2005

Self-adaptable trust mechansim for web services

Oana Andreea Dini
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Dini, Oana Andreea, "Self-adaptable trust mechansim for web services" (2005). *Master's Theses*. 2708.
DOI: <https://doi.org/10.31979/etd.qpbf-jwj6>
https://scholarworks.sjsu.edu/etd_theses/2708

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

SELF-ADAPTABLE TRUST MECHANISM FOR WEB SERVICES

A Thesis

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Oana Andreea Dini

May 2005

UMI Number: 1427163

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI Microform 1427163

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

© 2005

Oana Andreea Dini

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

Melody Moh

3/31/05

Dr. Melody Moh

Chris Tseng

Dr. Chris Tseng

Alexander Clemm

Dr. Alexander Clemm, Cisco Systems, Inc.

APPROVED FOR THE UNIVERSITY

Therese L. Williamson

ABSTRACT

SELF-ADAPTABLE TRUST MECHANISM FOR WEB SERVICES

by Oana A. Dini

The scope of the thesis is twofold: (i) study trust models synchronized with levels of security and (ii) propose an extension to the current WS-Trust mechanisms. The extension will consider the evaluation of the past interactions between WSs and metrics associated with trust violations of these interactions. The results can be applied to monitor the SLA in WS-based systems and for early threats detection.

We considered a new mechanism to build and self-adapt the mutual trust of the interacting WSs. On the client side, we enforce a start with the less critical applications supported by the client-WS (use-context), while on the provider side, we consider various levels of trust. The levels of trust are induced based on the statistical interactions.

The proposed approach is introducing the trust profile and defining mechanisms to evaluate the evolution of trust based on current interactions. The theory is supported by experiments.

ACKNOWLEDGEMENTS

I am very thankful for the continued support of my supervisor, Dr. Melody Moh, for her input and guidance during my graduate studies. Her enthusiasm always inspires students to do their best.

I would also like to thank Dr. Alexander Clemm, my co-supervisor, who has given me invaluable input from the industry perspective.

I wish to thank the committee members for taking the time to read my thesis and provide me with useful comments.

I am indebted to my family for their support and patience with my hectic schedule while completing this thesis. I am grateful for the constructive input from my father-in-law, Dr. Petre Dini.

Last, I want to thank my husband, Cosmin, for his continuous support and encouragement during this entire work.

TABLE OF CONTENTS

LIST OF FIGURES	viii
LIST OF ACRONYMS	ix
1 INTRODUCTION.....	1
1.1 TOWARDS WEB SERVICES AND TRUST IN WEB SERVICES APPLICATIONS.....	2
1.2 SECURITY AND TRUST ASPECTS WITHIN WEB SERVICES-BASED APPLICATIONS..	5
2 WEB SERVICES	7
2.1 WEB SERVICE DISCOVERY (UDDI)	9
2.2 WEB SERVICE DESCRIPTION	12
2.2.1 <i>Document Description Languages</i>	12
2.2.2 <i>Resource and Services Description Languages</i>	13
2.3 COMMUNICATIONS PROTOCOL FOR WEB SERVICES	14
2.3.1 <i>HTTP</i>	14
2.3.2 <i>SOAP</i>	15
3 SECURITY FOR WEB SERVICES	16
3.1 WEB SERVICE SECURITY LANGUAGE.....	16
3.2 SECURITY ASSERTION MARKUP LANGUAGE.....	18
3.3 WS POLICY FRAMEWORK	20
4 TRUST IN WEB SERVICES-BASED APPLICATIONS.....	22
4.1 MODELS OF TRUST.....	23
4.2 WEB SERVICE TRUST LANGUAGE	27
4.3 FORMAL TRUST LANGUAGES	29
5 DESIGN AND DEVELOPMENT OF SELF-ADAPTABLE TRUST MECHANISMS.....	33
5.1 THEORETICAL DEVELOPMENTS.....	34
5.1.1 <i>Trust Profiles</i>	34
5.1.2 <i>Trust Profile Anatomy: Reputation, Credibility, and Context</i>	38
5.1.3 <i>Formalizing Trust Building Process</i>	41
5.2 SELF-ADAPTABLE MECHANISMS	47
5.2.1 <i>Trust Profile Revisited</i>	47
5.2.2 <i>Embedded-Functions</i>	47
5.2.3 <i>Self-Adaptable Trust Architecture</i>	48
5.2.4 <i>Self-Adaptability Trust Functional Architecture</i>	50
5.2.5 <i>Context Roaming for Updated Context Lists</i>	51
5.2.6 <i>Update the Lists with Original Invocations</i>	53
5.2.7 <i>Invocation Handling</i>	54

5.2.8	<i>Associate Validity Period</i>	56
5.2.9	<i>Trust Anomaly Report (notifications)</i>	57
6	IMPLEMENTATION, VALIDATION, EVALUATION	57
6.1	PRACTICAL SOLUTIONS AND VALIDATION	57
6.2	WEB SERVICES-BASED APPLICATION.....	61
6.3	IMPLEMENTATION	61
6.4	RESULTS AND INTERPRETATION.....	64
7	CONCLUSION AND FUTURE WORK	71
7.1	FUTURE WORK RELATED TO THE THESIS EXPERIMENTATION.....	72
7.2	FUTURE WORK RELATED TO THE CONCEPTS PROPOSED IN THE THESIS.....	73
7.3	OTHER ONGOING WORK ON COMPLEMENTARY ASPECTS.....	73
	REFERENCES	75

LIST OF FIGURES

FIGURE 1. WEB SERVICE ARCHITECTURE	8
FIGURE 2. MODELS OF TRUST	23
FIGURE 3. INTERACTION BETWEEN WSSs AND THIRD PARTY ENTITIES	39
FIGURE 4. REPUTATION STATE TRANSITIONS ON THE RECOMMENDER SITE	42
FIGURE 5. CREDIBILITY INDEX	44
FIGURE 6. TRUST LEVELS.....	46
FIGURE 7. SELF-ADAPTABLE TRUST ARCHITECTURE.....	49
FIGURE 8. SELF- ADAPTABLE ENGINE ARCHITECTURE	50
FIGURE 9. CONTEXT LISTS UPDATES.....	52
FIGURE 10. INVOCATION HANDLING	55
FIGURE 11. TRUST DOMAINS ALONG AN END TO END INTERACTION	59
FIGURE 12. ENVIRONMENT DEPENDENCE OF WSSs	60
FIGURE 13. SCREEN SHOT OF THE SIMULATION APPLICATION.....	61
FIGURE 14. DIALOG WINDOW SPECIFYING A SIMULATION	62
FIGURE 15. TRUST LEVEL (VERTICAL BAR DENOTES POINT AT WHICH WS1 IS ADDED TO WS10'S WHITE LIST AND FREQUENTLY INTERACTING LIST)	64
FIGURE 16. COUNT OF BAD REPLIES RECEIVED BY WS6 FROM WS7.....	66
FIGURE 17. CREDIBILITY INDEX OF WS6 FOR WS7	67
FIGURE 18. COUNT OF GOOD REPLIES RECEIVED BY WS6 FROM WS7	67
FIGURE 19. TRUST OF WS6 FOR WS7.....	68
FIGURE 20. COUNT OF BAD REPLIES RECEIVED BY WS12 FROM WS1	69
FIGURE 21. CREDIBILITY INDEX OF WS12 FOR WS1	69
FIGURE 22. COUNT OF GOOD REPLIES RECEIVED BY WS12 FROM WS1	70
FIGURE 23. TRUST OF WS12 FOR WS1	70

LIST OF ACRONYMS

Term	Definition
3GPP	3 rd Generation Partnership Project
BAL	Business Anchor List
BL	Black List
CA	Certificate Authority
CL	Candidate List
ECP	Enhanced Client and Proxy
FIL	Frequently Interacting List
FSM	Finite State Machine
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol
HTTPS	Secure Hyper Text Transfer Protocol
IdP	Identity Provider
OWL	Web Ontology Language
PKI	Public Key Infrastructure
RDF	Resource Description Framework
SAE	Self Adaptable Engine
SAML	Security Assertion Markup Language
SGML	Standard Generalized Markup Language
SLA	Service Level Agreement

SNMP	Simple Network Management Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
SSL	Secure Socket Layer
SSO	Single Sign-On
STS	Security Token Service
TAL	Trust Anchor List
TCP/IP	Transport Control Protocol / Internet Protocol
TMG	Trust Models Guidelines
TTP	Trusted Third Party
UDDI	Universal Description, Discovery and Integration
W3C	World Wide Web Consortium
WL	White List
WS	Web Services
WSDL	Web Services Description Language
XKML	XML Key Management Specification
XML	Extensible Markup Language

1 Introduction

In today's business world, customers demand complex answers delivered in a timely manner. To that respect, it is more desirable to find a complex of solutions rather than a complex solution. In other words, it is better to create an application from building blocks, rather than having monolithic software. Monolithic software is difficult to build, debug, maintain, update, or improve; putting together building blocks is a more suitable way, where each individual software component delivers a well-defined service.

Building blocks are developed by individuals (teams), and most of the time they are developed with different technologies. Composing larger software from such building blocks is cumbersome due to the heterogeneity of technologies, access invocations, data representation, entity semantics, and additionally to interaction testing and validation. Thus, a quick and easy integration of all these building blocks is needed (Daconta, Obrst, & Kevin, 2003). This integration has to be done in an automatic manner for purposes of productivity (automated interaction reduce the manual intervention), accuracy (man is less reliable performing repetitive tasks), safeness, and solving of complex problems (through cooperation).

The idea of integrating different software components sounds appealing. However, there are specific concerns that need to be addressed before such an approach can be turned into a viable solution. These issues can be separated in three categories: discovery, security, and trust. Discovery refers to the process of locating the necessary building

blocks. Depending on the specifics of the constraints, the discovery can be trivial or it can be very challenging. Security is related to the integrity of the communication between the components. Trust addresses the issue of the extent to which one can rely on a specific component. The question is: does a component perform the claimed operations?

1.1 Towards Web Services and Trust in Web Services Applications

In order to increase the availability of specific components, a good approach would be to have them accessible over the Web. The Web has wide acceptance, so this should only add to the success of such a deployment. One problem that we are faced with is that in most of the current applications of the Web, people have a priori knowledge of the servers that have the needed information. For example, the Yahoo search web page is usually available from www.yahoo.com. In this case, the trust is inherently built, and is reinforced through social networks, progressive experience with using the service, and provider persuasion by continuously improving the offered services. This feature is specific to the client-server paradigm.

With distributed, stand-alone designed and implemented software blocks, the trust problem is somehow different. Building blocks that provide certain services can be connected so that they form a web of services, hence the notion of Web Services (WS). Surprisingly, Web Services is not an extension of the known notion of client-server web paradigm, but more related to interacting building blocks. In fact, a building block is a service offering a set of operations (it could be only one operation as well) that can be

invoked by other building blocks (services). As they interact in a very complex way, with many simultaneous invocations of a particular operation (service), they form a web. In a web of services, any service could be at the same time invoker and invoked. To bypass the weaknesses of trust in interacting with a priori not known services, services must be defined in a very formal manner, understandable for any potential invoker.

Therefore, Web Services Description Language (WSDL) was created to fill this request. WSDL is used within the description layer of web services. WSDL specifies how to describe a web service by using XML grammar. A WSDL specification includes information on what a service does, how a service should be called (invoked), and where the needed service can be found. The information is used to make possible the communication between two web services, following the same specification approach.

Web services request a series of resources in order to run. To express these requests, resources must have an understandable qualitative and quantitative specification to guide the potential users, in occurrence, web services to refer to them. The Resource Description Framework (RDF) was proposed to accurately describe systems and networks resources. RDF is using XML style to describe resources available on the Internet (Manola & Miller, 2004). The purpose of its development was to ease the Web Service-based applications in correctly identifying the required resources. It provides a mechanism to catalog data (i.e., resource specification) in a machine-parsable format, allowing for efficient queries and manipulations when Web Services are deployed and

instantiated (Powers, 2003). It is assumed that the resources will provide the features as captured by their RDF specification. Therefore, the trust of using needed resources is embedded into Resource and Web Services specifications and it is not the subject of our research. We are targeting trust mechanism only at the Web Services layer.

The nature of resources is quite large, spanning from a type of port, to a type of processor, editor, compiler, security mechanism, and so forth. Therefore, it becomes too complex and RDF has some shortcomings on this aspect, especially related to limited richness of relationships between resources and dependencies among their characteristic. Semantic Web appeared as a solution to alleviate the issue. Building ontologies and bridging them semantically led to the notion of Semantic Web. Again, Semantic Web is not a relative of Web Services at all; it is, in fact, a web of various semantic specifications of various entities. The Web Ontology Language (OWL) is a language that is adopted by 3rd Generation Partnership Project (3GPP) and used to specify resources and relate different resource ontologies.

Web Services is a new approach, which still has unanswered questions. Semantic Web may help guiding some requests to appropriate resources and/or web services, but the inherent problems of interaction between a priori known entities (specifically Web Services) is still there. Even after a web service is identified, building a semantic trust based on the semantic web services (offered by applications implementing semantic web), the concerns are still valid. There are some concerns regarding the accuracy of the service implementation. Others are related to trust and confidence as well as past

experience with the specific service.

We focus on a particular aspect of this myriad of issues: identify different trust models between Web Services and build a self-adaptable trust mechanism allowing one web service to automatically tune its trust behavior according to the criteria we will identify or/and propose. Traditionally, the concept of trust is associated with aspects of authentication, that is, trust in that the other party is indeed who it claims it is. In our thesis, the term trust is more related to the idea of confidence, i.e. confidence that the other party will deliver within the claimed parameters.

To better explain our approach for building self-adaptable trust mechanisms, we will present the foundation of the web services and the mechanisms surrounding this paradigm. Our contribution is based on the current achievements, existing supporting platforms, specification languages, and protocols. We will build a tiny trust layer and appropriate formalisms to implement it.

1.2 Security and Trust Aspects within Web Services-based Applications

The challenge of using Web Services is that they bring with them new security aspects that are still not fully understood. Classically, the security risks are related to authentication, access control, integrity, and privacy (Negm, 2004).

In Web Services, both parties, requester and provider, have to be authenticated. Usually only the requester is authenticated by the provider, but in Web Services-based

applications the authenticity of both parties has to be ensured. An example where this requirement should be satisfied would be when the requester sends sensitive information, such as credit information, to the service provider. Therefore, both parties should sign their messages in order to ensure the integrity of it.

Access control is another security issue that has to be addressed. Usually access control is on a per user basis, but in Web Services the access has to be associated with the content also, and with the conditions the content is provided. The access control does not depend only on the identity of the user, but on the content of resource also. That is because Web Services offers the possibility to connect different users from different security domains, which leads to undefined authorization procedures. Therefore, new approaches need to be considered in order to ensure appropriate secure and trusted access (Negm, 2004).

The integrity of the messages' content is of importance also. Communications/transport protocols are also important to guarantee the preservation of expected quality of services (i.e., Web Services). Commonly, in Web Services-based applications, Transport Control Protocol / Internet Protocol (TCP/IP) transport stack is used, while (HTTP/SOAP) protocols are higher layer communications protocols. The communicating parties need to make sure that an unauthorized third party has not modified the received messages. To do this, public key cryptography can be used. The sender signs the message with its private key and sends it to the other party. The receiver can verify the signature by using

the sender's public key. This approach also provides non-repudiation. The receiver has proof that a transaction took place. The proof relies in the signature of the sender. Only the owner of the private key can sign a message with that key (Kaufman, Perlman, & Speciner, 2002). These mechanisms apply when exchanging trust information, and it must be part of any trust-related activities.

Web Services have to ensure the privacy of the exchanged messages also. Only the intended party should be allowed to read the contents of a message. In order to do this, the message has to be encrypted. Since most of the time the message has to pass through more than one intermediate point, the encryption has to be in such a way that needed routing information remains without encryption (Negm, 2004). However, the information pertaining to trust should be encrypted.

Several standard organizations and international fora developed and they continue to improve mechanisms and procedures to ensure the security requirements related to Web Services. These organizations proposed WS-Security, SAML, and XML Signature, which are used for specifying and verifying security and trust properties. Additional mechanisms are still needed to deal with trust and trust guarantees. We first introduce the fundamentals leading to the improvements we will present throughout this thesis.

2 Web Services

The exact definition of Web Services is still under debate. The simplest definition there

is states that a web service is any application that is available over the Internet and uses XML as message specification language (Cerami, 2002).

A more involved definition and probably one that summarizes most of the definitions available at this time states that a Web Service is a “software application that can be *discovered, described, and accessed* based on XML and standard Web protocols over intranets, extranets, and Internet” (Daconta et al., 2003). This definition mentions three properties that a web service should have: it has to be discovered, described, and accessed, as summarized in the architecture below. The entities mentioned in the architecture will be explained later in the document.

Figure 1. Web Service Architecture

The discovery of a web service is done through Universal Description, Discovery and Integration (UDDI) protocol. UDDI allows the users to find Web Services that are of interest to them. It also gives the possibility to register a web service so that potential

customers can find the service.

The description of a service is done through WSDL. WSDL is an XML document that provides the grammar used to specify the format of the messages.

2.1 Web Service Discovery (UDDI)

Universal Description, Discovery, and Integration (UDDI) is a protocol that specifies a standard method for describing, discovering, and integrating Web Services (Bellwood, Clement, & von Riegen, 2003). Simply put, UDDI is an XML-based web registry used by businesses to make themselves visible to other businesses. It enables businesses to register themselves together with their services, and it also allows anyone to search for Web Services that are listed in the UDDI Registry (Daconta et al., 2003).

The technical architecture of UDDI is composed of three parts (Cerami, 2002), (i) UDDI data model (an XML Schema for describing web services), (ii) UDDI API (a SOAP-based API for searching and publishing UDDI data), and (iii) UDDI cloud services (a collection of UDDI nodes coordinated to provide the same information from any node). The first two parts are used to access the data while the latter is used for hosting it.

UDDI data model defines four basic data element (Cerami, 2002): `businessEntity`, `businessService`, `bindingTemplate`, `tModel`, and `publisherAssertion`.

- The `<businessEntity>` element includes basic information about a business such as business name, description, address, and contact information.

- The <businessService> element includes information about a single service provided by a business (Cerami, 2002). This includes the name of the service, the description, and one or more <bindingTemplate> elements. The <businessService> element is contained within <businessEntity> element.
- The <bindingTemplate> element provides the technical descriptions of Web Services. It specifies how and where to access a specific web service. There is an *accessPoint* element that gives the network address where the web service can be invoked. This is usually in the form of a URL, but it can also be an e-mail address or even a telephone number (Bellwood et al., 2003).
- The UDDI <tModel> is mainly used to provide a pointer to external technical specifications. These specifications are usually WSDL files, but they can also be web pages that have instructions on how to interface with a service (Newcomer, 2002). Companies can use the information obtained from these files to determine if a web service is compatible with the requirements of their business (Chappell, 2002). Another role of tModel is to help in searches for a specific service. For example, if one wants to find a certain service, what it has to do is obtain a tModel key value that is associated with the desired service and search UDDI for businesses that offer such a service (Newcomer, 2002).
- The <publisherAssertion> element describes the relationship that one businessEntity has with another businessEntity. This structure is usually useful within large companies that cannot be represented by one businessEntity only since they have multiple departments or subsidiaries (Bellwood et al., 2003). The relationship between these entities can be made visible within their UDDI registrations if both parties publish identical <publisherAssertion>. By having this reciprocal publishing, one cannot claim a relation with another party if that relationship does not exist at that moment in time (Bellwood et al., 2003).

The UDDI API is a SOAP-based API that defines the functions needed for interacting with UDDI registries (Cerami, 2002). The API is divided into inquiry API and publishing API. The inquiry API provides functions for finding information about businesses, whereas the publishing API provides functions for inserting, deleting, and updating information in a UDDI registry.

The inquiry API helps the invoker in finding information about businesses by providing three kinds of query patterns: browse pattern, drill-down pattern, and invocation pattern. The *browse pattern* usually allows the user to perform a generic request that will return a list of businesses that match the search key word (Bellwood et al., 2003). The functions

used to retrieve information through this pattern begin with word “find”. Compared to *browse pattern*, the *drill-down pattern* allows the user to get more details about a business. The *invocation pattern* is used with Web Services so that when an application invokes a Web Service more than once, it does not have to query the UDDI server every time (Bellwood et al., 2003).

The publishing APIs are used for publishing, deleting, and updating information in UDDI registry. Before a business can use any of the publishing APIs, it must obtain an authentication token from an operator. Since each operator can implement its own authentication mechanism, each business must choose one operator for all publishing transactions (Cerami, 2002).

The UDDI cloud services provide a distributed directory where all the data is stored. Even though businesses store data at one operator node, the data is being replicated across all the other nodes. The data within a UDDI registry can be divided into three categories (Cerami, 2002): *white pages*, *yellow pages*, and *green pages*. The *white pages* include general information about a business such as the business name, phone numbers, address, and other contact information. It also provides unique business identifiers such as Dun&Breadstreet D-U-N-S numbers. The *yellow pages* include information that categorizes businesses. This information offers the possibility to find a service based on its category (e.g., software or hardware business). The *green pages* contain technical information about a web service, such as how to find a service and how to invoke it.

2.2 Web Service Description

2.2.1 Document Description Languages

2.2.1.1 HTML

Hyper Text Markup Language (HTML) is a computer language used for creating websites. HTML consists of a limited list of markup tags that are used for creating HTML documents (Shannon, 2004). The purpose of these tags is to tell the browser how to display the page. HTML defines a set of tags that are fixed and they cannot get a different meaning than the initially defined ones. Also, if a tag is not present in the HTML tag set, a user cannot add his/her own tag. A more powerful meta-language intended for the text documents, Standard Generalized Markup Language (SGML), that does not have these shortcomings is too complex.

2.2.1.2 XML

The Extensible Markup Language (XML) is a more flexible markup that provides rules for specifying tags, which are needed by the user to organize documents in a certain way. XML was created with the intention of having an easier and more flexible language that can have the functionality of HTML and SGML. XML allows a user to create new tags whenever they are needed and also to define the relationship between the tags (Harold, 2001) (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2004). XML is designed in such a way that the content of a document is separated from the display. The content of a document is developed through the use of XML syntax.

2.2.2 Resource and Services Description Languages

2.2.2.1 RDF

Resource Description Framework (RDF) is an XML-based language used to describe resources available on the Internet (Manola & Miller, 2004). These resources can be any information available on the Web, spanning from large files to individual values (e.g., price of an item, or type of a processor). RDF provides a mechanism to catalog data in a machine-parsable format, allowing for efficient queries and manipulations (Powers, 2003).

RDF is based on an object-attribute-value triple, which is usually written as A (O, V) (Davies, Fensel, & Van Harmelen, 2003). This notation can be understood as an object “O” having an attribute “A” with a value “V.” As an example, let us consider the sentence “The name of this person is John.” In RDF terms, this statement can be converted to a *triple*. The object O is the part that is being described (“this person”), the attribute A is a property of the object (“name”), and the value V is a specific instance of the attribute (“John”).

2.2.2.2 WSDL

The Web Services Description Language (WSDL) is used for the description layer of web services. WSDL specifies how to describe a web service by using XML grammar. WSDL specification includes information on what a service does, how a service should be called (invoked), and where the needed service can be found. The information is used

to make possible the communication between two Web Services.

A WSDL specification is composed of six main elements: definitions, types, message, port type, binding, and service. The *definition* element shows the name of the document. It also contains the namespaces that are used throughout the document. These namespaces are used to properly identify the elements needed to be referenced in the document (e.g., XML Schema, SOAP specifications) (Cerami, 2002). The *message* element defines the messages that are used (e.g., request, response). The *portType* element defines the methods provided. This method is listed under the <operation> tag. The *binding* element provides details on the transmission protocol (soap:binding), the format of the message (soap:body), and operations (soap:operation) for a particular port type (Newcomer, 2002). The *service* element indicates the location of the service (e.g., soap:address)

2.3 Communications Protocol for Web Services

2.3.1 HTTP

HyperText Transfer Protocol (HTTP) is an application-level protocol that has been used by the World Wide Web since 1990 (Fielding et al., 1999). Since this protocol is based on a request/response model, it is implemented in two programs: a client program and a server program.

Usually the client initiates the communication by sending a request to the server. This

message contains information such as the method used (e.g., GET), the URL, the HTTP version, the host where the objects can be found, etc. The server responds with a message, which has three parts: (i) status line (protocol version, status code, and the appropriate error message), (ii) header lines (e.g., time, date of HTTP response), and (iii) the entity body (the requested object). After sending the response, the server closes the connection (HTTP is a stateless protocol.)

There is also a secure version of HTTP called HTTPS (HTTPS, 2004), which stands for HTTP over SSL (Secure Socket Layer). This protocol is used to ensure secure communications over the Web by providing encryption and authentication. The level of protection depends on the browser's implementation of the protocol. There have been flaws in some implementations that made possible attacks such as man in the middle attack.

2.3.2 SOAP

The Simple Object Access Protocol (SOAP) is a protocol used for transporting information between two computers. According to World Wide Web Consortium (W3C), SOAP is “a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML, an extensible messaging framework containing a message construct that can be exchanged over a variety of underlying protocols.” (Gudgin, Hadley, Mendelsohn, Moreau, & Nielsen, 2003) The three main aspects of this protocol are extensibility, usability over different transport protocols, and independence from programming models

and platforms (Gudgin et al., 2003). The extensibility stems from the simplicity of the protocol. The authors of SOAP wanted to make sure that the basics of the protocol would be supported in all implementations, so they kept the specifications simple. Different features such as reliability, security, and routing were omitted for that reason. SOAP is layered on top of the network and transport layer, so as an added benefit, it is not tied to a specific transport protocol (Snell, Tidwell, & Kulchenko, 2002). However, HTTP is the most widely used transport for exchanging XML-based SOAP messages. It therefore appears more often in literature implementation specific details. The SOAP messaging building block is the one-way message.

A SOAP message contains three major blocks: the envelope, the header, and the body. The envelope, which may contain an optional header, marks the start and the end of the SOAP message. It also has the namespace that identifies the SOAP version used in the message. If the header is present, it usually gives information on how the message should be processed. Examples of such information would be routing information and authentication. The body element is required and it contains the actual message that has to be delivered (Snell et al., 2002).

3 Security for Web Services

3.1 Web Service Security Language

Web Services Security (WS-Security) is a specification that describes improvements to

SOAP messaging that can satisfy message integrity, confidentiality, and security token passing (Nadalin, Kaler, Hallam-Baker, & Monzillo, 2004). These SOAP extensions are referred to as “Web Services Security: SOAP Message Security” or “WSS: SOAP Message Security”. It is possible that trust requirements will ask for specific extensions.

The main goal of this specification is to provide applications with a secure way for exchanging SOAP messages. Since there are a number of security protocols available, the intention of this specification is to fill the gap between these security protocols and Web Services, by providing an interface between them that allows the use of such standards (X.509, Kerberos, and SAML) for securing Web Services messages (MacVittie, D., 2004). The developers of this specification defined different “profiles” that describe how authorization or authentication standards should interact with WS-Security (e.g., X.509 Profile, SAML Profile). This security information is present in the SOAP header, and the fields used within the header depend on the Profile.

In order to provide secure message exchange, WS-Security language must satisfy a number of requirements: *multiple security token formats, multiple trust domains, multiple signature formats, multiple encryption techniques, and end-to-end message content security* (Nadalin et al., 2004). Just to see the importance of these requirements, take as an example a message that has to traverse multiple hops from source to destination. Every hop encountered by the message has a certain trust associated with it. Also, different encryption keys can be used along the message path. The security associated

with the message has to be preserved over the entire path. This cannot be done by HTTP-based security since this security technique only addresses point-to-point security (Seely, 2002). WS-Security ensures this end-to-end security by satisfying the above requirements.

In addition to integrity and confidentiality, WS-Security also defines a mechanism for passing the security tokens. This is done through the use of <UsernameToken> and <BinarySecurityToken> elements. The former element offers a way for providing a username. The latter element is used for including the binary security tokens such as X.509 certificate and Kerberos ticket. All security information is specified within the <Security> header element. This header is designed to be flexible so that it can support different types of security mechanisms. For example, there are multiple security tokens used for identifying the user. These tokens have to be carried by the header. Also, if a message is signed, the header has to give information about the manner in which it was signed and the storage location of the key. Other information included within this header regards the encryption (Seely, 2002).

3.2 Security Assertion Markup Language

Security Assertion Markup Language (SAML) is a standard that defines an XML-based mechanism for passing security information, such as authorization and authentication, between parties (Daconta et al., 2003). SAML has four components: *assertions*, *protocols*, *bindings*, and *profiles*.

Assertions are a set of claims made by an Asserting Party regarding a user (Hughes & Maler, 2004). An Asserting Party is a system that can provide assertions about a user. For example, it can assert that a user has been authenticated and it has some specific privileges. In SAML, an Asserting Party is also known as Identity Provider (IdP) or SAML authority. Parties that have some services and want to control the access to them can use these assertions. These parties are known as Service Providers (SP) or Relying Parties (they “rely” on the assertions given by the Identity Provider) (Cantor, Kemp, & Maler, 2004).

SAML specifies three types of assertions: authentication, authorization, and attribute. The authentication assertion contains information about who provided the assertion, the authenticated party, and the time period for which the assertion is valid. The authorization assertion defines the access permissions of a user to a certain resource. The attribute assertion provides detailed information regarding the user (e.g., VIP status) (Hughes & Maler, 2004).

SAML *protocols* specify the syntax of request and response messages. There are six protocols defined so far: assertion query and response protocol, authentication request protocol, artifact protocol, name identifier management protocol, single logout protocol, and name identifier mapping protocol (Hughes & Maler, 2004).

The *Bindings* define how SAML requests and responses should be transported (Hughes &

Maler, 2004). Most of the time, SAML messages are sent via SOAP over HTTP, but the next step is to remove SOAP and have HTTP as the transport protocol (MacVittie, L., 2003). There are six bindings defined: SAML SOAP Binding, Reverse SOAP (PAOS) Binding, HTTP Redirect Binding, HTTP Post Binding, HTTP Artifact Binding, and SAML URI Binding (Cantor, Hirsch, Kemp, Philpott, & Maler, 2004).

The *profiles* specify how to combine SAML assertions, protocols, and bindings in order to use them in different cases, e.g., Single Sign-on (SSO) (Hughes et al., 2004). The communicating parties have to know these rules in order to properly insert and extract the assertions present in the message. There are eight profiles defined so far (Hughes et al., 2004): Web Browser SSO Profile, Enhanced Client and Proxy (ECP) Profile, Identity Provider Discovery Profile, Single Logout Profile, Name Identifier Management Profile, Artifact Resolution Profile, Assertion Query/Request Profile, and Name Identifier Mapping Profile.

3.3 WS Policy Framework

Web Services Policy Framework (WS-Policy) specifies a model and a syntax used for describing policies for Web Services (Bajaj et al, 2004). The grammar is flexible and it can be used by different Web Services specifications to describe their own requirements and capabilities. This specification does not define how a policy is associated with a service. Web Services Policy Attachment (WS-PolicyAttachment) specifies mechanisms used for associating policies to subjects (Bajaj et al., 2004).

WS-Policy defines policy as being a group of policy alternatives, where a policy alternative is a group of policy assertions. A policy assertion is the fundamental element of a policy (Nolan, 2004). It usually represents a requirement of a policy such as the choice of transport protocol or the authentication method. Therefore, a policy is constructed by combining policy assertions into sets that form policy alternatives. The grouping of assertions is done through the use of policy operators, which are `<wsp:ExactlyOne>` and `<wsp:All>`. The alternatives are not ordered and so are the assertions within an alternative. There can be zero or more alternatives within a policy (Bajaj et al., 2004).

In Web Services, policies are used to communicate the requirements that need to be satisfied in order for the parties to interact (Bajaj et al., 2004). Usually, Web Services providers make their policies public so that potential clients know the conditions under which they can get a certain service. If a client cannot satisfy the requirements, then it cannot make use of the service. Thus, by making the policies visible, a client can decide if it will use a certain service or not. Policies are disclosed in an XML-based format called policy expressions. These policy expressions can be represented in a *normal* form or in a *compact* form. The *normal* form is a simple XML representation of a policy expression. It contains a listing of all the alternatives, which contain all their assertions. The user can only choose one alternative and this is expressed in the format of the policy expression through the use of `<wsp:ExactlyOne>` operator.

The *compact* form of a policy expression is still represented in an XML format, but it specifies an additional attribute called @wsp:Optional and a technique called *inclusion*. The attribute @wsp:Optional is used to indicate that an assertion is optional. The inclusion technique offers a way for sharing assertions between policy expressions. It specifies an element called <wsp:PolicyReference> that can be included in a policy expression to reference another policy. The contents of the referenced policy replace the element <wsp:PolicyReference> (Bajaj et al., 2004).

Another technique defined by WS-Policy is called *intersection*. It allows the potential interacting parties (a service provider and a potential user) to compare their policy alternatives and identify the ones that are compatible. Two alternatives are compatible if they have the same vocabulary. A vocabulary is defined as being a set of the policy assertion types (XML QNames such as <someName =...>) that are used within a policy. If two alternatives are found to be compatible, then they are combined in a new alternative. In order to determine if alternatives are compatible, domain knowledge is needed since the meaning of some assertion is specific to the domain to which it pertains (Nolan, 2004).

4 Trust in Web Services-based Applications

Trust between different parties can be achieved in multiple ways, but it is better when there are some guidelines that can help in the process of trust achievement. The “Trust Models Guidelines” (TMG) provides a guideline that can be used in creating trust

relationships between different parties (Linn, 2004). Its focus is on authentication and business relations between parties. Depending on what kind of relationships the involved parties have, a level of trust can be achieved.

4.1 Models of Trust

TMG describes three big models of trust, each of them having two cases. The three models are Pairwise Trust model, Brokered Trust model, and Community Trust model.

I. In the *Pairwise Trust* model, both parties have direct business agreements with one another. The authentication can be done directly (an out-of-band key exchange) or indirectly (use of Certificate Authority (CA)). These two types of authentication lead to this model's two cases: Pairwise/Direct and Pairwise/Indirect.

Figure 2. Models of Trust

In the Pairwise/Direct model, the interacting parties have direct business and authentication trust established. The recipient has the other party's identity listed in its Business Anchor List (BAL) and the key used for authentication listed in its Trust Anchor List (TAL). In the Pairwise/Indirect model, the interacting parties have a direct business trust established, but not a direct authentication trust. In this case, the recipient has the identity of the other party listed in its BAL, but the key is not present in its TAL. An assertion can be trusted if within the receiver's TAL there is a CA that can start a trusted path that leads to the other party's certificate (Linn, 2004).

In Pairwise Trust Model, the level of trust depends on the business agreements that are established among parties. Because of the requirement for a signed business agreement, this model leads to the formation of small communities that are hard to extend (Linn, 2004).

II. In the *Brokered Trust* model, the two parties do not have direct business trust with one another, which means that the identity of one party is not listed in the other party's BAL, but they have agreements with other parties that can help building a trusted path between the two parties.

Since there can be direct and indirect authentication trusts, there can also be two cases for this model: Brokered/Direct model and Brokered/Indirect model. In the Brokered/Direct model, the recipient has a direct authentication trust with the other party, which implies

that the other party's key is listed in the recipient's TAL. In order to establish complete trust, a business trust needs to be achieved. This can be done with the help of an entity that has a business trust with the recipient and with the party that sent the assertion. This entity can act as an intermediary broker. In the Brokered/Indirect model, the two parties do not have a direct authentication trust, which means that the key of one party is not present in the other party's TAL. Since there is also no direct business trust, the identity of one party is not present in the other party's BAL either. The business trust can be achieved as in the case of Brokered/Direct model, but the authentication trust depends on the methods used for authentication. These methods can be Public-Key infrastructure (PKI), Kerberos, or SAML. For example, if PKI is used, there are two manners in which the recipient of a signed assertion can check if the signature can be trusted. If the CA that signed the certificate of the local identity provider signed the certificate of the sending party, then the signature can be trusted. If the certificate was not signed by the same CA, but a trusted path from the recipient to the certificate of the sender can be built through the use of other CAs, then the signature can be trusted (Linn, 2004).

The Brokered Trust model allows for a larger set of entities that can interact with each other. Entities that do not have direct business trust or direct authentication trust can still interact if a valid chain of trust can be obtained.

III. In the *Community Trust* model, the two parties do not have a direct or indirect business agreement. They pertain to the same community that has an authentication

model, which is common to all the community's members. Based on this type of authentication, the members of the community form a business trust among themselves, trust that can be used to allow communication between parties. This model, like the other two models described before, has two variations, which are Community/Direct model and Community/Indirect model.

In Community/Direct model, the authentication between two entities is done directly. An example would be a community that allows its members to sign their own certificates. The certificates and keys are exchanged between the entities directly, without the involvement of a Third Trusted Party (TTP). This method can have a few disadvantages. Since the community members are responsible for the verification of the other party's identity, they cannot be entirely trusted to follow the recommended procedures such as the verification of a certificate. Also, the management of the keys used within a community is done by a designated officer. As the community grows, the keys management becomes harder and less secure.

The Community/Indirect model makes use of an already established trust infrastructure service. An example would be a community that has in its TAL a list of TTPs that provide PKI services. Any member of the community can verify the validity of a certificate by performing some method that determines if the certificate respects the policy of at least one of the listed TTPs. This approach has the advantage of using a technology that is well established and tightly controlled.

4.2 Web Service Trust Language

Web Services Trust Language (WS-Trust) specifies a way for requesting and emitting security tokens and also for managing trust relationships (Anderson et al., 2004). WS-Security provides a way for a secure exchange of SOAP messages. WS-Trust specification adds new elements that allow a party to decide if it can trust the credentials of the other party. It also specifies methods used for emitting, renewing, and validating security tokens.

A simple example of this specification would be a client sending a request to a Security Token Service (STS) for a security token. The client can send this token to a service provider in order to obtain permission to use a service. If the service provider trusts the STS that emitted the token, then it trusts the client that has the token and allows it to use the requested service (Madsen, 2004).

From this example, one can see that there are different elements needed for this process to take place. One of them is *getting* the security tokens and the other one is *trusting* the tokens. WS-Trust has two techniques for getting security tokens and three techniques for evaluating the level of trust of a token.

A token can be emitted as a result of a request or it can be an “out-of-band token acquisition” (Anderson et al., 2004). In the first case, the request can be performed either by the client or by a second party that acts as an agent for the client. If the STS trusts the

received token and the party that sent the request can prove that it has that token, the STS will exchange the token with another token in the form needed by the client (Anderson et al., 2004). In the second case, a token is sent to a party without being specifically requested by that party.

Services can decide if they could trust certain security tokens by using one of the three techniques described in WS-Trust.

(i) The simplest technique involves “fixed trust roots” (Anderson et al., 2004).

The party that uses this mechanism has a fixed group of trusted roots from where it accepts tokens. If the received tokens are not from any of those roots, then the transaction is not accepted.

(ii) Another technique used for establishing trust uses a hierarchy of nodes. When a request is received, the recipient will evaluate the entire hierarchy and, if it leads to a trusted root, the transaction is accepted.

(iii) The last technique involves the use of an “authentication service”. In this case, the recipient sends the token to the trusted authentication service and waits for a response. If the response is positive, then the transaction is accepted.

WS-Trust describes additional mechanisms that can improve the process of requesting security tokens and responding to such requests. One of these mechanisms uses challenges (Anderson et al., 2004). A client can send a request for a token to a certain service. The recipient can send back an answer that contains a challenge. If the client

can give the expected answer to the challenge, then the recipient sends a message to the client containing a security token.

An issued token can also be renewed if it has an expiration period associated with it. This attribute can be specified within the `<wst:Lifetime>` tag. The party that requests token renewal has to prove that it has the right to use the token. This proof is not needed in the case where the recipient trusts the requesting party (Anderson et al., 2004).

WS-Trust also provides a way to validate a security token. The response of a validity request can be another *token* or a *status*. This preference is specified within the `<wst:TokenType>` tag.

4.3 Formal Trust Languages

Most of the WS-Trust features deal with authentication of the cooperating parties, i.e., an invoker is either authenticated or not. Less is captured on the operational side, i.e., if a trusted and authenticated party provides the services as claimed. As an example, the Quality of Service (QoS) of a particular service may not fulfill the Service Level Agreement (SLA), yet the service is accepted by its invoker. We will hereafter refer to this aspect as *operational trust* and define it properly later. In this case, it is assumed that the interacting parties have an agreement on what their operational trust is. Levels of operational trust and appropriate SLA trust violations must be in place. Before developing around our theory on operational trust, we will briefly present several formal

trust languages and trust logics for measuring and expressing these kinds of features.

Language/ Logic	Applicability		Potential for self- adaptability techniques
	<i>Authentication Trust</i>	<i>Operational Trust</i>	
History-based signature (Bussard, Molva, & Roudier, 2004)	Yes	no	Only the idea partial trust could be extended on the operational trust
Decentralized reputation-based logic (Pavlov, Rosenschein, & Topol, 2004)	Yes	partial yes	The advanced functions on feedback data could apply
Reputation-based trust (Ruth, Xu Bhargava, & Regnier, 2004)	No	yes	Create credentials for the purpose of recommending trust;
Trust Requirements (Giorgini, Massacci, Mylopoulos, & Zannone, 2004)	yes (+ access control)	no	Trust Delegation
Items Recommender Systems (Massa & Bhattacharjee, 2004)	No	yes (but at the level of item)	Computability of trust and selecting among similar partners
Formal Logic Framework (Demolombe, 2004)	general	general	Graded trust and conditional trust could be extended and formalized
Trust Mediation Theory (Casltelfranchi, 2004)	general	general	Degree of trust and predictions
Use of Formal Methods fro Trust (Butler, Leuschel, Lo Presti, & Turner, 2004)	No	no	Use of formal methods [Z, B, T-SAS]
XML-based Trust Negotiation	yes	no	

Language (Bertino, Ferrari, & Squicciarini, 2003)			
---	--	--	--

Table 1. Formal Trust Languages and Trust Logics

Bussard (2004) provides a logic for describing the “proof of knowledge” in the sense that one party knows some of the secrets of the other party, without revealing this. A history-based signature scheme is used to provide different context proofs (recommendations) on locations, interactions, etc. A history-based signature is derived considering partial trusts and relationships between parties entitled with partial trust.

The authors of (Pavlov et al., 2004) introduce a decentralized reputation system based on the degree of satisfaction and dissatisfaction between interacting parties. The reputation is used for witness selection. While the system is decentralized, the protocols are based on external reference.

The paper (Ruth et al., 2004) presents a reputation based system used for accountability. The role-based trust management language is used for defining trust views and trust judgments based on the exchange history captured into a Direct Acyclic Graph (DAG). The language is based on the model proposed by Abdul-Rahman & Hailes.

A formal framework for gathering and specifying trust requirements at the enterprise level is presented in (Giorgini et al., 2004). The framework considers the authorization

trust (authentication + access control) and trust delegation.

In (Massa & Bhattacharjee, 2004), an analysis of Recommender systems based on Pearson correlation coefficient and cosine similarity is used to derive statistics of community interchanges. It is shown that recommender systems incorporating trust mechanisms are more effective than systems using classic techniques.

The framework from (Demolombe, 2004) proposes several formal definitions of trust, based on trust epistemic properties. The framework supports the notion of “graded trust” which will be enriched and formally defined in our theory later. Conditional trust will be also reconsidered in defining the *trust context*.

The theory in (Casaltelfranchi, 2004) introduces a cognitive anatomy of trust, focusing on quantitative elements of trust. The notions of *internal and environmental trust* and *degrees of trust* will be further exploited in our proposal.

The paper (Butler et al., 2004) identifies ways to formalize different aspects of trust. An approach for modeling and validation of trust is presented in the context of Trusted Software Agents and Services in Pervasive Information Environment (T-SAS).

In (Bertino et al., 2003) an XML-based trust negotiation language is presented. The main contribution is representing the authentication credentials as XML tokens and

emphasizing how to encode certificates and policies during trust negotiation.

In conclusion, most of the state-of-the-art proposals focus on authentication trust.

Operational trust requires more accurate historical data on previous interactions between parties, specifying some QoS/SLA parameters as well. Some of the formal principles used for modeling and validating the authentication trust can be extended to cover requirements of operational trust. Specific requirements and proposed solutions for operational trust will be discussed in the next section.

5 Design and Development of Self-adaptable Trust Mechanisms

Most of the current models and implementations rely on the WS-Security Language and SAML and make use of WS-Policy Language for defining the type of trust Web Services may either negotiate or query.

In the context of networks-on-demand, autonomic computing, and utility computing, there is a major request for Web services to become autonomous. This means, among other things, that a Web Service possesses mechanisms to select, tune, and adapt the type of trust and trust levels.

Before presenting our theoretical proposals for operational trust, we position the work as follows:

1. We distinguish between authentication (authorization) trust and operational trust. The latter implies specific QoS/SLA features and guarantees/violations policies. It constitutes the target of our proposal.
2. We identified centralized and decentralized trust establishment schema. The proposal we make is referring to the decentralized scheme; in our framework the components are self-managing operational trust. Additionally, every such a component may use the external services for complementary information (aka, behaving as in a centralized system, but not exclusively).
3. Related to the above item, the decisions are first based on internal references of the components (in occurrence, a WS keeps core-data to process and select the most trustable WS, or a WS with a given trust level), as opposite to external reference only.
4. Trust is manifested at the WS level (cooperation trust), and transport support level (transport trust). While we may simulate for the purpose of implementation some parameters of transport trust, we focus our proposal on cooperation trust.
5. Feedback for attempted cooperation has been used in the past for reporting on failures or trust violations, rather than to proactively select the most trusted service/component. Our proposal is focusing on the latter.

5.1 Theoretical Developments

5.1.1 Trust Profiles

Trust profiles are artifacts to capture essential information that constitutes the basis for

establishing a trusted cooperation between WSs.

The following parameters are proposed as core-features for a trust-profile:

Interaction mode

Context

Trust level

Credibility Index (associated with a trust level based on individual experience)

Reputation (associated with a trust entity, based on common experience)

Black list (non-trusted for requesting services form)

Denial list (not accepting requests from)

White list (by default trusted services)

Frequently interacting partners

Validity period

Renewal mode

Interaction mode is a set value with a minimum content of {operational, transport}.

Other values may be added.

Context expresses a peer relationship <WS-ID, WS-Service>, where WS-ID is the registered name for WS (as shown in UDDI), and WS-Service is one of the service offered by a WS, as presented by WSDL specification. Therefore, a WS may have a set of peers representing such contexts.

Trust levels represent the degrees of trust a WS can have with another WS. While the granularity of the degree can be as finely grained as desired, we do adopt the commonly used set of levels {*very untrustworthy*, *untrustworthy*, *trustworthy*, *very trustworthy*, and *unknown*}. For the reason of computation, the trust levels will be associated with thresholds, e.g.,

IF more than “10”, “requests” result in “QoS violations”

THEN “trust level” is “very untrustworthy.”

We introduced a new level “*unknown*” that reflects the trust level for a newly introduced context, or for a context that was not yet invoked.

Credibility index associated with a trust level expresses the nature of the trust level. For example, if a trust level reaches the level of “trustworthy” for the very first time, the credibility is still low. Only by reinforcing the trust level through multiple interactions, the credibility will increase. For the purpose of illustration, the set {original, medium, outstanding} can express three increasing stages of credibility index. For the computation reasons, each credibility index should be associated with a threshold, e.g., IF the “number of interactions” ever performed, is less than “3”

THEN “credibility index” is “original.”

Credibility index is based on direct experience in a given context.

Reputation contains the collective assessment of the working results in a particular context. While reputation is based on external sources, we do integrate it within the WS profile to validate the credibility index versus the general experience. For the purpose of complete theoretical presentation, we do propose the following simple value set {good, bad}.

Black list represents those contexts that are not recommended to be invoked. This could be due to various reasons, from individual experience, common experience, or as a

management/ monitoring directive. We adopt the context-based black list, as, for some context, the same entity may be on the black list, while for other context, it will not be. There is no historic information or feedback gathering on black list contexts.

Denial list contains those contexts whose invocations must be denied. This is for protecting a given WS.

White list contains those contexts that do not require additional computation for invocation. They are by default exposing “very trustworthy” and “good” in terms of trust level and credibility respectively.

Validity period represents the duration when a trust level is valid for a given context. The validity period is either set when a given context is instantiated, or it is tuned based on the interacting experiences. The validity period could be measured in epochs (when a context has a stable trust level), or as time intervals $\{(t_i, t_j)\}$.

Renewal mode expresses the way a validity period becomes not valid. In our scheme, the renewal period is $\{expiration, revocation\}$. Expiration is determined by validity period. Revocation is handled by special events occurring in the system.

Frequently interacting partners represent a set of contexts that form the core-entities a WS is exchanging on regular basis. This is relevant in the situation when a WS is

interacting with a large set of WSs; only those frequently interacting services are checked against the profile lists identified as profile parameters.

Certain correlations among the values of profile instances are checked and maintained by a WS itself.

5.1.2 Trust Profile Anatomy: Reputation, Credibility, and Context

The basis of our proposal is derived from the model proposed for the non-automated interactions where trust decisions are made based on a third party's opinion. These opinions are the result of the third party experience with the recommended services. Since these are the third party's direct (or collected) experiences, the level of trust accorded to the service should not be the same as the level assigned by the third party. Abdul-Rahman proposes a trust model that allows building trust based on recommendations and provides a technique to compute the difference between recommendation and personal experience. In this model, each party ranks all personal experiences in four categories: very good, good, bad, and very bad. The recommendations are ranked based on the result obtained from the difference between recommendation and personal experience. For example, entity A considers entity B as being "good" and recommends it with this category. However, entity C may have personal experiences with B that are categorized as "bad". Therefore, C modifies the recommendations from A by reducing them with one level, in this case from "good" to "bad" (Abdul-Rahman & Hailes, 2000). However, no mechanisms on how to build a

personal experience and use it were proposed.

In a classical framework, the interaction scenario is the following: Web Services communicate with each other after certain requirements are satisfied; these requirements are usually about security (confidentiality, integrity, etc) and trust. The security requirements are met through the application of different security mechanisms such as PKI, Kerberos, and X.509. The trust between different Web Services is established after the interacting parties check with a trusted third party (e.g., STS) on the validity of the other party's claims. The level of trust depends on how much a Web Service trusts the authority (e.g., STS) that issued the tokens and also on the context in which the tokens are used.

Figure 3. Interaction between WSs and Third Party Entities

The figure above shows an interaction between Web Services and a third party entity (broker). The broker (e.g., STS) manages trust relationships between Web Services. The witness could be used as decision factor when trust violations arise (see below on trust

violations). The advisor does not have the authority of a broker, but it can advise on certain services. For example, if a client is looking for a service that offers language translation (e.g., English to French), it can ask an advisor for information on Web Services that provide such service. The advisor may not be aware of a Web Service that provides the appropriate service, but it may know a WS that offers a good dictionary. Since there is a very good chance that such WS may also have a translator, the advisor can suggest to the client to try this WS. The existing models try to evaluate the correctness of external recommenders.

In our proposal, there are intrinsic relationships between components of *trust profile*. These relationships cannot be quantified in a model where a WS_i is solely relying on external entities to derive the level of trust on another WS_j . Self-adaptable functionalities, which solve this problem, will be introduced later. Such a mechanism will require minimum memory and computational power.

Rule 1: A trust violation represents a quantified deviation from the credibility index. A quantified deviation from reputation is not a trust violation.

Rule 2: A trust violation occurs in a given context. Trust violations of different contexts are not cumulative for changing the trust levels. Trust violations in contexts referring to the same WS-ID could lead to register that WS on the Black List. Most often trust violations trigger transition between the levels of trust (represented as a Finite State

Machine (FSM)).

Rule 3: The revocation can be recurrently applied for renewal; this means that a trust can decrease with more than one level within the same validity period. Ultimately, revocation may lead to a transition from the “very trustworthy” to “very untrustworthy.”

Rule 4: Trust level transitions are only driven by individual measurements of credibility. A transition could be marked as ‘forced’ when the reputation conflicts with the trust level tendency.

5.1.3 Formalizing Trust Building Process

5.1.3.1 Reputation

Reputation was introduced for two reasons, (i) as a guideline to drive the trust level when the trust level is *unknown*, and (ii) for exporting by request or notification, the level of trust in a given context, as perceived by a particular WS. This information is useful to Recommenders (brokers) to build a community sense for reputation of a given context (and WS, as well).

Figure 4. Reputation State Transitions on the Recommender Site

Reputation status could be either unknown, or good/bad. A Recommender has mechanisms to capture the threshold crossing once a context is known. For example, for particular types of WSs, one may define

$WS_{good} := x\% \text{ of WSs report trust_level in } \{vt, t\}$

$WS_{bad} := y\% \text{ of WSs report trust_level in } \{ut, vut\}$

$WS_{unknown} := \langle \text{no reports} \rangle \text{ or } \# \text{reports less than thresh-reports}$

Every recommender (broker) has its own reputation schema for establishing the reputation of a WS. For example, an aggressive reputation schema may require $x = 95$, while other weak reputation schema may only impose $x = 80$. Additionally, it is possible that a WS is considered good, even though a limited number of reports about that WS classify the trust-level in $\{ut\}$.

Variations of required x and/or y on the above expressions trigger appropriate transitions in the reputation FSM. In our approach, the reputation strictly depends on recommender's (broker's) policy to evaluate WS and/or contexts. Recommenders have their own translation schemes for inferring the reputation of a WS from the trust levels reported by WSs.

Alternatively, a WS may have special mechanisms to differentiate reputations coming from various Recommenders. For example, a good reputation coming from a recommender may conflict with a bad one coming from another recommender. In this case, the reputation is neglected. Assuming that reputation is a driving factor to adjust trust-level (this particular aspect it is not in our theory), each WS must have a reverse translation, per recommender, to estimate the trust level. This is done by combining the reputation with credibility index, separately and internally computed.

5.1.3.2 Credibility Index

Credibility index associated with a trust level expresses the nature of the trust level. For example, if a trust level reaches the level of “trustworthy” for the very first time, the credibility is still low.

Credibility index has a simple computation formula:

$$\begin{aligned} C &= [\#invocations - \#violations] / \#invocations \\ &= 1 - (\#violations / \#invocations) \end{aligned}$$

According to various thresholds, credibility index may define a few given states to identify a coarse-grained set of states when combining credibility index and the number of invocations.

Figure 5. Credibility Index

Credibility index is computed for every context populating a context list. It is used to enforce a trust level and also to roam context from one list to another. Roaming helps self-adaptable mechanisms to avoid additional computation and/or checking every time a context is invoked. Credibility index is a relative measure depending on the number of

invocations of the same WS and the number of revocations. While a differential credibility index can be derived, considering different request parameters that are violated, for the sake of simplicity, we consider a single type of violation, namely a bad reply.

5.1.3.3 Trust Levels

While the degree of granularity can be as finely grained as desired, we do adopt the commonly used set of levels $\{very\ untrustworthy, untrustworthy, trustworthy, very\ trustworthy, and\ unknown\}$. For updating the trust level, a function is used to control transition between the defined states.

**Trust_level = F (context, credibility, reputation,
renewal periods, #invocations, #violations, control_mode)**

Where:

F: {set of values} $\rightarrow$ {vt, t, ut, vut, unknown}

For example, one can define the level “*t*” as follows (triggered by the transaction 3, from “*vt*” or transaction 5, from “*ut*”):

vt:= = credibility = outstanding
& revocations < thr-rev
& # violations < thr-violations
& renewal period > thr-renewal

& # invocations > thr_invocations

where the thresholds are defined by each WS in turn, in general common for all contexts.

Figure 6. Trust Levels

For every transition, the function F must be defined. It is not mandatory that all the parameters vary from one trust level to another. As the trust is particular to every WS, the transitions within the trust FSM are specific to each WS. While a uniform trust level FSM will normalize the recommender's reputation, it is our thesis that transitions are particular to each WS. They are dictated by the WS design requirements, robustness of the WS, mission critical aspects of a WS, real-time aspects of a WS, etc. WS of a same

type may have a similar trust level FSM. A trust level is always associated with a validity period.

5.2 Self-Adaptable Mechanisms

5.2.1 Trust Profile Revisited

We identified key attributes of the trust profile needed for self-adaptation, as summarized below:

- *Context* $\langle ws-id, ws-s-id \rangle$
- *Trust level* $\langle vt, t, vt, vut, ukn \rangle$
- *Credibility index* $\langle original, medium, outstanding \rangle$
- *Reputation* $\langle h, dh, ukn \rangle$
- *Black list* {context}
- *Denial list* {context} [to refuse from]
- *White List* {context} [no additional computation]
- *Frequently interaction partners* {contexts} [no additional computation][no additional checking]
- *Validity period* $[ti, tj]$, associated with a level of trust
- *Renewal mode* {exp, rev}

As our target is operational trust, the component *ws-s-id* is very relevant. On one hand, it expresses a given operation by a WS in particular context. On the other hand, the service definition itself may contain various QoS/SLA parameters that are subject of operational trust violation.

5.2.2 Embedded-Functions

We identify four groups of functions that are specific for self-adaptable engine. They

allow a WS to interact with another WS by considering their mutual level of trust.

1. Profile handling

- **Update** *<profile, parameter>*
- **Read** *<profile, parameter>*

2. Operational functions

- **WS invocation** *<context>*
- **WS-response** *<context>*

3. Exchange with Recommenders (brokers)

- **Request external reputation** *<context, reputation>*
- **Provide external trust levels** *<context, trust-level>*

4. Report incidents

- **Send** *<notifications>*, where notifications may be of many types

5.2.3 Self-Adaptable Trust Architecture

The architecture captures essential features proposed to measure and keep updated a trust profile for a given context.

Figure 7. Self-Adaptable Trust Architecture

The main specification parts are identified as following:

There are four typed interfaces, allowing the operations identified in the previous section to occur. Each WS posses its own operational behavior (regardless of the fact that it embeds or not a self-mechanisms) as expressed through WSDL. Only QoS/SLA parameters, if mentioned, are of particular interest for self-adaptable trust.

The functions pertaining to the mechanisms for measuring, storing, and handling trust-related information, are collectively considered into the Self-Adaptable Engine (SAE).

5.2.4 Self-Adaptability Trust Functional Architecture

The functional architecture details the high-level architecture presented above.

Figure 8. Self- Adaptable Engine Architecture

SAE's functions are grouped into functional blocks. A set of functions ensure measurement, maintenance, and exchange on trust, reputation, and credibility. A second block is responsible of associating trust level with validity period and renewal mode. The third block of functions is responsible for updating the profile lists, while the last block handles the navigation of an invocation across the existing list contents. Explanation of the mechanisms will be given later.

The SAE functions can be embedded into the WS itself, or be outside, acting as a proxy. While the last situation is related to collocation of different pieces, in our proposal there is one WS and one SAE.

In our model, a WS possesses some mechanisms to react to peer WSs. The level of adaptability we introduced is captured by the credibility index and trust levels that are automatically tuned by a given WS itself as a response of how it adapts to other WS's behavior. The SAE is modular and extendable, therefore new adaptability functions can be added.

5.2.5 Context Roaming for Updated Context Lists

The following flowchart summarizes the activities a SAE is fulfilling when an invocation is triggered.

Figure 9. Context Lists Updates

A WS invocation will simply specify a given. For the candidate list and white list, there are two processes running in parallel. For example, for the candidate list, the process “a” decides when a context is transferred to the white list; the process “b” detects anomalies and updates the black list with that context. A similar mechanism is applied to the white lists as well, through processes “c” and “d.”

For some contexts, after special computations, the black list is reached. For simplifying the model, there is no rehabilitation in our proposal. Once on frequently interacting list or black list, a manual process is used to restate the level of trust.

Usually, the first invocation is based on the context and its reputation, or simply context. The minimum record per context the candidate list must contain is <contextID, #invocations, #violations, trust_level, reputation>. Note that reputation may be imposed by some WS specifications as mandatory.

5.2.6 Update the Lists with Original Invocations

Initially all the lists are empty. The update of the lists starts with the candidate list. This is a temporary list from where a context, after reaching some maturity from the trust-level computation perspective, migrates to other lists. For the first attempt, if the invocation succeeds, the context enters the CL. Otherwise, if violation is received, or no response at all, the context enters the black list. This is a hard-method, with “no pardon” on the first attempt. A special mechanism will differentiate from violations caused by the operational mode or the transport mode. In the case that the cause is the transport mode, a second attempt is allowed.

- **Initial values:**

- [BL] = $\emptyset$
- [WL] = $\emptyset$
- [FL] = $\emptyset$
- [CL] = $\emptyset$

- **List update law:**

```

IF Invocation [<context> (< trust-level > | <r>) <QoS/SLA parameters>]
THEN context → CL
ASSOCIATE trust-level WITH invoked Trust-level
IF Invocation
RETURN violation on <SLA/QoS parameters>
OR
NO RESPONSE
THEN <context> → BL

```

After a cycle, the context enters one of the lists.

5.2.7 Invocation Handling

In general, the lists have different contexts, when an invocation is triggered.

The following flowchart exposes the path a context can follow, when invoked. For simplicity, let us consider a simple invocation with no required trust level. Note that even in this case, based on the definition of context, other QoS/SLA operational parameters are present.

After passing through the FIL and BL lists, the main computation is on the WL. If the level of trust is invoked, then an additional comparison is done; otherwise, the context enters the CL and get associated with the appropriate values of credibility, trust level, eventually reputation.

When invocation is enabled, a given context is certainly in one of the lists and the whole profile parameters are calculated and/or associated with the current values.

Figure 10. Invocation Handling

Perform Invocation is an operational activity. Its result may violate invocation parameters; it is the invocation-handling engine that updates the credibility index based on the type of violation, namely, operational and transport, in our model.

5.2.8 Associate Validity Period

For those contexts that are not in the FIL, the validity period is a vital characteristic. A special self-adaptable function associates a validity period to a trust level. There are different implementation algorithms for the renewal of the validity period. We present below a simple mechanism; if no renewal is associated, then the validity period must be in the revocation mode. When a time interval is associated, a special clock and time mechanism must automatically renew the validity period increments of its duration, as expressed below.

```
ASSOCIATE
[<context><trust_level>] WITH <validity_period>]
or
[<context><trust_level> WITH <any | [t1, t2]>]

- "any" is associated with "revocation mode"
- [t1, t2] is associated with "expiration mode"

- Renewal law in the expiration mode:
  IF  $t > t_2$  &  $\text{trust-level}(t) = \text{trust-level}(t_2)$ 
  THEN  $t'2 = t + (t_2 - t_1)$ 
```

Other more sophisticated renewal mechanisms can be used, especially grading the validity period with the trust level. Associating a larger validity period to contexts having high trust level is an example of such a mechanism.

5.2.9 Trust Anomaly Report (notifications)

The drawback of all self-adaptable systems is that in the case where no policies are prescribed by the specification, anomaly reports must be sent to high-level controller. A few of such anomalies are listed below.

- Trust anomaly report (sent to Recommenders)
 - Trust-level < threshold
 - Credibility index < threshold
 - Reputation contradicts local trust level
- Trust violation
 - Wrong update of “trust level”
 - Send “notification” to Recommenders

A type of anomaly is related to internal errors in setting current values for profile parameters. Another type of anomaly could be on wrong updates of the trust level, or in the interaction with recommenders.

6 Implementation, Validation, Evaluation

6.1 *Practical Solutions and Validation*

We will make use of UDDI's tModel and of UDDI API to specify the extensions proposed in our thesis. Mainly, the trust information will use the browse pattern, while the invocation pattern will provide means for building trust profiles. The trust profiles are updated in real-time by a Web Service. Self-adaptable trust mechanisms may ask for

SOAP extensions as well, but this should be clarified in the second term of the study.

The implementation is done in a simulation environment. Several Web Services are considered and modeled together with the supporting transport/communications networks. Only those network parameters that may affect the trust are considered (e.g., delay). From the Web Services perspective, only properties related to trust and trust management are considered (e.g., levels of trust, trust renewal lifecycle, etc.).

Testing the mechanism we proposed is done as follows. A set of Web Services are given an original validated trust levels against other Web Services, in well-defined contexts. Then, we vary both network parameters and the level and the number of trust violations of different interactions. We monitor the exchanges between the simulated Web Services as well as the changes in trust levels. We then identify the trust variation, i.e., an increased/decreased level of trust with respect to the expectations, and try to evaluate the false positive and false negatives, as well as the convergence in trust stabilization (done by the number of expectations and the timestamps they occurred).

Two more considerations for setting a system for proof of concepts are needed. First, from the interaction perspective, the distinction between operational and transport mode is essential. From the transport perspective, there are interacting trust domains. A trust domain is a sub network where the level of trust between interacting entities is “very trustworthy” and it is guaranteed.

Figure 11. Trust Domains Along an End to End Interaction

There are many transport parameters that can affect the WS interactions, such as the packet loss rate, delay, jitter, etc. The variation in some of these parameters may introduce distortions in the trust level perceived by a given WS. Our thesis is that any transport related behavior should be distinctly specified as part of the WS description.

As mentioned above, trust domains are considered “very trustworthy”. Trust violations may occur when passing from one trust domain to another, at the transport level, but this is beyond the scope of our thesis. We do not consider these distortions and assume a uniformly trusted transport environment.

Additionally, WSs are relying on other entities to exist and operate (Figure 11). In fact, a WS may differentiate between conditions required to its own environment, and conditions raised because of interactions with other WSs.

The figure below exposes the fact that there is a set of trust parameters between a WS and

its environment (called trust parameters). The model we presented covers only the trust profile among WSs. When information is passing from WS1 to WS2 through the transport infrastructure, the only way to identify the violations caused by a WS is by clear identification of violations coming from the environment from those coming from the WS behavior.

Figure 12. Environment Dependence of WSs

Therefore, we assume that a WS has a mechanism to identify and separate the two types of violation origins. As an example, when WS1 invokes WS2, it is possible that a port has a flip-flop behavior on the Env1 side and there is a service violation detected on the WS1 side. Only with the confirmation from WS2 WS1 can infer this abnormal behavior (e.g., by attaching the appropriate Syslog message ID or Simple Network Management

Protocol (SNMP) inform object identifier).

6.2 Web Services-based Application

6.3 Implementation

The implementation consists of two main parts: a module that can be used to design a network layout and parameters, and a second module that can handle a simulation over the designed layout using the specified parameters. The simulator has been developed in Java. It contains no third party libraries. In Figure 13, we can see the user interface used to construct the simulation setup.

Figure 13. Screen Shot of the Simulation Application

The software allows storing of several simulation definitions. Either of them can be loaded when a run is started. Shown in Figure 14 is a screenshot of how a simulation definition is specified.

New Simulation x

name: simulation2

Defaults

bad service %: 25

cloud delay: 100

http, ftp, smtp

excessive delay: 20

dropped packets: 20

wrong answer: 20

internal error: 20

out of resources: 20

acceptable delay: 300

show, set

Specifics

spec. cloud delay: 450

show, set

cloud3, cloud2, cloud1

ws4, ws3, ws2

bad service %: 40

excessive delay: 10

dropped packets: 20

wrong answer: 30

internal error: 10

out of resources: 30

show, set, defa...

http, ftp, smtp

use of service %: 10

show, set, default

OK, Cancel, Flush removals

Breakdown of bad service

Delay incurred while crossing a specific network cloud

Acceptable processing delay (e.g. 300 units for smtp service)

Specific bad service for a Web Service

Specific service use for each Web Service

Figure 14. Dialog Window Specifying a Simulation

In the top part, default values are specified. They are generically applied to all the elements. An example is the percentage of bad service generating bad replies. For the network clouds, a delay while crossing such clouds can be specified. The lower half of

the dialog is used to specify any values for specific elements that differ from the default values. This is how we can generate reliable and unreliable Web Services.

The part of the software that deals with the advancement of the simulation consists of a simulation engine and a database. The simulation engine deals with randomly generating requests from one Web Service to another. The events are placed in a timestamp ordered queue and are delivered to the recipients at the appropriate time. Upon receiving a requests, the recipient will generate a reply based on its simulation parameters, and will place the request in the queue.

The database is a mySQL database. It consists of four tables: *requests*, *replies*, *logs*, and *computations*. The *requests* table keeps a list of all the requests that have been sent. The *replies* table keeps a list of all the replies that have been sent. The *logs* table registers changes in the membership of Web Services to lists such as white list, frequently interacting list, blacklist, etc. At every 100 time unit intervals, a trust snapshot is taken and stored in the *computations* table. For every pair of Web Services, the trust level, credibility index, good replies count, and bad replies count are stored.

As a feature for modifying the course of a simulation during its run, a right click on the simulation panel brings up a dialog from which the congestion delay for a network cloud can be changed, as well as the reliability of a specific Web Service. This has been used in two experiments that characterize the trust level when a simulation parameter is varied. For graph plotting, MySQL Query Browser is used to export the data to an Excel

Spreadsheet, from where graphs are generated.

6.4 Results and Interpretation

We first focus on the behavior of the setup under non-varying parameters. We expect that in this case, the trust value would eventually converge. Figure 15 shows the trust level of Web Service WS1 as perceived by WS10. At the beginning, there is some instability regarding the trust level for ws1. The initial values are not always on the same profile as the subsequent values. However, the trust profile gets to reflect the true behavior of the Web Service as time progresses. (Note: Since this is a simulated setup, the exact time units are not relevant, only the relative time; this holds for figures 15 to 23)

Figure 15. Trust Level (vertical bar denotes point at which WS1 is added to WS10's white list and frequently interacting list)

The observations in the above experiment were focused around transitions and

stabilization of the system. The simulated WS system was initialized with various runtime behaviors. Some WSs would inherently provide consistently good replies. Others would consistently provide bad replies. Some WSs would provide a mixture of good and bad replies with a predefined probability.

The system simulation is initialized to default parameters: credibility index is unknown, broker input is unknown, etc. As the simulation progresses, the trust levels between WSs vary and tend to converge towards stable values.

We have monitored such conversions for several setups. In the figures above we present the behavior of the trust level in the case of a mostly good Web Service (>95% good replies) and mostly bad WS (< 20% good replies). We observe that under constant conditions, the trust level converges quickly and remains stable. As the portrayed Web Service gains in trust, other web services eventually place it on their white list and, if enough interactions take place, on their frequently interacting list.

Another experiment was done to see the effect of congestion on the perceived trust of a given Web Service. In this case, we monitor the internal data that WS6 gathers regarding WS7. The percentage of bad replies sent by WS7 is 20%. Under the thresholds set in our functions, such a percentage would secure WS7 a spot on WS6's white list, but not on the frequently interacting list. While the simulation proceeds, at time 1032, WS7 is indeed placed on WS6's white list. At this point, a congestion factor is applied to a network

cloud on the path between WS6 and WS7, namely C2. An excessive delay was introduced at that point.

In Figure 16, we note a steep increase in the number of bad replies. At time 2487, the WS7 is deemed unacceptable and placed on WS6's black list. From this point on, we note that both the number of good replies and bad replies level off due to the complete avoidance that WS6 has for WS7 in terms of requesting service, as seen in Figure 16 and Figure 18. Within short periods of being placed on the blacklist, both the credibility index and the trust level drop to minimal values, as shown in Figure 17 and Figure 19.

Figure 16. Count of Bad Replies Received by WS6 from WS7

Figure 17. Credibility Index of WS6 for WS7

Figure 18. Count of Good Replies Received by WS6 from WS7

Figure 19. Trust of WS6 for WS7

A different experiment that was done with the simulation was the change in the reliability of a Web Service midway through the simulation. The focus for this experiment is on Web Service WS12 and WS1. The activity during this experiment is captured in figures 20 to 23. WS1 is a Web Service that is very reliable at the beginning of the simulation. 99% of the replies by WS1 are good replies. It is expected that from WS12's point of view, WS1 is quickly placed on the white list (at time 1321) and then placed on the frequently interacting list (at time 1858).

At time 1500, the reliability of WS1 has been changed to return 80% bad replies. The consequence of this change gradually takes effect. At time 2096, WS1 is removed from the frequently interacting list, and at time 2367 it is removed from the white list.

As the simulation proceeds, the trust level and credibility index decrease. At time 2645,

the statistics reach the point where the WS1 is placed on the black list. All the interactions of WS12 with WS1 are suspended.

Figure 20. Count of Bad Replies Received by WS12 from WS1

Figure 21. Credibility Index of WS12 for WS1

Figure 22. Count of Good Replies Received by WS12 from WS1

Figure 23. Trust of WS12 for WS1

In the two experiments above, we obtained similar results: a reliable Web Service is ranked as trustable while the proportion of bad replies it produces is small. Due to

internal reasons (decrease in reliability) or external reasons (network congestion), the Web Service is seen as not providing good replies any more. In such cases the Web Service is seen as less and less trustworthy. It eventually ends up on black lists, at which point its services are no longer called upon.

Within the scope of this thesis, we do not differentiate between losses of reliability due to internal as opposed to external factors. We have used network parameters (large delay) to show how a situation of false negatives can occur. A mechanism to detect such a case can be composed of Management Information Base (MIB) queries in order to extract true network status for the nodes along the path between the two Web Services.

7 Conclusion and Future Work

The thesis presented a new approach for rendering Web Services self-adaptable. The model proposed and the mechanism is not overwhelming the transport/communications. We introduced the concept of *trust profile*, proposed a *self-adaptable engine* associated with a Web Service and shown their implementation (as a proof of concept). Our proposal provides Web Services more autonomy with respect to their mutual trust.

We suggested a mechanism to build, negotiate, and self-adapt the mutual trust of the interacting WSs. On the client side, we will enforce a start with the less critical applications supported by the client-WS (use-context), while on the provider side, we will

consider various levels of trust. The levels of trust are induced based on the statistical interacting on (i) trust in timely delivery and (ii) trust on SLA. We introduce the notion of credibility index to grade the various granularities of trust levels.

We conducted experiments to show that under normal circumstances, the trust levels converge. They will remain stable as long as the parameters of the system are not changed. Once the simulation parameters are changed during the simulation, the trust profile changes accordingly. These parameters can be network related (e.g. delay) or Web Service related (e.g. reliability). Within the scope of this thesis, we do not differentiate between the two sources that affect trust levels.

7.1 Future Work Related to the Thesis Experimentation

Further experiments can be conducted to observe the change in trust level when simulation parameters are altered only for a short period, only to be later restored. This would show how much variation the system can absorb before changes are observed in the trust levels.

Systems where there are constant variations have not been simulated. Constant changes in trust level are expected under such cases. Further study can refine the trust model to account for periodic variations.

7.2 Future Work Related to the Concepts Proposed in the Thesis

In the aim of lowering the activity devoted by a WS to the management of the trust profile, we identified a few enhancements.

- (i) Context aggregation around the same service offered by multiple WSs

E.g., context_1 <WS1, Service_ID-x>

context_2 <WS2, Service_ID_x>

.....

context_n <WS-n, Service_ID-x>

could be aggregated as

context_1-n <{WS1, ..., WS-n}, Service_x

- (ii) Context aggregation around the same WS offering multiple services

E.g., Similar approach as for (i), but on the WS side of the context.

- (iii) Compression XML-based SOAP event

Nine OASIS XACML access control policies of various XACML SOAP sizes are compressed using some well known and some emerging techniques in (Qing & Adams, 2005).

The first two are targeting the optimization of working within and across the lists. The third is considering optimization at event structure, complemented this time with specific information pertaining to operational trust.

7.3 Other Ongoing Work on Complementary Aspects

Two other complementary topics but related to trust are now under scrutiny in two

international fora.

(i) Global Grid Forum (GGF13) was on March 14-16 in Seoul.

A Birds of a Feather (BoF) session entitled "Trusted Computing for Grid Security", is intending to start a new working GGF group.

(ii) A security framework combining access control and trust management for mobile e-commerce applications is scheduled to be presented at the Network Challenges on Access Control (NCAC), Ottawa, April 24, 2005 (OCRI, 2005).

Finally, Methodologies to write trusted application are starting to be proposed. E.g., "Steps For Writing Trusted Applications," by Craig Werhan of Wave Systems within the Trusted Computing Group (Werhan, 2005).

In conclusion, the novelty of the requirements and the challenges of distributing computation power between operational activities and trust maintenance put our proposal and its preliminary results among pioneering work in the field for the self-adaptable view.

References

- Abdul-Rahman, A., Hailes, S. (2000). Supporting trust in virtual communities. *Proceedings of the 33rd Hawaii International Conference on Systems Sciences*. IEEE Computer Society, 2000.
- Anderson, S, Bohren, J., Boubez, T., Chanliau, M., Della-Libera, G., Dixon, B., et al. (2004, May). Web Services Trust Language (WS-Trust). Retrieved July 16, 2004 from Microsoft Development Network. (MSDN). <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/WS-Trust.asp>
- Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., et al. (2004). Web Services Policy Framework (WS-Policy). Retrieved December 12th, 2004 from MSDN. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policy.asp>
- Bellwood, T., Clement, L., & von Riegen, C. (2003, October 14.) UDDI Version 3.0.1. Retrieved July 27, 2004. from Organization for the Advancement of Structured Information Standards (OASIS). http://www.uddi.org/pubs/uddi_v3.htm
- Bertino, E., Ferrari, E., & Squicciarini, A. (2003). X-TNL: An XML-based Language for Trust Negotiations. *Fourth IEEE Intl Workshop on "Policies for Distributed systems and Networks"*, IEEE Computer Society, 2003, pp. 81-85.
- Box, D., Curbera, F., Bajaj, S., Chappell, D., Daniels, G., Hallam-Baker, P., et al. (2003, June 2). Web Services Policy Attachment (WS-PolicyAttachment). Retrieved July 15, 2004 from MSDN. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnglobspec/html/ws-policyattachment.asp>
- Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., & Yergeau, F. (2004, February). Extensible Markup Language (XML) 1.0 (Third Edition). *World Wide Web Consortium (W3C)*. Retrieved October 5, 2004 from <http://www.w3.org/TR/2004/REC-xml-20040204/>
- Bussard, L., Molva, R., & Roudier, Y. (2004). History-Based Signature or How to Trust Anonymous Documents. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 78-91.
- Butler, M., Leuschel, M., Lo Presti, S., & Turner, P. (2004). The Use of Formal Methods in the Analysis of Trust. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 333-339.

- Cantor, S., Hirsch, F., Kemp, J., Philpott, R., & Maler, E. (2004, December 14). Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0. Retrieved December 10, 2004 from OASIS. <http://www.oasisopen.org/committees/download.php/10618/sstc-saml-bindings-2.0-cd-03.pdf>
- Cantor, S., Kemp, J., & Maler, E. (2004). Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0. Retrieved August 1, 2004 from OASIS. <http://www.oasisopen.org/committees/download.php/7738/sstc-saml-core-2.0-draft-17-diff.pdf>
- Castelfranchi, C. (2004). Trust Mediation in Knowledge Management and Sharing. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 305-318.
- Cerami, E. (2002). *Web Services Essentials*. Sebastopol, CA: O'Reilly & Associates, Inc.
- Chappell, D. (2002, March). *Java Web Services*. Sebastopol, CA: O'Reilly & Associates, Inc.
- Dacosta, M. C., Obrst, L. J., & Smith, K. T. (2003). *The Semantic Web*. Indianapolis, Indiana: Wiley Publishing Inc.
- Davies, J., Fensel, D., & Van Harmelen, F. (2003). *Towards the Semantic Web*. West Sussex: John Wiley & Son Ltd.
- Demolombe, R. (2004). Reasoning About Trust: A Formal Logical Framework. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 291-303
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., et al. (1999, June). Hypertext Transfer Protocol – HTTP/1.1. Retrieved September 28, 2004 from Internet Engineering Task Force (IETF) <http://www.ietf.org/rfc/rfc2616.txt>
- Freier, A. O., Karlton, P., & Kocher, P. C. (1996). The SSL Protocol Version 3.0. Retrieved November 9, 2004 from Netscape. <http://home.netscape.com/eng/ssl3/index.html>
- Giorgini, P., Massacci, F., Mylopoulos, J., & Zannone, N. (2004). Requirements Engineering Meets Trust Management. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 176-190.

Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., & Nielsen, H. F. (2003, June 24). Simple Object Access Protocol. Retrieved July 20, 2004 from W3C. <http://www.w3.org/TR/soap12-part1/>

Hallam-Baker, P. (2004, April 4). XML Key Management Specification (XKMS 2.0). Retrieved August 2, 2004 from W3C. <http://www.w3.org/TR/2004/CR-xkms2-20040405/>

Harold, E. R. (2001). *XML Bible*. Foster City: IDG Books Worldwide, Inc.

HTTPS. (2004, September 22). Retrieved September 28, 2004 from Wikipedia <http://en.wikipedia.org/wiki/HTTPS>

Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., et al. (2004, December 14). Profiles for the OASIS Security Assertion Markup Language (SAML)V2.0. Retrieved December 10th, 2004 from OASIS. <http://www.oasis-open.org/committees/download.php/10636/sstc-saml-profiles-2.0-cd-03.pdf>

Hughes, J. & Maler, E. (2004, July 22). Security Assertion Markup Language (SAML) 2.0 Technical Overview. Retrieved December 8th, 2004 from OASIS. <http://www.oasis-open.org/committees/download.php/7874/sstc-saml-tech-overview-2.0-draft-01.pdf>

Kaufman, C., Perlman, R., & Speciner, M. (2002). *Network Security. Private Communication in a Public World*. New Jersey: Prentice Hall PTR.

Linn, J. (2004, February). Trust Models Guidelines. Retrieved December 15, 2004 from OASIS. <http://www.oasis-open.org/committees/download.php/6158/sstc-saml-trustmodels-2.0-draft-01.pdf>

MacVittie, D. (2004). *Security Pipeline*. WS-Security Spec Nearing Completion. Retrieved December 8, 2004 from <http://www.securitypipeline.com/specwatch/ws-security.jhtml>

MacVittie, L. (2003). *Network Computing*. Speaking SAML. Retrieved December 8, 2004 from <http://www.nwc.com/showitem.jhtml?docid=1422ws1>

Madsen, P. (2004, May). Trust Networks in a Web Services World. Retrieved December 14, 2004 from <http://webservices.xml.com/pub/a/ws/2004/05/26/trust.html>

Manola, F., & Miller, E. (2004, February 10). RDF Primer. Retrieved October 5, 2004 from W3C. <http://www.w3.org/TR/rdf-primer/>

- Massa, P., & Bhattacharjee, B. (2004). Using Trust in Recommender Systems: An Experimental Analysis. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 221-235.
- Nadalin, A., Kaler, C., Hallam-Baker, P., & Monzillo, R. (2004). "Web Services Security: SOAP Message Security 1.0(WS-Security 2004)". Retrieved November 30th, 2004 from OASIS. <http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-soap-message-security-1.0.pdf>
- Negm, W. (2004, January). *Network Computing*. Bringing Balance to Web Services Solving the Trust & Threat Equation. Retrieved November 15, 2004 from http://techlibrary.networkcomputing.com/detail/RES/1075225294_388.html
- Newcomer, E. (2002). *Understanding Web Services*. Indianapolis, IN: Independent Technology Guides.
- Nolan, P. (2004). Understand WS-Policy processing: Explore Intersection, Merge, and Normalization in WS-Policy. Retrieved December 12, 2004 from IBM. <http://www.106.ibm.com/developerworks/webservices/library/ws-policy.html>
- OCRI. (2005). New Challenges for Access Control Workshop. Retrieved March 1st, 2005 from <http://www.ocri.ca/accesscontrol/>
- Pavlov, E., Rosenschein, J. S., & Topol, Z. (2004). Supporting Privacy in Decentralized Additive Reputation Systems. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 108-119.
- Powers, S. (2003, July). *Practical RDF*. Sebastopol, CA: O'Reilly & Associates, Inc.
- Qing, X., Adams, C., (2005). A comparison of compression techniques for XML-based security policies in mobile computing environments, New Challenges on Access Control (NCAC); <http://www.ocri.ca/accesscontrol>
- Ruth, P., Xu, D., Bhargava, B., & Regnier, F. (2004). E-notebook Middleware for Accountability and Reputation Based Trust in Distributed Data Sharing Communities. *Trust Management, (Proceedings of the 2nd International Conference, iTrust 2004)*. Oxford, UK, LNCS 2995, Springer, pp. 161-175.
- Seely, S. (2002). Understanding WS-Security. Retrieved December 5, 2004 from MSDN. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/understw.asp>

Shannon , R. (2004). What is HTML? Retrieved September 29, 2004 from
<http://www.yourhtmlsource.com/starthere/whatishtml.html>

Snell, J., Tidwell, D., & Kulchenko, P.(2002). *Programming Web Services with SOAP*.
Sebastopol, CA: O'Reilly & Associates, Inc.

Werhan, C. (2005, March 8). Steps for writing trusted applications. Retrieved March 10,
2005 from Computerworld
<http://www.computerworld.com/newsletter/0,4902,100216,00.html?nlid=APP>