

Master Thesis
Computer Science
Thesis no: MCS-2005-15
08 2005



S-UDDI

- discovering Web services, the Secure and Trustworthy way

Fredrik Erlandsson, Daniel Evertsson

Department of
Interaction and System Design
School of Engineering
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden

This thesis is submitted to the Department of Interaction and System Design, School of Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full time studies.

Contact Information:

Authors:

Fredrik Erlandsson

Address: Fogdevägen 5A, 371 40 Karlskrona

E-mail: fredrik.e@gmail.com

Daniel Evertsson

Address: Gyllenstjärnas väg 9A, 371 40 Karlskrona

E-mail: daniel@evertsson.nu

External advisor:

Peter Bayer

AerotechTelub AB

Address: Ljungadalsgatan 2, 351 80 Växjö

Phone: +4647042000

University advisor:

Per Mellstrand

Department of Interaction and System Design

**Department of
Interaction and System Design
Blekinge Institute of Technology
Box 520
SE – 372 25 Ronneby
Sweden**

**Internet : www.bth.se/tek
Phone : +46 457 38 50 00
Fax : +46 457 102 45**

ABSTRACT

SOA and especially Web services are a big evolving market these days. SOA typically uses Web services when interacting between different parts of applications. Methods to easily discover the Web services must exist. For this UDDI has been introduced. The current implementation of UDDI has a weak security model. We have developed an extension to this model, which interacts with the current solution to make it more secure. Our solution, S-UDDI, enable a way to find and publish Web services in a secure and trustworthy way.

Keywords: UDDI, Web services, Security

CONTENTS

<u>1</u>	<u>INTRODUCTION</u>	<u>1</u>
1.1	AIM, OBJECTIVES AND RESULT	1
1.2	OUTLINE	2
<u>2</u>	<u>BACKGROUND</u>	<u>3</u>
2.1	SERVICE ORIENTED ARCHITECTURE	3
2.2	PRINCIPAL WEB SERVICES ACTORS	4
2.3	FUNCTION OF UDDI	4
2.4	WEB SERVICES AND UDDI	5
2.5	UDDI ARCHITECTURE	6
<u>3</u>	<u>THREAT MODELLING</u>	<u>8</u>
3.1	THREAT AGENTS	8
3.2	THREAT	8
3.3	INFERENCE	9
<u>4</u>	<u>CONTRIBUTION</u>	<u>10</u>
4.1	ARCHITECTURE	10
4.2	INTERFACE	11
4.3	DEFENCE PRINCIPALS	12
4.4	POLICIES	13
4.5	SUMMARY	14
<u>5</u>	<u>EVALUATION</u>	<u>16</u>
5.1	THREAT MODELLING S-UDDI	16
5.2	THREAT MAPPING	17
5.3	THREAT MODELLING UDDI v.3	17
5.4	COMPARISONS	18
<u>6</u>	<u>CONCLUSIONS</u>	<u>19</u>
<u>7</u>	<u>REFERENCES</u>	<u>20</u>

LIST OF TABLES

<i>Table 1 – Threat agents</i>	8
<i>Table 2 – Identified threats in UDDI</i>	8
<i>Table 3 – Summary of security mechanisms</i>	15
<i>Table 4 – Identified threats in S-UDDI</i>	16
<i>Table 5 – Mapping of UDDI threats to security mechanisms introduced in S-UDDI</i>	17
<i>Table 6 – Identified threats in UDDI v.3</i>	17

LIST OF FIGURES

<i>Figure 1 – Web service actors and components</i>	4
<i>Figure 2 – Architecture of UDDI</i>	6
<i>Figure 3 – The UDDI data model</i>	7
<i>Figure 4 - S-UDDI Architecture</i>	10
<i>Figure 5 – Flow chart of S-UDDI</i>	11
<i>Figure 6 – Levels of security</i>	12
<i>Figure 7 – UDDI requirements</i>	13

ABBREVIATIONS

API – Application Programming Interface
GUI – Graphical User Interface
HTML – Hyper Text Markup Language
SOA – Service Oriented Architecture
SSL – Secure Socket Layer
tModel – Technical Model
UDDI – Universal Description Discovery and Integration
UUID – Universally Unique Identifier
URL – Uniform Resource Locator
WSDL – Web services Description Language
WWW – World Wide Web
XML – eXtensible Markup Language

1 INTRODUCTION

The network revolution made it possible to exchange information between applications. This resulted in a series of new services such as e-mail and later led to the evolution of what we today call the web. The services on the web are passive to that degree that the service requests information from a server processes the answer and presents the information to the user.

To make it possible to use more advanced services on the network, sometimes Web services are used. These services are applications running on a server and can be very complex because of the fact that they may use information both from the host computer and the requesting client when processing data or performing calculations.

Conventional passive services are often found through knowledge of the address to each service. Web services are easily found by knowing only one address, the address to the register service. This register allows registered users to add whatever service the user wants. All information in the registry is by default open to be read by everyone. When a user authenticates itself to the register service, username and password are sent over the Internet in plain text. This means that it is trivial for a dishonest user to retain a valid user account and add malicious services to the registry. This is a problem in many networks because as a network user, you need to be able to trust the service not to be malicious. It may be hard to decide if a service is malicious or not, especially if you cannot trust to try it. You want the administrator in the network to moderate the services in the registry.

In this thesis, we present the results from our investigation regarding the security in Web services. Firstly, we present a number of principal security problems in current UDDI implementations, and secondly describe a method that allows us to address these issues. In principal, we propose the use of a transparent proxy-based authentication mechanism, which enables an administrator to control the Web services that are available on his network.

We have also implemented a prototype authentication proxy service, S-UDDI, based on our model, and present both practical results from experimentation with this prototype, as well as a discussion on the generalizability and generic security properties which we have identified through this experimentation.

1.1 Aim, objectives and result

The aim with our work was to evaluate the security in existing UDDI registry, which seemed to be weak. If we found the existing security method insufficient, we were going to develop a safer method.

As we found the existing method insufficient, we developed a method to secure the UDDI, S-UDDI.

1.2 Outline

This thesis is disposed as follows:

- In chapter 2 we describe the background to Service Oriented Architecture.
- In chapter 3 we describe threats in the context of Web services against current implementations that we identified.
- In chapter 4 we present our suggested solution – S-UDDI.
- In chapter 5 we compare our solution to other similar methods and present our conclusions.
- In Appendix I we present a technical specification of our laboratory environment and our sequence of work.

Throughout this thesis we will denote UDDI v.2 with UDDI. UDDI v.3 will be denoted with its version number.

2 BACKGROUND

The introduction of computer networks in general and Internet in particular enables software communication. Communication software typically consists of a client who requests information from a server and present it to a user, but may also involve serverless peer-to-peer communication. A simple example of a client-server application can be illustrated when a user visits homepages. The user is using a browser, the client, which sends requests to a World Wide Web, WWW, server. The server responds with the requested data that the client browser presents for the user.

To ease intercommunication between applications and enable more advanced services, Web services has been introduced. The Web service act as an application running as a server and allows clients to connect and share information much like a standard server. However by using Web services in applications the software can be spread over a computer network, due to the fact Web services render possibility to application-to-application communication.

2.1 Service Oriented Architecture

Service Oriented Architecture, SOA [1], is a technology that enables complex intercommunication between heterogeneous software applications. The benefits of using SOA involve easier method reuse and more effective application integration. Within a SOA environment, software is assembled by invoking different service. Web services enable interface for software to share information or access different computation solutions. Web services typically communicate by using protocols similar to the protocols used in normal WWW communication hence the name. Due to the fact Web services are used by applications to interact and not just displaying information to a user, the data must be represented in protocols that are more stingily defined than normal WWW protocols.

The following protocols are typically used by Web services. Extensible Markup Language [2], XML, a data describing language developed to allow any number of data types and data names. SOAP [3], which is using XML formatted documents to provide an information exchange protocol. SOAP is commonly used by Web services for information exchange. Web Services Description Language [4], WSDL, is a language using XML to describe which methods a Web service implements and how to invoke the specific Web service.

2.2 Principal Web services actors

There are a number of terms commonly used to describe different entities and relations in a Web service environment, and these are important to understand when reading this thesis. In this section, we provide background information to this terminology, and explain the meaning and interactions between these entities and relations.

The most basic entities used are "Subscriber", "Provider" and "Web Service", and the simplest relation is "Invoke". Figure 1, illustrates the relation between these entities and the "Invoke" relation.

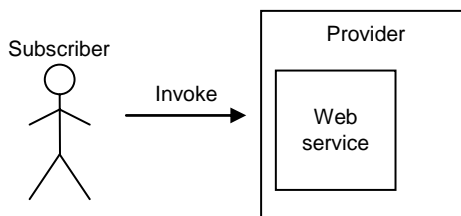


Figure 1 – Web service actors and components

When accessing a Web service the user making the request is called *subscriber*. When the subscriber uses a Web service, it *invokes* the service. The provider *provides* a service to be invoked by the subscriber. To know what methods a specific Web service implements and how to interact with those, the *interface* of the Web service, WSDL, are used. The Web service's WSDL document typically describes how a subscriber can invoke this service and what methods it implements. In order to make use of Web services, methods to discover Web services must exist. To enable this, UDDI has been created.

To illustrate these introduced Web service terms and put its actors in a context, we provide an example of how a Web service environment may behave. A user searches a known UDDI registry for available Web services. The result from the UDDI registry typically contains a description of the service with a WSDL describing how a subscriber should invoke the specific Web service. The result should also contain an Uniform Resolution Language (URL) to the specific Web service. To invoke the Web service the subscriber sends a SOAP formatted request to the provider, which executes the service and responds with a similarly formatted request.

2.3 Function of UDDI

To make use of Web services, methods to easily discover and publish such services must exist. By using the UDDI protocol this methods are made available. The UDDI protocol is implemented by an UDDI registry that enables its users to find Web services. The UDDI registry is itself a Web service, this render the possibility to invoke it with the same methods as a normal Web service. The user must know which UDDI registry to use and have the registry's URL provided in some other way. By using UDDI not only the Web services can be found, but it is also possible to find what methods a Web service implements and the responsible provider.

In the early versions, UDDI was designed as a public accessible directory. Due to this design, it is difficult to use UDDI in critical or hostile environments, where there is a risk that registry data is modified so that a subscriber finds an incorrect service. This might, for example, cause the subscriber to find old versions of a service that are no longer relevant, or, in a hostile environment, enable an adversary to publish hostile services that could pose a threat to the software environment. By using an unexpected service the subscriber may share sensitive information with the provider i.e. passwords, credit card information or personal information. However, with the latest version, UDDI v.3 the specification has introduced support for private registries. By means of a private registry it is possible to protect the registry from public use.

The information in an UDDI registry is categorized just like a telephone book with three categories: “white pages” of company context information; “yellow pages” which categorizes businesses by standard taxonomies; and “green pages” that document technical information about services that are exposed [5].

The UDDI registry and protocol is standardized by OASIS [6], the UDDI specification is designed to create an extensible searchable registry and have methods to both find and publish different Web services.

A commonly used definition of the service provided by a UDDI registry is given in [7]:

“The Universal Description, Discovery and Integration (UDDI) protocol is one of the major building blocks required for successful Web services. UDDI creates a standard interoperable platform that enables companies and applications to quickly, easily, and dynamically find and use Web services over the Internet. UDDI also allows operational registries to be maintained for different purposes in different contexts. UDDI is a cross-industry effort driven by major platform and software providers, as well as marketplace operators and e-business leaders within the OASIS standards consortium.”

2.4 Web services and UDDI

Today, most software that implements the UDDI protocol, implements version 2 of the standard. The UDDI v.2 does not support moderation of published data, i.e. the data published in the registry are made available directly after publication. As of July 2002 the UDDI version 3.0 were introduced and among others the changes of this version enables “registry interactions”, which means that the data integrity can increase when an administrator manually has to review data in a publish registry and approve it before it is moved to the searchable registry. The UDDI v.3 specification also enables support for authentication of the *inquirer*, the one searching an UDDI registry, which have not been possible in earlier versions. The UDDI v.3 has not yet made a big market share, the registry used in production registries is still UDDI v.2, and we cannot see indications of the market share of UDDI v.3 to grow either.

To enable usage of a secure Web service discovering within the limits of the protocol and standard, we have developed a method called S-UDDI.

2.5 UDDI architecture

UDDI is a Web service, which minimizes the external access requirements to a registry. Most of the UDDI registries existing today also provide a direct interface through WWW, which makes the data easy to access. By using the WWW interface the Web service discovering is made available even for those not familiar with the SOA environment.

The UDDI interface can be divided into two different parts; Inquire requests, that are the search for entities in the registry, and Publish requests, which are new registrations in the registry. The user who makes inquire requests is, in UDDI context, called *inquirer* and the user making publish requests is called *publisher*.

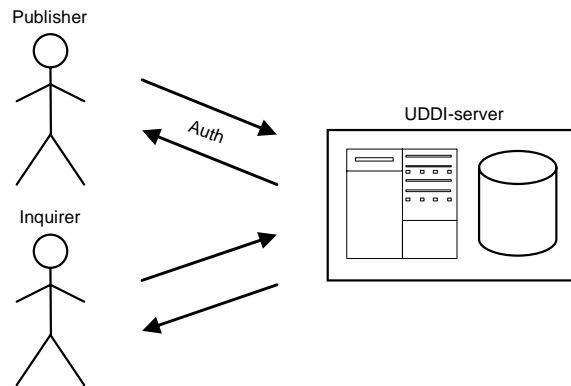


Figure 2 – Architecture of UDDI

As illustrated in Figure 2, only the publisher needs to be authenticated in order to send requests to the UDDI registry, the inquirer can access any information without prior authorization.

When considering an UDDI environment, one should know that a typical UDDI registry is supplied as a standalone application or service and requires of an external database for data storage. This is, however, not a formal requirement mandated by the standard as such, but the most common design.

2.5.1 UDDI data model

In this section we will provide you with a brief description of the UDDI data model, in order for you to understand the policies described in *4.4 Policies*. An UDDI registry consists of five data structures, namely *businessEntity*, *businessService*, *bindingTemplate*, *tModel* and *publisherAssertion*. Underlined fields in Figure 3 denote required elements.

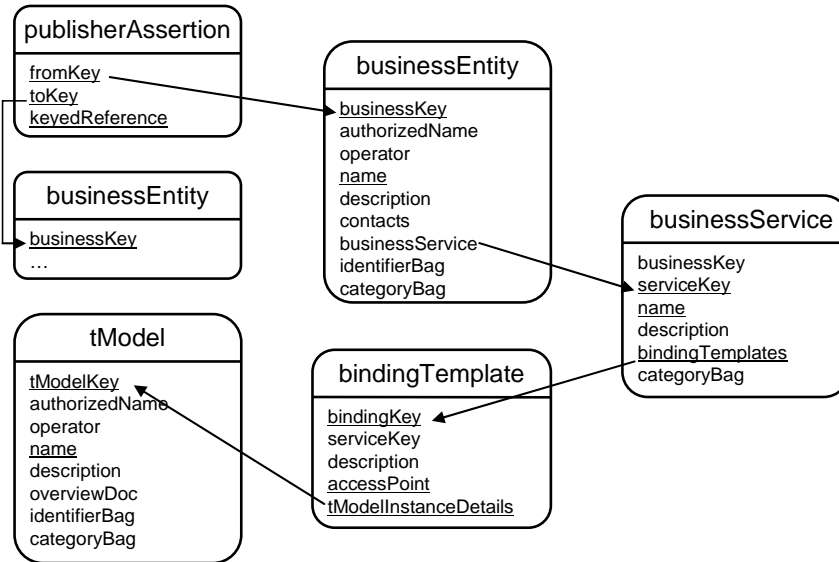


Figure 3 – The UDDI data model

The *businessEntity* is the top level entity, designed to act as the Provider in the UDDI relations. A business *name* is required in the *businessEntity* and when a business is added, a universally unique identifier, UUID [8], is calculated and stored in the *businessKey* element. Each key element in the structure is stored as an UUID. These UUIDs are used as reference between structure types.

A *businessService* reference may be added to the *businessEntity*. The *businessService* requires a *bindingTemplate* and a *businessEntity*. The *businessService* acts as a reference to a *bindingTemplate* which requires a reference to a *tModel*.

The *binding template* provides a mandatory *accessPoint* for information how to access a Web service. A business may register several *bindingTemplates* as reference to the same Web service, each *bindingTemplate* with a unique *accessPoint*, to give a subscriber more than one alternative when accessing the Web service.

The *tModel* is a general structure used to represent a technical specification of the Web service. This is done by a URL in the *overviewDoc* element pointing to a WSDL document. One *tModel* can be referenced from several different *bindingTemplates* implementing the same service. The *tModelKey* is a unique identifier to the *tModel* and is used as a technical fingerprint to a certain specification. When a subscriber searches the register to find if a business implements a *tModel*, it is the *tModelKey* that will be used in the search.

The *publisherAssertion* is used when a reference between two businesses are needed, e.g. when a large company uses a *businessEntity* to every department and wants them to reference to each other. [9]

3 THREAT MODELLING

In this chapter we address common threats to systems using UDDI. In the end of this chapter we will summarize the problems with UDDI today.

3.1 Threat agents

We have identified a number of threat agents for UDDI systems. In Table 1 we provide a list of these agents. When considering A4, the lower levels, A1-A3 are considered also, in contrary A1 is not considered in level A2-A4.

Threat agent A1 is a user with no access to the system, except a normal Internet connection. Threat agent A2 is a user who has no other access to the system than being on a, by the registry, trusted network. Threat agent A3 is a normal UDDI user, which can be any of the existing classes; publisher (allowed to add and remove data from an UDDI registry), coordinator (allowed to make all changes a publisher can do, but also manages UDDI Services data) or administrator (allowed making all possible changes to the registry). Threat agent A4 is a system administrator, i.e. a person who has full access to the computer running the UDDI registry.

Id	Who
A1	External intruder
A2	Local intruder
A3	UDDI user
A4	System administrator

Table 1 – Threat agents

3.2 Threat

In Table 2 we present a list of threats to UDDI, which we have identified. The threats are numbered incrementally with a facing letter T, the column ‘Who’ is the specified threat agent as used in Table 1.

Id	Who	What	How	Consequence
T1	A1	Server attack	DoS	Denial of Service
T2			state	
T3		Network attack	active	finds user data see A3
T4				change the data
T5				publish false service
T6			Passive	see A3
T7				unauthorized information
	A2	--	--	--
T8	A3	Modifies data	false UDDI save action	inconsistent registry data
T9			false UDDI delete action	
T10			erroneous UDDI save action	unreliable registry data
T11			erroneous UDDI delete action	
T12	A4	Modifies data	by direct usage of database	inconsistent registry data

Table 2 – Identified threats in UDDI

3.2.1 Description

The T1 attack can be accomplished by the threat agent A1 in several ways, for example by sending a vast amount of erroneous data. Another consequence to the system made by A1 can be the possibility to read or change unauthorized data. By analyzing the network traffic, A1 may gain access to unauthorized data.

The threats from A2 are similar to T1 – T7; the major difference is that A2 is on a by the registry trusted network. Since UDDI registries handle network traffic similarly regardless of the source of the traffic, there are no differences between A1 and A2.

The A3 threat agent might alter data it does not have sufficient privileges to rightfully modify, which will result in an inconsistent registry. A3 might also save erroneous data which will result in an unreliable registry.

The threat agent A4 might modify data by altering the database as a result of physical access to the server; this will result in an inconsistent registry.

3.2.2 Focus

We focus on the threats constituted by the threat agent A3 which is a normal UDDI user. The threats T1 to T7, which represent external threats, will be handled by policies. The reason to create policies is that the threats T1 to T7 are related to weakness in networks today, rather than weakness of the UDDI protocol.

Motivation and Delimitation

We have focused on how a typical user interacts with the UDDI registry. This is relevant because we do not expect a normal user to have the appropriate knowledge of the definition of correct data and how it should be published in a UDDI registry.

We will not consider direct tampering with the database containing the UDDI registry data. Neither do we discuss external threats such as direct security violation against the server hosting the UDDI registry.

3.3 Inference

According to our findings and conclusions drawn the main threat is the way a user interacts with the UDDI registry. A user may, without any restrictions read the published data in the registry. A malicious user may with a small effort, by traffic analyzing, also publish a Service or even a new Business with several Services as a front for malicious methods. If such a Service is published to the registry, it is impossible to know if it executes the wanted method or any other malicious methods.

The solution presented by us handles both the problem with the unauthorized access as well as the latter problem with unwanted services in the registry.

4 CONTRIBUTION

In order to increase the dependability of an UDDI registry we introduce S-UDDI, short for Secure-UDDI. To do this, S-UDDI provides a proxy service for the communication between the server and its clients. The main task for this proxy is to divide different UDDI requests into two distinct UDDI registries; one handling publish requests and one handling inquire requests. The proxy is constructed to be totally transparent, creating the illusion of being connected directly to an UDDI registry. This is interpreting the incoming UDDI requests to identify if it is an inquire- or a publish request. The proxy is also capable to handle authentication.

4.1 Architecture

The S-UDDI environment consists of two distinct UDDI registries, the first handles publish requests, and the second handles inquire requests. These UDDI registries do not have to originate from any specific software provider, but must be created to handle the UDDI protocol. The proxy has the same role as the UDDI registry normally would have.

Upon receiving a request; the proxy determines which type it is. If it is correctly formatted, the proxy forwards the whole request to the appropriate UDDI registry, otherwise it is discarded.

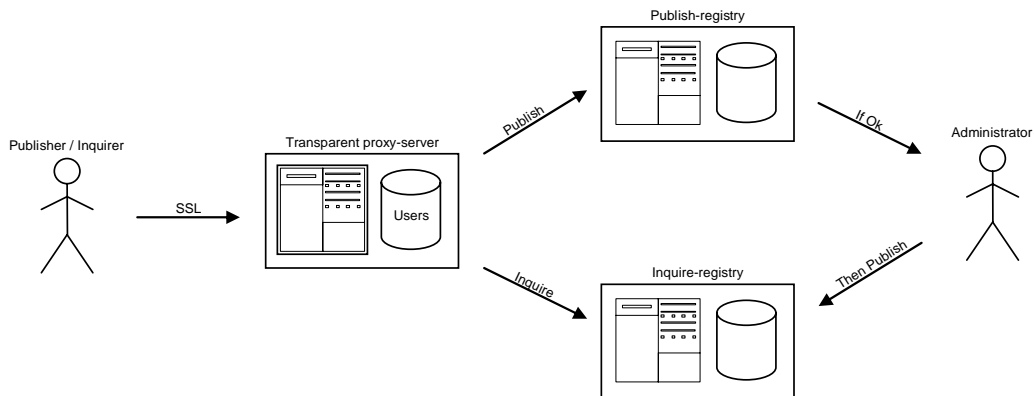


Figure 4 - S-UDDI Architecture

As illustrated in Figure 4, inquire and publish requests are handled in fundamentally different ways. There is no way, not even for a publisher, to send inquire requests to the publish registry. To access the data in the publish registry, you must be an administrator and access it directly. If the publisher publishes a service already published, it is the administrators work to correct the error. The publisher should not have to worry how the data in the publish-server is organized; methods to solve this issue are already implemented in UDDI. In a UDDI registry it is always possible to publish data, the data cannot be duplicated due to the fact that the UUID is generated by methods to ensure this.

When the publisher has published its data to the publish registry, an administrator manually has to review and approve the proposed data to make it searchable. This method does not support the possibility of publishing anything directly to the inquire registry; this registry only provides inquire possibility.

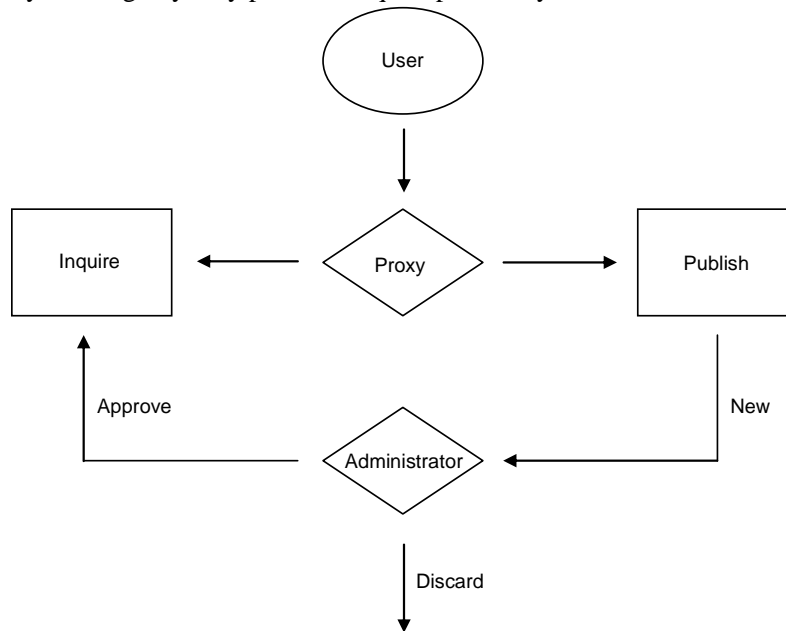


Figure 5 – Flow chart of S-UDDI

As illustrated in the flow chart in Figure 5 the user sends the requests to the proxy, which in turn determines if it is a inquire or publish request. Inquire requests are redirected to the inquire registry directly, whereas publish requests are redirected to the publish registry to be further analysed. The administrator gets a notification about a new service published, verifies it and approves or discards it. The approved requests are added to the inquire register for subscribers to use.

4.2 Interface

The access interfaces to the S-UDDI environment are limited to the same methods used in UDDI, there will be no other methods to make inquire or publish requests to the registries.

Because of the transparency in our proxy, regular UDDI requests are interpreted by the S-UDDI solution. However we do not permit direct access through a Web interface, but require usage of the UDDI Web service interface. With the usage of the UDDI specification we do not force the user to use a specific environment. Since the proxy is implemented to support the UDDI Web services interface totally transparent, all environments supporting Web services can implement a UDDI interface.

Since the web interface is not supported, we suggest that the proxy provider also should provide an application that makes UDDI requests to the S-UDDI solution and presents them in a browsable way. The application may also handle publish requests with support for authentication of publishers as well as publishing of new services. In order to publish data in the registry the publisher must first authenticate to the proxy server.

For administration, we suggest using a complementary tool that will show differences in the two registries and presents them to the administrator who has to approve or discard the changes. The presented difference is partly automatically verified, i.e. if a new web service interface is published, the administrator tool verifies that the specified service implement the methods it are supposed to. However, the administrator must also verify the data manually before he approves it and transfer the data to the inquire registry; this transfer is handled by the administrator tool. The policy for data approval is specified in *4.4 Policies*.

4.3 Defence principals

In order to achieve a more secure UDDI environment and be able to increase data integrity, S-UDDI introduces a number of defence principals.

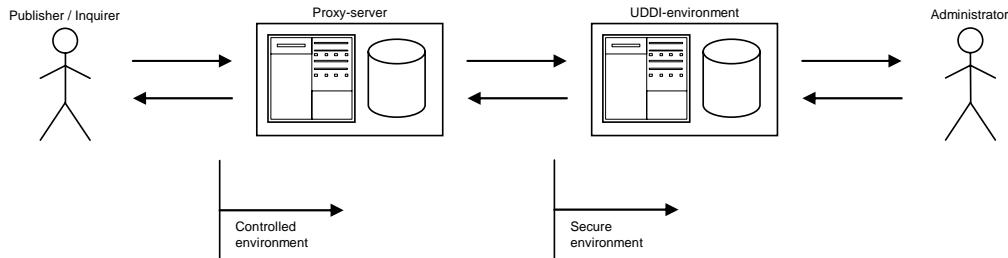


Figure 6 – Levels of security

The environment is divided into three levels of security, as shown in Figure 6. One is unmanaged and one is controlled, i.e. the number of users in this environment is limited. The last one is considered to be secure, i.e. not directly accessible from the Internet. The security mechanism is centralized to the proxy who makes this a single point of failure, if the integrity of the proxy cannot be guaranteed; the system must be considered compromised. The two UDDI registries are not accessible in any way but through the proxy and only accept requests from the proxy.

To ensure the data integrity in the UDDI registry, the registries are configured to accept incoming request from the Proxy and Administrator only. To secure the access to the registry we suggest the use of a traditional packet filtering firewall [10,11]. This provides the system with an extra level of security.

Using the S-UDDI approach the specified proxy handles the authorization with its own user control mechanism. In UDDI, only the publish interface has support for authentication, which leads to a publicly searchable registry which in some cases not desirable. A solution to this issue is to force SSL-usage and verify the client's SSL-certificate. Due to the fact that S-UDDI runs by the regular UDDI specifications, we cannot introduce authentication to the inquire requests. Though, we can force users to use SSL when communicating with the S-UDDI. Doing this, we gain control over the users and get the ability to authenticate via the certificate [12]. The decision of accepting a certain SSL certificate on the administrator and is controlled by the Access Policy, chapter 4.4.2.

For an adversary to publish an unauthorized/malicious service into this system, he first needs to gain control to the proxy and get authenticated to add a service to the temporary UDDI registry. Then he needs to convince the Administrator to approve the service in some way, e.g. by social engineering.

4.4 Policies

In this section we describe the different policies we have developed. The data policy assists an administrator approving or denying data published to the registry. When it comes to the matter of access to the environment, the Access policy addresses this.

4.4.1 Data policy

A few things need to be considered by the administrator, when approving new data in the publish registry, before it can be published into the inquire registry.

When a new Service is published, the publisher needs to be trusted according to the Access policy. The S-UDDI solution has a wider set of requirements than the standard UDDI solution. In Figure 7 the S-UDDI requirements are illustrated, the underlined elements are requirements in UDDI. The optional elements are discarded in this figure, for a complete list of elements, see Figure 3.

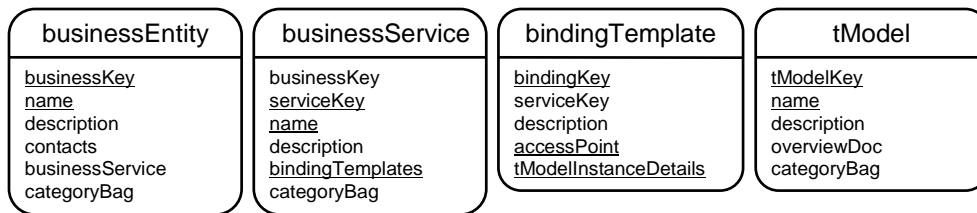


Figure 7 – UDDI requirements

In a businessEntity, S-UDDI requires besides the standard businessKey and name also a businessService along with a description, a contact and a categoryBag. The businessService is a reference to a regular businessService entity. The description and categoryBag have the same function, to ease the search of a new business in the S-UDDI inquire registry. The description describes the business as the categoryBag holds the category of the business. Besides these, a contact is mandatory in the businessEntity, this to enable the subscriber to contact the business, if needed, when finding a new service.

The businessService requires a businessKey as a reference to a businessEntity, a serviceKey as a unique identifier and a name of the service. The description and categoryBag holds the same type of information as in the businessEntity. The bindingTemplate element is a reference to a bindingTemplate structure.

The bindingKey is a unique identifier in the bindingTemplate and the serviceKey refers back to the businessService. The description contains information about the bindingTemplate and the accessPoint holds a reference to the Web service and the tModelInstanceInfo is a reference to a tModel.

The tModel needs, besides a unique identifier, tModelKey, a name, a description and a categoryBag. The description holds a description of the tModel and the categoryBag categorizes it. The element overviewDoc in the tModel are also required and is a reference to the WSDL document where information about the Web service is found.

4.4.2 Access policy

The S-UDDI design developed by us shall implement the methods described below to ensure accessibility to the registry. By implementing these methods the S-UDDI environment accessibility can be increased to a higher level. The access policy is developed with the identified threats illustrated in Table 2 in mind. By the use of the methods described below the identified threats are eliminated.

To decrease Denial of Service [13] attempts, such as sending large quantities of data, an inquirer cannot send more than one inquire request per previously defined time unit. Another way of making a DoS-attack is to send a vast amount of erroneous data to the registry. To avoid this, the inquirer's data is discarded if sending more than three erroneous packages in a sequence. When the system is blocked, the access is retained after a number of seconds.

To avoid state attack on the server, where a malicious hacker consumes a lot of the server resources, a time limit for each connection is used. If the connection time exceeds this specified time the connection will be closed, forcing the client to establish a new connection

To increase the security in the transfer and minimize the risk of successful traffic sniffing, a valid SSL-certificate is demanded by both the inquirer and the publisher. To give a subscriber access to the Inquirer register, the administrator must accept the user's SSL-certificate. The issue of a valid SSL certificate is controlled with an internal policy.

To access the publisher registry, a username and password are needed to be authenticated to the environment. The issue regarding distribution of trusted publisher accounts is controlled with an internal policy. The structure of a valid password is also controlled with an internal policy.

4.5 Summary

By using S-UDDI, it is possible to use web services in a more dependable fashion. Thanks to the server being transparent to the users, this system can be integrated into an existing infrastructure without requiring extensive modification to existing components. By separating requests into distinct registries and using a trusted path from the less secure to the more secure registry, it is possible to increase the dependability in a principled way.

4.5.1 Security Mechanisms

To conclude the introduced security mechanisms presented through this chapter we present them in Table 3.

Id	What	How
S1	Proxy service	Divide requests according to action
S2	Administrator verification	Verify data according to policy before approval
S3	UDDI registries in secure environment	Blocking network traffic, not from the trusted proxy or administrator
S4	Proxy authentication	The proxy is using internal authentication
S5	Force SSL-encryption	All traffic to and from the proxy must be encrypted
S6	DoS	Only a number of requests are allowed to originate from the same place per time unit
S7	DoS	Only three erroneous requests in sequence are allowed to originate from the same place
S8	State	Time limit of connection to the proxy
S9	SSL-certificate verification	The user of S-UDDI must have a, by the proxy, approved SSL-certificate

Table 3 – Summary of security mechanisms

4.5.2 Problems with UDDI solved by S-UDDI

The main problem with UDDI is the lack of data integrity. With the S-UDDI solution, a method to solve this shortage exists. By using S-UDDI all threats identified by the threat agent A3, presented in Table 2, can be handled. The problem with UDDI's authentication for the inquire methods have also been solved i.e. threat T7 identified in 3.2 *Threat*. With our solution, methods to solve most of the threats by the threat agent A1 have also been presented, there are some DoS attacks that we do not handle.

5 EVALUATION

In this chapter we present a threat model of S-UDDI, and make a threat mapping according to the security mechanisms introduced in the S-UDDI solution. We also provide a comparison of our solution with other, similar, approaches, such as those mandated by newer versions of the standard. Principal advantages with our solution are e.g. the transparent proxy and the moderation of data in the registry.

5.1 Threat Modelling S-UDDI

With S-UDDI there are still a few threats left that a system administrator must have knowledge about. The threat agents described in section 3.1 *Threat agents* are applied even on our contribution S-UDDI.

Table 4 contains a list of identified threats to S-UDDI; it is similar to Table 2 with the difference that the solved issues are removed.

Id	Who	What	How	Effect
T1	A1	server attack	DoS	insufficient availability
T18	A2	Network attack	passive	unauthorized information
	A3	--	--	--
T12	A4	Modifies data	by direct usage of database	inconsistent registry data

Table 4 – Identified threats in S-UDDI

5.1.1 Description

Distributed DoS are a server attack not handled by the policies described in 4.4 *Policies*; we are eliminating the other DoS attacks addressed by us. The threat agent A4 might still modify data by changing the database due to physical access to the server; this will result in an inconsistent registry. This problem does not have a solution other than place trust on A4.

In S-UDDI we introduced a new threat T18; this is a threat that arises from the introduction of the controlled environment. The threat agent A2 has access to this environment and by this access gathers unauthorized information, through traffic analysis between the proxy and the UDDI registries. Because threat agent A2 operates in a controlled environment and are well known to the system, the threat agent should not be considered to be a severe threat.

5.2 Threat Mapping

The identified threats to a UDDI registry, listed in Table 2, are mapped to the security mechanisms, found in Table 3. In the table below, we show which threats are eliminated with each security mechanism.

	S1	S2	S3	S4	S5	S6	S7	S8	S9
T1						X	X		
T2								X	
T3					X				
T4				X	X				
T5				X	X				
T6					X				
T7					X				X
T8	X	X							
T9	X	X							
T10	X								
T11	X								
T12									

Table 5 – Mapping of UDDI threats to security mechanisms introduced in S-UDDI

In Table 5 the security mechanism S3 is not mapped to a threat, this because S3 handles S-UDDI threats not previously introduced. In the S-UDDI design we introduce a controlled environment, this is where the threat agent A2 operate. To eliminate modification in the registries by A2 the security mechanism S3 is used.

5.3 Threat Modelling UDDI v.3

In UDDI v.3 two new types of registries have been introduced. A production registry where verified data exists to be inquired and a staging registry where new data is placed. The staging registry is to UDDI v.3 as the publish registry is to S-UDDI, data is added to let the administrator user class, in UDDI v.3 called *approver*, approve the data and by this move it to the production registry to make it searchable.

There are a number of threats identified by us in UDDI v.3 that we believe have not been solved, these are presented in Table 6.

Id	Who	What	How	Consequence
T1	A1	Server attack	DoS	insufficient availability
T2			State	
T3		Network attack	Active	finds user data see A3
T4				change the data
T5				publish false service
T6			Passive	see A3
T7				unauthorized information
T13	A2	Modifies data	false UDDI approval	inconsistent production data
T14	A3	Modifies data	false UDDI save action	inconsistent staging data
T15			false UDDI delete action	
T16			erroneous UDDI save action	unreliable staging data
T17			erroneous UDDI delete action	
T12	A4	Modifies data	by direct usage of database	inconsistent registry data

Table 6 – Identified threats in UDDI v.3

5.3.1 Description

Since the existing methods to secure traffic are not required according to the UDDI v.3 specification, we believe this to be a severe threat. It is realistic to say that in a real environment, the developers will choose to make traffic encryption a requirement. Due to the fact that until this day it only exist a limited number of UDDI v.3 registries commercial available, it has not been possible for us to investigate this any further.

In UDDI v.3 the threat agent A2 may falsely send data approval to data in the staging registry making the staging registry contain inconsistent data. This is a major threat since the data integrity in the production registry cannot be guaranteed, the whole staging/production concept introduced in UDDI v.3 can be considered pointless.

The threats identified by A3 in UDDI v.3 are significantly different to the one identified in Table 2, due to the fact that it is impossible for A3 to publish data in the production registry. In our sense, A3 may still be a threat to the registry because the traffic to the staging registry may be tampered with, this enable A1 to falsely publish erroneous data. The data published to the staging registry should not be considered to be a very large threat since an approver must approve the data before it becomes available in the production registry.

5.4 Comparisons

There exists a wide amount of both external- and internal threats in UDDI. By external threats we address the issue of an attacker outside the system and by internal threats we address the issue of an attacker authorized to the system.

The identified threats presented in Table 2 can be reduced to two of the identified threats in UDDI, by using S-UDDI. The threats identified in S-UDDI, presented in Table 4, introduces one new threat. This is not a severe threat due to the fact the threat agent is a trusted user inside the S-UDDI environment. Since the number of threats in S-UDDI is reduced by ten , this gives us a brief understanding that S-UDDI is superior to UDDI. Even if UDDI v.3 has some of the security mechanism we suggest in S-UDDI, it still lacks security methods, i.e. securing the network traffic and protecting the registry against server attacks.

The new version of the standard, UDDI v.3 is similar to our method of securing the UDDI register. However, version 3 of UDDI lacks a few important security mechanisms. The main difference between S-UDDI and UDDI v.3 is the, in S-UDDI, mandatory moderation of the added services, the mandatory SSL-encryption and the mandatory secure environment. The S-UDDI environment also includes policies handling e.g. the moderation as well as the access to the environment.

6 CONCLUSIONS

From our perspective, the main problem with UDDI is, as we described in 3.4 *Inference*, that the published data are not reliable. As a result of the published data are not moderated and all traffic from and to the register is sent in plaintext and therefore a malicious user may, with small means, get hold of publisher account information. If a malicious user gains control over a publisher account, the user may publish whatever service the user wants to publish, malicious or not. This makes the registry unreliable in an environment where you cannot trust every component.

It is not realistic to assume that it only exist good publishing on the Internet, or even on a corporate network. On the Internet, it is not at all unlikely to be a victim of intentional manipulation in order for a malicious hacker to steel information. Even on a corporate network, this is possible, as a consequence of viruses or Trojans.

As a result of this, we think that a security layer needs to be added to the existing UDDI, if it should be used outside a laboratory environment. The version of UDDI used today, has no requirements of encrypted authentication or data moderation and the new standard are today not widely deployed. Not even the new version of UDDI requires data encryption, but contains support for data moderation.

We have presented a method, S-UDDI, addressing a lot of the problems existing in today's implementations of UDDI. S-UDDI makes it possible to use UDDI registries in corporate networks as well as in other networks requiring data integrity and availability. We have described the method and compared it to other existing methods. We have found that our method makes it, to a wider extent, possible to trust the information in the registry and only trusted users can read the information in the registry. We have also found that a higher availability of the searchable registry can be achieved.

Finally we think that Web services and Service Oriented Architecture is a valuable step towards a more effective system development, even though it is in need of good security models to penetrate this market. We think that it is a requirement to guarantee the security and integrity in the central parts of the architecture, the UDDI registry. S-UDDI represents the next step in this direction and is a technique that should be considered when implementing this type of architecture.

7 REFERENCES

- 1 Hasan, J 2004, *Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0*, Apress, Berkeley
- 2 *Extensible Markup Language (XML) 1.0 (Third Edition)* 2004
Retrieved August 11, 2005, from
<http://www.w3.org/TR/2004/REC-xml-20040204>
- 3 *SOAP Version 1.2 Part 1: Messaging Framework* 2003
Retrieved August 11, 2005, from
<http://www.w3.org/TR/2003/REC-soap12-part1-20030624>
- 4 *Web Services Description Language (WSDL) 1.1* 2001
Retrieved August 11, 2005, from
<http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- 5 Johnson, M 2001 *UDDI : A Phone Book for Web Services*
Retrieved August 11, 2005, from
http://www.itworld.com/nl/xml_prac/05102001
- 6 *Organization for the Advancement of Structured Information Standards*
Retrieved August 11, 2005, from
<http://www.oasis-open.org/who>
- 7 *Universal Description, Discovery and Integration*
Retrieved August 11, 2005, from
<http://www.uddi.org>
- 8 *UUID, ISO/IEC 11578:1996*
- 9 *UDDI Version 2.03 Data Structure Referenc* 2002
Retrieved August 11, 2005, from
<http://uddi.org/pubs/DataStructure-V2.03-Published-20020719.pdf>
- 10 *PF: The OpenBSD Packet Filter*
Retrieved August 11, 2005, from
<http://www.openbsd.org/faq/pf>
- 11 Deshmukh, D, Gopalan A, *Firewalls Feature Focus*
Retrieved August 11, 2005, from
<http://www.ewh.ieee.org/r10/bombay/news1/2.htm>
- 12 *RFC 2246 - The TLS Protocol Version 1.0* 1999
Retrieved August 11, 2005, from
<http://www.faqs.org/rfcs/rfc2246.html>
- 13 *Denial of Service Attacks*
Retrieved August 11, 2005, from
http://www.cert.org/tech_tips/denial_of_service.html

APPENDIX I

Laboratory environment

To enable laboratory work we have set up a small laboratory environment described by the picture below.

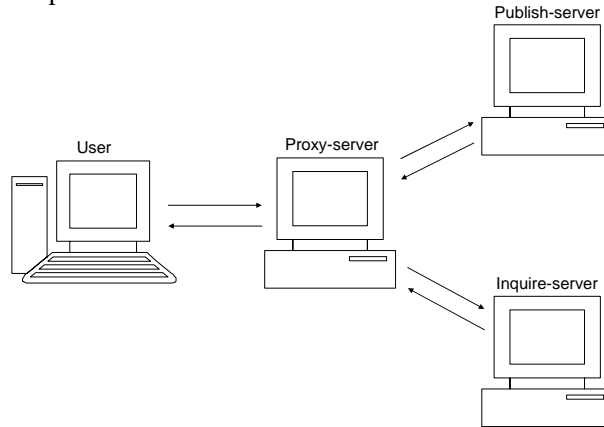


Figure 8 – Laboratory environment

The user's computer ran Microsoft Windows XP SP2 and we developed test applications in Microsoft Visual Studio 2003. The Proxy-server's computer ran Microsoft Windows XP SP1 and the Proxy-server was created in Microsoft Visual Studio 2003 C#. Both the Publish- and the Inquire-server were running Microsoft Windows 2003 and its built-in UDDI-server.

In order for us to understand and move towards a solution we have made a couple of laboratory experiments.

7.1.1 Web services

We started developing a Web service and invoked it in different ways. By analyzing the TCP/IP traffic to and from this service, we saw all the communication made. The communication is SOAP objects formatted by XML. We came to the conclusion that methods to verify that the provider actually do execute the expected service, is a very extensive problem without any suggested solution, other than the web service consumer must trust the provider to execute the expected service.

7.1.2 UDDI registry

Later we decided to set up our own UDDI registry to extend the simple laboratory environment. For this we used the UDDI registry provided by Microsoft in Windows 2003.

UDDI registry through WWW

The laboratories started by simple trial and error in the UDDI registry's WWW interface. In this registry we saw that internal methods for data verification do not exist. Without any knowledge of a UDDI registry's structure, a user can alter the information, both intentionally and unintentionally, in a UDDI registry so bad that all information in it can be considered futile.

UDDI registry through API

The UDDI API provided by Microsoft¹ contains some simple examples. Because we did not have any experience in the UDDI environment, we chose to analyze these examples. The provided examples varied in quality; however we found one good example, with the ability to send inquire request to a UDDI registry. This example (called 'uddiexplorer') was modified to enable finding of a specific tModel, i.e. the created web service described above and which was previously added to the UDDI registry. We also added methods for discovering the access point to the selected tModel. Support was also added to enable invoking of a tModel at a selected access point. With the use of our tool we evaluated and tested different SOA actions.

Traffic analysis to understand and find a way to develop a good proxy

In order for gain larger understanding in the UDDI chain and Web service environment we made some traffic analysis. For this matter, a tool called Ethereal [2] was used. The messages used were XML formatted data extracted from traffic analysis i.e. data in plaintext.

The results and conclusions drawn from the traffic analysis helped us to develop a tool with the ability to send raw TCP messages containing UDDI requests directly to the UDDI proxy.

We also developed a UDDI simulator with the capability to simply expect a specific UDDI request and replying to this request with static data. This simulator were tested both against applications using Microsoft's UDDI API and our own 'send raw UDDI commands' application.

7.1.3 Laboratory Proxy development

At this stage we have gained understanding for what to be done in terms of creating a more secure UDDI registry, a proxy that filters inquire and publish request and sends them to separate servers.

From the static UDDI simulator we developed a proxy with the abilities to handle simultaneous connections. It was also developed to handle a number of UDDI requests, with the work scheme;

```
If <request> equals a inquire request
  Do
    Send the whole request to the inquire server
  Done
Else If <request> equals a publish request
  Do
    Authorize the requester
    Send the whole request to the publish server
  Done
Else
  Reject the request with an error to the requester
```

7.1.4 Laboratory Client

In order for us to make tests against the proxy we developed a UDDI client. This was created as a demonstrator. It is capable to handle the most common UDDI requests, sufficient to show how our proxy works.

¹ <http://www.microsoft.com/downloads/details.aspx?FamilyID=542991cf-9056-49fd-babf-7ff3638ee6b1&DisplayLang=en>

² <http://www.ethereal.com/>

The client development started as stated above with a dummy interface to send and receive plain text to an UDDI registry. This was used to send SOAP requests to the UDDI registry in order for us to make some of the laboratories described above. The SOAP messages used were gathered by network traffic analyses using the UDDI examples provided with the UDDI API. We also did laboratory work and changed the SOAP messages manually to see how a UDDI registry react and handle both correctly formatted and erroneous requests.

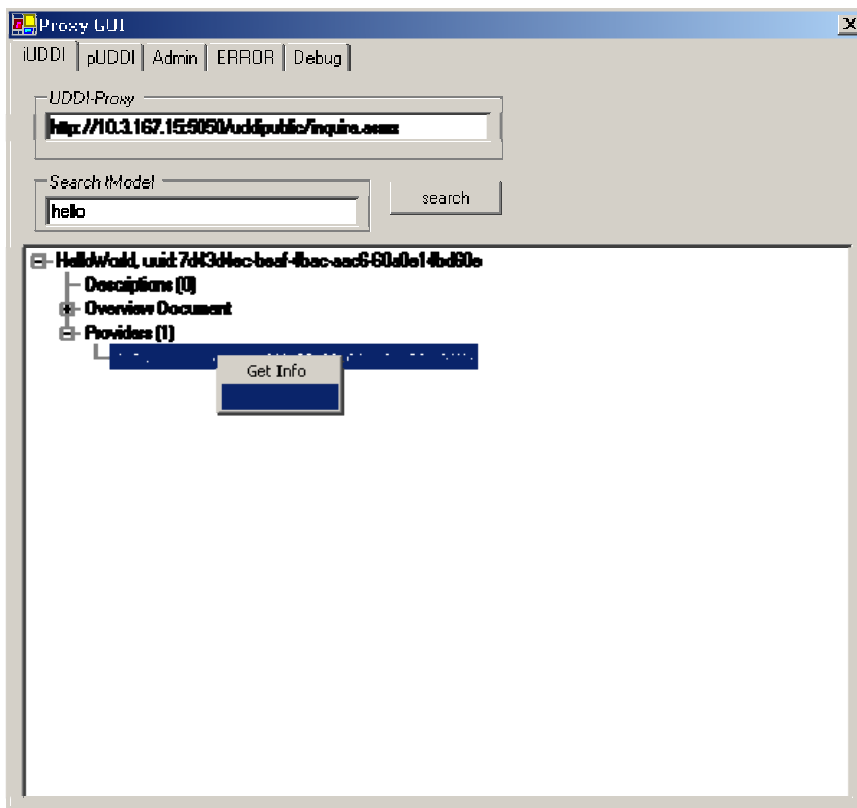
Code analysis

From the simple “send messages over TCP” interface we decided to provide GUI using the provided API. As a consequence of the insufficient documentation provided to the API, we were forced to make code analysis of the available examples and modify them to suite our need.

Due to bad documentation of the API we tried to generate our own API according to the UDDI’s WSDL, using the ‘wsdl.exe’ application provided with Visual Studio. “wsdl.exe” is a utility to generate code for an xml Web service client. However after some time without success we decided to use Microsoft’s API even of its lack of documentation.

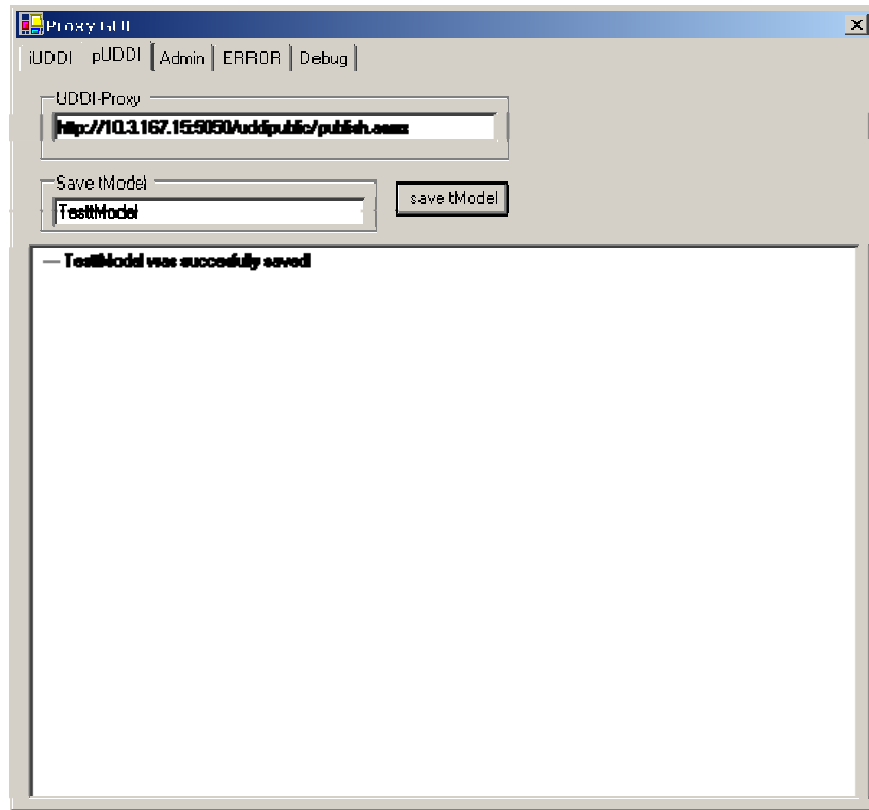
Inquire Interface

The developed method in the inquire interface was the ‘find tModel’ method. A user can search for a specific tModel; the results are presented in a clickable list, if the user clicks on one of the entities in the list, detailed information about the selected entities appears. The interface also has support for invoking a found service, described by a tModel, with the limitation that the found service must implement a specific WSDL.



Publish Interface

In the publish interface we have implemented methods to publish a new tModel, with a provided name.



Administrator Interface

The administrator interface implements methods to show differences in the publish registry and the inquire registry. This makes it possible to verify the presented changes and discard/approve the changes.

