchemaPath may not define *context-dependent* rules, one of Schematron's virtues.

In particular, ISO Schematron has a macro facility called abstract patterns [184] that facilitates a more declarative way of labeling the items in a schema(-constraint) relationship. Consider Listing 5.8 that gives a generic reusable abstract pattern that represents the relationship of a parent to its child. The single rule that the `required-child` pattern defines checks if a `$parent` has at least one `$child`.

Listing 5.8: Abstract Schematron pattern

```
1 <sch:pattern name="required-child" abstract="true">
2    <sch:rule context="$parent">
3      <sch:assert test="$child">
4        The parent <name path="parent::*" /> should have a child <name/>.
5      </sch:assert>
6   </sch:rule>
7 </sch:pattern>
```

The $ tokens in the schematron are treated as macros by the schematron implementation (analogous to function symbols in functional programming), each being replaced by their invocations during the transformation chain execution, in order to amount to conventional Schematron schemata. See Listing 5.9 for two concrete patterns extending the abstract pattern.

Listing 5.9: Concrete Schematron pattern implementing the abstract pattern from 5.8

```
1 <sch:pattern name="parent-child-instance1" is-a="required-child">
2   <sch:param name="parent" value="\Person"/>
3   <sch:param name="child" value="\Name"/>
4 </sch:pattern>
5 <sch:pattern name="parent-child-instance2" is-a="required-child">
6   <sch:param name="parent" value="\Car"/>
7   <sch:param name="child" value="@Owner"/>
8 </sch:pattern>
```

Note the `sch:param` elements that reference the macros, as well as the `is-a` attributes for inheritance. The sample provides two concrete implementations of

the abstract pattern `required-child`, i.e. `parent-child-instance1` and `parent-child-instance2`. The two implementations would yield validity in case of schema assessment of Listing 5.2 and Listing 5.5, respectively. In addition to the parameters referenced in this sample, we could have also introduced new parameters or use `let` expressions to extend one of the patterns beyond its abstract definition. In fact, provided with these two schematron fragments in a combined Schematron schema, it is possible to drive a custom processor using the abstract pattern and to generate XML instances that are either valid against one or against both concrete pattern implementations. Abstract patterns are capable of supplying enough mark up for implementing such a Schematron processor and to generate code according to concrete patterns that extend invariants defined in the abstract patterns. This way, a Schematron is used for schema documentation in addition to its primary focus on schema(-constraint) validation. The extra labeling represented in the abstract patterns enable identification of parts of constraints (i.e. rules) and assertion (i.e. assertion tests) and opens the door to reuseability by retargetting schematron schemata for code generation of various kinds. In other words, abstract patterns overcome concomitant lack of declarative expressiveness, leveraging the power of XPath validation while exhibiting much less terseness. This heralds significant advancements in home-made schema languages and bears endless extension opportunities [185].

SPath [240, 71] is another integration approach of schemata into the data model of XPath, facilitating co-occurrence constraints and thereby type reflection and introspection. Inspired by the unified approach of RelaxNG, SPath tries to unite the worlds of element and attribute declaration, which W3C's XML Schema so clearly circumscribes. *Occurrences* represent both at one time, element and attribute usages in *types*, whereas node tests are usable to gain a distinction of element and attribute. SPath defines a set of new *axes*, which are navigational facilities to walk the document model as well as *axis modifiers* that are more specific and get appended to the an axis' unique name. Furthermore, SPath allows for both *name tests* and *kind tests* analogous to XPath, the former requiring a colon-separated combination of a prefix, wildcard and a local-name with wildcard, or a single wildcard (*). Kind tests in contrast cover all node kinds encountered in a query path and have the appearance of functions accepting a variable number of arguments. *Predicates* in SPath leverage the full set of XPath expression as predicates and are used to filter nodes based on some criterion. SPath's *functions* are assigned two categories: one denoting functions which are impossible to express as axes without changing XPath's axis syntax, and the other referring to functions that get hold of literal properties, such as `name`, `namespace URI`, `selector`, etc.. Theoretically, with appropriate extensions, Schematron is able to (re-)express the constraint taxonomy yielded by the SPath language, as SPath is likewise based on context-dependent XPath queries.

5.2 Security Policy Validation and Authoring

For as much as almost four decades, extensive research has been undertaken in exploring techniques for verifying that security policies are correctly enforced [225], in such a way that the system, and, in particular its software components, exhibit the intended security properties [151, 102]. Especially against the background of the service-oriented evolution, the interoperable end-to-end security behavior of computing systems is a major concern in security policy

research. A good survey of three decades of information-flow based security is provided by [209, 140], culminating in programming languages for specifying and enforcing information-flow security policies. A variety of approaches exist that take a type theoretic approach to policy validation ([189, 234, 170, 248, 129, 221, 169, 51, 235, 126], to name a few) and use *security-typed languages* to augment variables and expressions with specific policies that get verified by security-type checking when enforced. Most approaches fall back on the original lattice model of information-flow control in terms of type systems for static program verification [83] and capitalize on the virtue of type-abstracted policies being rendered composable, as long as the signatures of the two sub-type systems are in line with each another.

Another approach for policy specification, composition, validation and enforcement are *semantic-based* security policy models, such as [199, 30, 200, 207, 55, 152, 49]. Here, powerful reasoning is possible on security-typed information-flow policies, as security properties can be specified in terms of the semantics of the end-to-end program behavior, yielding a unified vocabulary for end-to-end policies. Methods include extensions of the λ-calculus for information-flow analysis, such as the constraint-based Core ML [199, 200] for decidable type inference and the Dependency Core Calculus [30] with corresponding semantic model capable of noninterference proofs. Some approaches are specific to a certain policy domain, such as authorization and trust policy frameworks [207, 62, 208, 130, 12], while others concentrate on specific programming languages for implementation, like Java [55, 49]. Also methods for general policy compilation, composition and parameterization in terms of other policies [54, 55] exist.

In the SOA and component software domain, there has been extensive research in similar directions, such as [186, 146, 120, 182, 17, 178, 56, 128, 34, 245, 133] and many more. In the domain of Web Services as an instantiation of SOA, policies are a hot field in general, whereby semantically rich security policy management approaches in particular ([32, 143, 187, 148, 217, 198], to name a few), open a wide area of research for SOA practitioners and practitioners from the Semantic Community. Such semantic policy management approaches range from simple utilizations of ontologies standards for policy representation [32], to combinations of various Web Services security standards with ontologies, such as integrations with WS-Policy [143, 148] and WS-Agreement [187], as well as with Apache's WSS4J [217] for providing the functionality of XML-Signature and XML Encryption, respectively. A method that uses a similar notion of abstract policy template as our Schematron approach is presented in [198], however inventing a new language called High-Level Objective-based Policy for Enterprises (HOPE) for generating WS-Policy assertion vocabulary, as opposed to our mediation approach. Though major portions of the mentioned research aim to overcome limitations of syntactic policy validation and authoring approaches, we believe that policy syntax validation will stay an indispensable method in the future that benefits from emerging semantic methods. In particular where policies, enforcement and validation are complex, such as for the WS-Security composite standard, syntactic constraint schemata like the ones we provide with Schematron, will remain very valuable.

5.2.1 WS-Policy and WS-Security Validation

An example of a loosely-coupled and widely standardized approach to information-flow policies at the operational level is WS-SecurityPolicy for end-to-end security of WS-Security-augmented

Web Services communications. WS-SecurityPolicy represents a security-typed information-flow control policy language typed by XML Schema, which is embedded in the logic of WS-Policy and used for Web Services security transactions. WS-Policy lends itself to implementing services registration and discovery techniques (e.g. [196]), and, naturally for transaction management [167]. Different policy models and languages (including WS-Policy) are reviewed in [197] as well as their applicability for SOA Policy Management. The authors argue that WS-Policy is specific to Web Services only, and, as being a low-level policy language, WS-Policy is not very suitable for (1) policy derivation in a service composition, nor for (2) modeling "Change" support and generally (3) lack a means for business-oriented policy specification. In contrary, the authors deem policy frameworks such as Rei [142] and KAoS [231, 232] being more suitable for high-level SOA policy representation. Nonetheless, they identify mechanisms and tooling support for the translation from high level business-oriented policies into low-level WS-Policy expressions, i.e. for *policy refinement*, to be particularity useful. This is because WS-Policy is standardized and widely adopted for defining requirements and capabilities of Web Services. Policy refinement implies a *compliance validation* of higher against lower level policy (and/or vice versa).

In our eyes and in objection to [197], the WS-Policy grammar yields a generally applicable policy language and is not necessarily restricted to Web Services (a policy may but does not have to be attached to a Web Service using the framework). Moreover, with respect to Web Services, goal (1) from above is accomplishable also by the WS-Policy framework, as the specification outlines ways to obtain effective policies as an aggregate of all policies attached to an endpoint. Moreover, research exists (e.g. [167, 79]) for policy compositions along sequences of endpoint invocations (i.e. service compositions). Obviously, WS-Policy's boolean aggregation model is well-suited for policy intersection and composition in the course of orchestrating or choreographing composite services. Likewise, statement (2) misses the fact that also WS-Policy's grammar is arbitrarily extensible (just as Rei and KAoS) with a richer formal semantics - such as ontologies - if desired. With respect to clause (3), the mentioned SCA specifications and corresponding SCA Policy Framework [48] is an attempt to provide a frame for business-oriented policy specification based on so called SCA Intents [211]. Consequently, the real inhibitor for SOA policy management with the help of WS-Policy is the lack of tools, frameworks and foremost the absence of sophisticated high level vocabularies, rather than the nature of WS-Policy by itself. The domain-specific standard language extensions currently available for WS-Policy all treat of low-level Web Services capabilities and requirements, such as WS-ReliableMessaging, WS-Transaction and WS-Security for reliable messaging, transaction management and end-to-end security in Web Services communications, respectively.

In further discussions we will concentrate on the WS-Security composite standards, as this is a central element to this thesis. Methods verifying and choreographing WS-Security by utilizing WS-SecurityPolicy [123, 114] deal with controlling the flow of information within a security transaction as well as with the mediation of that transaction between two Web Services. This is what standards implementors of Web Services security engines will usually be concerned with. For meta-specifications on WS-Security, like WS-I profiles and individual application profiles of WS-Security, it is desirable to provide a validation mechanism independently, such that their machine readable definitions can be reused. This happens either dynamically at runtime by

checking SOAP messages or statically, or by constraining the valid set of policies [212] (such as WS-SecurityPolicy in case of BSP - compare Chapter 2).

Web Services Policy Language (**WSPL**) [40] is an attempt for abstract policy specification, in parts meta-modeling Web Services policy specifications, in particular XACML. WSPL is being sheparded by OASIS and is meant to fulfill a very specific purpose: in essence, WSPL will become the XACML token profile (compare the discussion on token profiles in Section 2.2). It is therefor less generic compared to the Schematron approach and not as wide in its applicability, while our method allows for both, vertical and horizontal policy compliance. See Listing 5.10 that shows an WSPL policy in an abbreviated syntax.

Listing 5.10: A WSPL policy set using an abbreviated syntax

```
1  PolicySet (target=<port type>) {
2    PolicySet (target=<operation/message>) {
3       Policy (target=<aspect>) {
4          Rule {
5             <predicate>, ...
6          } ...
7       } ...
8    } ...
9    Policy (target=<aspect>) {
10      Rule {
11         Rule {
12           Member-level = 'Gold',
13           Transaction-Fee = 5 }
14         Rule {
15           Member-level = 'Gold',
16           Time >= 9pm,
17           Time <= 6am }
18         Rule {
19           Member-level = 'Tin',
20           Transaction-Fee = 25 }
21      } ...
22   } ...
23 }
```

The `PolicySet` reminds of the SCA notion of policy, which assigns high-level policy directives - likewise specified in sets of policy - to SCA components. WSPL defines a complex policy model that is largely in line with the definition of IETF's Policy Core Information Model (PCIM) [165] which supplies an object model for representing policy information. As the Listing conveys, policies can be part of sets of policies and define rules which in turn define predicates. Lines (9-22) show a fictitious sample policy that captures certain QoS security aspects, which the authors call Quality of Protection (QoP) in [40].

WS-PolicyAssertion [42] and WS-Security policy profile of WS-PolicyConstraints [41] has some affinities with our approach of validating WS-Security at runtime, as was described in Chapter 3. WS-PolicyConstraints is limited in its applications however, not the least because WS-PolicyAssertions uses a subset of XPath that only supports absolute XPath expressions (i.e. all start with "`//<doc root>`"). Moreover, it does not support any numbered elements (e.g. `x[1]`), is incapable of defining query functions in the XPath expressions (e.g. `[@PasswordType="PasswordDigest"]`) and references are only possible to text elements or the values of XML attributes in the terminal element of the XPath expression. This subset of XPath is inadequate and overly constrained to meet requirements for complex (secu-

rity) requirements, such as co-occurrence constraints. The language is not expressive enough to capture natural language conditions, such as the ones the BSP defines, at a degree schematrons are capable of doing so. Though Schematron does not define an optimal subset of XPath, the subset is far more flexible and capable of each of the exemplified XPath capabilities that WS-PolicyAssertion lacks. As mentioned, the efforts to provide an XSLT 2.0 skeleton version of Schematron along the ISO standard is a major advancement in the support for powerful XPath queries. Further research is needed to specify an optimal subset that retains as much expressivity as possible while preserving the ability to match the potential nodesets specified by each XPath expression. The existent BP validation engine for the WSRR (compare Section 3.1.1.1) supplies an extension of Schematron's reference implementation with more powerful, custom functions.

There are a number of model-driven approaches in the domain of WS-Security, ranging from methods that verify Web Services compositions [110, 70], to new languages to enforce user-defined Security [225, 55, 33], to UML [188] modeling approaches that supply a graphical languages for modeling WS-Security [125] and access control policies [53]. Approaches that define security-typed languages and use a type-theoretic approach for policy verification are often used for runtime SOAP validation against the abstract policy specifications (e.g. [118, 124, 123]) and in essence do what we described as dynamic message validation, however for a policy standard as opposed to the BSP interoperability profile. But also static propositions exist, such as [247], who introduce a framework for validating electronic health records against XML abstracts. A method similar to our static policy validation method is proposed by the authors of [60], who supply a formal semantics for WS-SecurityPolicy in terms of an abstract link-language. Abstract policies written in this language are used to compile concrete policies and to verify these policies against the goals defined in the abstract policies. A theorem prover is called and provides assurances that the policy goals are met for any number of receivers and consumers governed by that policy.

The static validation of BSP conformance presented in this thesis is inspired by the preliminary research of [210, 179], that uses PROLOG [223] to verify BSP meta-requirements statically in WS-SecurityPolicy policies. We expanded on these works as well as on [168]: in addition to the Schematron validation of policies, we add an XML mediation architecture that we motivate with BSP conformance and verify by a proof-of-concept implementation. Sources for general discussions on contract logic approaches for SLA Management with the help of prolog and logic programming can be found in [195, 60]. An approach that discusses a language for domain independent policy assertions embedded within WS-Policy [43] also has similarities with our mediation method. The difference is, that we provide a standardized assertion language with Schematron and a self-contained custom APM, as opposed to WS-Policy's grammar. This can be seen as a drawback of our approach, as WS-Policy is standardized and able to express the policy requirements we express in the APM. Another paper notable in this context is [33], that describes the application of WS-Policy for automatizing services interactions based on services preferences specified in the WS-Policy policies.

A good overview of *XML Mediation* techniques and objectives is given by [229], who analyze various aspects of interoperability for data integration in SOA-based systems. An interesting approach building upon model-driven mediation methods is outlined in [227], who describe a rule-based *XML Mediation* that works separated from the actual application of the XML data

that is used. This is affine to our approach, however for data validation of complex messages in health-care SOA environments, in particular for privacy anonymization. So the closest overlap to our work is the dynamic BSP validation of WS-Security, as discussed in the implementation in Section 3.2.1. The authors' policy authoring method requires the definition of rules utilizing a *conceptual data notation*. Rules consist of two components: *constraint conditions* and *actions*, the latter being taken *iff* the respective constraint condition applies. This is analogous to Schematron's rule and assert/report element constructs. To retain independence from both industry-specific standards and implementation-specific data representation, the authors define two mappings: the obvious mapping between the data notation in the mediation rules to the concrete representation of the implementation, and another mapping that represents the implied knowledge inherent in the rules. As this implied knowledge is usually dependent on data structures used by the industry-specific standards in use, the authors propose ways to import this knowledge independently and thereby meet major concerns in SOA migration and interoperability. They give examples with respect to the HL7 [91] health care standard.

Further, the approach distinguishes rules into *privacy constraint* rules and *data constraint* rules and a formalism is given by which the rules are sufficiently described. Constraint Conditions that represent the predicates of actions are divided into *Simple Constraint Conditions* (**SCC**) and *Complex Constraint Conditions* (**CCC**), the former belonging to rules that depend on the evaluation results of single simple conditions, while the latter are part of rules that depend on evaluation results of combinations of more than one condition. The work then describes a set of privacy constraint rule (R1-R4) and a data constraint rule (R5) and chooses XACML and Schematron as the implementation-specific standards for formalizing the rules in XML. They give translations of SOAP envelope requirements R1 through R5 to both of these technologies, and leverage existing XSLT transformation patterns of rule R1 to R4 for XACML and of rule R5 for Schematron. With the help of this mediation method it is easier to maintain consistency of validation rules, in case they are updated in accord with the change in the specifications. The rules are reusable without modification for the specifications (and versions) for which the rules have been implemented.

CHAPTER 6

Conclusions

"Technical skill is mastery of complexity while creativity is mastery of simplicity."

E. Christopher Zeeman [249]

6.1 Policy Authoring

The Schematron validation approach elaborated in the course of this thesis, is a promising mechanism for general verifications of security profiles in terms of policy validation, but also for profiles originating from other areas than security. Schematron is standard and due to the simplicity of this language, while being extraordinarily expressive, the implementation of validation rules for policies is easily adaptable also to custom (e.g. legislative) requirements. Since schematron is explanatory, meaning that it is able to provide (human-readable) validation responses, a wide range of profiles and specifications apart from WS-I profiles can be covered, including arbitrary specialized application profiles. Requirement representation as schematron schema encapsulates validation rules form actual implementations, expressing XML validation in XML, while the schemata remain adaptable to changes in the targeted specifications or data models. Schematron allows for validation beyond XML Schema conformance, by the definition of conditional assertions on the syntax of XML documents (i.e. XML policies).

Because Schematron is simple, scalable, rich in expressiveness and explanatory, this highly intuitive validation language is able to express non-functional natural-language (security) requirements as syntactic constraints on (security) policies in a machine tractable manner, lending itself to a mighty validation mechanism. Our SVP authoring approach supports multiple levels of expertise, allowing also policy authors that are not domain experts to succeed in policy management with minor efforts. The schematrons we provide for policy mediation are totally model-agnostic and system-independent, allowing for horizontal and vertical policy compliance validation, while policy rule adaptions/redefinitions at the operation level in accordance to higher-level policy changes (or changes in the specifications or profiles to validate) are easy to

be realized. This renders the approach highly flexible, in addition to the virtue of the validation method following a normed procedure that produces output in a standardized format.

In the course of a service's life-cycle also the IT policies change frequently, since business is ever-changing. Hence, a (partial) automation of the correlation of IT policies with current business-level requirements is very desirable. The (semi-)automatic policy authoring process we establish, informs authors of policy expressions that violate current architectural requirements. This makes the developer capable of meaningful (security) governance and enables her to adapt IT policies to (ever-changing) business situations. The validation of operational-level IT policies against architectural-level requirements and the result thereof can be used to guide a policy developer in creating or selecting IT policies that correctly and fully map the current business policy. Existing standards for finding Web services are keyword-based (e.g. UDDI) and therefor require a lot of manual effort to select services with desired capabilities. Though efforts in the semantic community (e.g. [65]) promise to amend this drawback in the future, a (simpler) syntactic validation, such as our SVP mediation provides with the Schematron standard, is an accurate supplement that can help improve search and selection of services according to business requirements. The SVP authoring enhances the consumability of policy development through policy reuseability and complexity reductions by means of abstract architecture policy templates. This reduces workload for the policy author and increase the value of products delivering such tools.

6.2 BSP Conformance

BSP compliance is needed to make web services securely interoperable within heterogeneous IT infrastructures and multiple (legally separated) organizations, rendering BSP conformance validation a must. Adherence to BSP's requirements diminishes vulnerabilities for services and their interactions, inter alia helping to prevent security token substitutions, replay attacks on security tokens, leaking user name token passwords and plain text guessing attacks. Considering the amount of sensitive information that crosses enterprise boundaries in a SOA (possibly passing intermediaries), helping the prevention from these vulnerabilities is an absolute necessity. BSP easily lends itself to validating compliance by checking WS-SecurityPolicy features for each message at runtime. However the overhead of doing this is significant. Our dynamic BSP conformance validation interceptors provide us with a good indicator for the processing overhead introduced through checking message exchanges on the wire, and serves as the basis for the estimation of the performance overhead introduced through runtime validation. We measured it as being in order of a tenth of a second each time validation is performed for all 81 testable BSP requirements. Though runtime validation of BSP is obviously costly, it provides great coverage, as more than a third of all profile requirements BSP defines can be covered by this method. However, message validation has the major drawback of procuring the developer no guarantees re. BSP compliance until runtime.

Therefor we explored a novel approach based on statically checking WS-SecurityPolicy syntax with the help of Schematron. This approach infers from the policy assertions the families of WS-Security-enriched SOAP messages that can be generated and hence the BSP compliance. BSP conformance validation within "static" WS-SecurityPolicy policies simplifies the policy au-

thoring by assisting authors in the difficult process of correct policy development itself; correct in the sense that a policy fully maps the business requirement of BSP conformance, and difficult, because SOA security is complex. Authors often have no idea if their produced policy is WS-I compliant (i.e. secure and interoperable) or not. BSP conformance validation considerably improves consumability, as preventing policy authors from being forced to immerse deeply into the realms of WS-Security, WS-SecurityPolicy and SOA security, while being able to provide assurances on compliance before real-time. We adopt schematrons on the class of WS-SecurityPolicy policies for a guided validation that instructs policy authors on violations with respect to a given requirement, supplying suggestions of alternative compliant policy syntax.

The usefulness of static WS-SecurityPolicy policy validation against BSP is underlined by the fact that it provides build-time compliance guarantees, partially avoiding the WS-Security message validation overhead by obviating the need to test requirements within message exchanges that have been covered already by the static policy validation. This advantage is valid for all (application) profiles that can be accounted for by the method described in this thesis. Moreover, static validation is able to extend coverage of BSP beyond those requirements verifiable from the SOAP envelope, for a subset of BSP's requirements that are amenable to validation within the policy model, that is, for 12 requirements (compare Table 4.1, last column). However, the overall coverage of WS-SecurityPolicy validation against BSP is limited (it covers only 18 BSP requirements and two recommendations), since it is non-trivial to map BSP requirements to policy assertions (and in many cases impossible). Even when possible, it requires intimate knowledge of BSP itself and the WS-Security headers, as well as knowledge of WS-SecurityPolicy and the mechanism by which a SOAP runtime produces messages in order to comply with a deployed policy.

Moreover, BSP, WS-SecurityPolicy and WS-Security are specifications and therefor innately open to interpretation to a certain degree - as all specifications. These sources of ambiguity can be overcome, by testing the behavior of different platforms off-line, given a particular set of policy assertions. We can leverage the dynamic validation implementation to test the Axis2's policy behavior and to cross-check the rules inferred from the policy standard. This way, we can unambiguously derive WS-SecurityPolicy assertions expressed with static schematron rules that render runtime message exchanges (produced by the tested platform) compliant to a subset of the dynamic BSP requirements. The successful implementation of this subset of BSP as static schematron rules for WS-SecurityPolicy validates our SVP policy mediation method with respect to complex (Architecture level) security requirements. The implementation highlights the broad applicability of Schematron for the management of (the security life-cycle of) policies.

6.3 Future Work

First of all, it must be stated, that this work is of a very practical nature, lacking any thorough theoretic considerations. In the future, we hope to explore formal foundations of schema languages and policy specification. There are a number of challenging research areas in the field of SOA (security) policy management, in particular meta-modeling tools, transformation technologies and general MDD/MDS engineering techniques for best-practice guidance harbor a great potential for future research. We hope to study other kinds of policies beyond WS-Policy and

WS-SecurityPolicy in more detail, and other kinds of profiles beyond BSP. Though it makes sense to adopt an intermediate model between the schematron and the operational policies, because we engineer security requirements form scratch in our approach, one could think of a method that is superior in its application to many heterogeneous sources of (security) policy requirements that are potentially available on hands as a prerequisite and likely to adhere to different policy models at the architecture level of abstraction. An idea would be to leave out a custom intermediate architecture model separated from the SVP but use Schematron alone or Schematron in combination with WS-Policy as the architecture model: due to Schematron's abstract patterns, we could provide base patterns that are to be extended for specific solutions. Referring to Listing 3.10, an abstract pattern could be defined that matches the availability of `wssp:SignedElements` assertion. When providing this pattern as standard base validation rule for general payload integrity protection, any extension would conform to that pattern and validation could be unified. The validation rules get customizable to the message payload schema used in individual solutions, and, reuseability is further promoted.

Furthermore, the schematron rules could be grouped using the WS-Policy model instead of our custom APM operators, as this would follow a widely used Web Services standard. This way, we could define WS-Policy meta-policies on WS-Policy policies, mediated by embedded schematron rules. Another way to combine WS-Policy and Schematron would be to augment the XML Schema definition of a policy model extension with advanced schema-constraints, such as schematron augmentations within the WS-SecurityPolicy schema definition. We plan to examine this idea against the background of BSP, WS-(Security-)Policy and Schematron. Another interesting research direction is the combination of Schematron's rule-based syntactic validation with semantic validation approaches. For example, a semantically rich architecture policy model (i.e. a policy ontology) could be introduced that allows the verification of complex semantic rules that are valid independent of heterogeneous operational policy standards. Many policy standards and languages that are available semantically do the same thing and could be abstracted at the architecture level that way, possibly embedding (supplementary) standard-specific schematrons for individual policy languages/models in addition to the semantic rules for conceptional validation.

The limited scope of this thesis got into the way of implementing XPath indexing for speeding up the SVP authoring: if each requirement defined in an APT is indexed with xpath, the `context` of pattern requirements (i.e. `Requirement` elements that do define a `Property` child) could be constructed automatically, when creating an APS. This way, only the xpath for the test attributes of the respective `Assert` elements (which map to Schematron's `assert` elements) would have to be specified by the policy developer, when constructing an architecture policy, while she would not need to care about the rule's XML context. This however implies that requirements have an exact match in the policy and that sub requirements of requirements in the APM map to XML sub-element of the respective containing XML policy elements. Otherwise architecture requirements can not be correctly validated. Our prototype requires the policy author to type the xpath expressions for the assertion `context` and `test` attributes, respectively, using the prototype eclipse UI, which constitutes another feature that deserves considerable improvement. An option would be to provide a graphical view of the policy model's XML Schema that allows the selection of a schematron rule's `context` and `test` xpath in terms of defined

schema types.

After the three step policy authoring model was implemented, we recognized that an extension with a forth step could be desirable in some situations. Between step (1) and step (2) an architectural requirement configuration (**ARC**) could be introduced that confers the conjunction and (exclusive) disjunction of subsets of APSs. Hence, APSs requirements could be reused with multiple groupings in different ARCs. The current approach would require to define a new APS with corresponding grouping that fully incorporates all requirements that an existent APS defines as opposed to a reuse of specific individual requirements an APS defines when introducing the ARC as intermediary step. A particularly promising area of research in the context of SOA policy and SOA security are Natural Language Processing (**NLP**) methods, since security and policy requirements are in general often stated in natural language without any formalization. NLP technologies for the automatic processing of (application) profiles and translation to machine verifiable build-time and run-time rules would obviate manual examination. Our prototype is designed in such a way, that a NLP pre-processing step could be integrated with the current architecture with minor efforts.

With regards to WS-I compliance, the automatic translation of BSP requirements to dynamic SOAP header syntax rules with the help of NLP is realistic to be implemented, considering the scheme of BSP requirements and the specification's document properties that in sum should exhibit enough information as to be able to extract the SOAP envelope rules. However, deriving static policy validation rules by NLP methods will be overly difficult, as this implies inferring the families of SOAP header meta-data from the policy standard. Hence, also the WS-SecurityPolicy specification document would have to be automatically processed, in order to account for BSP requirements statically. We see no obvious possibility how to provide this automated, as the interrelationships between WS-Security, BSP and WS-SecurityPolicy are too complex and ambiguous for this task (compare the discussion on WS-I conformance in Section 3.1.1).

APPENDIX A

A.1 List of Abbreviations

Table A.1: Abbreviation Index

APC	Architectural Policy Configuration
API	Application Programming Interface
APM	Architectural Policy Model
APS	Architectural Policy Selection
APT	Architectural Policy Template
BP	Basic Profile
BSP	Basic Security Profile
C14N	Canonicalization
CA	Certificate Authority
CCC	Complex Constraint Conditions
COM	Component Object Model
CORBA	Common Object Resource Broker Architecture
DNF	Disjunctive Normal Form
DoS	Denial of Service
DTD	Document Type Definition
EaaS, XaaS, *aaS	Everything as a Service
EAI	Enterprise Application Integration
EMF	Eclipse Modeling Framework
HIPAA	Health Insurance Portability and Accountability Act
ID-WSF	Identity Web Services Framework
IETF	Internet Engineering Task Force
J2EE	Java 2 Enterprise Edition
MDD	Model-Driven Development
MDS	Model-Driven Security

Continued on Next Page...

NLP	Natural Language Processing
OASIS	Organization for the Advancement of Structured Information Standards
OPM	Operational Policy Model
PKI	Public Key Infrastructure
PSVI	Post-Schema-Validation-Infoset
QoS	Quality of Service
RBAC	Role Based Access Control
REL	Rights Expression Language
RPC	Remote Procedure Call
SAML	Security Assertion Markup Language
SCA	Services Component Architecture
SCC	Simple Constraint Conditions
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SoC	Service-oriented Computing
SSL	Secure Socket Layer
STS	Security Token Service
SVP	Schematron Validation Pipeline
SVRL	Schematron Validation Report Language
SwA	SOAP with Attachments
TC	Technical Committee
TCP/IP	Internet Protocol Suite
TLS	Transport Layer Security
UN/CEFACT	United Nations Center for Trade Facilitation and Electronic Business
W3C	World Wide Web Consortium
WS-I	Web Services Interoperability Organization
WSDL	Web Services Description Language
WSPL	Web Services Policy Language
WSRR	WebSphere Services Registry and Repository
WSSS	Web Services Standards Stack
XACML	Extensible Access Control Markup Language
XMKS	Extensible Key Management Specification
XML	Extensible Markup Language
XrML	Extensible Rights Markup Language
XSLT	Extensible Stylesheet Language Transformation

A.2 Architectural Policy Model

Listing A.1: XML Schema Definition of the APM

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:tns="http://www.ibm.com
       /ns/policy"
3      targetNamespace="http://www.ibm.com/ns/policy" blockDefault="#all"
4      elementFormDefault="qualified">
5      <xs:element name="Requirement">
6          <xs:complexType>
7              <xs:complexContent>
8                  <xs:extension base="tns:operatorContentType">
9                      <xs:sequence minOccurs="0" maxOccurs="unbounded">
10                         <xs:element name="Property" type="tns:PropertyType"/>
11                     </xs:sequence>
12                     <xs:attribute name="name" type="xs:anyURI" use="required"/>
13                     <xs:anyAttribute namespace="##any"/>
14                 </xs:extension>
15             </xs:complexContent>
16         </xs:complexType>
17     </xs:element>
18     <xs:complexType name="operatorContentType">
19         <xs:sequence>
20             <xs:choice minOccurs="0" maxOccurs="unbounded">
21                 <xs:element ref="tns:Requirement"/>
22                 <xs:element ref="tns:AllOf"/>
23                 <xs:element ref="tns:OneOf"/>
24                 <xs:any namespace="##other"/>
25             </xs:choice>
26         </xs:sequence>
27     </xs:complexType>
28     <xs:element name="AllOf" type="tns:operatorContentType"/>
29     <xs:element name="OneOf" type="tns:operatorContentType"/>
30     <xs:complexType name="PropertyType">
31         <xs:sequence maxOccurs="unbounded">
32             <xs:element name="Assert" minOccurs="1" maxOccurs="unbounded">
33                 <xs:complexType>
34                     <xs:simpleContent>
35                         <xs:extension base="xs:string">
36                             <xs:attribute name="test" type="xs:string"/>
37                         </xs:extension>
38                     </xs:simpleContent>
39                 </xs:complexType>
40             </xs:element>
41         </xs:sequence>
42         <xs:attribute name="name" type="xs:string" use="required"/>
43         <xs:attribute name="context" type="xs:string" use="required"/>
44     </xs:complexType>
45 </xs:schema>
```

A.3 BSP Experiments

A.3.1 WS-SecurityPolicy Example Policy

Listing A.2: Example policy used for estimating the overhead introduced through dynamic BSP validation

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <wsp:Policy wsu:Id="SigEncr" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis
       -200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://schemas.xmlsoap.org/
       ws/2004/09/policy">
3    <wsp:ExactlyOne>
4      <wsp:All>
5        <sp:AsymmetricBinding xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
             securitypolicy">
6          <wsp:Policy>
7            <sp:InitiatorToken>
8              <wsp:Policy>
9                <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/
                     securitypolicy/IncludeToken/AlwaysToRecipient">
10                 <wsp:Policy>
11                   <sp:RequireThumbprintReference/>
12                   <sp:WssX509V1Token10/>
13                 </wsp:Policy>
14               </sp:X509Token>
15             </wsp:Policy>
16           </sp:InitiatorToken>
17           <sp:RecipientToken>
18             <wsp:Policy>
19               <sp:X509Token sp:IncludeToken="http://schemas.xmlsoap.org/ws/2005/07/
                     securitypolicy/IncludeToken/Never">
20                 <wsp:Policy>
21                   <sp:RequireThumbprintReference/>
22                   <sp:WssX509V3Token10/>
23                 </wsp:Policy>
24               </sp:X509Token>
25             </wsp:Policy>
26           </sp:RecipientToken>
27           <sp:AlgorithmSuite>
28             <wsp:Policy>
29               <sp:TripleDesRsa15/>
30             </wsp:Policy>
31           </sp:AlgorithmSuite>
32           <sp:Layout>
33             <wsp:Policy>
34               <sp:Strict/>
35             </wsp:Policy>
36           </sp:Layout>
37           <sp:IncludeTimestamp/>
38           <sp:OnlySignEntireHeadersAndBody/>
39           <sp:SupportingTokens xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
                 securitypolicy">
40             <wsp:Policy>
41               <sp:UsernameToken>
42                 <wsp:Policy>
43                   <sp:HashPassword/>
44                 </wsp:Policy>
45               </sp:UsernameToken>
46             </wsp:Policy>
47           </sp:SupportingTokens>
```

```
48          </wsp:Policy>
49        </sp:AsymmetricBinding>
50        <sp:Wss10 xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
51          <wsp:Policy>
52            <sp:MustSupportRefKeyIdentifier/>
53            <sp:MustSupportRefIssuerSerial/>
54          </wsp:Policy>
55        </sp:Wss10>
56        <sp:SignedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy
              ">
57          <sp:Body/>
58        </sp:SignedParts>
59        <sp:EncryptedParts xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/
              securitypolicy">
60          <sp:Body/>
61        </sp:EncryptedParts>
62        <ramp:RampartConfig xmlns:ramp="http://ws.apache.org/rampart/policy">
63          <ramp:user>client</ramp:user>
64          <ramp:encryptionUser>service</ramp:encryptionUser>
65          <ramp:passwordCallbackClass>com.ibm.axis2.sample.PWCBHandlerASYM</
                ramp:passwordCallbackClass>
66          <ramp:signatureCrypto>
67              <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin">
68              <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.type"
                    >JKS</ramp:property>
69              <ramp:property name="org.apache.ws.security.crypto.merlin.file">client.
                    jks</ramp:property>
70              <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.
                    password">apache</ramp:property>
71            </ramp:crypto>
72          </ramp:signatureCrypto>
73          <ramp:encryptionCrypto>
74            <ramp:crypto provider="org.apache.ws.security.components.crypto.Merlin">
75              <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.type"
                    >JKS</ramp:property>
76              <ramp:property name="org.apache.ws.security.crypto.merlin.file">client.
                    jks</ramp:property>
77              <ramp:property name="org.apache.ws.security.crypto.merlin.keystore.
                    password">apache</ramp:property>
78            </ramp:crypto>
79          </ramp:encryptionCrypto>
80        </ramp:RampartConfig>
81      </wsp:All>
82    </wsp:ExactlyOne>
83  </wsp:Policy>
```

A.3.2 WS-Security example

Listing A.3: Example SOAP envelope resulting from the enforcement of the policy from Listing A.3.1

```
<?xml version='1.0' encoding='utf-8'?>
<soapenv:Envelope xmlns:soapenv="..." xmlns:xenc="...">
  <soapenv:Header xmlns:wsa="...">
    <wsse:Security xmlns:wsse="..." soapenv:mustUnderstand="1">
      <wsu:Timestamp xmlns:wsu="..." wsu:Id="Timestamp-9716945">
        <wsu:Created>2009-01-15T20:47:55.649Z</wsu:Created>
        <wsu:Expires>2009-01-15T20:52:55.649Z</wsu:Expires>
      </wsu:Timestamp>
      <xenc:EncryptedKey Id="EncKeyId-urn:uuid:39596182BE34C3C7D5123205247568320">
        <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"
            />
          <ds:KeyInfo xmlns:ds="...">
            <wsse:SecurityTokenReference>
              <wsse:KeyIdentifier EncodingType="..." ValueType="...">
                HYL3...
              </wsse:KeyIdentifier>
            </wsse:SecurityTokenReference>
          </ds:KeyInfo>
          <xenc:CipherData>
            <xenc:CipherValue>N3COu4...</xenc:CipherValue>
          </xenc:CipherData>
          <xenc:ReferenceList>
            <xenc:DataReference URI="#EncDataId-3278348" />
          </xenc:ReferenceList>
        </xenc:EncryptedKey>
        <wsse:BinarySecurityToken xmlns:wsu="..." EncodingType="..." ValueType="...
            x509v1" wsu:Id="CertId-148082">MIICTDCC...</wsse:BinarySecurityToken>
        <ds:Signature xmlns:ds="..." Id="Signature-32486590">
          <ds:SignedInfo>
            <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-
                c14n#" />
            <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-
                sha1" />
            <ds:Reference URI="#Id-3278348">
              <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
              </ds:Transforms>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
              <ds:DigestValue>V9mIm1la8gUegbVArobZfUI2ixs=</ds:DigestValue>
            </ds:Reference>
            <ds:Reference URI="#Timestamp-9716945">
              <ds:Transforms>
                <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
              </ds:Transforms>
              <ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
              <ds:DigestValue>ynE5lTvNOgYDX4ru9oBD0M0OmAo=</ds:DigestValue>
            </ds:Reference>
          </ds:SignedInfo>
          <ds:SignatureValue>LsUET6...</ds:SignatureValue>
          <ds:KeyInfo Id="KeyId-4414010">
            <wsse:SecurityTokenReference xmlns:wsu="..." wsu:Id="STRId-30983464">
              <wsse:Reference URI="#CertId-148082" ValueType="...#X509v1" />
            </wsse:SecurityTokenReference>
          </ds:KeyInfo>
        </ds:Signature>
    </wsse:Security>
```

```
53      <wsa:To>http://127.0.0.1:8080/axis2/services/secureSample</wsa:To>
54      <wsa:MessageID>urn:uuid:E914DD4C86122F03E21232052475681</wsa:MessageID>
55      <wsa:Action>urn:echo</wsa:Action>
56    </soapenv:Header>
57    <soapenv:Body xmlns:wsu="..." wsu:Id="Id-3278348">
58      <ns1:echo xmlns:ns1="http://sample.axis2.ibm.com">
59        <param0>Hello world</param0>
60      </ns1:echo>
61    </soapenv:Body>
62  </soapenv:Envelope>
```

A.4 WS-SecurityPolicy Conformance Validation against BSP

A.4.1 Static Validation Rules

Table A.2: Rule for **R3212**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(0)	MUST	6.7 [Security Header Layout] Property 7.1 Layout Assertions

There is no need to validate a policy against this requirement, since WS-Security defines an ordering of the SECURITY_HEADER according to the "prepending rule" already and the runtime platform has to handle correct ordering.

Table A.3: Rule for **R3227**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(0)	MUST	6.2 [Timestamp] Property

This requirement is innately fulfilled by any policy. No matter how many `sp:IncludeTimestamp` assertion a policy contains, the boolean [Timestamp] property renders a SECURITY_HEADER to either contain a timestamp or to contain none.

Table A.4: Rule for **R5421**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(0)	SHOULD	7.1 AlgorithmSuite Assertion

The default SIGNATURE_METHOD defined in WS-SecurityPolicy by the [Sym Sig] and [Asym Sig] properties are equal to "http://www.w3.org/2000/09/xmldsig#hmac-sha1" or "http://www.w3.org/2000/09/xmldsig#rsa-sha1" by default.

Table A.5: Rule for **R5621**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(0)	MUST	7.1 AlgorithmSuite Assertion

No need to validate this requirement, since the [Asym KW] property defined in WS-SecurityPolicy has either a value of "http://www.w3.org/2001/04/xmlenc#rsa-1_5" or "http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p".

Table A.6: Rule for **R3002**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(1)	MUST	6.1 [Algorithm Suite] Property 7.1 AlgorithmSuite Assertion
	Assertions	

```
1 /wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
```

To check whether there is a SIG_REFERENCE to an element with missing ID attribute is only possible with runtime information. Nonetheless, making the `wssp:XPathFilter20` assertion a type (2) assertion will force any SIG_REFERENCE to contain a xmldsig-filter2 transform, no matter if it is referencing elements with or without `id` attribute. But note that security policies asserting for other transforms, like `wssp:XPath10` or `wssp:AbsXPath`, could still result in valid SOAP runtime messages, since they might very well contain an `id` attribute. So this rule is not unambiguous.

Table A.7: Rule for **R5423**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	6.1 [Algorithm Suite] Property 7.1 AlgorithmSuite Assertion
	Assertions	

```
1  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
2  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
3  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
4  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
5  wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
6  wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
7  wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
8  wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:STRTransform10
9  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
10 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
11 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
12 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
13 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
14 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:XPathFilter20
```

The WSSP specification makes no clear statements about how to define SIG_TRANSFORM algorithms. So we make the assumption, that `wssp:STRTransform10` or `wssp:XPathFilter20` assertions as descendants of an `wssp:AlgorithmSuite` assertion determine the signature transformation algorithm to use and will render a runtime platform to use one of these two algorithms and thereby make it comply to R5423. There are several type (1) assertions possible that would violate R5423, however, there is no need to check for these, since the specified type (2) assertion is stronger.

Table A.8: Rule for **R5412** $\wedge$ **R5404**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(1)	MUST	7.1 AlgorithmSuite Assertion
	Assertions	

```
1  /sp:AlgorithmSuite/wsp:Policy/sp:InclusiveC14N
```

The default CANONICALIZATION_METHOD method defined by WS-SecurityPolicy is exclusive C14N, so without any assertion concerning canonicalization, a policy is valid with respect to R5412 and R5404 in either case. However the `wssp:InclusiveC14N` assertion enforces the usage of inclusive C14N at runtime and is therefor violating.

Table A.9: Rule for **R5420**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	SHOULD	7.1 AlgorithmSuite Assertion
	Assertions	

```
1  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
2  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
3  wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
4  wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
5  wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
6  wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
7  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
8  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
9  wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
10 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
11 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
12 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192
13 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
14 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
15 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
16 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
17 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
18 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
19 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
20 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
21 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
22 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
23 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
24 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
25 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
26 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
27 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
28 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
29 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
30 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Rsa15
31 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
32 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
33 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
34 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
35 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
36 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic192Rsa15
37 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
38 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
39 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
40 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
41 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
42 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Rsa15
43 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
44 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
45 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
46 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
47 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
48 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesRsa15
```

See notes on R5620, R5625 and R5626.

Table A.10: Rule for **R5620** ∧ **R5625** ∧ **R5626**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	6.1 [Algorithm Suite] Assertion 5.4.3 X509Token Assertion
	Assertions	

```
1  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
2  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
3  wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
4  wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
5  wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
6  wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256
7  wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
8  wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
9  wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
10 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
11 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
12 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128
13 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
14 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
15 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
16 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
17 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
18 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDes
19 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
20 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
21 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
22 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
23 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
24 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256
25 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
26 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
27 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
28 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
29 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
30 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256
31 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256
32 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256
33 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256
34 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256
35 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256
36 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/
       wssp:TripleDesSha256
37 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256Rsa15
38 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256Rsa15
39 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256Rsa15
40 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256Rsa15
41 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic256Sha256Rsa15
42 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/
       wssp:Basic256Sha256Rsa15
43 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256Rsa15
44 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256Rsa15
45 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256Rsa15
46 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256Rsa15
47 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:Basic128Sha256Rsa15
48 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/
       wssp:Basic128Sha256Rsa15
49 wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256Rsa15
50 wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256Rsa15
51 wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256Rsa15
52 wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256Rsa15
53 wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/wssp:TripleDesSha256Rsa15
54 wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy/
       wssp:TripleDesSha256Rsa15
```

Note that R5620, R5625 and R5626 match exactly the same set of assertions. Note in addition, that algorithms in the rule for R5420 are partly disjunct to the algorithms in this rule. For Example, if `wssp:Basic256Sha256Rsa15` is asserted for in a policy, R5620 is not violated, whereas R5420 is violated. On the other hand, if wssp:Basic192Rsa15 is an assertion in the policy, R5420 is not violated, whereas R5620 is violated. However, even if both `wssp:Basic256Sha256Rsa15` and `wssp:Basic192Rsa15` are specified in the same policy, it is not clear which algorithms would be used for DIGEST_METHOD and which algorithm would be utilized for ED_ENCRYPTION_METHOD at runtime. So there is no point in combing the rule for R5420 and the rule for R5620, R5625 and R5626 to one unified rule, intersecting over the algorithms of each requirement.

Table A.11: Rule for **R3033**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	5.4.3 X509Token Assertion
	Assertions	

```
1 wssp:X509Token/wsp:Policy/wssp:WssX509V3Token10
2 wssp:X509Token/wsp:Policy/wssp:WssX509V3Token11
```

WS-Security does not specify a default ValueType of an X509_TOKEN, therefor one of the two specified assertions has to be included in a policy in order to conform to BSP.

Table A.12: Rule for **R6902**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	5.4.4 KerberosToken Assertion
	Assertions	

```
1 /wssp:KerberosToken/wsp:Policy/wssp:WssGssKerberosV5ApReqToken11
```

Is clear from the WS-SecurityPolicy specification.

Table A.13: Rule for **R6903** $\wedge$ **R6904**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	5.1.1 Token Inclusion Values
	Assertions	

```
1 /wssp:KerberosToken@wssp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/
      IncludeToken/Once"
```

This single type (2) assertion makes a policy non-violating to both, R6903 and R6904, since a token inclusion value of Once means for this token, that "[...] References to the token MAY use an internal reference mechanism. Subsequent related messages sent between the recipient and the initiator may refer to the token using an external reference mechanism." However, this rule cannot guarantee, that a KERBEROS_TOKEN is an INTERNAL_SECURITY_TOKEN in the initial message exchange, nor can it guarantee that a KERBEROS_TOKEN is an EXTERNAL_SECURITY_TOKEN in subsequent changes. This must be done at runtime.

Table A.14: Rule for **R6905**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	5.4.4 KerberosToken Assertion
Assertions		

```
1 /wssp:KerberosToken/wsp:Policy/wssp:RequireKeyIdentifierReference
```

The single assertion of this rule must exist in a policy, because of the rule for R6903 and R6904. In the definition of a token inclusion value of Önceïnside the WS-SecurityPolicy specification (see comment on rule for R6903 and R6904) MAY is used twice, so there can be internal and external references to the to KERBEROS_TOKEN. The wssp:RequireKeyIdentifierReference assertion forces both, internal and external references to contain a STR_KEY_IDENTIFIER, resulting in R6905 to be valid for external references and not violated for internal references.

Table A.15: Rule for **R6302**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	6.7.1 Strict Layout Rules for WSS 1.0 7.2 Layout Assertion
Assertions		

```
1 /wssp:Layout/wsp:Policy/wssp:Strict
```

No default SECURITY_HEADER layout is defined in the WS-SecurityPolicy specification. Although layout assertions other than wssp:Strict will not necessarily violate R6302 - this depends on how the runtime platform interprets these assertions – the strict layout will guarantee that any referenced token that shall be singed occurs before the SIGNATURE that signs it and thereby makes any policy compliant to R6302.

Table A.16: Rule for **SEC_17_9**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	MUST	5.4.1 UsernameToken Assertion
	Assertions	

```
1  /wssp:UsernameToken/wsp:Policy/wssp:NoPassword
2  ( /wssp:UsernameToken/wsp:Policy/wssp:RequireDerivedKeys AND /wwsp:SignedSupportingTokens/
       wsp:Policy/UsernameToken )
3  ( /wssp:UsernameToken/wssp:Policy/wssp:HashPassword AND /wwsp:SignedSupportingTokens/wsp:Policy/
       UsernameToken )
```

This recommendation demands integrity protection if the WS-Security elements `wsse:Created` and `wsse:Nonce` elements are used inside a `wsse:Username` in conjunction with a wsse:Password of type wsse:PasswortText. Whether that is the case for a message header can only be ascertained at runtime. WS-Security however defines a digest password to be a combined hash value of `wsse:Created`, `wsse:Nonce` and a `wsse:Password` of type `wsse:PasswordText` and BSP on the other hand defines the wssp:HashPassword assertion to make usage of digest passwords mandatory for wssp:UsernameTokens. This means, we can indirectly fulfill the recommendation by requiring integrity protection for those wssp:UsernameTokens that include a wssp:HashPassword assertion, since this will force a runtime message to use `wsse:Created` and `wsse:Nonce` together with the wsse:PasswordText password (in order to be able to derive a combined hash value of all three elements). Additionally, if a wsse:Password of type wsse:PasswordText is used to derive keys for subsequent encryption, it is recommended to sign the corresponding username token. In WS-SecurityPolicy a password of type `wsse:PasswordText` can be expressed by the absence of any kind of password assertion, since `wsse:PasswordText` is the default password type. When now combining these two recommendations to a rule, we can say, that any username token utilizing digest passwords, as well as any username token that requires key derivation and which does not specifies a password assertion (i.e. type is `wsse:PasswordText`), must be signed. Note that a username tokens without password do not have to be signed.

Table A.17: Rule for **SEC_17_13**

Rule Type	*Requirement Level*	*WS-SecurityPolicy Specification*
(2)	SHOULD	5.4.1 UsernameToken Assertion 8.5 SignedEncryptedSupportingTokens Assertion 8.7 EndorsingEncryptedSupportingTokens Assertion 8.8 SignedEndorsingEncryptedSupportingTokens Assertion
		Assertions

```
1  wssp:UsernameToken/wsp:Policy/wssp:NoPassword
2  ( wssp:HashPassword AND ( wssp:SingedEncryptedSupportingTokens/wsp:Policy/wssp:UsernameToken OR
       wssp:EndorsingEncryptedSupportingTokens/wsp:Policy/wssp:UsernameToken OR
       wssp:SingedEndorsingEncryptedSupportingTokens/wsp:Policy/wssp:UsernameToken ) )
```

These assertions say, that if a password is used with a username token, a digest password should be used. In other words, a `wssp:UsernameToken` assertion without a password assertion is invalid, since this corresponds to the default behavior of using a password of type `wsse:Passwordtext`. So username tokens should either contain a `wssp:NoPassword` or `wssp:HashPassword` assertion. As the recommendation specifies further, any form of password demands confidentiality protection, which in this context corresponds to the rule that those username tokens asserting for `wspp:HashPassword` must in addition be descendant of either `wssp:SignedEncryptedSupportingTokens`, `wssp:EndorsingEncryptedSupportingTokens` or `wssp:SignedEndorsingEncryptedSupportingTokens`. Note that if one would combine this recommendation with SEC_17_9, combining assumptions mentioned for both rules, one could interpret the two recommendations as requiring username tokens containing a password to be signed as well as encrypted, resulting in the vast majority of `wssp:UsernameToken` assertions to be invalid, except those part of signed as well as encrypted supporting tokens, i.e. of either `wssp:SignedEncryptedSupportingTokens` or `wssp:SignedEndorsingEncryptedSupportingTokens`.

A.4.2 Minimal Schematron for the static BSP Conformance

```
<schema id="WS-I Basic Security Profile" xmlns="http://purl.oclc.org/dsdl/schematron">
  <title>WS-I Basic Security Profile</title>
  <ns prefix="wssp" uri="http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702" />
  <ns prefix="wsp" uri="http://schemas.xmlsoap.org/ws/2004/09/policy" />
  <pattern id="R5404">
    <rule id="Rule for R5404"
 context="wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
     wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy | wssp:SupportingTokens/
     wsp:Policy/wssp:AlgorithmSuite/wsp:Policy | wssp:SignedSupportingTokens/wsp:Policy/
     wssp:AlgorithmSuite/wsp:Policy | wssp:EndorsingSupportingTokens/wsp:Policy/
     wssp:AlgorithmSuite/wsp:Policy | wssp:SignedEndorsingSupportingTokens/wsp:Policy/
     wssp:AlgorithmSuite/wsp:Policy">
      <assert id="Assert for R5404" role="MUST" test="not(wssp:InclusiveC14N)">
        Inclusive Canonicalization algorithm must not be used.
      </assert>
    </rule>
  </pattern>
  <!-- R5412 is the same as the above R5404 -->
  <pattern id="R5420">
    <rule id="Rule for R5420" context="wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/
        wsp:Policy |wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy">
      <assert id="Assert for R5420" role="SHOULD" test="wssp:Basic256 or wssp:Basic192 or
          wssp:Basic128 or wssp:TripleDes or wssp:Basic256Rsa15 or wssp:Basic192Rsa15 or
          wssp:Basic128Rsa15 or wssp:TripleDesRsa15">
       The <name path="parent::*" /> must have a <name />
 child element of one of wssp:Basic256, wssp:Basic192, wssp:Basic128, wssp:TripleDes,
     wssp:Basic256Rsa15, wssp:Basic192Rsa15,wssp:Basic128Rsa15 or wssp:TripleDesRsa15.
      </assert>
    </rule>
  </pattern>
  <pattern id="R5620">
    <rule id="Rule for R5620" context="wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/
        wsp:Policy | wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy">
      <assert id="Assert for R5620" role="MUST" test="wssp:Basic256 or wssp:Basic128 or
          wssp:TripleDes or wssp:Basic256Sha256 or wssp:Basic128Sha256 or wssp:TripleDesSha256
          or wssp:Basic256Sha256Rsa15 or wssp:Basic128Sha256Rsa15 or wssp:TripleDesSha256Rsa15"
          >
        The <name path="parent::*" /> must have a <name /> child element of one of wssp:Basic256
            or wssp:Basic128 or wssp:TripleDes or wssp:Basic256Sha256 or wssp:Basic128Sha256
           or wssp:TripleDesSha256 or wssp:Basic256Sha256Rsa15 or wssp:Basic128Sha256Rsa15 or
           wssp:TripleDesSha256Rsa15
      </assert>
    </rule>
  </pattern>
  <!-- R5625 and R5626 are the same as the above R5620 -->
  <pattern id="R5625">
    <rule id="Rule for R5625" context="wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/
        wsp:Policy | wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
        wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy">
      <assert id="Assert for R5625" role="MUST" test="wssp:Basic256 or wssp:Basic128 or
          wssp:TripleDes or wssp:Basic256Sha256 or wssp:Basic128Sha256 or wssp:TripleDesSha256
          or wssp:Basic256Sha256Rsa15 or wssp:Basic128Sha256Rsa15 or wssp:TripleDesSha256Rsa15"
          >
        The <name path="parent::*" /> must have a <name /> child element of one of wssp:Basic256
            or wssp:Basic128 or wssp:TripleDes or wssp:Basic256Sha256 or wssp:Basic128Sha256
           or wssp:TripleDesSha256 or wssp:Basic256Sha256Rsa15 or wssp:Basic128Sha256Rsa15 or
           wssp:TripleDesSha256Rsa15
      </assert>
    </rule>
```

```
36  </pattern>
37  <pattern id="R5626">
38    <rule id="Rule for R5626" context="wssp:SymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/
          wsp:Policy | wssp:AsymmetricBinding/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
          wssp:SupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
          wssp:SignedSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
          wssp:EndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy |
          wssp:SignedEndorsingSupportingTokens/wsp:Policy/wssp:AlgorithmSuite/wsp:Policy">
39      <assert id="Assert for R5626" role="MUST" test="wssp:Basic256 or wssp:Basic128 or
            wssp:TripleDes or wssp:Basic256Sha256 or wssp:Basic128Sha256 or wssp:TripleDesSha256
            or wssp:Basic256Sha256Rsa15 or wssp:Basic128Sha256Rsa15 or wssp:TripleDesSha256Rsa15"
            > The <name path="parent::*" /> must have a <name /> child element of one of
            wssp:Basic256 or wssp:Basic128 or wssp:TripleDes or wssp:Basic256Sha256 or
            wssp:Basic128Sha256 or wssp:TripleDesSha256 or wssp:Basic256Sha256Rsa15 or
            wssp:Basic128Sha256Rsa15 or wssp:TripleDesSha256Rsa15
40      </assert>
41    </rule>
42  </pattern>
43  <pattern id="R6902">
44    <rule id="Rule for R6902" context="wssp:KerberosToken/wsp:Policy">
45      <assert id="Assert for R6902" role="MUST" test="wssp:WssGssKerberosV5ApReqToken11">
46        The <name path="parent::*" /> must have a <name /> child element
              wssp:WssGssKerberosV5ApReqToken11.
47      </assert>
48    </rule>
49  </pattern>
50  <pattern id="R6903_R6904">
51    <rule id="Rule for R6903 and R6904" context="wssp:KerberosToken">
52      <assert id="Assert for R6903 and R6904" role="MUST" test="@wssp:IncludeToken='http://docs.
            oasis-open.org/ws-sx/ws-securitypolicy/200702/IncludeToken/Once'>
53        The <name path="parent::*" /> must have a <name /> attribute with value 'Once'.</assert>
54     </rule>
55  </pattern>
56  <pattern id="R6905">
57    <rule id="Rule for R6905" context="wssp:KerberosToken/wsp:Policy">
58      <assert id="Assert for R6905" role="MUST" test="wssp:RequireKeyIdentifierReference">
59        The <name path="parent::*" /> must have a <name /> child element of one of
              wssp:RequireKeyIdentifierReference.
60      </assert>
61    </rule>
62  </pattern>
63  <pattern id="R5423">
64    <rule id="Rule for R5423" context="wssp:AlgorithmSuite/wsp:Policy">
65      <assert id="Assert for R5423" role="MUST" test="wssp:STRTransform10 or wssp:XPathFilter20"
            >
66        The <name path="parent::*" /> Must have a <name /> child element wssp:STRTransform10 or
              wssp:XPathFilter20.
67      </assert>
68    </rule>
69  </pattern>
70  <pattern id="R3033">
71    <rule id="Rule for R3033" context="wssp:X509Token">
72      <assert id="Assert for R3033" role="MUST" test="wsp:Policy/wssp:WssX509V3Token10 or
            wsp:Policy/wssp:WssX509V3Token11">
73        Any X509_TOKEN ValueType attribute MUST have a value of http://docs.oasis-open.org/wss
              /2004/01/oasis-200401-wss-x509-token-profile-1.0#X509v3. So only
              wssp:WssX509V3Token10 and wssp:WssX509V3Token11 are valid assertions as a child of
              <name /> that is a child of <name path="parent::*" />
74      </assert>
75    </rule>
76  </pattern>
77  <pattern id="R6302">
78    <rule id="Rule for R6302" context="wssp:Layout/wsp:Policy">
79      <assert id="Assert for R6302" role="MUST" test="wssp:Strict">
80        Any SECURITY_HEADER child elements MUST be ordered so that any SIGNATURE necessary to
              verify the issuance of an REL_TOKEN precedes the first SECURITY_TOKEN_REFERENCE
              that refers to that REL_TOKEN. So a wssp:Strict assertion must be included.
81      </assert>
82    </rule>
83  </pattern>
84  <pattern id="SEC_17_9">
85    <rule id="Rule for SEC_17_9" context="wssp:UsernameToken/wsp:Policy">
```

```
86        <assert id="Assert for SEC_17_9" role="MUST" test="wssp:NoPassword or (
               wssp:RequireDerivedKeys or wssp:RequireExplicitDerivedKeys or
               wssp:RequireImpliedDerivedKeys) and (../../../../wssp:SignedSupportingTokens or
               ../../../../wssp:SignedEndorsingSupportingTokens or ../../../../
               wssp:SignedEncryptedSupportingTokens or ../../../../
               wssp:SignedEndorsingEncryptedSupportingTokens) or wssp:HashPassword and (../../../../
               wssp:SignedSupportingTokens or ../../../../wssp:SignedEndorsingSupportingTokens or
               ../../../../wssp:SignedEncryptedSupportingTokens or ../../../../
               wssp:SignedEndorsingEncryptedSupportingTokens)">
87          If a wsse:PasswordText is being used to derive a key for a subsequent encryption of a
                 response, it should be signed to ensure that an attacker does not substitute an
                 alternative, but valid wsse:Username and wsse:PasswordText. So a wssp:UsernameToken
                  with password thatrequires derived keys must be descendant of wssp:Signed*
                 SupportingTokens.
88        </assert>
89      </rule>
90    </pattern>
91    <pattern id="SEC_17_13">
92      <rule id="Rule for SEC_17_13" context="wssp:UsernameToken">
93        <assert id="Assert for SEC_17_13" role="SHOULD" test="wsp:Policy/wssp:NoPassword or (
               wsp:Policy/wssp:HashPassword and (../../../
               wssp:SignedEndorsingEncryptedSupportingTokens or ../../../
               wssp:SignedEncryptedSupportingTokens or../../../
               wssp:EndorsingEncryptedSupportingTokens))">
94          <name path="../../../wssp:SignedEndorsingEncryptedSupportingTokens"/> is the name. When
                 sending any form of a password, cleartext or digest, confidentiality services are
                 strongly recommended to prevent its value from being revealed or from offline
                 guessing. This can be done
95   by a wssp:SingedEncryptedSupportingToken, wssp:EndorsingEncryptedSupportingToken or
          wssp:SingedEndorsingEncryptedSupportingToken, that encapsulates the UsernameToken.
96        </assert>
97      </rule>
98    </pattern>
99    <pattern id="R3002">
100     <rule id="Rule for R3002" context="wssp:AlgorithmSuite/wsp:Policy">
101       <assert id="Assert for R3002" role="MUST" test="wssp:XPathFilter20">
102         Any SIG_REFERENCE to an element that does not have an ID attribute MUST contain a
                 TRANSFORM with an Algorithm attribute value of "http://www.w3.org/2002/06/xmldsig-
                 filter2". wssp:XPathFilter20 must be child of algorithm suite.
103       </assert>
104     </rule>
105   </pattern>
106 </schema>
```

Bibliography

[1] Common Object Request Broker Architecture (CORBA). http://www.corba.org/.

[2] Component Object Model (COM). http://www.microsoft.com/com/default.mspx.

[3] Document Structure Description (DSD). http://www.brics.dk/DSD/. visited: 2009-09-02.

[4] Java 2 Enterprise Edition (J2EE). http://java.sun.com/j2ee/overview.html.

[5] Organization for the advancement of structured information standards (OASIS). http://www.oasis-open.org. visited: 2009-09-02.

[6] Policy design tool. http://www.alphaworks.ibm.com/tech/policydesigntool.

[7] Standards and Web Services. http://www.ibm.com/developerworks/webservices/standards/.

[8] Web Services Interoperability Organization (WS-I). http://www.ws-i.org.

[9] World wide web consortium (W3C). http://www.w3.org.

[10] *IEEE standard computer dictionary : a compilation of IEEE standard computer glossaries*. IEEE Computer Society Press, New York, NY, USA, January 1991.

[11] XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, November 1999. W3C Recommendation.

[12] Trust-adapted enforcement of security policies in distributed component-structured applications. In *ISCC '01: Proceedings of the Sixth IEEE Symposium on Computers and Communications*, page 2, Washington, DC, USA, 2001. IEEE Computer Society.

[13] eXtensible rights Markup Language (XrML) 2.0. http://www.xrml.org/index.asp, November 2002.

[14] Security in a web services world: A proposed architecture and roadmap. http://www.ibm.com/developerworks/library/specification/ws-secmap/, April 2002. IBM Corp.

[15] Conformance claim attachment mechanisms version 1.0. http://www.ws-i.org/Profiles/ConformanceClaims-1.0-2004-11-15.html, November 2004. WS-I Final Material.

[16] Web services architecture. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/, February 2004. W3C Working Group Note 11.

[17] Efficiently managing security concerns in component based system design. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 1*, pages 164–169, Washington, DC, USA, 2005. IEEE Computer Society.

[18] eXtensible Access Control Markup Language (XACML) version 2.03. Technical report, February 2005.

[19] Namespaces in XML 1.0 (second edition), August 2006. W3C Recommendation.

[20] Tutorial 6: Security in SOA and web services. In *Web Services, 2006. ICWS '06. International Conference on*, 2006.

[21] Web services security: SOAP message security 1.1. Technical report, OASIS Web Services Security (WSS) TC, February 2006.

[22] XML Schemas: Best Practices. http://www.xfront.com/BestPracticesHomepage.html, November 2006. visited: 2009-09-02.

[23] *Understanding SOA Security Design and Implementation*. IBM Corp., Riverton, NJ, USA, 2007.

[24] Web services policy 1.5 - guidelines for policy assertion authors. http://www.w3.org/TR/2007/WD-ws-policy-guidelines-20070330/, March 2007. W3C Working Draft.

[25] Web services policy 1.5 - primer. http://www.w3.org/TR/ws-policy-primer/, November 2007. W3C Working Group Note.

[26] WS-securitypolicy 1.2. Technical report, OASIS Web Services Security (WSS) TC, July 2007.

[27] XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20/, January 2007. W3C Recommendation.

[28] *SOA Policy Management*. IBM Corp., 2008.

[29] SOA policy management. *IBM redpaper REDP-4463-00*, 2008.

[30] Martín Abadi, Anindya Banerjee, Nevin Heintze, and Jon G. Riecke. A core calculus of dependency. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 147–160, New York, NY, USA, 1999. ACM.

[31] Nayef Abu-Ghazaleh, Michael J. Lewis, and Madhusudhan Govindaraju. Differential Serialization for Optimized SOAP Performance. In *HPDC '04: Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing*, 2004.

[32] aes Garcia, Diego Zuquim Guimar} and Maria Beatriz Felgar de Toledo. Web service security management using semantic web techniques. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 2256–2260, New York, NY, USA, 2008. ACM.

[33] S. Agarwal, S. Lamparter, and R. Studer. Making web services tradable: A policy-based approach for specifying preferences on web service properties. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(1):11–20, January 2009.

[34] Sudhir Agarwal and Barbara Sprick. Specification of access control and certification policies for semantic web services. In *6th International Conference on Electronic Commerce and Web Technologies, volume 3590 of Lecture 2 https://javacc.dev.java.net*, pages 348–357. Springer, 2005.

[35] Dakshi Agrawal, Seraphin B. Calo, James Giles, Kang-Won Lee, and Dinesh C. Verma. Policy management for networked systems and applications. In *Integrated Network Management*, pages 455–468. IEEE, 2005.

[36] B. Agreiter, M. Alam, M. Hafner, J. P. Seifert, and X. Zhang. Model driven configuration of secure operating systems for mobile applications in healthcare. In *Proceedings of the 1st International Workshop on Mode-Based Trustworthy Health Information Systems*, 2007.

[37] E. S. Al-Shaer and H. H. Hamed. Firewall policy advisor for anomaly discovery and rule editing. In *Integrated Network Management, 2003. IFIP/IEEE Eighth International Symposium on*, pages 17–30, 2003.

[38] Alessandro Aldini, Gilles Barthe, and Roberto Gorrieri. *Foundations of Security Analysis and Design V: FOSAD 2007/2008/2009 Tutorial Lectures*. Springer-Verlag, Berlin, Heidelberg, 2009.

[39] Alessandro Aldini, Roberto Gorrieri, and Fabio Martinelli. *Foundations of Security Analysis and Design III: FOSAD 2004/2005 Tutorial Lectures (Lecture Notes in Computer Science)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[40] Anne H. Anderson. An introduction to the web services policy language (WSPL). *Policies for Distributed Systems and Networks, IEEE International Workshop on*, 0:189, 2004.

[41] Anne H. Anderson. WS-Security policy profile of WS-PolicyConstraints. OASIS, December 2005.

[42] Anne H. Anderson. Domain-Independent, Composable Web Services Policy Assertions. 2006.

[43] Anne H. Anderson. Domain-independent, composable web services policy assertions. In *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 149–152, Washington, DC, USA, 2006. IEEE Computer Society.

[44] Apache Foundation. Apache AXIOM. `http://ws.apache.org/commons/axiom/index.html`. Visited: 2009-07-28.

[45] Apache Foundation. Apache Axis 2. `http://ws.apache.org/axis2/`. Visited: 2009-07-27.

[46] Apache Foundation. Apache Rampart - Axis2 Security Module. `http://ws.apache.org/rampart/`. Visited: 2009-07-28.

[47] (OSOA) Open Service Orientated Architecture. Service Component Architecture Specifications. `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`. visited: 2009-11-25.

[48] (OSOA) Open Service Orientated Architecture. SCA policy framework V1.00. `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`, March 2007. visited: 2009-11-25.

[49] Anindya Banerjee and David A. Naumann. Secure information flow and pointer confinement in a java-like language. In *CSFW '02: Proceedings of the 15th IEEE workshop on Computer Security Foundations*, page 253, Washington, DC, USA, 2002. IEEE Computer Society.

[50] Abbie Barbir, Martin Gudgin, Michael McIntosh, and K. Scott Morrison. WS-I basic security profile version 1.0. WS-I, Nortel Networks, Microsoft, IBM, Layer 7, August 2005.

[51] G. Barthe and B. Serpette. Partial evaluation and non-interference for object calculi. In A. Middeldorp and T. Sato, editors, *Proceedings of FLOPS'99*, volume 1722 of *lncs*, pages 53–67. Springer, 1999.

[52] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security for process-oriented systems. In *SACMAT '03: Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 100–109, New York, NY, USA, 2003. ACM.

[53] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security: from UML models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15:2006, 2006.

[54] Lujo Bauer, Jarred Ligatti, and David Walker. A calculus for composing security policies. Technical Report TR-655-02, Princeton University, August 2002.

[55] Lujo Bauer, Jay Ligatti, and David Walker. A language and system for composing security policies. Technical Report TR-699-04, Princeton University, January 2004.

[56] Trevor Bench-capon, Grant Malcolm, and Michael Shave. Semantics for interoperability: relating ontologies and schemata. In *and Expert Systems Applications, 14th International Conference, DEXA 2003*, pages 703–712. Springer, 2002.

[57] Tim Berners-Lee. WWW: Past, present, and future. *Computer*, 29(10):69–77, 1996.

[58] Bastian Best, Jan Jürjens, and Bashar Nuseibeh. Model-based security engineering of distributed information systems using UMLsec. In *ICSE*, pages 581–590, 2007.

[59] Geert Jan Bex, Wim Martens, Frank Neven, and Thomas Schwentick. Expressiveness of XSDs: from practice to theory, there and back again. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 712–721, New York, NY, USA, 2005. ACM.

[60] Karthikeyan Bhargavan, Cédric Fournet, and Andrew D. Gordon. Verifying policy-based security for web services. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 268–277, New York, NY, USA, 2004. ACM.

[61] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes. Recommendation, World Wide Web Consortium, May 2001. See `http://www.w3.org/TR/xmlschema-2/`.

[62] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. In *In Proceedings of the 1996 IEEE Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, 1996.

[63] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. XQuery 1.0: An XML Query Language. Recommendation, World Wide Web Consortium, January 2007. See `http://www.w3.org/TR/xquery/`.

[64] Napoleon Bonapartes. *Aphorisms and thoughts*. 1838.

[65] Piero Bonatti and Daniel Olmedilla. Rule-based policy representation and reasoning for the semantic web. pages 240–268. 2007.

[66] R. T. Braden. RFC 1122: Requirements for Internet hosts — communication layers, October 1989. See also STD0003. Status: STANDARD.

[67] R. T. Braden. RFC 1123: Requirements for Internet hosts — application and support, October 1989. See also STD0003. Updates RFC0822. Updated by RFC2181. Status: STANDARD.

[68] R. T. Braden and J. Postel. RFC 1009: Requirements for Internet gateways, June 1987. Obsoleted by RFC1812. Obsoletes RFC0985. Status: HISTORIC.

[69] S. Bradner. RFC 2119: Key words for use in RFCs to indicate requirement levels, March 1997. See also BCP0014. Status: BEST CURRENT PRACTICE.

[70] Barbara Carminati, Elena Ferrari, and Patrick C. K. Hung. Security conscious web service composition. pages 489–496, 2006.

[71] Federico Cavalieri, Giovanna Guerrini, and Marco Mesiti. Navigational path expressions on XML schemas. In *DEXA '08: Proceedings of the 19th international conference on Database and Expert Systems Applications*, pages 718–726, Berlin, Heidelberg, 2008. Springer-Verlag.

[72] Charles André, Frédéric Mallet, and Marie-Agnés Peraldi Frati. Non-functional property analysis using UML 2.0 and model transformations. Research Report RR-5913, INRIA, 2006.

[73] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web services description language (WSDL) 1.1. World Wide Web Consortium, Note NOTE-wsdl-20010315, March 2001.

[74] James Clark and Steve DeRose. XML Path Language (XPath). Recommendation, World Wide Web Consortium, November 1999. See `http://www.w3.org/TR/xpath/`.

[75] Claudio Sacerdoti Coen, Paolo Marinelli, and Fabio Vitali. Schemapath, a minimal extension to xml schema for conditional constraints. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 164–174, New York, NY, USA, 2004. ACM.

[76] LLC Ken North Computing. Vulnerabilities and Threats. `http://www.webservicessummit.com/Vulnerabilities.htm`, March 2008.

[77] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[78] John Cowan and Richard Tobin. XML Information Set (Second Edition). Recommendation, World Wide Web Consortium, February 2004. See `http://www.w3.org/TR/xml-infoset/`.

[79] Francisco Curbera. Component contracts in service-oriented architectures. *Computer*, 40(11):74–80, 2007.

[80] Francisco Curbera, Matthew Duftler, Rania Khalaf, William Nagy, Nirmal Mukhi, and Sanjiva Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2), 2002.

[81] Francisco Curbera, Donald F. Ferguson, Martin Nally, and Marcia L. Stockton. Toward a programming model for service-oriented computing. In *ICSOC*, pages 33–47, 2005.

[82] Thomas DeMartini, Anthony Nadalin, Phil Griffin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web Services Security Expression Language (REL) Token Profile 1.1, February 2006.

[83] Dorothy E. Denning and Peter J. Denning. Certification of programs for secure information flow. *Commun. ACM*, 20(7):504–513, 1977.

[84] Steven DeRose, Ron Daniel Jr., Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. XML Pointer Language (XPointer). Working draftrecommendation, World Wide Web Consortium, August 2002. See `http://www.w3.org/TR/xptr/`.

[85] Steven J. DeRose, Deborah Lapeyre, B. Tommie Usdin, James Clark, Murata Makoto, Martin Gudgin, Henry S. Thompson, and Priscilla Walmsley. XML 2001 Schema Language Comparison Town Hall. `http://nwalsh.com/xml2001/schematownhall/`, 2001.

[86] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746.

[87] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), April 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746.

[88] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878.

[89] Betty J. T. Dobbs. Newton's Commentary of The Emerald Tablet of Hermes Trismegistus: Its Scientific and Theological Significance. *Hermeticism and the Renaissance - Intellectual History and the Occult in Early Modern Europe*, pages 182–191, 1988.

[90] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, Stanford, CA, USA, 1981.

[91] R. H. Dolin, L. Alschuler, C. Beebe, P. V. Biron, S. L. Boyer, D. Essin, E. Kimber, T. Lincoln, and J. E. Mattison. The HL7 clinical document architecture. *J Am Med Inform Assoc*, 8(6):552–569, 2001.

[92] Document Type Definition Languages (DSDLs). Part 3: Rule-based validation - schematron. `http://standards.iso.org/ittf/PubliclyAvailableStandards/c040833_ISO_IEC_19757-3_2006(E).zip`. INTERNATIONAL STANDARD ISO/IEC 19757-3.

[93] Nicodemos Damianou Naranker Dulay. The ponder policy specification language. In *Lecture Notes in Computer Science*, pages 18–38. Springer-Verlag, 2001.

[94] D. Eastlake, J. Reagle, and D. Solo. *XML-Signature Syntax and Processing*. XML Signature Working Group, 2002. W3C Recommendation.

[95] Donald E. Eastlake, Joseph M. Reagle, Takeshi Imamura, Blair Dillaway, and Ed Simon. XML encryption syntax and processing. World Wide Web Consortium, Recommendation REC-xmlenc-core-20021210, December 2002.

[96] Eclipse. Eclipse Modeling Framework Project (EMF). `http://www.eclipse.org/modeling/emf/`. Visited: 2009-07-27.

[97] Eclipse. Eclipse Platfrom. `http://www.eclipse.org/platform`. Visited: 2009-07-27.

[98] Eclipse. Eclipse Plugin Central. `http://www.eclipseplugincentral.com/`. Visited: 2009-07-27.

[99] J. Epstein, S. Matsumoto, and G. Mcgraw. Software security and SOA: danger, will robinson! *Security and Privacy Magazine, IEEE*, 4(1):80–83, 2006.

[100] Thomas Erl. *Service-Oriented Architecture. Concepts, Technology, and Design*. Prentice Hall, 2005.

[101] Catherine Everett. Cloud computing - A question of trust. *Computer Fraud and Security*, 2009(6), 2009.

[102] Richard J. Feiertag and Peter G. Neumann. The foundations of a provably secure operating system (PSOS). In *IN PROCEEDINGS OF THE NATIONAL COMPUTER CONFERENCE*, pages 329–334. AFIPS Press, 1979.

[103] Christopher Ferris, Canyang Kevin Liu, Mark Nottingham, Prasad Yendluri, Martin Gudgin, Keith Ballinger, and David Ehnebuske. WS-I basic profile version 1.1. WS-I, Microsoft, IBM, SAP, BEA Systems, webMethods, August 2004. (Editors).

[104] Paris Flegkas, Panos Trimintzios, George Pavlou, and Antonio Liotta. Design and implementation of a policy-based resource management architecture. In *Integrated Network Management*, pages 215–229, 2003.

[105] Riccardo Focardi and Roberto Gorrieri, editors. *FOSAD '00: Revised versions of lectures given during the IFIP WG 1.7 International School on Foundations of Security Analysis and Design on Foundations of Security Analysis and Design*. Springer-Verlag, London, UK, 2001.

[106] Riccardo Focardi and Roberto Gorrieri. *Foundations of Security Analysis and Design II*. Springer-Verlag, 2004.

[107] Organization for the Advancement of Structured Information Standards (OASIS). RELAX NG specification. `http://www.oasis-open.org/committees/relax-ng/spec-20011203.html`, 2001. Visited: 2007-07-27.

[108] Organization for the Advancement of Structured Information Standards (OASIS). Reference Model for Service Oriented Architecture 1.0. `http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf`, 2006. Visited: 2007-07-31.

[109] Online Community for the Security Assertion Markup Language OASIS Standard. Security assertion markup language v2.0. `http://saml.xml.org/saml-specifications`, October 2009. OASIS Standard.

[110] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *Automated Software Engineering, 2003. Proceedings. 18th IEEE International Conference on*, pages 152–161, 2003.

[111] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[112] Thomas L. Friedman. *The world is flat : a brief history of the twenty-first century*. Farrar, Straus and Giroux, 2005.

[113] Henrik Frystyk Nielsen and Hervé Ruellan. SOAP 1.2 attachment feature. World Wide Web Consortium, Note NOTE-soap12-af-20040608, June 2004.

[114] Stephen Gilmore, Valentin Haenel, Leïla Kloul, and Monika Maidl. Choreographing security and performance analysis for web services, 2005.

[115] Rodney Gleghorn. Enterprise application integration: A manager's perspective. *IT Professional*, 7(6):17–23, 2005.

[116] Charles F. Goldfarb. The roots of SGML: A personal recollection. *Technical Communication*, 46:75–78, February 1999.

[117] Marc Goodner and Anthony Nadalin. Web Services Federation Language (WS-Federation) Version 1.2 . Technical report, May 2009. OASIS Standard.

[118] Andrew D. Gordon and Riccardo Pucella. Validating a web service security abstraction by typing. In *XMLSEC '02: Proceedings of the 2002 ACM workshop on XML security*, pages 18–29, New York, NY, USA, 2002. ACM.

[119] Karl Gottschalk, Stephen Graham, Heather Kreger, and James Snell. Introduction to Web services Architecture. *IBM Systems Journal*, 41(2), 2002.

[120] Mark Grechanik, Dewayne E. Perry, and Don Batory. A security mechanism for component-based systems. In *ICCBSS '06: Proceedings of the Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems*, page 53, Washington, DC, USA, 2006. IEEE Computer Society.

[121] Paul Grosso, Eve Maler, Jonathan Marsh, and Norman Walsh. XPointer Framework. Recommendation, World Wide Web Consortium, March 2003. See `http://www.w3.org/TR/xptr-framework/`.

[122] Nils Gruschka and Luigi Lo Iacono. Vulnerable cloud: SOAP message security validation revisited. *Web Services, IEEE International Conference on*, 0:625–631, 2009.

[123] Nils Gruschka, Meiko Jensen, and Torben Dziuk. Event-based application of ws-security policy on soap messages. In *SWS '07: Proceedings of the 2007 ACM workshop on Secure web services*, pages 1–8, New York, NY, USA, 2007. ACM.

[124] Nils Gruschka, Norbert Luttenberger, and Ralph Herkenhöner. Event-based SOAP message validation for WS-securitypolicy-enriched web services. In Hamid R. Arabnia, editor, *SWWS*, pages 80–86. CSREA Press, 2006.

[125] Li guo Wang and Lyndon Lee. UML-based modeling of web services security. In *IEEE European Conference on Web Services Poster session*, 2005.

[126] Brant Hashii, Scott Malabarba, Raju Pandey, and Matt Bishop. Supporting reconfigurable security policies for mobile programs. *Computer Networks*, 33(1-6):77–93, June 2000.

[127] Rachel Heery and Manjula Patel. Application profiles: mixing and matching metadata schemas. *Ariadne*, (25), September 2000.

[128] Sandra Heiler. Semantic interoperability. *ACM Comput. Surv.*, 27(2):271–273, 1995.

[129] Nevin Heintze and Jon G. Riecke. The SLam calculus: programming with secrecy and integrity. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 365–377, New York, NY, USA, 1998. ACM.

[130] Peter Herrmann. Formal security policy verification of distributed component-structured software. In *23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'2003), IFIP, LNCS*, pages 257–272. Springer-Verlag, 2003.

[131] Diane I. Hillmann, and Jon Phipps,. Application profiles: exposing and enforcing metadata quality. In *DCMI '07: Proceedings of the 2007 international conference on Dublin Core and Metadata Applications*, pages 53–62. Dublin Core Metadata Initiative, 2007.

[132] Frederick Hirsch. Web Services Security SOAP Messages with Attachments (SwA) Profile 1.1, February 2006.

[133] Dong Huang. Semantic descriptions ofweb services security constraints. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering*, pages 81–84, Washington, DC, USA, 2006. IEEE Computer Society.

[134] Jane Hunter, and Carl Lagoze,. Combining RDF and XML schemas to enhance interoperability between metadata application profiles. In *WWW '01: Proceedings of the 10th international conference on World Wide Web*, pages 457–466, New York, NY, USA, 2001. ACM.

[135] IBM. WS-I Validator. `http://publib.boulder.ibm.com/infocenter/sr/v6r3/index.jsp?topic=/com.ibm.sr.doc/cwsr_wsi_validator.html`. Configuring the WSRR provided plug-ins.

[136] IETF (internet engineering task force). `http://www.ietf.org`.

[137] Stefan Illner, Andre Pohl, and Heiko Krumm. Model-driven security management of embedded service systems, November 2005.

[138] Bart Jacob, Michael Brown, Kentaro Fukui, and Nihar Trivedi. *Introduction to Grid Computing*. IBM Redbook SG24-6778-00. IBM International Technical Support Organization, December 2005.

[139] Rick Jelliffe. The current state of the art of schema languages for XML. `http://chinese-school.netfirms.com/computer-article-XML.html`, 2001. XML Asia Pacific 2001 Conference.

[140] Anita K. Jones and Richard J. Lipton. The enforcement of security policies for computation. In *SOSP '75: Proceedings of the Fifth ACM Symposium on Operating Systems Principles*, pages 197–206, New York, NY, USA, 1975. ACM Press.

[141] W. H. S. Jones. *Hippocrates, with an English translation*, volume 4. William Heinemann, London, 1931. Section xi.-xiii.

[142] Lalana Kagal. Rei : A Policy Language for the Me-Centric Project. Technical report, HP Labs, September 2002. http://www.hpl.hp.com/techreports/2002/HPL-2002-270.html.

[143] Lalana Kagal, Massimo Paolucci, Naveen Srinivasan, Grit Denker, Tim Finin, and Katia Sycara. Authorization and privacy for semantic web services. *IEEE Intelligent Systems*, 19(4):50–56, 2004.

[144] L. M. Kaufman. Data security in the world of cloud computing. *Security and Privacy, IEEE*, 7(4):61–64, July 2009.

[145] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, January 1883.

[146] Khaled M. Khan and Jun Han. Composing security-aware software. *IEEE Softw.*, 19(1):34–41, 2002.

[147] Nils Klarlund,, Anders Möller,, and Michael I. Schwartzbach,. The DSD schema language. *Automated Software Engg.*, 9(3):285–319, 2002.

[148] Vladimir Kolovski, Bijan Parsia, Yarden Katz, and James A. Hendler. Representing web service policies in OWL-DL. In *International Semantic Web Conference*, pages 461–475, 2005.

[149] Mikko Kontio. Architectural manifesto: An introduction to the possibilities (and risks) of cloud computing.

[150] Dimitrios Koutsomitropoulos,, George Paloukis,, and Theodore S. Papatheodorou,. From metadata application profiles to semantic profiling; ontology refinement and profiling to strengthen inference-based queries on the semantic web. *Int. J. Metadata Semant. Ontologies*, 2(4):268–280, 2007.

[151] C. E. Landwehr. The best available technologies for computer security. *Computer*, 16(7):86–100, 1983.

[152] D. Lekkas and D. Spinellis. Handling and reporting security advisories: A scorecard approach. *Security and Privacy Magazine, IEEE*, 3(4):32–41, 2005.

[153] David S. Linthicum. *Enterprise application integration*. Addison-Wesley Longman Ltd., Essex, UK, UK, 2000.

[154] Torsten Lodderstedt, David A. Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML '02: Proceedings of the 5th International Conference on The Unified Modeling Language*, pages 426–441, London, UK, 2002. Springer-Verlag.

[155] Emil C. Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Trans. Softw. Eng.*, 25(6):852–869, 1999.

[156] Piyush Maheshwari. Enterprise application integration using a component-based architecture. *Computer Software and Applications Conference, Annual International*, 0:557, 2003.

[157] Satoshi Makino, Kent Tamura, Takeshi Imamura, and Yuichi Nakamura. Implementation and performance of WS-security. *Int. J. Web Service Res.*, 1(1):58–72, 2004.

[158] Eve Maler and Norman Walsh. XML pipeline definition language version 1.0. W3C Note, February 2002.

[159] Jonathan Marsh, David Orchard, and Daniel Veillard. XML Inclusions (XInclude) Version 1.0 (Second Edition). Recommendation, World Wide Web Consortium, November 2006. See `http://www.w3.org/TR/xinclude/`.

[160] Francis G. Mccabe. Service oriented architecture reference architecture. Technical report, April 2008.

[161] Michael McIntosh and Paula Austel. XML signature element wrapping attacks and countermeasures. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 20–27, New York, NY, USA, 2005. ACM.

[162] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.

[163] Alfred J. Menezes,, Scott A. Vanstone,, and Paul C. Van Oorschot,. *Handbook of Applied Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 1996.

[164] Jonathan D. Moffett and Morris Sloman. Policies hierarchies for distributed systems management. *IEEE Journal on Selected Areas in Communications*, 11(9):1404–1414, 1993.

[165] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy core information model – version 1 specification. RFC 3060 (Proposed Standard), February 2001. Updated by RFC 3460.

[166] Haralambos Mouratidis, Paolo Giorgini, and Gordon A. Manson. Integrating security and systems engineering: Towards the modelling of secure information systems. In *CAiSE*, pages 63–78, 2003.

[167] Nirmal K. Mukhi and Pierluigi Plebani. Supporting policy-driven behaviors in web services: experiences and issues. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 322–328, New York, NY, USA, 2004. ACM.

[168] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, and Satoshi Hada. XML access control using static analysis. *ACM Trans. Inf. Syst. Secur.*, 9(3):292–324, 2006.

[169] Andrew C. Myers. JFlow: practical mostly-static information flow control. In *POPL '99: Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 228–241, New York, NY, USA, 1999. ACM.

[170] Andrew C. Myers and Barbara Liskov. A decentralized model for information flow control. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 129–142, New York, NY, USA, 1997. ACM.

[171] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. Ws-secureconversation 1.3. Technical report, March 2007. OASIS Standard.

[172] Anthony Nadalin, Marc Goodner, Martin Gudgin, Abbie Barbir, and Hans Granqvist. Ws-trust 1.3. Technical report, March 2007. OASIS Standard.

[173] Anthony Nadalin, Phil Griffin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web services security kerberos token profile 1.1, February 2006.

[174] Anthony Nadalin, Phil Griffin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web services security: Saml token profile 1.1, February 2006.

[175] Anthony Nadalin, Phil Griffin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web services security x.509 certificate token profile 1.1, February 2006.

[176] Anthony Nadalin, Chris Kaler, Phillip Hallam-Baker, and Ronald Monzillo. Web services security usernametoken profile 1.1, February 2006.

[177] Martin Naedele. Standards for XML and web services security. *Computer*, 36(4):96–98, 2003.

[178] Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, John Miller, and Jon Lathem. Semantic interoperability of web services - challenges and experiences. In *Proceedings of the Fourth IEEE International Conference on Web Services (ICWS 2006*, pages 373–382. IEEE CS Press, 2006.

[179] Yuichi Nakamura, Fumiko Sato, and Hyen-Vui Chung. Syntactic validation of web services security policies. pages 319–329. 2007.

[180] Christian Nentwich, Licia Capra, Wolfgang Emmerich, and Anthony Finkelstein. xlinkit: a consistency checking and smart link generation service. *ACM Trans. Internet Technol.*, 2(2):151–185, 2002.

[181] C. Neuman, T. Yu, S. Hartman, and K. Raeburn. The Kerberos Network Authentication Service (V5). RFC 4120 (Proposed Standard), July 2005. Updated by RFCs 4537, 5021.

[182] Nimal Nissanke. Component security: Issues and an approach. In *COMPSAC '05: Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05) Volume 2*, pages 152–155, Washington, DC, USA, 2005. IEEE Computer Society.

[183] International Organisation of Standardisation. Information processing systems – open systems interconnection – basic reference model – part 2: Security architecture, 1989. ISO 7498-2:1989.

[184] Uche Ogbuji. Discover the flexibility of schematron abstract patterns. Technical report, October 2004. See `http://www.ibm.com/developerworks/xml/library/x-stron.html`.

[185] Uche Ogbuji. Tip: Use data dictionary links for XML and web services schemata. Technical report, May 2004. See `http://www.ibm.com/developerworks/xml/library/x-tipdict.html`.

[186] Ebenezer A. Oladimeji and Lawrence Chung. Analyzing security interoperability during component integration. In *ICIS-COMSAR '06: Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse*, pages 121–129, Washington, DC, USA, 2006. IEEE Computer Society.

[187] Nicole Oldham, Kunal Verma, Amit Sheth, and Farshad Hakimpour. Semantic WS-agreement partner selection. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 697–706, New York, NY, USA, 2006. ACM.

[188] OMG. Unified modeling language specification v2.0. Technical Report 05-07-04, Object Management Group, 2005.

[189] P. Orbaek and J. Palsberg. Trust in the lambda-calculus. *Z. Funct. Program.*, 7(6):557–591, 1997.

[190] OSGi Alliance. OSGi - The Dynamic Module System for Java. `http://www.osgi.org/Main/HomePage`. Visited: 2009-07-27.

[191] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing: Introduction. *Communications of the ACM*, 46(10):24–28, 2003.

[192] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented Architectures: Approaches, Technologies and Research Issues. *The VLDB Journal*, 16(3), 2007.

[193] Mike.P. Papazoglou. Service -Oriented Computing: Concepts, Characteristics and Directions. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE)*, 2003.

[194] Mike.P. Papazoglou and Dimitrios Georgakopoulos. Service-oriented Computing. *Communications of the ACM*, 46(10), 2003.

[195] Adrian Paschke. Verification, validation and integrity of distributed and interchanged rule based policies and contracts in the semantic web. *CoRR*, 2006.

[196] Tan Phan, Jun Han, Jean-Guy Schneider, Tim Ebringer, and Tony Rogers. Policy-based service registration and discovery. pages 417–426. 2007.

[197] Tan Phan, Jun Han, Jean-Guy Schneider, Tim Ebringer, and Tony Rogers. A survey of policy-based management approaches for service oriented systems. In *ASWEC '08: Proceedings of the 19th Australian Conference on Software Engineering*, pages 392–401, Washington, DC, USA, 2008. IEEE Computer Society.

[198] Tan Phan, Jun Han, Jean-Guy Schneider, and Kirk Wilson. Quality-driven business policy specification and refinement for service-oriented systems. pages 5–21. 2008.

[199] François Pottier and Sylvain Conchon. Information flow inference for free. In *ICFP '00: Proceedings of the fifth ACM SIGPLAN international conference on Functional programming*, pages 46–57, New York, NY, USA, 2000. ACM.

[200] François Pottier and Vincent Simonet. Information flow inference for ML. *ACM Trans. Program. Lang. Syst.*, 25(1):117–158, 2003.

[201] Stefan Prennschütz-Schützenau, Nirmal K. Mukhi, Satoshi Hada, Naoto Sato, Fumiko Satoh, and Naohiko Uramoto. Static vs. Dynamic Validation of Basic Security Profile Conformance. In *ICWS '09: Proceedings of the IEEE International Conference on Web Services (ICWS'09)*, 2009.

[202] Ravi, Edward, Hal, and Charles. Role-based access control models, 1996.

[203] Martin Raepple Rich Levinson, Tony Gullotta, Symon Chang. WS-securitypolicy examples. `http://docs.oasis-open.org/ws-sx/security-policy/examples/ws-sp-usecases-examples-cd-01.pdf`, June 2009. Editors, OASIS Committee Draft 01.

[204] John W. Rittinghouse and James F. Ransome. *Cloud Computing: Implementation, Management and Security*. CRC Press, Boca Raton, 2009.

[205] Eddie Robertsson. Combining Schematron with other XML Schema languages. `http://www.topologi.com/public/Schtrn_XSD/Paper.html`, July 2002. visited: 2009-09-02.

[206] Jothy Rosenberg and David Remy. *Securing Web Services with WS-Security - Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption*. Pearson, 2004.

[207] T. Ryutov and C. Neuman. The specification and enforcement of advanced security policies. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, page 128, Washington, DC, USA, 2002. IEEE Computer Society.

[208] Tatyana Ryutov and Clifford Neuman. The set and function approach to modeling authorization in distributed systems. In *In Proceedings of the Workshop on Mathematical Methods and Models and Architecture for Computer Networks Security*, 2001.

[209] A. Sabelfeld and A. C. Myers. Language-based information-flow security. *Selected Areas in Communications, IEEE Journal on*, 21(1):5–19, 2003.

[210] Fumiko Satoh, Hyen-Vui Chung, Naohiko Uramoto, and Naoto Sato. Security policy conformance validation for preventing security attacks on web services. Technical report, IBM Tokyo Research Laboratory, 2008.

[211] Fumiko Satoh, Nirmal K. Mukhi, Yuichi Nakamura, and Shinichi Hirose. Pattern-based policy configuration for SOA applications. In *SCC '08: Proceedings of the 2008 IEEE International Conference on Services Computing*, pages 13–20, Washington, DC, USA, 2008. IEEE Computer Society.

[212] Fumiko Satoh, Yuichi Nakamura, Nirmal Mukhi, Michiaki Tatsubori, and Kouichi Ono. Methodology and tools for end-to-end SOA security configurations. In *SERVICES I*, pages 307–314, 2008.

[213] Rudolf Schreiner and Ulrich Lang. Protection of complex distributed systems. In *MidSec '08: Proceedings of the 2008 workshop on Middleware security*, pages 7–12, New York, NY, USA, 2008. ACM.

[214] Ra Schäfer and Werner Rechert. Security and web services. Technical report, IT Transfer Office, 2004. Technical Report, No. ITO 2004-1.

[215] SGML. Information processing – text and office systems – standard generalized markup language (SGML). ISO 8879:1986(e), ISO, October 1986.

[216] Claude E. Shannon. Communication theory of secrecy systems. *Bell Systems Technical Journal*, 28:656–715, 1949.

[217] Brian Shields, Owen Molloy, Gerard Lyons, and Jim Duggan. Using semantic rules to determine access control for web services. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 913–914, New York, NY, USA, 2006. ACM.

[218] Satoshi Shirasuna, Aleksander Slominski, Liang Fang, and Dennis Gannon. Performance comparison of security mechanisms for grid services. In *In 5th IEEE/ACM International Workshop on Grid Computing*, pages 360–364. IEEE Computer Society Press, 2004.

[219] M. Sloman and E. Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10–19, 2002.

[220] Morris Sloman. Policy driven management for distributed systems. *Journal of Network and Systems Management*, 2:333–360, 1994.

[221] Geoffrey Smith and Dennis Volpano. Secure information flow in a multi-threaded imperative language. In *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 355–364, New York, NY, USA, 1998. ACM.

[222] S. Staab, W. van der Aalst, V. R. Benjamins, A. Sheth, J. A. Miller, C. Bussler, A. Maedche, D. Fensel, and D. Gannon. Web services: been there, done that? *IEEE Intelligent Systems*, 18:72–85, 2003.

[223] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT Press, Cambridge (MA), 1986.

[224] Toyotaro Suzumura, Toshiro Takase, and Michiaki Tatsubori. Optimizing Web Services Performance by Differential Deserialization. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, 2005.

[225] Nikhil Swamy, Brian J. Corcoran, and Michael Hicks. Fable: A language for enforcing user-defined security policies. In *In IEEE Symposium on Security and Privacy*, 2008.

[226] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[227] Masayoshi Teraguchi, Issei Yoshida, and Naohiko Uramoto. Rule-based XML mediation for data validation and privacy anonymization. *Services Computing, IEEE International Conference on*, 2:21–28, 2008.

[228] Henry Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures. Recommendation, World Wide Web Consortium, May 2001. See `http://www.w3.org/TR/xmlschema-1/`.

[229] Andreas Tolk. XML mediation services utilizing model based data management. In *WSC '04: Proceedings of the 36th conference on Winter simulation*, pages 1476–1484. Winter Simulation Conference, 2004.

[230] UN/CEFACT (united nations center for trade facilitations and electronic business).

[231] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *POLICY '03: Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, page 93, Washington, DC, USA, 2003. IEEE Computer Society.

[232] Andrzej Uszok, Jeffrey M. Bradshaw, Matthew Johnson, Renia Jeffers, Austin Tate, Jeff Dalton, and Stuart Aitken. KAoS policy management for semantic web services. *IEEE Intelligent Systems*, 19(4):32–41, 2004.

[233] Eric van der Vlist. Examplotron. `http://examplotron.org/`, 2003.

[234] Dennis Volpano, Cynthia Irvine, and Geoffrey Smith. A sound type system for secure flow analysis. *J. Comput. Secur.*, 4(2-3):167–187, 1996.

[235] Dennis Volpano and Geoffrey Smith. Probabilistic noninterference in a concurrent language. *J. Comput. Secur.*, 7(2-3):231–253, 1999.

[236] World Wide Web Consortium (W3C). SOAP Message Transmission Optimization Mechanism. `http://www.w3.org/TR/soap12-mtom/`, 2005. Visited: 2007-07-28.

[237] World Wide Web Consortium (W3C). Extensible markup language (XML) 1.0 (fifth edition). `http://www.w3.org/TR/REC-xml/`, 2008. W3C recomendation.

[238] Norman Walsh and John Cowan. Schema language comparison. `http://nwalsh.com/xml2001/schematownhall/slides/index.html`, December 2001.

[239] E. Rowland Watkins, Mark McArdle, Thomas Leonard, and Mike Surridge. Cross-middleware interoperability in distributed concurrent engineering. In *E-SCIENCE '07: Proceedings of the Third IEEE International Conference on e-Science and Grid Computing*, pages 561–568, Washington, DC, USA, 2007. IEEE Computer Society.

[240] Erik Wilde and Felix Michel. SPath: a path language for XML schema. In *WWW*, pages 1343–1344, 2007.

[241] Raymond L. Wilder. *Introduction to the Foundations of Mathematics*. John Wiley and Sons, New York, 1952.

[242] Andrew Wilson. *The Dark Side of the Flat World*. CSC, 2007.

[243] Ludwig Wittgenstein. Tractatus logico-philosophicus. Suhrkamp Taschenbuch Wissenschaft, 1989. Translation from `http://www.gutenberg.org/files/5740/5740-h/5740-h.htm`.

[244] WS-I. Basic security profile [1.0] test assertions version 1.0. http://www.ws-i.org/Testing/Tools/2004/12/BasicSecurityTestAssertions.htm.

[245] M. I. Yagüe and J. M[a] Troya. A semantic approach for access control in web services. In *In Proc. of the W3C Euroweb 2002 International Conference*, pages 483–494, 2002.

[246] Mamoon Yunus and Rizwan Mallal. An empirical study of security threats and countermeasures in web services-based services oriented architectures. In *WISE*, pages 653–659, 2005.

[247] Tun Z, Bird L. J, and Goodchild A. Validating electronic health records using archetypes and XML abstract.

[248] Steve Zdancewic and Andrew C. Myers. Secure information flow and CPS. In *ESOP '01: Proceedings of the 10th European Symposium on Programming Languages and Systems*, pages 46–61, London, UK, 2001. Springer-Verlag.

[249] E. C. Zeeman. *Catastrophe Theory Selected Papers, 1972-1977*. Addison-Wesley, Reading, MA, 1977.

[250] Liang-Jie Zhang. Services computing: a new discipline. *International Journal on Web Services Research JWSR*, 2(1), 2005.

[251] Liang-Jie Zhang, Jia Zhang, and Hong Cai. *Services Computing*. Springer, 2007.

[252] H. Zimmermann. OSI reference model: The ISO model of architecture for open systems interconnection. *IEEE Transactions on Communications*, 28(4):425–432, 1980.