MASTER'S THESIS

Trust-based Security in Web Services

Information Systems Institute
Distributed Systems Group
Technical University of Vienna

supervised by
Ao.Univ.Prof. Mag. Dr. Schahram Dustdar
and
Univ.Ass. Dipl.-Ing. Dr.techn. Clemens Kerer

by

Christian Platzer
Samerweg 35
6060 Hall in Tirol

Vienna, May 2nd 2004 ______________________

Für meine Eltern

Danksagung

Für sein persönliches und fachliches Engagement, sowie für die ausgezeichnete Betreuung während der Erstellung dieser Arbeit, gilt mein persönlicher Dank Clemens Kerer.

Ganz besonderer Dank gilt meiner Familie, die mich auf meinem Weg immer unterstützt hat, und mir in schwierigen Zeiten den nötigen Rückhalt gab. Nur dadurch war es mir möglich, die richtigen Entscheidungen zu treffen und auch dazu zu stehen.

Kurzfassung

Im Zuge dieser Arbeit wurde *SimOffice*, ein auf Vertrauen basierendes System zur dynamischen Verwaltung von Zugriffsrechten für Web services entwickelt. Web services gewinnen als neue Technologie zur Entwicklung von verteilten, service-orientierten Applikationen zunehmend an Bedeutung. Mit der wachsenden Zahl dieser Dienste - speziell im firmeninternen Bereich - steigen auch die Kosten zur Wartung der dazugehörigen Benutzerkonten rapide an.

Die Grundidee hinter dieser Arbeit war, ein sich selbst verwaltendes System zu schaffen, in dem Zugriffsrechte automatisch vergeben werden. Zu diesem Zweck wurde ein Algorithmus entwickelt, der versucht, die menschliche Art der Urteilsfindung zu imitieren. Dieser auf Vertrauen basierte Ansatz wurde schließlich in einer Fallstudie implementiert, um zu zeigen, dass eine deartige Form der Verwaltung von Benutzerkonten auch in einem rein auf Web services basierten System zu einer deutlichen Aufwandsminderung führt.

Weiters wurden verschiedene Möglichkeiten diskutiert, um diesen Ansatz in weiterer Folge auch auf Systeme ausweiten zu können, denen andere Technologien zugrunde liegen.

Abstract

In the course of this thesis *SimOffice*, an environment with trust-based access control for Web services, was developed.
Web services are gaining more and more importance as a technology to develop distributed, service-oriented applications. With the growing number of services, especially within corporate networks, the expense to maintain user accounts for every single service grows tremendously.
The basic idea behind this concept was to create a self maintaining system, where access restrictions are set automatically. For this purpose an algorithm was developed, which mimics the way humans pronounce a judgement about another party. This trust-based approach was finally implemented in the *SimOffice* case study to demonstrate that this way to maintain user accounts eventually leads to a reduction of effort, even in a system completely based on Web services.
Furthermore, some ways to extend this approach to systems based on other technologies, were discussed.

Contents

List of Figures

List of Tables

1 Introduction

Today more and more companies employ administrators to maintain their network, adjust user privileges and keep everything up and running. With the growing size of these companies, the amount of work put into these activities grows enormously until a point is reached when a single person can no longer overlook this meshwork of permissions, exceptions and policies. Apart from the resulting overhead to add, delete or adjust permissions, this may also results in some severe security leaks. Wouldn't it be great if a computer system could decide by itself whether to grant access rights to a requesting user or not?

Another trend arising in the sector of software development is to reuse functionality in the form of Web services. Web services are a platform-independent way to establish communication between two applications connected through a network. Here the maintenance of policies is even more expensive. A distributed application can use many different functions and every single one needs to be secured in a proper way. It would be a hard task for a human to adjust all security levels properly because new functions need to be added while others are obsolete and not used any more.

SimOffice is a security-aware network environment that attacks these problems with a combination of common security standards and a completely new method to judge a user's intentions. The goal is to create an independent system where decisions are made based on trustworthiness. This way the 'thinking' is done by the system while a human supervisor can still influence the judgement in some special cases. The assessment itself tries to imitate human behavior and human trust. Of course this concept is not limited to Web services but can be applied to almost every security-sensitive area in computer networks.

SimOffice will simulate a possible environment within an office where even the coffee maker is accessible through a Web service. And what is a harder punishment for bad behavior than a denial of coffee!

1.1 Motivation

Today, the majority of software companies are implementing tools based around the new standards for Web services [41, 45, 46]. Considering the fast development and the strong commitment of several important software companies like IBM and Microsoft, a wide range of ready-to-use services throughout the entire Web can be expected soon. Google for example, one of the most popular search engines on the web, already offers a Web service for web queries [16]. An implementation based on the provided API is straight forward and requires but basic programming skills.
But Web services available on the Web will not remain the only application for these standards. Modularity and language-independency paved the way for combined applications, using basic Web services even in closed environments like corporate networks. Thus it is quite possible to implement a company's applications as Web services and make them accessible through published descriptions (see WSDL-Files in 2.5.1). The advantages of such an architecture are obvious: Services are completely independent of implementation or operating system and useable from everywhere within the entire network. Developers are encouraged to use the provided functionality without further knowledge of the involved code. Each Web service is registered at a centralized spot which makes service discovery much easier.
Another trend in the development of computer systems is to adapt the human way of thinking, to address security aspects like access rights or judgment of the other parties intent. Combining both, Web services and an intuitive way of access management is the main motivation for this thesis.

Let's take a look at a possible scenario to offer a better illustration:
Al is employed as a programmer at Bits-and-Bytes Industries. His task is to write a program which calculates the average income of his team's employees. Al knows an employee who works in the personnel department and he uses a program to read and write to and from the personnel database. Un-

fortunately this program was written in Visual C++, whereas he uses Java for development. He has no knowledge how to use C++ code in Java, so he decides to create his own database connection and rewrites the code for the same functionality.
Now we take the same scenario at Makrohard Ldt., where Web services are used to increase efficiency. The moment Al realizes that the code he desires may have been written before, he looks up the service description on the companies central server, downloads the WSDL-File and includes it in his application. All he has to do afterwards is load every employee via the Web service and calculate the mean value. This way he saved time and money for the company.
And thats what Web services are designed for after all. To increase efficiency and simplify software development. But there is a major drawback that needs to be addressed as well. To continue the above example we assume Bob worked at Makrohard Ldt. too but he was fired today because he refused to design his programs as Web services. He decides to write one last program and uses the same Web service Al used but he has something far more evil in mind. Instead of reading an employee's data, he deletes every single employee from the company's database. He can do this, because the service does not only provide functions to read records from the database but also to write and delete them.
This example is a little oversubtle of course. In reality access to Web services will be restricted depending on how important they are. This can be done by an administrator who sets access rights for every user account for example. But with increasing modularity the number of Web services will simultaneously grow and maintenance will become almost impossible. So what really is needed is some sort of self-organizing mechanism for access restriction. Section 1.2 describes arising problems when it comes to maintain security in Web services. It also outlines my approach towards a potential solution.
To continue the scenario above, Bob would have been unable to use the men-

tioned Web service in a self-organizing system, because the moment he was fired, he would have been classified as 'untrustworthy' by a central entity and his access rights limited to a innocuous level. Had he still tried to invoke the Web service it had simply ignored the request or even sent a warning to the system administrator. Either way Bob would have been unable to cause any damage to the system.

1.2 Problem Definition

The main problem is to create a reasonable combination of accessibility and access restrictions.

A federation of Web services must meet some requirements to retain a useful nature: First, access rights to a single Web service must not be fixed if the total number of available services grows too large to maintain it manually. A dynamic technique to adjust access levels automatically is needed. Nevertheless there must be a facility to change access rights manually too. Otherwise an administrator would be unable to customize access restrictions if it is necessary.
On the other hand the whole system has to be accessible to every authorized user. What's a perfect safe system worth if nobody can use it?
To meet the requirements above a mechanism is utilized that most people use everyday: **Trust**.
This approach intends to mimic the decisions taken by humans when it comes to judging wether an action of an opposing party is beneficial or not. The goal is to create a federation of Web services where security is assured both by common safety techniques for transport and privacy and a trust-based approach for access control.
Creating such a system is not an easy task because systems regulated by trust-based mechanisms tend to be unstable in long-term view. This would result in a complete distrust or the counterpart, a blind trust situation. Neither of this two conditions is suitable for a computer system whose main task is to provide a public service. This thesis will treat the problem of establishing trust relationships and evaluate the capabilities of trust-based access control.

1.3 Structure of this Thesis

Chapter 2 gives an overview of the state of the art for Web services and a description of the underlying technologies. Furthermore this chapter explains how to handle Web services and introduces the fundamental standards such as WSDL, XML and SOAP.

Chapter 3 keeps track of related work and papers this thesis is based on. It concentrates on trust and methods for trust management as well as punishment and other principles to keep an already established trust-based environment working.

Chapter 4 discusses the design of the trust-aware network environment *SimOffice* and gives insights into the architectural and design decisions made during the development of the SimOffice case study.

Chapter 5 focuses on the implementation, describes the tools used and shows the problems encountered during the development process.

Chapter 6 presents the SimOffice case study in detail. It demonstrates the finished implementation, provides an overview of the main features, and offers a walk-through for several typical scenarios.

In Chapter 7 an evaluation of the work presented in this thesis and ideas for further research are given.

Chapter 8 summarizes the contribution of this work and rounds off the thesis.

2 State Of The Art Review

This section introduces Web service technologies and explains how to invoke, publish and provide Web services in a distributed environment. Section 2.7 also gives a short overview of trust-related projects and their significance for the development of behavior-aware computer systems.

2.1 The Notion of the Web Service

In [39] the definition of a Web service is given as: "any process that can be integrated into external systems through valid XML documents over Internet protocols". This definition outlines the general idea Web services are built for.

Unlike services in general, Web services are based on specifications for data transfer, method invocation and publishing. This is often misunderstood and when a Web service is mentioned it sometimes refers to a general service provided on the Web, like the weather forecast on a Web page for example. The weather forecast is a service and provides its functionality for a variety of users but unless it comprises an interface to communicate with other applications via SOAP (see Section 2.4) it is no Web service by definition.

Web services can be seen as software components with an interface to communicate with other software components. They have a certain functionality that is available through a special kind of Remote Procedure Call. In fact they even evolved from traditional Remote Procedure Calls. The difference lies in the interface and the method for transportation. Furthermore Web services can not be viewed or used with an ordinary browser. They require a unified form of messaging embedded in a XML document. This communication architecture contains three subcomponents.

- **Consumer**: This denotes the entity utilizing the Web service. This is another application in most cases.

- **Transport**: It defines the means for the communication the Consumer uses while interacting with a service.
- **Provider**: The service provider.

In order to keep the whole system truly platform-independent, transport in both direction uses XML. This includes the description of an operation to execute and the data payload as well. Although transportation is not restricted to a specific protocol or method, HTTP became the most popular way to pass on XML documents between Web services. The following section will start with the first step in our course to understand Web services: *Transportation.*

2.2 HTTP

Found everywhere on the Internet, HTTP (**H**yper**T**ext **T**ransfer **P**rotocol) is a ubiquitous protocol for data connections between Web browsers and servers.
This protocol is the current standard for transferring HTML documents, although it is designed to be extensible to almost any document format like XML for example. HTTP Version 1.1 is documented in RFC 2068 [34].
It operates over TCP connections, usually to port 80, though any other port can be used. After a successful connection, the client transmits a request message to the server, which sends a reply message back.
The simplest HTTP message is "GET url", to which the server replies by sending the named document. If the document doesn't exist, the server may send an HTML-encoded message stating this. This form of communication represents a typical request/response mechanism. A client sends a request for a specific document to the server and waits for a response. If the server does not respond with the requested document it is up to the client to wait for the timeout and request the same document again. This loosely coupled type of communication is very common in client-server architectures.

In addition to GET requests, clients can also send HEAD and POST requests, of which POSTs are the most important. POSTs are used for HTML forms and other operations that require the client to transmit a block of data to the server. After sending the header and the blank line, the client transmits the data.
This way Web services utilize the HTTP protocol to transmit both Data payload and service request to a Web service. Now it is time to explain how the transmitted data looks like.

2.3 XML

XML is an abbreviation for Extensible Markup Language [44]. It is designed to describe data and improve the functionality of the Web by providing more flexible and adaptable ways of information representation. It is called extensible because its format is not fixed like HTML. Instead, XML is a meta-language which lets you design your own customized markup languages. A markup is a mechanism to specify structures within a document, whereas the way to add markup to a document is defined by the XML specification. But unlike HTML, XML does not specify semantics or a set of tags. There is no prescribed method for rendering XML documents, so semantics will be defined by the application using it or by style sheets.

The following example will show the structure of an XML document and how data is represented:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note noteID="1">
    <to>Bob</to>
    <from>Al</from>
    <heading>Our dilemma</heading>
    <body>Don’t dare to squeal on me man!</body>
</note>
```

This basic XML document starts with the XML declaration in the first line. It defines the XML version and the used character encoding. In this case the document conforms to the 1.0 specification of XML and uses the ISO-8859-1 (Latin-1/West European) character set. It is important to specify the character set to avoid misinterpretation of the provided data.
The next line describes the root element of the document. Elements are one way to store data in an XML document. The following 4 lines describe the child elements of root (to, from, heading and body). By looking at the elements it is easy to see that the XML document represents a message. The

last line finally describes the end of the root element, completing the note from Al to Bob. Along with the root element in the second line comes an attribute called noteID. Attributes are another way to store data and used to provide additional information about elements, also called meta-data. In this case it may be used to count the messages sent from Al to Bob.
A list of legal elements that defines the document structure is the Document Type Definition (DTD). A document with correct XML syntax is called "Well Formed" while a "Valid" XML document also conforms to a DTD.

More and more applications make use of XML to store information because of its benefits. Some of them are:

- The structure is well-defined and can be passed between different computer systems which would otherwise be unable to communicate.
- Data payload is encapsuled in tags and therefore readable by human viewers.
- Due to their textual nature, XML-Files are platform-independent.

These advantages made XML the perfect format to communicate between Web services. To ensure a platform and language independent use for every Web service, SOAP was developed. It is an XML application with defined elements and a predefined structure. The following section will treat SOAP in detail.

2.4 SOAP

SOAP, the Simple Object Access Protocol [46] was developed to enable a communication between Web services. It was designed as a lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP is an extensible, text-based framework for enabling communication between diverse parties that have no prior knowledge of each other [8]. This is the requirement a transport protocol for Web services has to fulfill. SOAP specifies a mechanism to perform remote procedure calls and therefore removes the requirement that two systems must run on the same platform or be written in the same programming language. SOAP also defines data encoding rules, called base level encodings or *Section 5* encodings. It is important to note that these *Section 5* encodings are not mandatory in any way, so clients and servers are free to use different conventions for encoding data as long as they agree on format.

All this is done in the context of a standardized message format. The primary part of this message has a MIME type of *text/xml* and contains the SOAP envelope which is an XML document.

The envelope consists of a an optional header which may target the nodes that perform intermediate processing, and a mandatory body which is intended for the final recipient of the message. This way a firewall can be adjusted to filter SOAP Messages with an inappropriate header for example. The Header may also hold digital signatures for a request contained in the body.

The body contains the serialized payload. For a request this is the method argument where the surrounding XML tag must have the same name as the called method. The response body contains the return value if it exists.

Data types are not delineated in the SOAP envelope explicitly so the type of a result parameter can not be discovered just by looking at the SOAP message. Client applications define data types either generically through *Section 5* encodings, or privately via agreed-upon server contracts.

The anatomy of a SOAP Envelope looks like this:

```
<SOAP-ENV: Envelope xmlns:
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
        <t:Transaction-ID xmlns:t="some-URI">
                552511951722
        </t:Transaction-ID>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
        <m:RemoteFunction xmlns:m="some-URI">
        <Parameter1>123</Parameter1>
        </m: RemoteFunction>
      </SOAP-ENV:Body>
</SOAP-Envelope>
```

In this example, the header contains some additional information enclosed by the Transaction-ID tag. This ID can be processed by any node before the final service node to ensure the request's correctness for example.
The body contains but one method call in the request. The called method's name is *RemoteFunction* whereas the methods parameter *Parameter1* is 123. The parameters type may be of integer type but could be a String as well. The client application must decide how to handle it.
SOAP messages are fundamentally one-way transmissions from a sender to a receiver, but they are often combined to implement a request/response mechanism.

Summing up, SOAP is an XML-based protocol for sending messages and making remote procedure calls in a distributed environment. Using SOAP, data can be serialized without regard to any transport protocol, although HTTP is typically the protocol of choice.

2.5 Publishing and Finding Web Services

2.5.1 WSDL

With SOAP, a communication between Web services is possible and structured and each participant knows how to send or receive the corresponding SOAP Message. The final step to complete the communication architecture of Web services is to define how to access a service once it is implemented. This is where the Web Service Description Language (WSDL, [45]) steps in. WSDL describes services as collections of network endpoints, or ports [45]. Again it is an XML document with a defined grammar where the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. WSDL documents use the following elements to describe a Web service:

- **Types:** A container for data type definitions. In contrast to SOAP, WSDL can define types using some type system (such as XSD).
- **Message:** A definition of the data being passed in a single RPC.
- **Operation:** A description of an action (method) supported by the service.
- **Port Type:** A set of operations supported by one or more endpoints.
- **Binding:** A concrete data format specification for a particular port type.
- **Port:** A single endpoint defined as a combination of a binding and the network address where it can be found.
- **Service:** A collection of related endpoints.

Now that a Web service can be described completely, the only remaining problem is how a potential user can find the corresponding description (WSDL document). The following section deals with this last problem.

2.5.2 UDDI

Universal Description Discovery and Integration (UDDI) provides a method for publishing and finding service descriptions [41]. It is a directory where Web services can be registered and assigned to a service provider, therefore forming a structure that resembles a yellow pages directory. Any user can browse the directory to search for a desired service and download the description in case of a match. UDDI is an industry initiative (ARIBA, IBM and Microsoft) to enable businesses to quickly and dynamically find and transact with each other. Searching a UDDI registry can be done by any UDDI browser or by the registry's Web interface if present. Microsoft for example, provides both a Web interface [32] and the original UDDI registry [31] to search for an entry. A query with 'weather' as the search string returns about 10 results at the Microsoft UDDI registry. The corresponding WSDL File can be found in the overview document if it was specified.
The UDDI registry is implemented as multiple peer nodes where the registration across all nodes is replicated by node operators. Again, the data structure of the registry is XML yet a WSDL document cannot be published without additional precautions. The structure of a UDDI registry differs from a WSDL file because UDDI also supports business and service information. As a result, UDDI has no direct support for WSDL or any other service description mechanism. After adjusting the WSDL service description the UDDI registry contains the services general description, port bindings and a reference to the original WSDL File as tModels [42]. Figure 1 shows how this mapping works in detail.

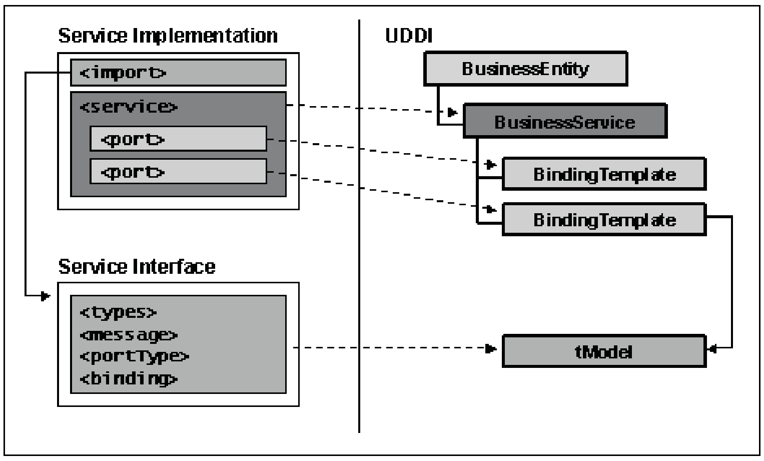


Figure 1: WSDL to UDDI mapping

Every service interface of a WSDL file is published as a tModel in the UDDI registry. A service interface includes types, message, portType and binding. Some elements of the business service are constructed using information from the WSDL service description. The service name for example, is used as the name of the UDDI's business service entry. A complete description how to map WSDL files to UDDI entities can be found at the IBM developers page [19].

Publishing the service description to a UDDI directory was the last step in the process to create a Web service.

2.6 Using Web Services

To finally use a Web service, several steps have to be performed. Figure 2 shows the order of the events, followed by a description of how to execute each step.

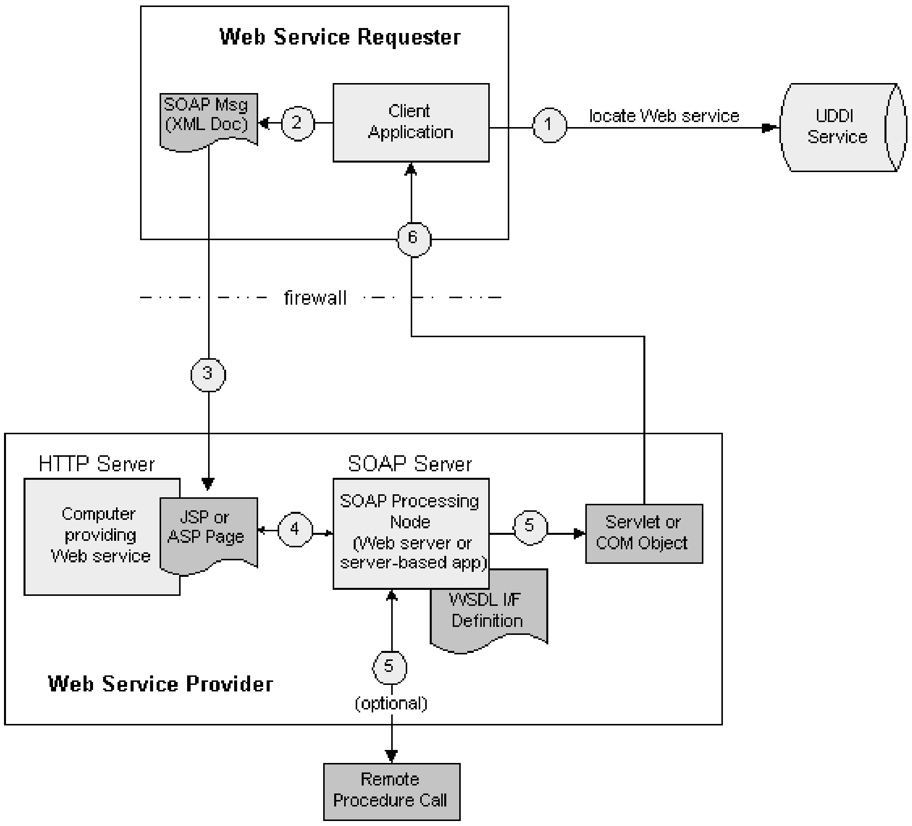


Figure 2: Invoking a Web service

1. Locating the Web service. This can either be done by browsing a public UDDI registry or by means of an existing WSDL document. It is possible to build a private UDDI registry as well. Private registries are easier to maintain due to their size but it can be hard to discover the UDDI registry's position. Sometimes, a company's main Web page is linked to WSDL documents, too (see [16]).

2. Creating the SOAP Message. This is done by the development tool in most cases. Tools like Weblogic Workshop from BEA or Web service Development Kit from Microsoft will create valid SOAP messages for the methods described in the WSDL document or UDDI registry.

3. Transmission. Another advantage of message transport via HTTP is the service providers firewall setting. If the firewall permits Port 80 (HTTP POST/GET) connections, a SOAP message is able to pass through as well. If the firewall is unable to filter and process SOAP requests on the other hand, it leaves the system vulnerable to attackers who use the Web service's functionality for a potential attack.

4. Parsing the SOAP message is done by the provider's Application Server. The parser decides if the request is valid and decides which procedure to call.

5. Processing. The service provider calls all necessary procedures, or even other Web services, to complete the requested task.

6. Return the result. The result is wrapped in a SOAP reply and returned to the requestor where the client application can parse the message and evaluate the included data.

2.7 Trust-related Projects

A state of the art analysis on trust-related topics is not easy to perform because there are no existing standards for this issue at the moment. There are some related standards worth mentioning though.
One of this standards is the Web Services Trust Language or WS-Trust [20]. This title is a bit confusing since the standard does not describe how to manage trust in a federation of Web services. It rather describes a mechanism to obtain a Security Token and establish a trusted relation among two parties. This relation is of no dynamic nature but of a static one. With other words, WS-Trust is able to handle complete trust or complete distrust but nothing in between. A requestor sends a request, and if the policy permits and the recipient's requirements are met, then the requestor receives a security token response. Once a Security Token is successfully obtained and used in a request, the SOAP Messages of the issuing party are trusted. A security token can be requested by a SOAP action that is defined as:

```
http://schemas.xmlsoap.org/security/RequestSecurityToken
```

while the SOAP action for security token responses on a port is defined as:

```
http://schemas.xmlsoap.org/security/RequestSecurityTokenResponse
```

The standard also allows a hierarchical structure to fetch the token for a specific user as long as the trust chain eventually leads to a trusted root. This structure represents a very special way of trust propagation which is an important issue in this thesis. However, WS-Trust is an insufficient tool to create a trust-aware environment if staggered levels of trust are required.
The second standard worth mentioning is the Web services Federation Language (WS-Federation, [21]). It defines mechanisms to allow different security realms to federate using different or like mechanisms by allowing and brokering trust of identities, attributes and authentication between participating Web services. In a federated single sign on, a user authenticates at a

portal within the federation of Web services. Once he is successfully authenticated, he can use other portals without authentication because the portals are linked through their identity providers. This way his user data is passed on from one portal to another. Managing identities is another important topic treated later in this thesis.
An example for a project dealing with role assignment based on a user's behavior is the TERM Server Architecture, done at Purdue University's Center for Education and Research in Information Assurance and Security (CERIAS, [7]). This project implements access control based on direct and recommended trust.
Furthermore it tries to establish a standard for trust in Computer Systems and deal with the arising problems for which there is no satisfying solution at the moment.

3 Related Work

3.1 Definition of Trust

Trust, as we know it, is a very important aspect of human life. We use it every day in one way or another. For instance, we go to work and trust airplanes not to crash our houses meanwhile, we trust our bank to give us the money we claim and some even trust their computers to work every time they switch them on. Trust is central to all transactions, where our own actions are dependent on the actions of others. Thus, excluding instances where trust in someone has no influence on our decisions. Trust can be strong or weak depending on the environment. Morton Deutsch defines trust as follows [25]:

- If an individual is confronted with an ambiguous path, a path that can lead to an event perceived to be beneficial (Va+) or to an event perceived to be harmful (Va-);
- He perceives that the occurrence of Va+ or Va- is contingent on the behavior of another person; and
- He perceives that strength of Va- to be greater than that strength of Va+.

In this definition, Deutsch describes an event with a beneficial outcome as Va+, while a harmful result is entitled as Va-. His view of a trust relationship is such that any event is linked with other events by either conducive or detrimental paths. To reach another event, one of the available paths has to be taken. This definition is based on psychology but outlines one of the most important requirements to establish trust. If the perceived benefit were greater than the perceived harmfulness then the significance of trust in the choice would not be that big. In other words, a trust relation requires a harmful path with more significance than the beneficial one. An employee for example, would not choose to hack a database and switch identity with another employee if the others salary was less than his own.

A more concrete and mathematical definition is given by Diego Gambetta [14]:

> *Trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action (or independently or his capacity ever be able to monitor it) and in a context in which it affects his own action.*
> *When we say we trust someone or that someone is trustworthy, we implicitly mean that the probability that he will perform an action that is beneficial or at least not detrimental to us is high enough fur us to consider engaging in some form of cooperation with him.*
> *Correspondingly when we say that someone is untrustworthy, we imply that that probability is low enough for us to refrain from doing so.*

Both definitions combined will give the requirements that have to be met when building an environment based on trust. Gambetta [14] measures trust with a range from 0 to 1, where 1 represents complete or blind trust and 0 is complete distrust. With that in mind, trust gains the following properties [10]:

1. *Trust is relativized to some transaction.* A may trust B to drive a car but not to baby-sit.
2. *Trust is a measurable belief.* A may trust B more than A trusts C for the same business.

3. *Trust is directed.* A may trust B to be a profitable costumer but B may distrust A to be a retailer worth buying from.

4. *Trust exists in time.* The fact that A trusted B in the past does not in itself guarantee that A will trust B in the future. B's performance and other relevant information may lead A to re-evaluate it's trust in B.

5. *Trust evolves in time, even within the same transaction.* During a transaction, the more A realizes it can depend on B for a service X the more A trusts B. On the other hand, A's trust in B may decrease if B proves to be less dependable than A anticipated.

6. *Trust between collectives does not necessarily distribute to trust between their members.* On the assumption that A trusts a group of contractors to deliver (as a group) in a collaborative project, one cannot conclude that A trusts each member of the team to deliver independently.

7. *Trust is reflexive, yet trust in oneself is measurable.* A may trust his lawyer to win a case in court more than he trusts himself to do it. Self-assignment underlies the ability of an agent to delegate or offer a task to another agent in order to improve efficiency or reduce risk.

Beside the above definitions and restraints, trust is treated differently in some other sectors. Sociology for example deals with responsibilities and moral in this context. These concepts, although strongly related to the notion of trust, are hard to measure or to include in a conceptual form. The ability to measure trust is the main prerequisite to create a concept of a trust-based security environment in a network. Gambetta's definition of trust forms the basis for this thesis. Deutsch on the other hand, gave a valuable hint on methods to keep a trust relationship balanced, once it is established. A balance of benefit and harmfulness is a main criterion for a working trust-based environment. Section 3.5 covers this subject in detail.

3.2 Establishing Trust

Trust relationships are usually based on identity [40]. Information about an agent's behavior that was gained earlier can only be applied in later transactions, if the agent is recognized as the very same. Thus a working authentication system is essential to establish a trusted environment. This system can be a private/public key infrastructure, certificates or some other method to ensure someone's identity.
This respect is the most limiting part in the adaption of trust in computer systems. A user who must first obtain a security token, get a private key or has to request a certificate before he can use a specific Service, is limited right from the beginning. Unfortunately there is no satisfying work-around for this problem. The reason is simple. Without any auxiliary form of authentication, the only way to identify a user is to identify the computer itself. If the same computer is then used by another person, it is impossible to distinguish between them. Even a public/private key structure is not a perfect proof for a given identity out of the same reason. It is still possible for multiple persons to share their access data but it is unlikely enough to be compromised and therefore a sufficient method to ensure a user's identity.

In this identity-based trust, a trust relationship will be of a strong or weak value. The prisoners' dilemma [24] is a common example for trust relations and the problem of unknown identities.
Consider two burglars, Bob and Al, got caught by the police. Each has to choose whether to confess or not and whether to implicate the other or not. The scenario is as follows [25]:

- If neither Bob nor Al confesses the both will serve one year for carrying a concealed weapon.
- If Bob confesses and implicates Al and at the same time Al confesses and implicates Bob the both will serve 10 years each.

- If Bob confesses and implicates Al but Al does not confess then Bob goes free and Al gets 20 years and vice versa.

The payoffs of this prisoners' dilemma are shown in Table 1. Now Bob and

	Al		
		Confess	Don't confess
Bob	Confess	10/10	0/20
	Don't confess	20/0	1/1

Table 1: Prisoners' dilemma payoff

Al are kept in two different cells, unable to communicate with each other. They will try to choose a strategy that will minimize the time they have to stay in prison. Without further knowledge of the others' behavior, everyone will choose the so called dominant strategy [24] for their best payoff. If Al confesses there are two options: If Bob confesses too, he gets 10 years but if he doesn't he will be free. If Al does not confess on the other hand, he has the option of 20 years in the case of Bob confessing and one year if he doesn't. So both of them will choose to confess to avoid the worst case, no matter what the other does and consequently get 10 years each, which is not the best possible outcome.

In the above example they both could have got away with one year each, if they had trusted that the other person would not confess. But trusting the other person in this kind of scenario can be difficult. They could have used a trusted third party for instance. Maybe a lawyer they both know and who suggests how both of them benefit most. If the identity of this lawyer is acknowledged by both burglars, he can establish a trust relation.
Another option would show if Al and Bob knew each other since they were children. In this case they would know each other well enough to assess whether the other confesses of not. This relation is called direct trust.

If trust between Al and Bob is not high enough, punishment may matter. Each of them will think twice if it is a wise decision to confess when he gets punished by the other's gang afterwards. How punishment influences a continuum of trust will be discussed in Chapter 3.5.
The following section explains how a trust relationship like this can be expressed in a more mathematical way and how a distributed system deals with it.

3.3 A Formal Representation of Trust

3.3.1 Basic Trust

Particular agents or entities are represented by the letters a to z, while each agent is a member of A, the set of all agents. Furthermore, situations are represented by greek letters α and ω. A situation is defined here as a specific point in time relative to a specific agent. Thus different agents at the same point in time will not consider themselves to be in identical situations [29]. Situations apply to agents and therefore the notation becomes α_x to ω_x for situations from the point of view of agent x, for example.

An agent x has a **basic trust value** T_x , derived from previous experience. If no previous experience is present an initial trust value is used. This value, which is dynamically altered in the light of all experience, is used in the formation of trusting relationships with unknown agents, and is represented as T_x normalized over (0,1) [29]. It is important to realize that it does not correspond to the amount of trust x has in any other agent, but only to the general trusting disposition of x and is less 'fluid' and changeable than a trust in any specific agent. This value is important for unknown users and defines the amount of trust available as a foundation to build up a mutual trust relationship.

3.3.2 The Trust Value

In [29] the representation of trust is defined as follows:
Given two agents, $x,y \epsilon A$, to say x trusts y, we write: $T_x(y)$. In addition to being a representation of the fact that x trusts y, this value is normalized over (0,1), of the amount of trust x has in y. In other words, should $T_x(y) = 1$, then x has complete or blind trust in y. The trust value is a view of a particular agent of another with regard to the trusted agent's general capabilities. Furthermore, different situations may require different views

of trust. The amount of trust in a particular agent in a given situation is represented in [29] as, for example, $T_x(y, \alpha_x)$, normalized over (0,1) for x's situational trust in, or reliance on, y to perform correctly in situation α_x. Note the interchangeability of the concepts of situational trust and reliance here. At present, they are considered to be identical. However, it may be feasible in the future to separate the two concepts, although closely related, to reflect more closely the more philosophical views in the subject [29]. For this work it does not matter if an agent's decision to cooperate with another agent is viewed as reliance or trust because it is a question of interpretation.

3.3.3 Importance

The agents's estimate of how important a situation is to itself is a value normalized over (0,1), represented by $I_x(\alpha_x)$. The importance is an estimation of x on how much situation α_x means to it. The importance of a situation to an agent is useful in determining the amount of situational trust to place in an agent at any given time. Related to the importance of a situation are the concepts of cost and benefits pertaining to that situation.
The costs of a situation are measured in terms of the problems associated with incompetent or malevolent behavior on the part of another agent in a relationship. This costs are represented by $C_x(\alpha_x)$, with a value normalized over (0,1). These potential costs are relative only to the agent concerned. As such, the potential costs of untrustworthy behavior remain the same. On the other hand, the benefits of trustworthy behavior from the agent(s) being worked with, are represented as $B_x(\alpha_x)$, normalized over (0,1).
Take the example of a bank account for a better understanding of importance and cost/benefit. From the owner's point of view, transactions *from* the account will be far more important than transactions *to* it. For simplicity, malevolent behavior occurs in the form of unauthorized transactions and therefore a unauthorized transaction to the account is not harmful to the account's owner. The costs for a wrong transactions *from* the account can

be very high though. The benefits for authorized transactions are similar. Accepting a incoming transaction will be more beneficial than authorizing an outgoing transfer.

3.3.4 Acquaintance

Since trust is based on previous interactions and situations to a large extent, some method of showing whether an agent is known to another or not is needed. This is referred to as acquaintance, represented as $K_x(y)$. Again, its value is normalized over (0,1). For simplicity it can also be treated as a boolean value.

3.4 Rules for Interaction

With this notation we can now express a trust relationship for a specific agent. The provided equations are neither intended to be a exact representation of human cognition nor as a final statement how trust works within agents. They help to make decisions in particular situations.

The 'amount' of direct trust an agent has in another, depends on the initial or general trust in that agent and the importance of the whole situation for the trusting agent:

$$T_x(y, \alpha_x) = f(T_x(y), I_x(\alpha_x))$$

In general, if $I_x(\alpha_x) > T_x(y)$, the resulting situational trust will be $T_x(y, \alpha_x) < T_x(y)$. If the importance of a specific situation is higher than the trust value for the other agent, the direct trust value for this particular action will always be lower than the initial trust in the other agent. This function calculates a value based on importance and previous experience only. If this method is sufficient the value can be obtained with the following equation:

$$T_x(y, \alpha_x) = T_x(y) + (T_x(y) * (T_x(y) - I_x(\alpha_x)))$$

Note that the decision to trust a specific agent may also be related to the other agent's competence in the given situation.
In Order to cooperate with agent y, the trust x has in y for that situation has to be above a certain threshold for this particular action. This threshold is a function of importance, costs and benefits:

$$\text{If } T_x(y, \alpha_x) > \text{Cooperation\_Threshold}_x(\alpha_x) \Rightarrow \text{Will\_Cooperate}(x, y, \alpha_x)$$

with:

$$\text{Cooperation\_Threshold}_x(\alpha_x) = \frac{\text{Perceived\_Risk}_x(\alpha_x)}{\text{Perceived\_Competence}_x(y, \alpha_x)} * I_x(\alpha_x)$$

Here, Perceived_Competence$_x(y, \alpha_x)$ reflects the influence of an agent's competence, and allows cooperation with an agent which is not trusted very much, or an agent in an important situation where the agent is known to be reliable and competent. The Perceived_Risk$_x(\alpha_x)$ represents the agent's best estimate of the situation's potential costs and benefits. Both terms are explained as follows:

Perceived Competence
If the trusting agent has no knowledge of the other agent, or the situation, then:

$$\text{Perceived\_Competence}_x(y, \alpha_x) = T_x$$

It is possible for the trusting agent to know the agent to be trusted though. If so, it may have some form of trust in that agent already. In this case competence of y is $T_x(y)$ instead of T_x.

Perceived Risk
Risk is a balance of cost, benefits and importance of a given situation. According to Marsh [29], a good estimate for the perceived risk is:

$$\text{Perceived\_Risk}_x(\alpha_x) = \frac{C_x(\alpha_x)}{B_x(\alpha_x)} * I_x(\alpha_x)$$

The threshold for a successful cooperation is higher, the higher the costs and the importance of the situation. If different degrees of cooperation are possible, staggered thresholds according to the different importance values are imaginable.

The last rule to complete agent's interactions concerns the propagation of trust. Marsh [29] did not take a recommendation into consideration, that allows an agent to rely on the trust of a third agent. The idea of small worlds is a fitting approach to this problem. Based on the Small World hypothesis

[18], members of any large network are connected to each other through short chains of intermediate acquaintances. In an experiment to prove the hypothesis, packages were sent to 100 random people in the USA with the order to transfer the package to the original recipient using only people known at a first-name basis. The astonishing result is one of 'six degrees of separation', which states that any two people in the U.S. population at the time are connected by no more than six other people.
This model can be mapped to smaller networks as well. It is possible to rely on the trust of a third agent which is connected to the trusted agent as well. If no path to the desired agent can be found, the initial trust value is used for the first interaction.
To stay true to the notation of Marsh, this recommendation affects the basic trust of x in z in case agent x has no prior knowledge about z. Provided that x already knows agent y and y already established a trust relation, its initial trust can be adjusted as follows:

$$T_x = T_x + \frac{T_x(y) * (T_y(z) - T_x)}{2}$$

This adjustment reflects a recommendation of a trusted third party and influences the basic trust value of the trusting agent. The assumption is such that agent y already established a relationship with agent z. How much influence y's assessment of z has on the initial trust of x, depends on x's previous experience with y. The more x trusts y, the more influence will y's recommendation have on x's initial trust. This function always takes x's own trust value into consideration. This way x will never completely rely on another's recommendation. In the most extreme case, when x has an initial trust of $T_x = 0$ and blind trust in y and y has blind trust in z, x's initial trust value is raised to:

$$T_x = 0 + \frac{1 * (1 - 0)}{2} = 0.5$$

This represents a change from a complete distrust situation to a neutral view.

The formulae above are of heuristic nature and intend to provide a useful tool for the evaluation of situations and cooperative relationships. A more theoretical trust derivation algorithm is given in [48]. This algorithm deals with entities and graphs for relations between different agents. It presents a formalism for expressing trust relations along with an algorithm for deriving trust relations from recommendations.
The next section finally deals with risk and punishment in an already established trusted environment.

3.5 Risk and Punishment

Punishment is introduced to balance a trust-based environment and to keep it from being unstable. If there is an absence of suitable punishment, that is incurred loss, for breaking agreements or contracts, individuals will not possess the appropriate incentives to fulfil them [9]. The result would be complete distrust for the multitude of the participating agents. Because this will be generally recognized within the population, agents will not choose to enter transactions with one another. Thus, what could in principle be a mutually beneficial relationship will not be initiated. Secondly, the threat of punishment for errant behavior must be credible, else the threat is no threat. The enforcement agency itself must be trustworthy and will do what it says and only what it says [9].

This way punishment can be used to balance an existing system and outweigh the costs and benefits a transaction implies. What is punishment for one agent, represents risk (to be punished) for the other. In a real example punishment could consist of stakes, an agent has do deposit before it may enter the trusted environment.

With this in mind, the following sections deal with the attempt to establish a working model of a trusted environment.

4 The SimOffice Concept

With all this theoretical knowledge about Web services, Internet technologies, trust and the various forms of trust relationships, it is time to make something useful out of it. This chapter gives an insight into the development process of the *SimOffice* environment. It discusses the most important decisions taken during the conceptual work, and how possible alternatives would look like.

4.1 Prerequisites

4.1.1 Hardware

To participate in the *SimOffice* environment all a user needs is a network interface and the proper connection. As long as the transport of messages based on internet protocols is possible, the basic hardware requirements for *SimOffice* are met. Depending on a user's processing power and connection speed, the time to complete a single transaction will vary. Different clients will require different response times. A second to complete a request for a coffee will be sufficient, but for a file server this is not sufficient at all. Therefore, performance-related hardware requirements must be chosen for each client individually.

4.1.2 Software

The biggest benefit of an architecture based on Web services becomes obvious here. A user is not bound to a certain piece of software, to use or even provide a Web service in this environment. What is needed though, is a proper way to de- and encode SOAP messages sent via HTTP for example. Furthermore, a Web browser will ease the process of discovering a Web service but is not strictly required. For the architectural design, this creates the following guidelines:

- *SimOffice*-related data transfer should only happen through SOAP messages. Any form of platform- or even language-dependent communication is a limiting factor and should be avoided.
- Arbitrary ways to establish communication between two points within the *SimOffice* environment are not desired and should be reduced to a minimum amount.
- Potential users should be able to participate in the *SimOffice* environment with no or a minimum of prior knowledge.

Section 4.3 explains how these guidelines are carried out in detail.

4.1.3 Availability

Apart from hardware and software requirements, availability is the third big issue that needs to be addressed before a meaningful concept can be created. Two options are possible:

- A Web service is always available. In this case, an unavailable service is considered erroneous. In general the service is expected to reply within a given time.
- A Web service is available only once or at random times. Unavailable services are expected and handled by the system.

Web services are in principle designed to be available as long as they exist. UDDI registries do not comprise a facility to check the availability status of a once registered Web service. As a result, public UDDI entries are often obsolete because they are used for testing purposes or not updated properly. On the other hand it is not mandatory to register every Web service at a UDDI registry. If the service discovery is solved otherwise, nothing speaks against the use of Web services in ad-hoc network environments.

SimOffice will definitely allow ad-hoc connections for the **consuming** Web service. Whether a **provider** of a Web service is treated as an erroneous node when it does not respond will be discussed in the next section.

Other important issues like scalability, network security or privacy are discussed later in this chapter because they directly influence the decisions made during the design process.

4.2 Architectural Design

As mentioned earlier, the *SimOffice* architecture consists of two main parts:

1. **Providing** Web Services: A providing Web service offers the functionality within the environment. A file server is a good example. It holds at least one Web service with different functions for creating, deleting or moving files. The whole file server is entitled as a *node* within the *SimOffice* environment. One node consists of one or more Web services.

2. **Consuming** Web services: A consuming Web service represents the 'user' within the system and is not a constant part of *SimOffice*. In the example with the file server, a service that invokes the methods to delete, move or create a file is treated as a user. A node can be both consuming and providing Web services at once.

Communication within *SimOffice* is entirely based on Web services. The concept presented from now on focuses on trust-based Web services. To build up an environment based on trust, the first decision is how to pass on trust-related information between two nodes.

4.2.1 Server Dependence

One method to pass information between two nodes is a direct connection. This connection is preferably established with a Web service on each side to stay language and platform independent.

Figure 3 shows how the nodes interact with each other.

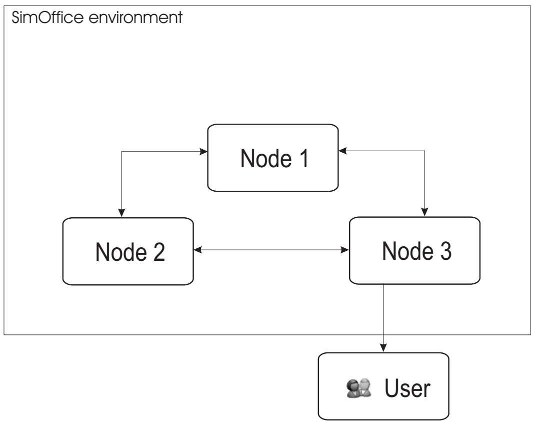


Figure 3: Architecture with directly connected nodes

The other alternative is a *trust server* or *trust engine* that establishes the connection between two nodes. Again the communication between trust server and nodes is achieved with Web services.

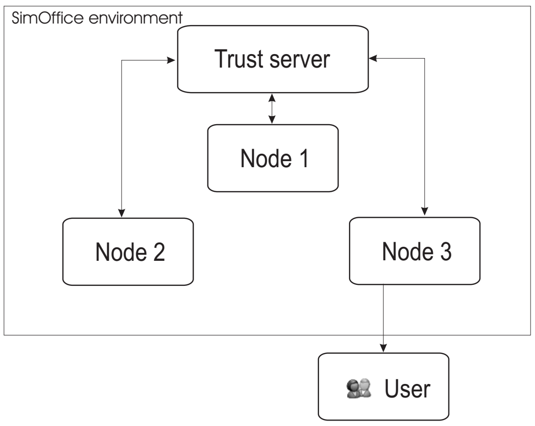


Figure 4: Architecture with a trust server

Either way, a node must know the correct way to receive or transmit trust-related data. This fact is designated as *trust-awareness* in this thesis. Without this knowledge, a node cannot join the *SimOffice* environment. The advantages and disadvantages of the different architectures are compared in

Table 2 for a better illustration.

Server-based architecture	**Server-less architecture**
Services are registered at the server, either in a private UDDI registry or as WSDL-Files to download.	Service discovery is very problematic without a UDDI registry or a central registration point. Some sort of broadcast for available services would be needed.
A provider asks the trust server to assess a requesting user.	Every service provider must comprise an algorithm for judgment and trust propagation as well as an interface for incoming requests to judge a third party.
The trust server deals with trust recommendation and only delivers the result to the requesting party.	A provider must choose whether to ask a third party for recommendation or rely on its own experiences.
The trust server is a single point of failure. In case of an attack or failure, the whole system is affected.	Security attacks or possible failures will only affect one node. The remaining system will still be functional.
New user accounts are available immediately after they are known to the trust server. A possible 'Superuser' can be created for all Web services by simply responding with the highest trust ratio of 1.	A distribution system for new user accounts has to be created. The generation of a 'Superuser' is not possible at a single point. Instead, every node must provide some sort of facility to create extraordinary accounts.

Table 2: Architecture comparison

The server-based architecture is the method of choice for *SimOffice*. Above all, the possibility to create user groups relatively easy at a single point was casting for this decision. Furthermore, the trust engine is responsible to evaluate the trust levels for a user at a specific node, which enables low-performance nodes like the coffee maker to participate in the environment too. Additionally, a node's required trust-awareness is minimal.

4.2.2 Identification

The next problem is **Authentication**. Who needs to be identified within the *SimOffice* environment? The answer is simple: Everyone that needs to be judged before access is granted or denied. This includes every Web service consumer, no doubt about that. For a service provider this is more difficult to decide. Let us take a look at a possible scenario.
SimOffice is extended with a node that provides functionality for printing various documents. This node is implemented directly at the printer server by Betty, one of the company's employees. When finished, it has to be published to the central trust server to join the trusted environment and receive recommendations for example. Before it is published, it will be checked for things like program errors and trust awareness by both, the programmer and the system administrator who is responsible for the trust server. This check ensures that this new node fits into the existing structure and works as intended. After the new node is approved and added to the environment, it will remain unchanged until it is unregistered. After a few days, Betty wants to use the Web Service she programmed herself to print a few pages. Like every other user she needs to authenticate first before she has access to the service. Fortunately for her she is a respected employee and her trust level is high enough for this operation.

It is helpful to look at *SimOffice* as a distributed application, where every node represents a set of procedures of the entire application. A node will not behave in an unexpected way and therefore it is not necessary to judge it's intent or identify it like a user. With other words, once a node is successfully integrated into the *SimOffice* environment, it is treated as a trusted, integral part or the system. If a node uses another node's functionality, the connection is represented as a blind trust relationship. This structure makes it difficult to handle guests on the other hand. If a user would like to provide his own Web services, their functionality have to be approved by the system administrator first. Even then, this 'guest' would represent a unreliable node which cannot guarantee it's availability. Therefore guests or generally ad-hoc services are not intended to extend the SimOffice environment.

Several ways to finally identify a user are possible. An idea mentioned earlier in this thesis was to use the IP or hardware address of a connection for authentication purposes. This 'passive' form of authentication has the big advantage that no user must be aware of the authentication method. Nevertheless, Section 3.2 explains why this form of identification is not sufficient in this context.
The most commonly used authentication mechanism is called basic authentication. To gain access to a specific resource, a user must provide an ID and a password. If the user is authorized to access the requested resource an answer is sent back. An application of this scheme in the *SimOffice* environment would result in the following characteristics:

- A user must register at a central point to create a user ID along with a password. The trust server can represent this point.
- A potential user must provide his ID and password in **every** request. Connections between Web services are of a request-response type and limited to one transaction. Persistent connections are not supported.
- ID and password are either sent in the SOAP header or as part of the data payload. In the latter case, both services must provide support for the additional parameters.
- After receiving a request, the providing node fetches the authentication data and queries the trust server for the user's access rights. If the user is trusted, the proper response is sent back.

This approach is very fitting and meets all requirements to authenticate a user in the *SimOffice* environment. Unfortunately there is one catch. User

ID and password are sent as plain text within the SOAP message. As a consequence this messages are susceptible to eavesdropper and therefore a severe security leak. Anyone with access to the SOAP message on its way from the sender to the receiver would be able to read the login data and use it to send messages with another identity.
The final step in this evolution of authentication methods is to encode the message's data. This leads to the very commonly used private/public key infrastructure. In this model, the central point of authentication is represented by a certification authority. Every user has to register there first, to receive a private key for his user ID. When sending a message, the user ID has to be sent along with the payload or in the header and has to be encoded with the corresponding private key. Once received, the public key, which is accessible at the certification authority, can be used to decode the message and verify the user's identity. This way a SOAP message is protected against eavesdropping as well as message tampering and replaying. This method to secure SOAP messages is part of the current security standards for Web services (WS-Security [22]).

The solution with a private/public key infrastructure requires a high amount of client awareness. A client must know how to sign a request with the private key. This involves some additional expenditure which is the price for the secure transmission.
The *SimOffice* case study will use the user ID/password method for authentication simply because it is much easier to implement and the way to authenticate a user has no influence on the functionality of trust relationships itself.
Based on these considerations, a SOAP request containing user ID an password would look like this:

```
<SOAP-ENV: Envelope xmlns:
SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV: encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <SOAP-ENV:Header>
```

```
        <t:user-id xmlns:t="some-URI">Johnny Mnemonic</t:user-id>
        <t:password xmlns:t="some-URI">pityyoucanreadthis</t:password>
      </SOAP-ENV:Header>
      <SOAP-ENV:Body>
        <m:RemoteFunction xmlns:m="some-URI">
        <Parameter1>123</Parameter1>
        </m: RemoteFunction>
      </SOAP-ENV:Body>
</SOAP-Envelope>
```

This completes the basic architecture of *SimOffice*. What remains is the question how to discover a Web Service once a user was successfully registered.

4.2.3 Discovery

Two ways are designated to service discovery. The first one is to make the WSDL documents of all nodes accessible at a specific point like the certification authority. Secondly, every Web service will be published at a private UDDI registry. What remains is the question how to find the UDDI-registry's location or the index page where the WSDL files are published. With the architecture existing so far, it is now possible to identify the first point a new participant must access before he is able to use a Web services within the *SimOffice* environment. The very first action will be the creation of the user account. For this purpose a Web interface has to be created where various information about the new user is gathered. Upon completion of the registration process, a link to both UDDI registry and WSDL download area is displayed.

With this last point, the coarse architecture of *SimOffice* and its most important elements is complete. The next section will explain how this fundament is used to finally establish the desired trust relationships.

4.3 Trust-related Design

As the most important part of my thesis, this section covers the process to assess an identified user based on previous experiences. Some methods and ideas to balance trust values will be refuted or proven during implementation and are of a theoretical nature at the moment.

4.3.1 Variables

The whole concept of trust is based on variables. Some of them are related to a node or a specific Web service, while others evolve in time to represent a relationship. With the architecture of a central trust server, the question where to keep these variables comes up.

The situational trust value $T_x(y, \alpha_x)$ can be viewed as the key value that is only needed where an assessment about another party is made. All other variables, including importance $I_x(\alpha_x)$ and initial trust value T_x are stored at the same position because they are used to calculate the trust value.

It is possible to either manage the variables at the trust server or at the concerning node itself. The following pros and cons for both methods will ease the decision which way to use.

Node manages variables: A simple scenario, where the node receives a single request from an already identified user, would cause the following order of events:

1. The node receives a request with authentication data.
2. The node sends the data (ID and password) to the trust server and asks for a recommendation. Along with the user's data, the node must also send the situational trust value which is evaluated by the node itself because all required variables are available.

3. The trust server evaluates the request and decides whether other nodes are required for recommendation. If so, the trust engine must request every node's recommendation and wait for their responses. Upon completion or a predesignated timeout the trust engine works out the recommendation value and sends it back to the invoking node.
4. The node receives the trust engine's assessment and finally grants or denies access to the requested service.
5. In case of an access violation or misbehavior, the node sends an additional message to the trust server with information like parameter or requested function to carry out punishment if necessary. Simultaneously, the node alters the situational trust value for the requesting user.

An advantage of this method is the independence of the different nodes. Adjustments of the variable's values take place directly at the node and do not concern the trust server, thus making it easier to balance new nodes without affecting other nodes. This approach involves a large amount of network traffic on the other hand. Every node that is taken into consideration for a recommendation causes an additional SOAP request as well as an additional delay if only one node does not respond. Furthermore a node must be capable of storing values and calculating trust values which involves a considerable amount of computing power. This fact probably rules out the coffee maker as a node.

Trust server manages variables: This approach deals with the mere opposite. The trust server manages as many variables as possible thus reducing the amount of server-node traffic. The same scenario mentioned above would appear in the following manner:

1. The node receives a request with authentication data.

2. The node passes the identification data on to the trust server, along with relevant parameters (e.g. the amount of pages to print at a printer service) and asks whether to carry out the operation or deny the access.
3. The trust server receives the request, checks if the ID is valid and calculates the trust value. Because the data of all other nodes is accessible at the trust server, the trust engine can immediately calculate a recommendation that takes all other nodes into account.
4. If the trust value is higher than the threshold, the trust server entitles the node to carry out the operation and complete the request. If the trust level is too low, the node simply denies to complete the request. In both cases the situational trust value is adjusted by the trust engine as well as possible punishment in case of a misbehavior.

This scenario makes it plain that the second approach comprises more benefits. It involves almost no effort for a single node to participate in the *SimOffice* environment. All a node has to do is pass on the request's data to the server and wait for the permission to carry out the requested operation. Even the coffee maker will be able to do that. On the other hand the trust server is both, a single point of failure and a limit for the environment's scalability. Depending on the trust server's performance, a limited amount of nodes can be handled by the trust engine at the same time.

The second approach perpetuates the thought of a completely server-based design and additionally comprises a very convenient way to add new functionality to the system if the trust server is designed cunningly. A well-designed trust engine will assign adequate default values to the variables of a newly added node. If chosen properly, there will be no need to adjust these values

in many cases.
This very important issue is discussed in the next section.

4.3.2 Default Values

Default Values are used for new nodes. As soon as a new node registers at the trust server, those values are used to initially fill the database. If not altered afterwards, they still have to represent a meaningful node, whereas 'meaningful' refers to a node where access rights are neither too low nor too high. If a user with an average trust level (e.g. 0.5) just manages to cross the threshold, the system has the desired behavior.
The following listing will give the default values for all variables as well as a detailed description of their influence on the system's behavior. A file server is consulted as an example with the whole file server representing a node, while three Web services to read, write and delete a single file are the node's functions.

Initial trust value $\mathbf{T_x}$: This variable represents the confidence that is put into an unknown user. The lower this value is, the more the user has to prove himself before he reaches a high trust level.
This value is global to the whole node and cannot be adjusted for a single function of the node. In case of the file server it means every new user is trusted equally, no matter which function he desires to use. To initially permit or grant access to the different functions, their threshold levels have to be set accordingly. For a neutral start the default for initial trust $T_x = 0.5$.

Importance $\mathbf{I_x}$: As the name implies, this variable is a measure of how important a node is. Again this variable applies to the whole node and is not altered individually for different functions. It would seem fitting to apply different importance values to different functions, like $I_{read} = 0.3, I_{write} = 0.5, I_{delete} = 0.7$ for instance, but this would result in two variables affecting the final accomplishment of the request and therefore distort the characteristic for a later trust adjustment. First the importance. The lower it is, the higher will be the situational trust. According to Section 3.4, the value for situational trust is obtained by the following equation:

$$T_x(y, \alpha_x) = T_x(y) + (T_x(y) * (T_x(y) - I_x(\alpha_x)))$$

This value is then checked against the *threshold* to figure out whether the operation is carried out or not. This threshold is the second variable that influences the final accomplishment of the operation. Both, a low importance and a low threshold would lead to a very low required situational trust. In the file server example, user with a low situational trust of 0.3 would still be able to access the read function if threshold and importance were set to 0.3 too. A file server is more important than that and therefore the importance to read, write and delete a file are set to 0.7 in this example. A neutral default is $I_x = 0.5$ again.

Cooperation threshold, cost and benefit $\mathbf{C_x, B_x}$: The cooperation threshold appoints how trusted a user must be, before an operation is carried out. It originally is a function of cost, benefit and competence. This is a fitting approach for equal trust relationships but not for *SimOffice* where a node always takes requests but never invokes a service of a client. As a result, an operation's benefit is a pointless value. A node will not benefit from any operation while a certain cost may exist. With the equation

$$\text{Perceived\_Risk}_x(\alpha_x) = \frac{C_x(\alpha_x)}{B_x(\alpha_x)} * I_x(\alpha_x) = \frac{C_x}{0} * I_x,$$

the result would be an infinite risk. Therefore, threshold levels are not calculated based on risk but defined with a fixed value for every single function in the *SimOffice* environment. The more significant a function is, the higher the threshold level. A default value of 0.5 for the threshold completes the

picture of a neutral assessment of new users, assuring that he is able to access functions with low and average importance.
The next section finally discusses how to properly adjust trust levels.

4.3.3 Trust Adjustment and Punishment

The adjustment of trust levels is the most important and by the way the most interesting point of my thesis. The whole concept heavily depends on the way that a user's behavior influences the system's reaction in the future. It is helpful to visualize how human assessment works and which rules it follows. Some of the most obvious properties are:

- Trust in a specific agent should never reach the values 0 and 1. Blind trust and complete distrust are undesirable conditions and have to be avoided if possible. It would cause the system to snap in a state where it is impossible to return. Blind trust would never result in misbehavior while complete distrust represses any chance to prove again.

- Trust alteration is based on importance. The more important the action is, the more the agent's behavior influences future assessments.

- To what extend the direct trust is altered also depends on the quantity of transactions. A single transaction of a common event will not influence the trust level as much as a transaction of a rare event.

An algorithm has to be found that alters the direct trust value of an agent according to these properties.

Double-sided exponential approach
This approach uses an exponential function for both malevolent and trustworthy behavior. Exponential functions satisfy the requirement to converge towards a specific value. First we have to set up an equation for both cases where the function for trustworthy behavior converges towards one, while the

function for untrustworthy behavior converges towards zero. Figure 5 shows the desired shape of both functions.

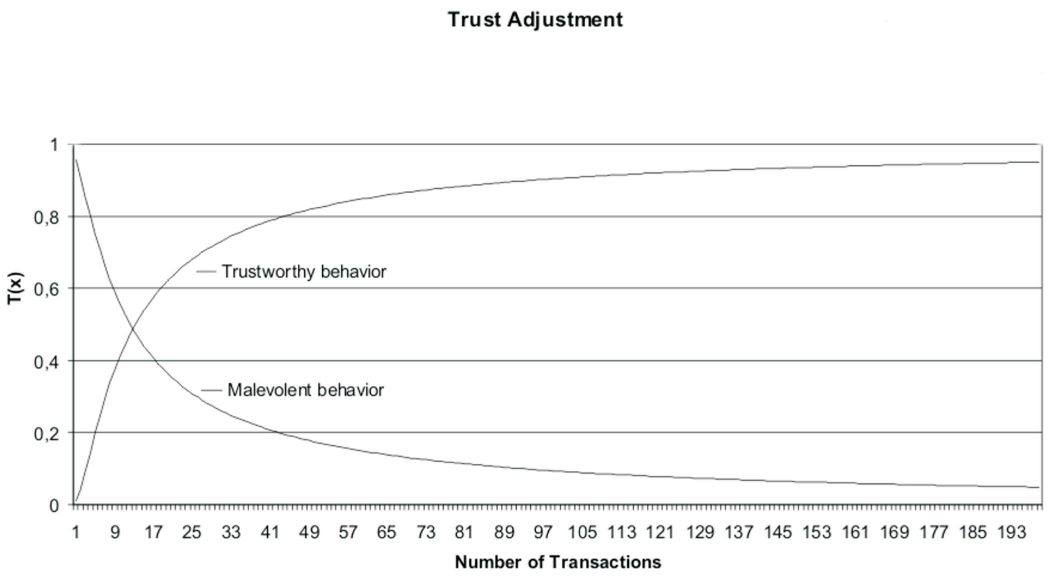


Figure 5: Double-sided exponential approach

The first equation that is applied in case of a trustworthy behavior is represented by a simple exponential function:

$$y = f_{(x)} = e^{-\frac{1}{x}} \qquad \forall x \in (0, \infty]$$

To calculate a subsequent value from a given trust level, the following equations have to be combined:

$$\begin{aligned} y_{initial} &= e^{-\frac{1}{x}} \\ \Rightarrow \ln(y_{initial}) &= -\frac{1}{x} \\ \Rightarrow x = -&\frac{1}{\ln(y_{initial})} \end{aligned}$$

and

$$y_{good} = e^{-\frac{1}{x+c}}$$

$$\Rightarrow y_{good} = e^{-\frac{1}{-\frac{1}{\ln(y_{initial})}+c}}$$

$$\underline{y_{good} = e^{-\frac{\ln(y_{initial})}{c*\ln(y_{initial})-1}}} \quad \forall\, y_{initial} \in (0,1].$$

The constant c defines the slope and is a function of importance I and increment s:

$$c = s * I_x$$

The function to compute loss of trust in case of an untrustworthy behavior is very similar:

$$y = f_{(x)} = 1 - e^{-\frac{1}{x}} \qquad \forall x \in (0,\infty]$$

Again, a combination of subsequent steps is used to calculate the adjustment for malevolent behavior:

$$\begin{aligned} y_{initial} &= 1 - e^{-\frac{1}{x}} \\ \Rightarrow 1 - y_{initial} &= e^{-\frac{1}{x}} \\ \Rightarrow \ln(1 - y_{initial}) &= -\frac{1}{x} \\ \Rightarrow x &= -\frac{1}{\ln(1 - y_{initial})} \end{aligned}$$

and

$$y_{bad} = 1 - e^{-\frac{1}{x+c}}$$

$$\Rightarrow y_{bad} = 1 - e^{-\frac{1}{c-\frac{1}{\ln(1-y_{initial})}}}$$

$$\underline{y_{bad} = 1 - e^{-\frac{\ln(1-y_{initial})}{c*\ln(1-y_{initial})-1}}} \quad \forall\, y_{initial} \in (0,1].$$

The constant c in this equation is the same as in the first equation for trustworthy behavior.
With these formulae it is now possible to create an example for a better illustration. The following conditions are assumed:

- The initial trust level is very low and represents a very suspicious view for the first interaction, where nothing is known about the agent ($T_x = 0.01$).
- The importance of the node is average ($I_x = 0.5$), and increment is set to $s = 0.2$.
- Out of 10 accesses, one malevolent behavior occurs.
- 200 transactions are traced.

The result is shown in Figure 6.

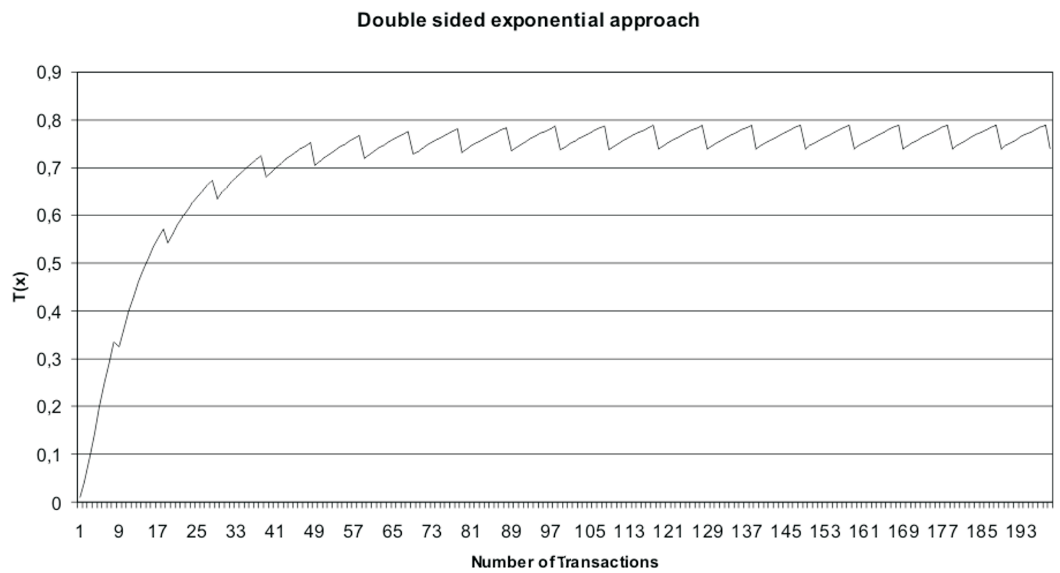


Figure 6: Example for double-sided exponential approach

This approach gives quite a good account of human trust. In the above example, the agent will never achieve a higher trust level than 0.8 and if the threshold is set to that amount, the operation simply requires a better behavior than one fault out of ten. The higher the trust level is, the more impact a misbehavior has. Therefore an agent cannot 'gather' trust for future actions. One last adaption is possible to reflect human trust even better. With the formulae used so far, an equilibrium of trustworthy and untrustworthy behavior would cause the situational trust to converge towards 0.5. That means an agent that alternately behaves trustworthy and untrustworthy is seen as neutral. If this is not intended, the value towards which the situational trust converges can be adjusted by the use of asymmetric importance values for the constant's (c) calculation. The adaption looks as follows:

$$c_{good} = s * (1 - I_x), \qquad c_{bad} = s * I_x$$

For the implementation of the *SimOffice* case study, this adaption will be used because malevolent behavior is expected to be a minority.

Single-sided exponential approach

In contrary to the previous method, this approach utilizes only one function to adjust trust levels. Unlike the double-sided approach, accumulation of trust is possible here. Although this method is not adequate for *SimOffice*, it is noteworthy for other fields where judgement is based on other aspects. A newly hired employee is the best example that utilizes this way of assessment. The business management will first assign projects with lower importance to the new employee to give him a chance to prove himself. If he does it right, new and more important projects will be assigned to him. If he fails he will not immediately get fired but assigned less important projects again. If he continues to fail he will eventually reach a point where he is useless, crosses the threshold and really gets fired.

Figure 7 shows the desired shape of the function.

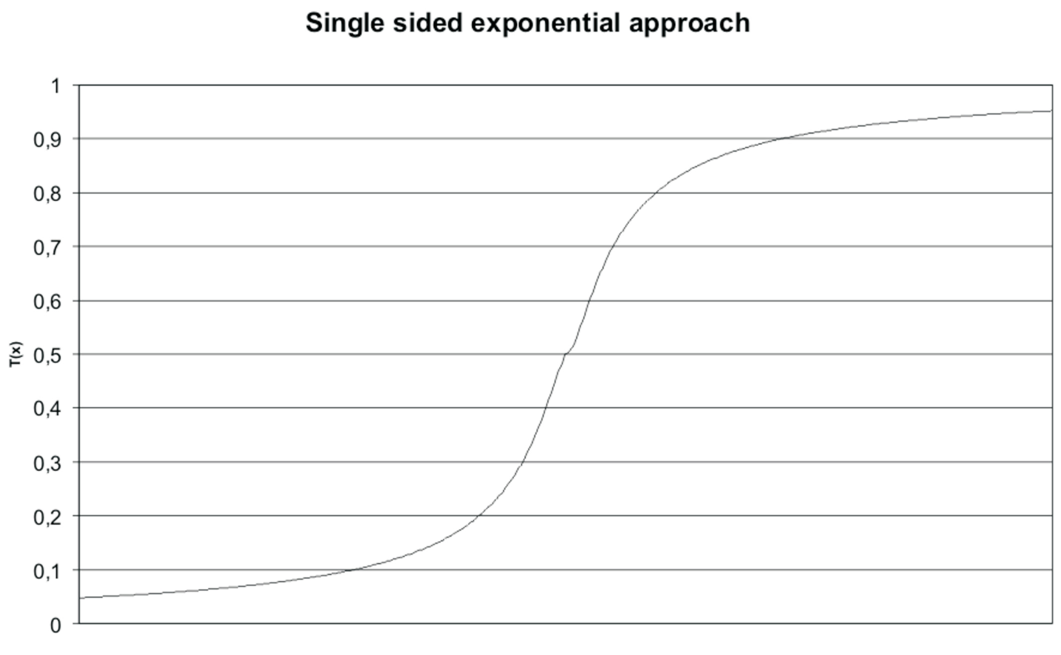


Figure 7: Single-sided exponential approach

To achieve this characteristic, the following function is used:

$$y = f_{(x)} = sign(x) * e^{-\frac{1}{|x|}} \qquad \forall\, x \in R$$

This functions produces values in an Interval of (-1,1). Normalized over (0,1) the function is:

$$y = f_{(x)} = \frac{(sign(x) * e^{-\frac{1}{|x|}}) + 1}{2} \qquad \forall\, x \in R$$

In contrast to the double-sided approach, values at the abscissa are not bound to a certain limit which facilitates the usage of this method enormously. Neutral assessment starts with $x = 0$. For trustworthy behavior, a constant c is added while it is subtracted for malevolent behavior. The constant c is again:

$$c = s * I_x$$

with this new value for x the according trust value is obtained with the above mentioned function.

Both presented approaches are expedient in a fitting environment and are highly flexible to meet the requirements of a given situation.

Another point worth consideration is punishment and what form of punishment is sufficient in the *SimOffice* environment. One form already exists as soon as the double-sided algorithm is applied. The trust engine can decide if a request poses a trust violation which leads to a removal of access rights. If the trust engine denies to carry out the requested operation, no harm is done and a decrease of the requestors trust level is a sufficient form of punishment. A different system is imaginable where an operation is carried out no matter if it poses an access violation or not. To ensure sufficient punishment a user can be forced to deposit a certain amount of cash at the certification author-

ity upon registration. In case of a malevolent behavior this cash is reduced according to the importance of the requested action.
SimOffice will not comprise functionality that is carried out by all means and therefore trust reduction is put down as a sufficient form of punishment.

4.3.4 Recommendation Trust

With the server based architecture of *SimOffice*, recommendation trust is relatively easy to cover. Section 3.4 already introduced an algorithm to adjust trust values according to the experiences of other parties. The original formula altered the basic trust value according to the amount of trust that is put into the node which presents the recommendation:

$$T_x = T_x + \frac{T_x(y) * (T_y(z) - T_x)}{2}$$

The *SimOffice* environment is a little different. Recommendations occur only between two nodes and their relationship is not expressed by a certain trust level. Instead the importance of the asked node is used to weight the recommendation, changing the formula to:

$$T_x = T_x + \frac{I_y * (T_y(z) - T_x)}{2}$$

In this method, where the user is represented by z, it is ensured that a recommendations from the coffee maker does not influence the initial trust level as much as a suggestion made by the file server. The proceeding to finally apply this formula is simple. When a node receives a request from an unknown user, the trust engine iterates through all other nodes within the environment. If an iterated node exhibits an existing assessment, the above function is applied. This way all nodes are taken into consideration.

4.3.5 User groups

The final step to complete the entire concept of *SimOffice* is to create a facility for different user groups. An administrator for example must be able to access every Web service, no matter how high his trust level is. On the other hand there may exist some employees that should never be able to access certain functionality. Again the server based design comes in handy. After the situational trust value is calculated (see Section 4.3.2), it is post-processed by the trust engine. Only two cases are possible:

Superuser: If the user is a member of the superuser group, the situational trust value is set to 1 and therefore represents blind trust. This user will always hit the threshold.

Ordinary User: All other accounts are represented by a maximum situational trust level, to which the situational trust is trimmed if the maximum is exceeded. Let us take Bob for an example again. He starts as a new employee and is a member of the user group 'standard' which limits situational trust to 0.5. He proves well and although he never behaves in a malevolent manner he is not able to access Web services with a higher threshold than 0.5. What he does not know is the fact that his trust level for the coffee maker is already at 0.73. After a year he is promoted to a member of 'valuable employees' where the maximum situational trust is 0.8. Now he is able to take full advantage of all his good behavior and enjoys the moccachino which is granted at a trust level of 0.7 or higher.

With this last point, the *SimOffice* design is complete. The trust-related design presented in Section 4.3 and it's subsections is a general concept and therefore not limited to *SimOffice* or even Web services. Instead, it can be applied to all kinds of systems where an assessment about another party is required to grant or deny access to a specific resource. It is required to be able to identify the other party though. Otherwise the assessment would be worthless. An adoption of this concept for E-Mails to filter Spam for example would fail because the origin of an E-Mail is not genuine and therefore not sufficient for a reasonable judgment. However, an application of this concept for peer to peer networks would be imaginable. There, the trust level could be used to distribute bandwidth between multiple users for example.

The next section will discuss how this concept was adopted and implemented in the course of the *SimOffice case study.*

5 Case Study

5.1 Implementation

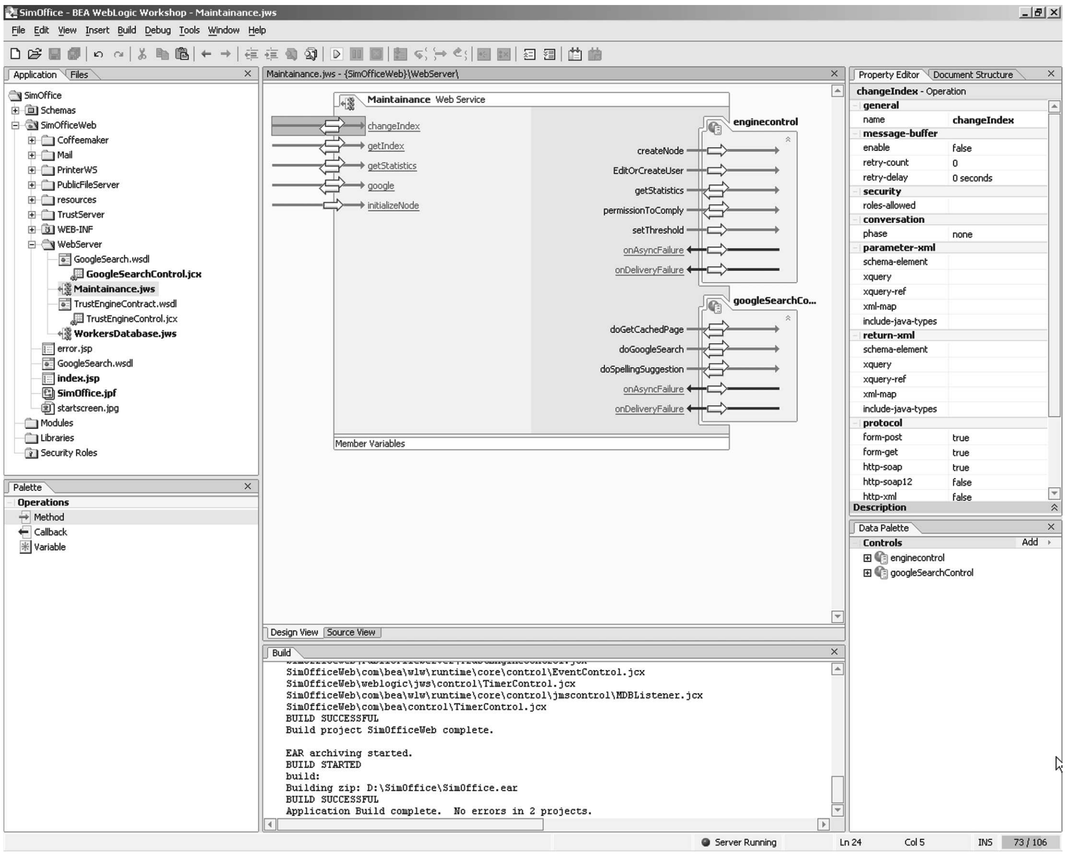
This section deals with the implementation of the *SimOffice* case study, the involved development tools and the different parts it consists of.
Basically, *SimOffice* entitles the environment, including the trust server and all participating nodes. The environment alone is not very ostensive though and that is why a client application was built to give a visual view of all provided features.

The back-end part of the case study implements the functionality discussed in the previous section. It's external appearance is limited to a set of WSDL files for all participating nodes, and the corresponding Web services waiting to accept incoming SOAP messages.

The front-end part of the *SimOffice* case study is an application which utilizes the provided WSDL documents to show how a possible client can look like. It is implemented in a different programming language to demonstrate platform independency again.

5.1.1 Back-end

The Weblogic Workshop 8.1 from Bea Inc. [4] was the development environment of choice for the *SimOffice* back-end. It is a very powerful integrated development environment for Web applications and Web services. Figure 8



changeIndex - Operation

general	
name	**changeIndex**
message-buffer	
enable	false
retry-count	0
retry-delay	0 seconds
security	
roles-allowed	
conversation	
phase	none
parameter-xml	
schema-element	
xquery	
xquery-ref	
xml-map	
include-java-types	
return-xml	
schema-element	
xquery	
xquery-ref	
xml-map	
include-java-types	
protocol	
form-post	true
form-get	true
http-soap	true
http-soap12	false
http-xml	false

```
SimOfficeWeb\com\bea\wlw\runtime\core\control\EventControl.jcx
SimOfficeWeb\weblogic\jws\control\TimerControl.jcx
SimOfficeWeb\com\bea\wlw\runtime\core\control\jmscontrol\MDBListener.jcx
SimOfficeWeb\com\bea\control\TimerControl.jcx
BUILD SUCCESSFUL
Build project SimOfficeWeb complete.

EAR archiving started.
BUILD STARTED
build:
Building zip: D:\SimOffice\SimOffice.ear
BUILD SUCCESSFUL
Application Build complete.  No errors in 2 projects.
```

Figure 8: Bea Weblogic Workshop 8.1

shows the Weblogic desktop during the implementation phase.
Weblogic is compliant with Java 2 Enterprise Edition version 1.3. Web services are developed as Java methods with primitive data types for input and output. As soon as a Java method is marked as a common operation, Weblogic accepts SOAP messages with the corresponding parameters to use the method as a Web service.

Weblogic Workshop provides a built-in server which is permanently updated during the development process. Therefore it is not necessary to separately deploy the Web services during the debug process. It is possible though, to compile the complete application to an EAR archive and deploy it to a production server.
For *SimOffice*, every node including the trust engine was implemented as a single Web service with interfaces for every function. The 'maintenance' Web service for example consists of five methods. To test one of these, Weblogic provides a test browser to access the Web services without the need

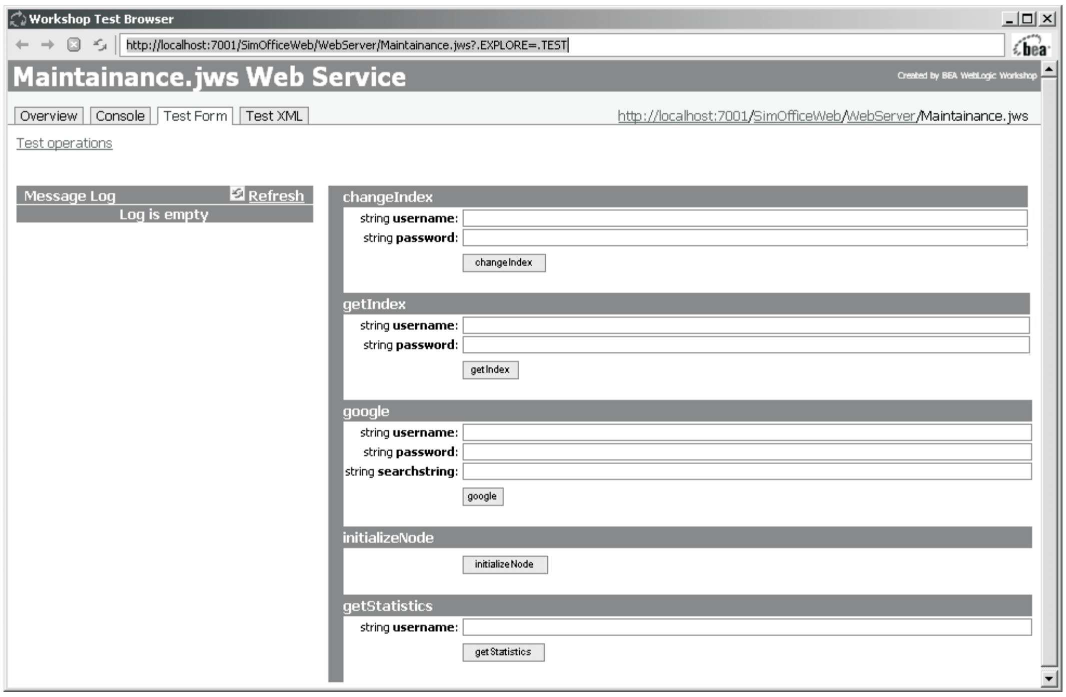
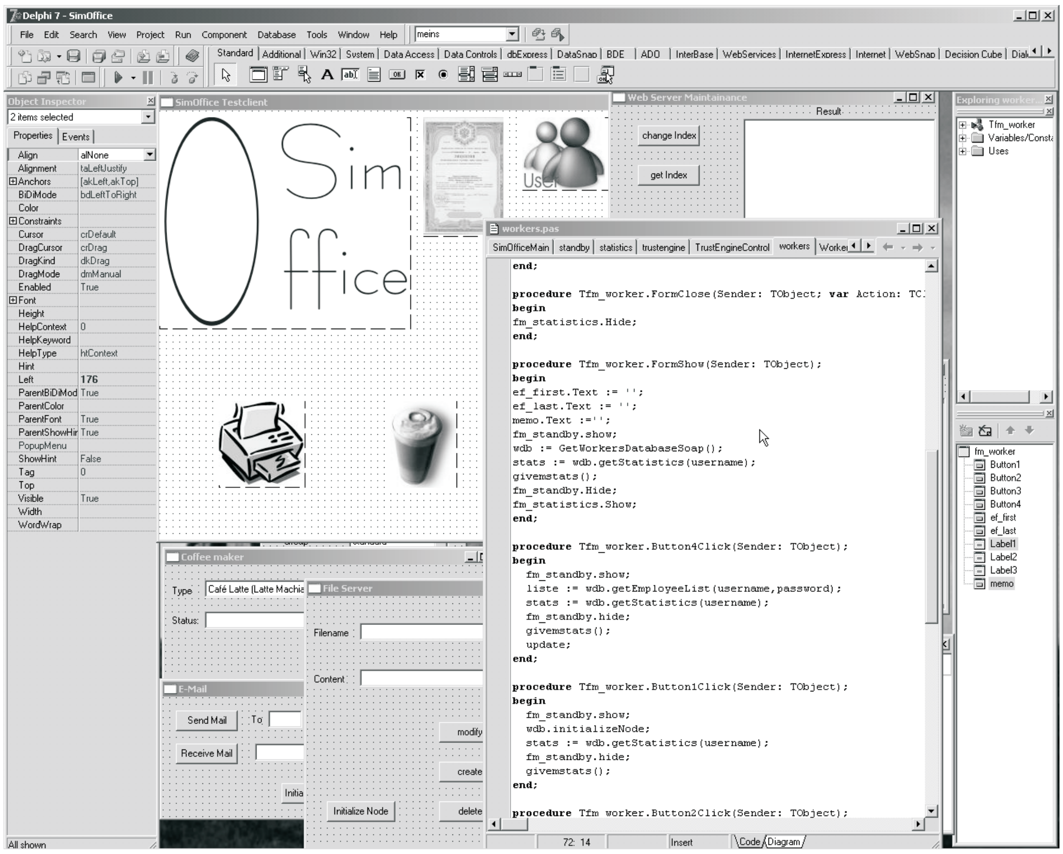


Figure 9: Workshop Test Browser

to implement a test client first. Upon completion of the back-end, a client application was developed to provide a better visual representation of the *SimOffice* environment.

5.1.2 Front-end

The client application which represents the front-end of *SimOffice* is not bound to a program language or platform. To demonstrate the possibilities



Object Inspector — 2 items selected — Properties | Events

Align	alNone
Alignment	taLeftJustify
⊞Anchors	[akLeft,akTop]
BiDiMode	bdLeftToRight
Color	
⊞Constraints	
Cursor	crDefault
DragCursor	crDrag
DragKind	dkDrag
DragMode	dmManual
Enabled	True
⊞Font	
Height	
HelpContext	0
HelpKeyword	
HelpType	htContext
Hint	
Left	**176**
ParentBiDiMod	True
ParentColor	
ParentFont	True
ParentShowHir	True
PopupMenu	
ShowHint	False
Tag	0
Top	
Visible	True
Width	
WordWrap	

All shown

Type: Café Latte (Latte Machia

Status:

Filename:

Content:

```
end;

procedure Tfm_worker.FormClose(Sender: TObject; var Action: TC
begin
fm_statistics.Hide;
end;

procedure Tfm_worker.FormShow(Sender: TObject);
begin
ef_first.Text := '';
ef_last.Text := '';
memo.Text :='';
fm_standby.show;
wdb := GetWorkersDatabaseSoap();
stats := wdb.getStatistics(username);
givemstats();
fm_standby.Hide;
fm_statistics.Show;
end;

procedure Tfm_worker.Button4Click(Sender: TObject);
begin
  fm_standby.show;
  liste := wdb.getEmployeeList(username,password);
  stats := wdb.getStatistics(username);
  fm_standby.hide;
  givemstats();
  update;
end;

procedure Tfm_worker.Button1Click(Sender: TObject);
begin
  fm_standby.show;
  wdb.initializeNode;
  stats := wdb.getStatistics(username);
  fm_standby.hide;
  givemstats();
end;

procedure Tfm worker.Button2Click(Sender: TObject);
```

Figure 10: Borland Delphi 7.0

of the *SimOffice* architecture, the client is implemented in Borland Delphi 7.0 as a single executable.

It is possible to create a WSDL file for every Web service in Weblogic Workshop. Delphi on the other hand, offers the facility to import WSDL documents and create service controls for the contained method descriptions. Finally making the remote method call can therefore be done in two lines of code. The only problem here is the server location which is an integral part of every WSDL document because it defines the URL where the Web service can be reached. When generated in Weblogic, this URL defaults to 'localhost' which is inappropriate unless client and server are running at the same machine. If the server is deployed somewhere else or the client is executed at a remote computer, the 'localhost' in the WSDL files must be changed to the correct IP before they can be used. Figure 11 shows the main screen of

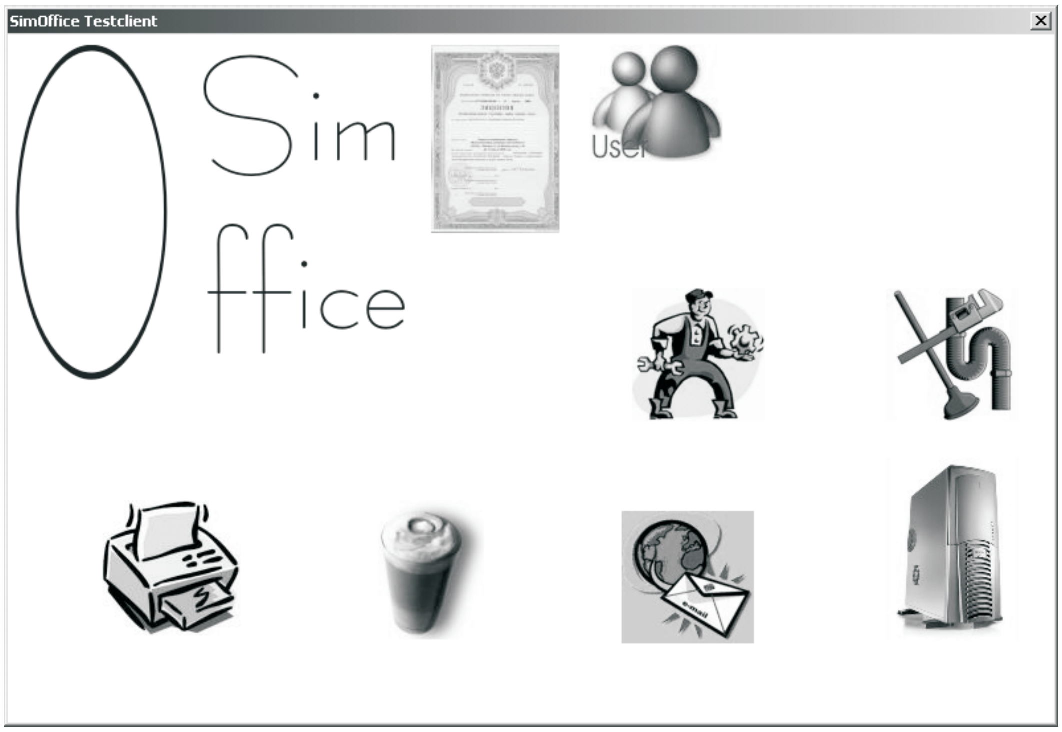


Figure 11: SimOffice client application

the *SimOffice* client application example.

The following section describes the application's functioning in detail.

5.2 Sample Setup

The most descriptive way to explain the finished *SimOffice* case study is a detailed example.
For this purpose, Al is consulted one last time. He heard of Locosoft Unlimited, a company which is looking for a new staff member. He decides to apply for the work and attends the job interview. Meanwhile Locosoft's system administrator completes the last steps for the setup of *SimOffice*, their new trust-based working environment.

5.2.1 Deployment and Server Setup

The *SimOffice* back-end is delivered in a single Web archive with the name SimOfficeWeb.ear. Locosoft's system administrator deploys the application archive at the local production server. After deployment, *SimOffice* is ready to take up nodes. The trust engine offers five methods for server operation:

editOrCreateUser: This method creates a user for *SimOffice* with a password and the associated user group. If the user already exists, password and group are overwritten.

createNode: New nodes are registered with this method. The node has a unique ID as well as node importance and the initial trust value.

setThreshold: This method sets the threshold for a specific service at a given node.

getStatistics: This method returns a list of strings with valuable information about a node and a specified user.

permissionToComply: This is the most important method. Whenever a user tries to access a Web service, the trust engine is asked for permission. Trust levels are altered in this method too, depending on the requested action and the user's access rights.

New nodes can now be added to the environment.

5.2.2 Node Setup

To successfully add a node to *SimOffice*, two steps are necessary:

1. Register the node at the trust engine
To initialize a node at the trust engine, the administrator uses the server's *Create Node* method. It is accessible as a Web service and has to be called, to store the node's importance and initial trust values. After this, the administrator registers every function and the related threshold with the server's *set Threshold* method. To keep track of all values, the administrator decides to create an 'initialize' method at every participating node. This way, new functionality can easily be registered at the trust engine, if the node is extended someday. The initialize method for the Mail server looks like this:

```
public void initializeNode()
{
   enginecontrol.createNode(2,0.4,0.4); // ID=2,I=0.4,T(x)=0.4
   enginecontrol.setThreshold(2,1,0.4); // sendmail
   enginecontrol.setThreshold(2,2,0.6); // receive mail
}
```

This node's ID is 2 and it holds two functions with Thresholds 0.4 and 0.6. The initialize method is not available as a Web service and only the administrator is able to call it. See Appendix A for a lists of initial node configuration.
The second step to complete the node setup is to ask for permission every time a user requests an operation.

2. Add node awareness
This part is quite easy to achieve. Instead of accomplishing the requested operation immediately, the node must first ask the trust engine for the permission to do so. For this purpose, the node invokes the server's *permission to comply* method, with username and password of the requestor as parameters. If the answer is true, the operation is carried out.

The adjusted nodes are now ready to accept requests and therefore, their WSDL descriptions can be published. The administrator makes all files available for download at the company's Intranet. This way they can be used to build personal applications that use the provided functions.

5.2.3 Use Case Scenario

Meanwhile, Al finished his job interview and is hired as a new employee. The system administrator now creates Al's user account for the *SimOffice* environment. Like every new employee, Al is assigned to the 'standard' user group which limits his maximum situational trust to 0.5. Al knows nothing about this. All he sees is the SimOffice client application that is installed on his desktop and an e-mail from his system administrator with his access data. He decides to give it a try, logs in with his user data and tries to order a black coffee (see Table 6). His request arrives at the trust engine and initiates the following order of events:

- Al is unknown to the coffee maker. Therefore his initial trust is set to $T_x = 0.3$.
- The trust engine asks every known node for recommendation but Al has not caused any events so far. Thus his initial trust value is not altered in any way.
- The situational trust for the coffee maker is now computed as: $T_x(Al) = T_x + (T_x * (T_x - I_x)) = 0.3 + (0.3 * (0.3 - 0.5)) = \underline{0.24}$.
- The situational trust value is checked against the threshold for black coffee which is 0.5. Al is not authorized to order black coffee. The coffee maker refuses to accomplish his request.
- Al's unauthorized request demands punishment and so his initial trust is altered as:
  $T_{x_{bad}} = 1 - e^{-\frac{\ln(1-T_x)}{c*\ln(1-T_x)-1}}$
  with $c = s * I_x = 0.1 * 0.5 = 0.05$
  $\Rightarrow T_{x_{bad}} = 1 - e^{-\frac{\ln(1-0.3)}{0.05*\ln(1-0.3)-1}} = \underline{0.2956}$

Figure 12 visualizes the trust levels for Al within the SimOffice environment. All Al sees, is that the coffee maker does not trust him. A little confused, he

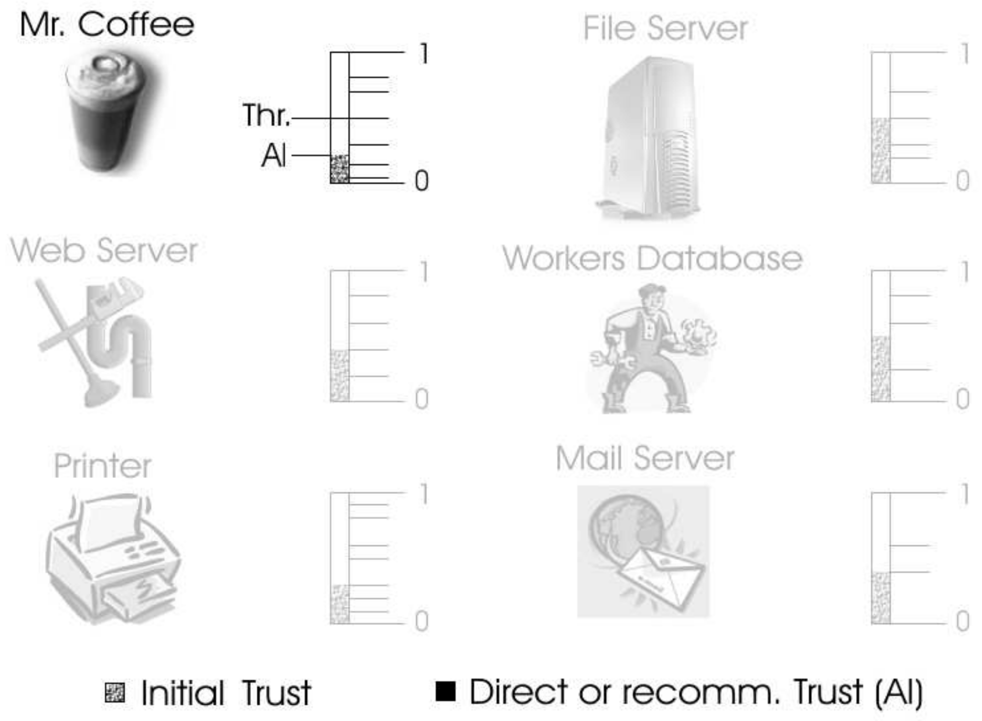


Figure 12: Scenario 1 - Initial trust

tries to order a Café Latte and he succeeds since the threshold is only 0.15 for this one.
One month and 31 Café Latte later, Al tries to order a black coffee again. Now the situation is different:

- Al's month of good behavior at the coffee maker raised his trust level to $T_x = 0.65$.
- Again, the trust engine asks for recommendation but Al has not used another node yet. Therefore his initial trust value is not altered.
- The situational trust for the coffee maker is computed as:
  $T_x(Al) = T_x + (T_x * (T_x - I_x)) = 0.65 + (0.65 * (0.65 - 0.5)) = \underline{0.7475}$.
  He is a member of the 'standard' user group, so his situational trust is

trimmed to $T_x(y) = 0.5$.

- The situational trust value is checked against the threshold for black coffee which is 0.5 too. Now Al is authorized to order this coffee and his request is carried out.
- Al's authorized request demands reward and so his initial trust is altered as:
  $T_{x_{good}} = e^{-\frac{\ln(T_x)}{c*\ln(T_x)-1}}$
  with $c = s*(1 - I_x) = 0.1*(1 - 0.5) = 0.05$
  $\Rightarrow T_{x_{good}} = e^{-\frac{\ln(0.65)}{0.05*\ln(0.65)-1}} = \underline{0.6559}$

While sipping his coffee, Al decides to try one of SimOffice's other function-

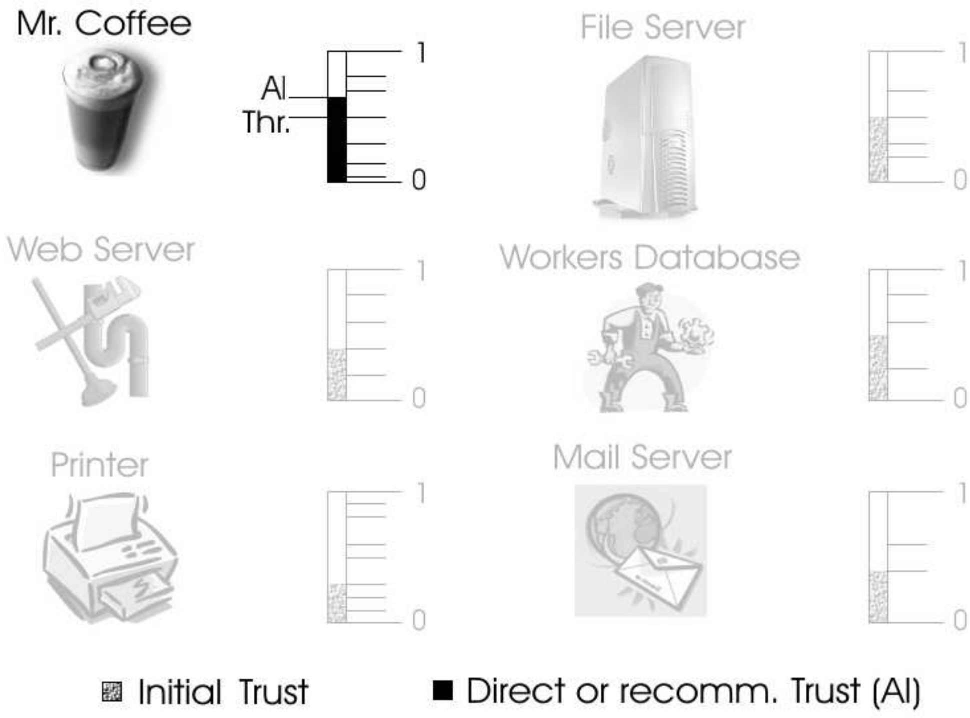


Figure 13: Scenario 2 - Direct trust

alities. He connects to the printer and tries to print 15 copies of the latest joke he received as a junk mail. He submits his request, causing the following order of events:

- Al is unknown to the printer at the moment. According to Table 7, his initial trust value is set to $T_x = 0.3$.
- The trust engine asks for recommendation at all known nodes and because the coffee maker already knows Al, his initial trust is altered as follows:
  $T_{printer} = T_{init} + \frac{I_{coffee} * (T_{coffee}(Al) - T_{init})}{2} = 0.3 + \frac{0.5*(0.6559-0.3)}{2} = 0.389$
- Now the situational trust can be calculated:
  $T_x(Al) = T_x + (T_x * (T_x - I_x)) = 0.389 + (0.389 * (0.389 - 0.5)) = \underline{0.346}$.
- The threshold to print between 11 and 20 pages is set to 0.3. Without the coffee maker's recommendation, Al would not have been able to print the pages but his good behavior enabled him to do so.
- Al's authorized request is again rewarded, by adjusting his trust to:
  $T_{x_{good}} = e^{-\frac{\ln(T_x)}{c*\ln(T_x)-1}}$
  with $c = s * (1 - I_{printer}) = 0.1 * (1 - 0.5) = 0.05$
  $\Rightarrow T_{x_{good}} = e^{-\frac{\ln(0.389)}{0.05*\ln(0.389)-1}} = \underline{0.404}$

Figure 14 visualizes the influence of one recommendation for the printer.

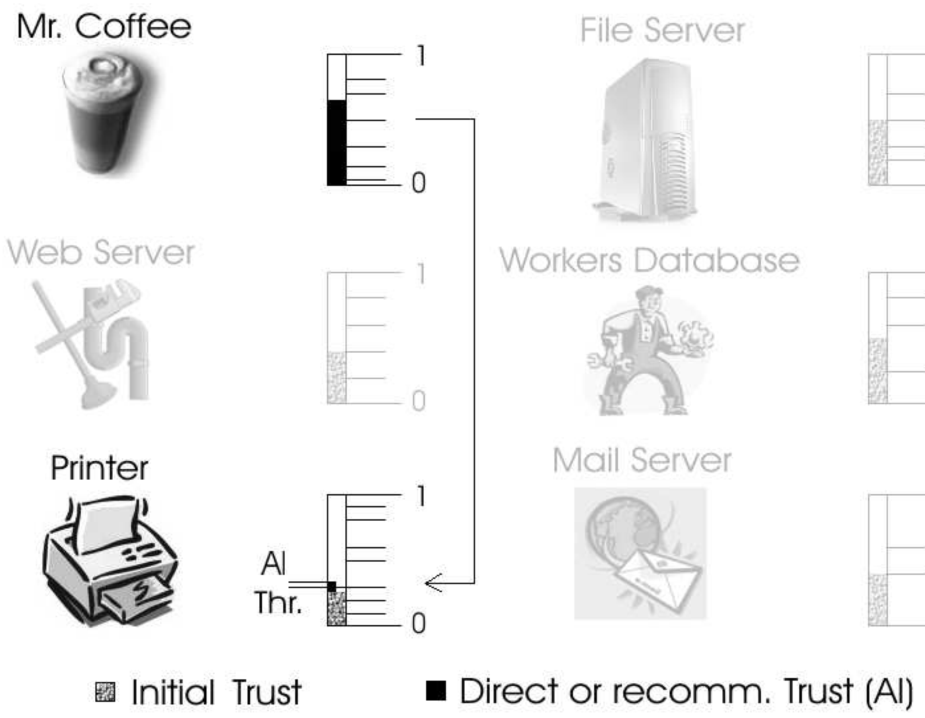


Figure 14: Scenario 3 - One recommendation

Happy with his funny mail, Al continues his work. After some time, his superior assigns him the task to access the worker's database and fire Bob. For this purpose the administrator is told to assign temporary superuser rights to Al. Again, Al logs in and accesses *SimOffice* as he used to do. The events he causes are a bit different this time:

- Al's Identity is checked with the provided username and password.
- The trust engine recognizes Al as a member of the superuser group and immediately grants permission for the requested action. Bob is fired without further questions.
- No adjustment of Al's trust value is carried out, because he is unable to cause malevolent actions and therefore no reward (or punishment)

is necessary.

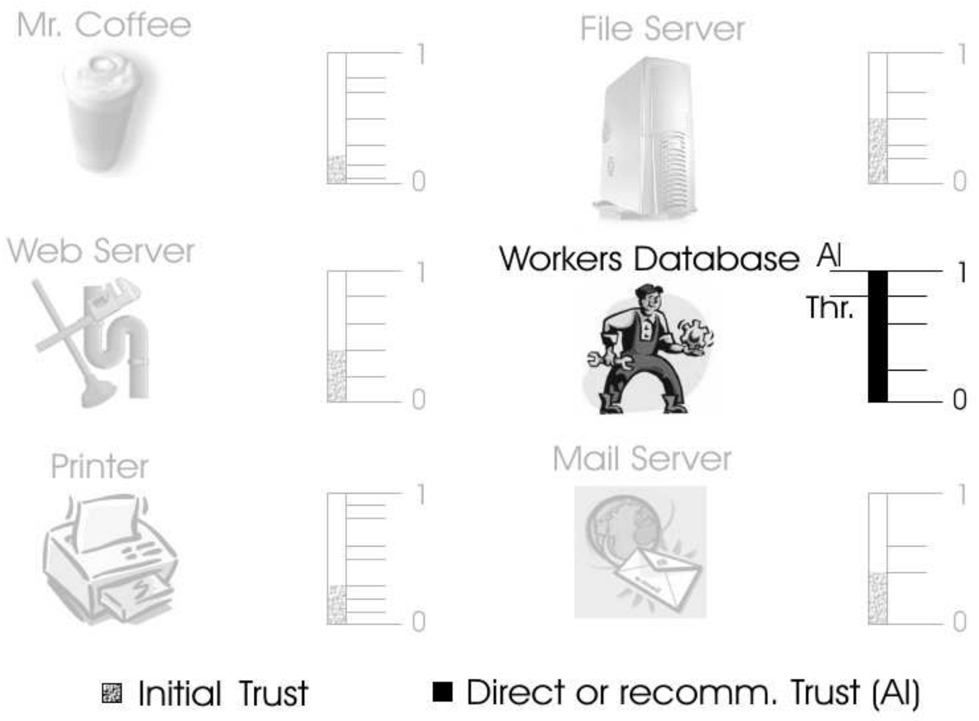


Figure 15: Scenario 4 - Superuser

One year passes and Al is promoted to a higher position. At the same time his SimOffice account is changed to a member of the 'advanced' group, limiting his maximum situational trust to 0.7 now. He starts to grow tired of black coffee and so he tries to order a moccachino. He receives the following result:

- Al's reputation at the coffee maker is legendary. Over the year, his initial trust grew to $T_x = 0.7$.
- The trust engine does not ask other nodes for their opinion because Al is already known here.
- The situational trust for Al is:
  $T_x(Al) = T_x + (T_x * (T_x - I_x)) = 0.7 + (0.7 * (0.7 - 0.5)) = \underline{0.84}$. This

would be enough for the strong mocca, but his user group trims the situational trust to 0.7 again.

- The situational trust value is checked against the threshold for the moccachino which is 0.7. Luckily, Al is able to match the threshold and his coffee is granted.
- Al's request with the new group membership causes the trust engine to alter his trust value to:
  $T_{x_{good}} = e^{-\frac{\ln(T_x)}{c*\ln(T_x)-1}}$
  with $c = s * (1 - I_{coffee}) = 0.1 * (1 - 0.5) = 0.05$
  $\Rightarrow T_{x_{good}} = e^{-\frac{\ln(0.7)}{0.05*\ln(0.7)-1}} = \underline{0.7043}$

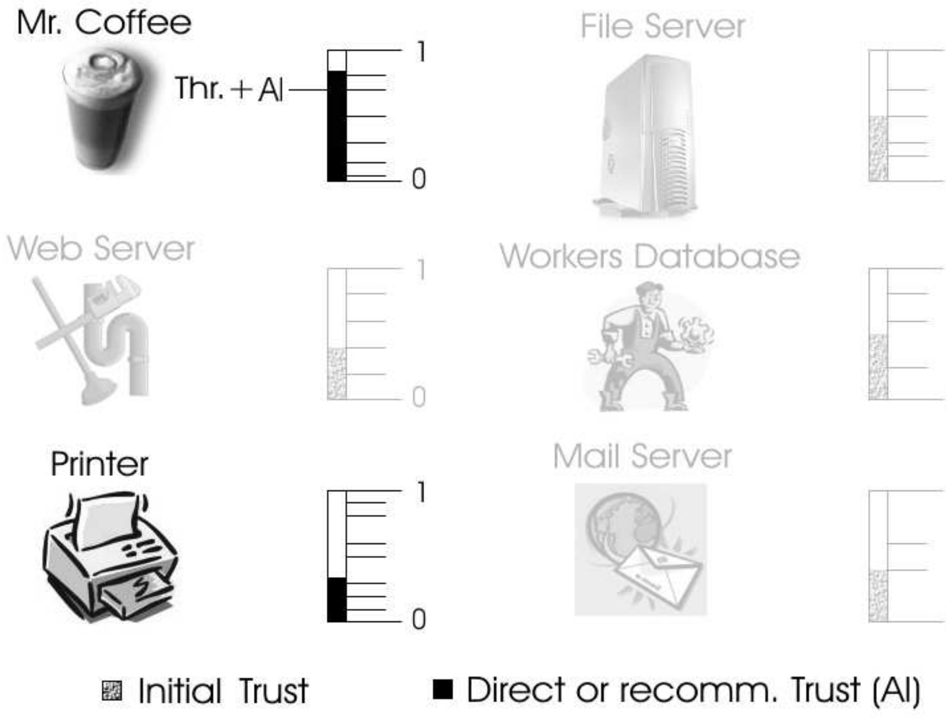


Figure 16: Scenario 5 - Group dependent trust limit 0,7

One day, when Al is promoted again and his group membership is changed to 'full user', he will eventually be able to order the extra strong mocca.
To take some further advantage of his new position, Al decides to use the Mail service provided by SimOffice and write a mail. Now he is known at more than one point within the SimOffice environment which leads to the following cycle:

- Al never interacted with the mail server before. Therefore his trust level is set to the initial trust value of $T_x = 0.4$ for this node (see Table 3).

- The trust engine asks the existing nodes for their recommendation trust about Al. At this point he is known at three other nodes and every one is taken into consideration, altering his initial trust to:

  Coffee maker: With $I_{coffee} = 0.5$ and $T_{coffee}(Al) = 0.706$ Al's initial Trust is altered to:
  $T_{mail} = T_{init} + \frac{I_{coffee} * (T_{coffee}(Al) - T_{init})}{2} = 0.4 + \frac{0.5*(0.706-0.4)}{2} = 0.4765$.

  Printer: With $I_{printer} = 0.5$ and $T_{printer}(Al) = 0.803$ Al's initial Trust is altered to:
  $T_{mail} = T_{mail} + \frac{I_{printer} * (T_{printer}(Al) - T_{mail})}{2} = 0.4765 + \frac{0.5*(0.803-0.4765)}{2} = 0.5581$.

  File Server: With $I_{file} = 0.7$ and $T_{printer}(Al) = 0.7588$ Al's initial Trust is finally altered to:
  $T_{mail} = T_{mail} + \frac{I_{file} * (T_{file}(Al) - T_{mail})}{2} = 0.5581 + \frac{0.7*(0.7588-0.5581)}{2} = 0.6283$.

- The situational trust for Al is:
  $T_x(Al) = T_x + (T_x * (T_x - I_x)) = 0.6283 + (0.6283(0.6283 - 0.4)) = \underline{0.77}$.
  As an 'advanced' user, this value is trimmed to 0.7 again.

- The situational trust value is checked against the thresholds for the operation he requested which is 0.6. The recommendation from all the other nodes was so high, that he is instantly able to use all functions of the mail server.

- Al's first request at the mail server raised his trust value from initially 0.4 to:
  $T_{x_{good}} = e^{-\frac{\ln(T_x)}{c*\ln(T_x)-1}}$
  with $c = s*(1-I_{mail}) = 0.1*(1-0.4) = 0.06$
  $\Rightarrow T_{x_{good}} = e^{-\frac{\ln(0.0.6283)}{0.06*\ln(0.0.6283)-1}} = \underline{0.631}$

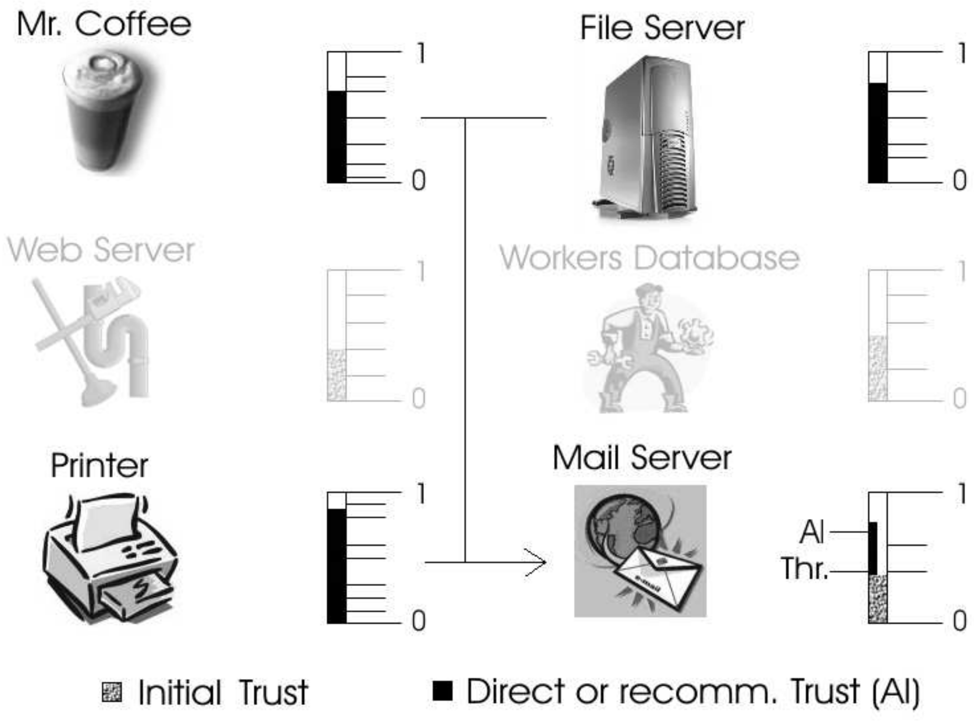


Figure 17: Scenario 6 - Three recommendations

Figure 17 shows the high impact of three recommending sources for the Mail service. The initial trust value is almost doubled because of Al's good behavior at the other services.

After his good experiences with Locosoft and SimOffice, Al decides to change his job and gather some additional experience in another company. The day he leaves, the administrator deletes his user account for SimOffice which includes the experiences at the nodes he already made. Should Al ever decide to come back to Locosoft, he would have to start his journey all over again. But before he decides that, he better gets used to Cappuchino again.

With this last cycle, the most common events happening within *SimOffice* were displayed. Based on this experience, the following section will give an evaluation of the presented work and ideas for further extensions.

6 Evaluation

6.1 Concept-related Aspects

With the implemented *SimOffice* environment, it is obvious that the overall concept works. A far more important question though is whether the whole concept even makes sense.
This section mentions the major problems arising in the *SimOffice* environment and how a possible solution can look like. Furthermore some of the major advantages of the whole concept are discussed.

6.1.1 Repeated Actions

The easiest way to trick *SimOffice* is with repeatedly executed actions. Once a user has access to a specific action, it is treated as trustworthy behavior when the function is executed. In junction with Web services as the underlying technology, this fact is like a death blow for a trust-based architecture. For example, the moment an agent has access to the file server's modify-function, he could use the WSDL file and easily write a program that modifies the same file over and over again. Every access would represent trustworthy behavior, raising the concerning trust level to values like 0.9 or higher without even clicking a button. This problem is not a result of the concept. It is simply caused by a lack of properly rated actions. *SimOffice* is purely action based. That means only actions or called methods are able to influence a users trust level. For some nodes this is actually sufficient. The coffee maker is a good example. If someone likes to order 1000 Cappuchinos only to raise his trust level, he may do so. The file server on the other hand is different. Like mentioned above, it may matter in which time intervals accesses occur.

The solution for this problem is a proper node setup. Some nodes will demand to track time to prevent exploiting the trust based system. The printer

for example may limit it's printing capacity to a specified amount per day instead of pages printed at one time. The file server could track the intervals between file accesses and then conclude if the behavior is trustworthy or not. Another solution is, to exclude certain functions from awarding and punishing operations. This primarily affects functions for status retrieval or polling operations. The functions are still limited by a user's trust level and are therefore accessible with a sufficient trust level but invocation of the operation does not affect the trust level. This way only 'important' operations are subject to the punishment/award system.
Giving a general solution for this problem is impossible, because every node needs individual setup to represent human trust as accurately as possible.

6.1.2 Opportunity for Errors

This issue is closely related to the problem above. In an action-based environment it is impossible to behave untrustworthy, if the trust level is higher than the highest threshold. Again, a more accurate node setup will solve this problem. Even if an agent is able to print 200 pages at once, he can still be treated untrustworthy when his daily limit is exceeded. A well designed node always offers a chance for bad behavior as long as the trust level is below 1 (blind trust). The more things influence the final assessment, the harder the system is to compromise.

6.1.3 The Situational Trust Value

The situational trust value as described in Section 3.3 has the purpose to alter the trust in a specific agent, according to the importance of the node. The function however, does not limit the result to an interval between $(0, 1)$. As a result, the situational trust which is calculated with every new transaction, distorts the exponential function for reward an punishment.
The trust characteristic looks even better when the situational trust value is omitted and the direct trust in the agent respectively the recommended trust value, is used to compare to the threshold instead. The node's importance still influences the reward and punishment functions to make sure that trust levels on more important nodes do not raise as fast as trust levels of unimportant nodes. This way it is impossible to obtain values above one or lower than zero.

Omitting the situational trust value for an assessment will be sufficient for most applications. Which method is finally used, does not influence the systems behavior. It merely presents an additional way to balance the entire environment.

6.1.4 Guest accounts

Not implemented in the *SimOffice* environment is a facility for guest accounts. A guest account would be used for unrecognized usernames and/or passwords. In this case, instead of refusing access to *SimOffice*, the user could be treated as a guest, very similar to the administrator, where the trust level is a fixed value like 0.2 for instance. Thus, enabling unregistered users to access functionality with low priority. *SimOffice* basically represents a closed environment with one network and no access control based on location. A guest account is a convenient way to enable access to the network from the Internet for example. Every request that arrives at the Web server could be treated as a guest and therefore grant access to some basic functions.

6.1.5 Callbacks

More complex operations that need some time to execute are currently designed with an extra function to poll the status. A much better solution for time-intensive actions is the 'callback'. Callbacks invoke a Web service at the client application upon completion of the requested action. The advantage is, that the client application does not block until an answer is received. An example for a callback is the coffee maker. Right now, it is implemented with a function to poll the status of the coffee maker. By using a callback service, the coffee maker would be able to notify the requestor when the coffee is ready.
Unfortunately, callbacks are not widely supported yet. Delphi for example, does not support callbacks and therefore a design which includes them would again limit the usability for some program languages.

6.1.6 Privacy

Another arising question concerns privacy. Users tend to feel offended when every single action they make is traced by a system administrator. This is possible in *SimOffice* too, but a balanced system will not frequently demand human interaction except for account management. As a result, every user is judged the same way and it does not matter if the user's current trust level is a result of frequent misbehavior or just infrequent usage (with low start level). So a trust-based access control in fact improves privacy for the users.

6.1.7 Node extension

Adding new nodes to the system is very easy and does not require a server downtime. This important point is one of the major benefits of the concept and a basic requirement for a dynamic system. This feature could be extended to a system, where nodes are added and removed ad-hoc. The result would be an environment, where users are able to register their own Web services at the server and provide their functionality to the network. On the other hand, nodes would become subject to trust-based assessment too, because their functionality is not guaranteed and therefore not initially trusted. The trust engine would react by deleting a node's experiences as soon as the node is removed. Upon re-registration, this experiences would have to be built up again.

An ad-hoc capable system based on Web services causes a serious problem for the users nevertheless. Building distributed applications based on a system with unknown availability is almost impossible. An application using 10 different Web services for example, would cease function if one of the services is not functional. This is the reason why *SimOffice* was built as a static environment with guaranteed availability of it's components.

6.2 Future Work

6.2.1 P2P Architecture

Many extensions of *SimOffice* are possible in the future. A big task is to change the architecture to a pure peer-to-peer environment, where no server presents a single point of failure. The arising problems with user accounts and peer awareness are challenging tasks for further research. Two possible approaches are described below:

Broadcast Messages: A common problem along peer-to-peer architectures is how to pass on data and reach every node within the system. One way to replace the trust server, is to transfer trust- and user-related data with broadcast messages. The node itself judges a requesting agent based on it's own experience and recommendation from other nodes. This method generates a massive amount of network traffic and is therefore not suitable for large networks. Additionally, the environment is restricted to a single network and therefore subject to it's boundaries. Only nodes within the same sub net can be addressed.

Node List: Far less network traffic would be caused by a system, where every node keeps a list of participating nodes. In addition to the algorithm necessary for judgment, every node must also comprise a facility to decide which nodes to ask for a possible recommendation. Furthermore, an algorithm to find other nodes and make a node known to the network must be implemented for every agent.

Both approaches require a higher amount of processing power than a server-based architecture. Furthermore, authenticating a user is still a problem unless a certification authority is involved. In this case a single point of failure would still be present.
Although finding a solution for this problems is not part of this thesis, it is interesting to think about the complexity of these issues.

6.2.2 Content-based Judgment

Another idea is to extend the conditions for trustworthy and untrustworthy behavior to content-related aspects as well. Although it reaches more to the field of artificial intelligence, it would be an interesting extension nevertheless. In such a system, the judgement would be based on the actions taken, as well as the related content that is produced or consumed. In case of the File server, a content-aware trust engine would keep track of the files a user tries to access and decide how to deal with this user. Even if the user's trust level is high enough to read files, the trust engine could decide to deny access to some files because of their confidential content. Deciding if a file is confidential or not, is a most challenging task and object for further research.

With all the mentioned aspects, *SimOffice* is still an innovative way to control access within a distributed federation of Web services. Although the implemented framework is far from being perfect, it nevertheless points the direction for trust-controlled access monitoring in tomorrow's computer systems. It's extendability and ability to adept different environments, makes *SimOffice* a fitting solution to secure various kinds of computer systems.

7 Conclusion

The *SimOffice* environment developed in the course of this diploma thesis represents a trust-based access control for federated Web services. The design is based on exponential functions to adjust trust levels according to a user's behavior. A trust engine was designed that comprises an algorithm to assess incoming requests and permit or deny access to the demanded resource. A facility to add, delete and modify users, passwords and group membership was created.
The implemented case study showed that *SimOffice* offers an extensible solution for various environments. The concept is independent of platform and programming language and offers a flexible way to connect applications through Web services. *SimOffice* comprises a trust server which dynamically administrates participating nodes.
The concept describes an innovative approach for automatic access control and presents two algorithms for trust adjustment.

For future distributed systems, trust will constitute an upcoming method to build behavior-aware architectures. This new perspective will enable developers to keep pace with the growing amount of administrative complexity.

A Appendix - Node configuration

Variable	**Value**
Initial trust value	0.4
Importance	0.4
Threshold for receive mail	0.4
Threshold for send mail	0.6

Table 3: Mail Server Setup

Variable	**Value**
Initial trust value	0.4
Importance	0.3
Threshold for google search	0.2
Threshold for get index	0.4
Threshold for set index	0.6

Table 4: Web Server Maintenance Setup

Variable	**Value**
Initial trust value	0.5
Importance	0.8
Threshold for list of employees	0.3
Threshold for hire	0.6
Threshold for fire	0.8

Table 5: Workers Database Setup

Variable	**Value**
Initial trust value	0.3
Importance	0.5
Threshold for status retreival	0.05
Threshold for Cafe Latte	0.15
Threshold for Cappuchino	0.3
Threshold for black Coffee	0.5
Threshold for Moccachino	0.7
Threshold for extra strong Mocca	0.8

Table 6: Mr. Coffee Setup

Variable	**Value**
Initial trust value	0.3
Importance	0.5
Threshold for one copy	0.1
Threshold for 2-10 copies	0.2
Threshold for 11-20 copies	0.3
Threshold for 21-50 copies	0.5
Threshold for 51-100 copies	0.6
Threshold for more than 100 copies	0.8
Threshold for real print	0.9

Table 7: Printer Setup

Variable	**Value**
Initial trust value	0.5
Importance	0.7
Threshold for file list retrieval	0.2
Threshold for modify	0.3
Threshold for create	0.5
Threshold for delete	0.7

Table 8: File Server Setup

References

[1] A. Abdul-Rahman. Supporting trust in virtual communities, 1998.

[2] M. D. Abrahams. Trusted system concepts, 1995.

[3] P. Bateson. The biological evolution of cooperation and trust, 2000.

[4] Bea Systems Inc. *Bea Weblogic Workshop 8.1, http://www.bea.com/*, 2002.

[5] T. Beth, M. Borcherding, and B. Klein. Validation of trust in open networks, 2000.

[6] M. Blaze, J. Feigenbaum, and P. Resnick. Managing trust in an information-labeling system, 1996.

[7] Center for Education and Research in Information Assurance and Security (CERIAS). *Formalizing Trust, Fraud, and Vulnerability, http://www.cs.purdue.edu/homes/bb/NSFtrust.html*, 2003.

[8] T. Clements. Overview of SOAP. *Java Developers Forum, http://java.sun.com/developer/technicalArticles/xml/webservices/*, 2001.

[9] P. Dasgupta. Trust as a commodity.

[10] T. Dimitrakos. A Service-Oriented Trust Management Framework. In *Trust, Reputation, and Security: Theories and Practice*, pages 53–72. Rino Falcone, Suzanne Barber, Larry Korba and Munindar Singh, 2003.

[11] J. Dunn. Trust and political agency, 2000.

[12] C. English, P. Nixon, and S. Terzis. Dynamic trust models for ubiquitous computing environments, 1996.

[13] L. Eschenauer, V. D. Gligor, and J. Baras. On trust establishment in mobile ad-hoc networks, 2002.

[14] D. Gambetta. *Can We Trust Trust?*, chapter 13, pages 213–237. Basil Blackwell, 1988. Reprinted in electronic edition from Department of Sociology, University of Oxford.

[15] D. Good. Individuals, interpersonal relations, and trust, 2000.

[16] Google Inc. *Google Web Api's Developers Kit, http://www.google.com/apis/*, 2004.

[17] E. Gray, P. O'Connel, C. Jensen, S. Weber, J. Seigneur, and C. Yong. Towards a framework for assessing trust-based admission control.

[18] E. Gray, J.-M. Seigneur, Y. Chen, and C. Jensen. Trust propagation in small worlds.

[19] IBM. *Understanding WSDL in a UDDI registry, http://www-106.ibm.com/developerworks/webservices/library/ws-wsdl/*, 2002.

[20] IBM. *Web Services Trust Language (WS-Trust), http://www.ibm.com/developerworks/library/ws-trust/index.html*, 2002.

[21] IBM. *Web Services Federation Language (WS-Federation), http://www.ibm.com/developerworks/library/ws-fed/*, 2003.

[22] IBM. *Web Services Security (WS-Security), http://www-106.ibm.com/developerworks/webservices/library/ws-secure/*, 2003.

[23] C. E. Irvine. A national trusted computing strategy, 2002.

[24] W. King. The prisoners' dilemma.

[25] P. Lamsal. Understanding trust and security, Oct. 2001. http://www.cs.Helsinki.FI/u/lamsal/papers/UnderstandingTrustAndSecurity.pdf.

[26] N. Luhmann. Familiarity, confidence, trust: Problems and alternatives, 2000.

[27] S. Marsh. Investigating trust between users and agents in a multi agent portfolio.

[28] S. Marsh. Trust and reliance in multi agent systems, 1992.

[29] S. Marsh. Trust in distributed artificial intelligence. In *Modelling Autonomous Agents in a Multi-Agent World*, pages 94–112, 1992.

[30] S. Marsh. Formalising trust as a computational concept, 1994.

[31] Microsoft. *Microsoft UDDI Business Registry Node, http://uddi.microsoft.com/inquire*, 2004.

[32] Microsoft. *Microsoft UDDI Business Registry Node: Web Search, http://uddi.microsoft.com/search*, 2004.

[33] S. Mishra and R. D. Schlichting. Abstractions for constructing dependable distributed systems, 1992.

[34] Network Working Group. *Hypertext Transfer Protocol – HTTP/1.1, ftp://ftp.rfc-editor.org/in-notes/rfc2068.txt*, 1997.

[35] P. G. Neumann. Practical architectures for survivable systems and networks, 2000.

[36] P. Nikader and K. Karvonen. Users and trust in cyberspace, 2001.

[37] S. Overhagen and P. Thomas. WS-Specification: Specifying Web Services Using UDDI Improvements. In *Web, Web-Services, and Database Systems*, pages 100–110. Akmal B. Chaudrin, Mario Jeckle, Erhard Rahm and Rainer Unland, 2002.

[38] P. F. Pires, M. R. Benevides, and M. Mattoso. Building Reliable Web Services Compositions. In *Web, Web-Services, and Database Systems*, pages 59–72. Akmal B. Chaudrin, Mario Jeckle, Erhard Rahm and Rainer Unland, 2002.

[39] S. Robak and B. Franczyk. Modeling Web Services Variability with Feature Diagrams. In *Web, Web-Services, and Database Systems*, pages 120–128. Akmal B. Chaudrin, Mario Jeckle, Erhard Rahm and Rainer Unland, 2002.

[40] N. Shankar and W. A. Arbaugh. On trust for ubiquitous computing.

[41] UDDI.org. *UDDI Technical White Paper, http://www.uddi.org/pubs/Iru-UDDI-Technical-White-Paper.pdf*, 2000.

[42] UDDI.org. *Using WSDL in a UDDI Registry, http://www.uddi.org/pubs/wsdlbestpractices-V1.07-Open-20020521.pdf*, 2000.

[43] B. Williams. Formal structures and social reality, 2000.

[44] World Wide Web Consortium (W3C). *Extensible Markup Language, http://www.w3.org/XML/*, 1996.

[45] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/wsdl/*, 2001.

[46] World Wide Web Consortium (W3C). *SOAP Version 1.2: Messaging Framework, http://www.w3.org/TR/SOAP/*, 2003.

[47] R. Yahalom. Trust relationships in secure systems, a distributed authentication perspective, 1999.

[48] R. Yahalom, B. Klein, and T. Beth. Trust relationships in secure systems–A distributed authentication perspective. In *RSP: IEEE Computer Society Symposium on Research in Security and Privacy*, 1993.

[49] Y. Zhang. Intrusion detection in wireless ad-hoc networks, 2000.

[50] L. Zhou. Securing ad hoc networks, 2000.