



SIMDAT

Data Grids for Process and Product Development using Numerical Simulation
and Knowledge Discovery

Project no.: 511438

Grid-based Systems for solving complex problems – IST Call 2
Integrated project



Deliverable

D 14.1.2 Production of first working prototype(pharma)

Start date of project: 1 September 2004

Duration: 48 months

Due date of deliverable: 1 September 2005

Actual submission date: 14 October 2005

Lead contractor for this deliverable: NEC Europe Ltd.

Revision: 1.0

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination level		
PU	Public	X
PP	Restricted to other programme participant (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history

Date	Version	Author	Modification
23.08.	0.1	Falk Zimmermann	Template
30.08.	0.2	Federico Crazzolaro Falk Zimmermann	NEC's contribution on security
05.09.	0.3	Chris Dodge	LION's contribution
05.09.	0.4	Changtao Qu Falk Zimmermann	NEC's contribution on the semantic broker
05.09.	0.5	Kai Kumpf	SCAI contribution on semantic annotation
13.09.	0.6	Peer Hasselmeyer	Added motivation, partly proof-read
14.09.	0.7	Peer Hasselmeyer	Editorial changes
14.09	0.8	Martin Weindel	Added knowledge model chapter
15.09	0.9	Falk Zimmermann	Final editing of prototype version
22.09	0.91	Falk Zimmermann	Integration of Lessons Learned section Extended section on conclusions
25.09	0.92	Federico Crazzolaro	Deployment Picture (section 3.5)
26.09	0.93	Falk Zimmermann	Modified introduction for chapter 5
30.09	1.0	Falk Zimmermann	Final editing

Copyright

This report is property of the partners of the SIMDAT consortium 2005. Its duplication is restricted to the personal use within the consortium, funding agency and project reviewers.

1 Introduction

1.1 Motivation

SIMDAT is an application-driven project with applications from four different domains. The output of each domain is a prototypical implementation of an application solving real-world problems. During the first 12 months of the project, the pharma sector developed a prototype that demonstrates the integration of bioinformatics applications with basic Grid technologies.

This document describes the pharma activity prototype. The problem addressed by the SIMDAT pharma prototype is the discovery and use of distributed pharma databases and analysis tools. Different installations of SRS (a popular bioinformatics software package) provide different bioinformatics databanks and analysis tools. Users wanting to access them need to know who provides what and how to access them. The more databases and tools there are, the more tedious this task becomes. The SIMDAT pharma prototype tries to alleviate this problem by offering a portal that knows about the accessible SRS servers and the services they provide. Which service is actually fulfilling a user's request is hidden from that user. Finding appropriate databases and analysis tools is further facilitated by the use of semantic technology which allows for finding and using similar or even equivalent services that use a terminology a user does not know about.

The prototype is an extension of the SRS software used widely within the bioinformatics community. It adds features for integrating multiple SRS servers deployed across provider domains and for virtualising access to them. Chapter 1.3 contains some background on the SRS software.

The SIMDAT pharma prototype is made up of a number of building blocks, integrated into one coherent prototype. This document contains a detailed technical description of the individual building blocks and the technologies used within the prototype. The architecture of the prototype is described in chapter 2.1 while the individual building blocks are detailed in the later chapters.

1.2 The Problem Domain

Many aspects of the pharmaceutical research and discovery process depend on data in electronic formats, originating from multiple sources within the academic community, or from commercial data providers or through a company's own research. This data holds key information about the complex biological mechanisms that scientists attempt to understand when looking for new medicines. Due to the underlying complexity of biological processes, there are now hundreds of different databases containing data associated with different parts of these processes.

One of the basic challenges when attempting to use this data is making it and the analysis tools that use the data available to scientists. This problem has been solved to a certain extent with data integration products like SRS as described in the next section. However, as the number and size of datasets grows, and approaches to the use and management of electronic data evolve single site data and tool integration is no longer going to be sufficient. In the area of distributed use of biological data, there are certain problems to be addressed.

1.2.1 Finding Remote Data Sets & Analysis Tools

It is inevitable that the current approach of downloading all required datasets to a central location for use will cease to be viable. So to find a remote dataset for use will necessitate some method of description for registration and discovery. Analysis tools also need to be suitably described. What level of description is appropriate; abstract biological descriptions or detailed technical descriptions? The prototype takes a first step in experimenting with ontologies to describe SRS servers (i.e. the data and tools available) and allowing the system to discover their location via a broker.

1.2.2 Access to Remote Data Sets & Analysis Tools

In the longer term, access to datasets and tools has to take place via standard mechanisms, such as web services. Where bioinformaticians would normally run multi-step analyses manually, or have to hardcode scripts, then systems such workflow engines can be used to simplify the use of GRID based resources. The prototype uses the SRS web services client side layer, WSOjects to allow access as a first step, however further work is to be done on making APIs more standard.

1.2.3 Security

Commercial and proprietary datasets are of significant value, so their remote usage has to be subject to strict security. In addition, the usage patterns of public servers by commercial organisations may provide clues to areas of research activity, so also need to be subject to appropriate security measures. The prototype includes a security component that supports these scenarios and lays the foundation for more comprehensive distributed working such as virtual organisations.

1.3 SRS

1.3.1 Introduction

Over the past few decades, advances in the field of molecular biology, coupled with advances in genomic technologies, have led to an explosive growth in the biological information generated by the scientific community. This mass of genomic information has, in turn, led to a requirement for computerized databases to store, organize, index, and link the data and for specialized tools to view and analyze the data.

SRS is a platform for life-science database and application integration [17] [18], providing rapid, easy and user-friendly access to the large volumes of diverse data stored in over 1000 public domain databases [19] along with the proprietary data generated internally. SRS also contains a set of data mining tools which allow the user to access all of this diverse biological and life science data through a single user interface. Since these databases may hold references to other sources of information, i.e. other databases, SRS is able to use these references to explore the relationships between the different sources of biological data. SRS is the most widely used tool for accessing life-science databanks and is used by over 300 commercial and academic sites delivering genomic data to thousands of users.

SRS solves the problem of differing library formats by building an index for each field in each database available to it. The indices, rather than the full database, can then be searched rapidly. Entries in one index can be related to corresponding entries in any other database, allowing users to retrieve, and access entries from all the interconnected data sources.

1.3.2 Architecture

Figure 1 gives an overview of the architecture of SRS 8.1, the main sections being described in the paragraphs below.

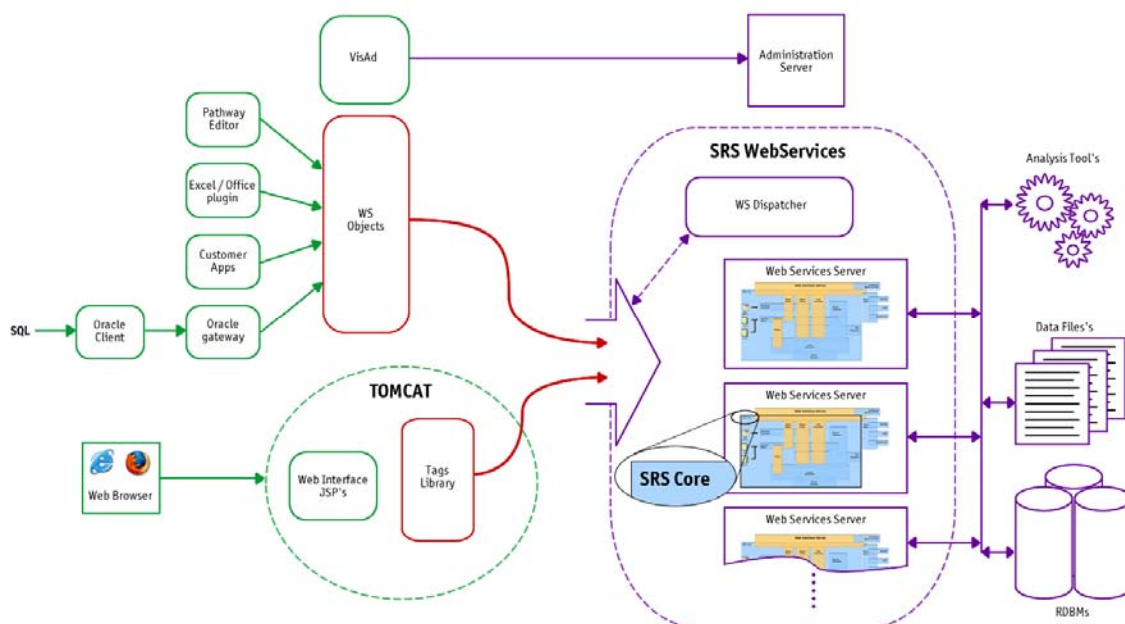


Figure 1: An Overview of the Architecture of SRS

1.3.2.1 Data and Analysis Tool Integration

Flat-files, XML data and relational databases are all fully described using meta-data, including the type and structure of data, relationships to other data sources, how the data should be indexed or presented to users, and how it can be mapped to external object models.

1.3.2.2 The SRS Core

This contains the main modules for data processing, including indexing data sources, parsing & loading data, processing queries, performing set operations on result sets and launching analysis tools. Each data source integrated into the system can be understood and manipulated by the core.

1.3.2.3 SRS Web Services

The operations that the core supports are wrapped in infra-structure code so that they operate as a server (Web Services Server). A single installation will normally comprise multiple servers for redundancy and load-balancing purposes.

The web services also present a set of APIs that allow access to the core functionality, i.e. data querying, loading etc.

1.3.2.4 WSOjects

Programmatic access to SRS is used in many situations, especially within a project such as SIMDAT, and is done through the client-side support library called WSOjects. The primary reason for this layer is that the use of the SRS server's API requires a detailed knowledge of the internal *dynamic object model* used by the core to hold and manipulate data.

1.3.2.5 The Web Interface (Tomcat & JSPs)

This is the SRS end-user application that is built on top of the SRS server. The server does not “understand” the data at all; it could in fact be used with any type of data (however it is very good at dealing with the complexity and inter-connectedness of biological data). The web interface thus introduces the concepts and application structure that make SRS a bioinformatics tool.

1.3.3 SRS Federation

The SRSFed consortium (www.srsfed.org) was founded by LION and three EMBnet nodes (Belgium, Sweden and Slovakia) in 2004 (www.embnet.org). The aim of the consortium is to provide a network of “co-operating” SRS servers providing high-quality information that may be located on geographically separated servers, but provided through a single, common user interface. This reduces the need for data to be duplicated or maintained at multiple sites, increasing research efficiency and reducing costs.

1.3.3.1 Participating Institutes

At the time of writing (Sept 2005) there are six participating institutes in Slovakia, Belgium, Sweden, Poland, Colombia and Brazil.

1.3.3.2 Current Project Status

The first phase of the federation project is complete, and has seen the creation of a network of “master” servers which create data/index sets which are then distributed to “slave” servers. This allows the CPU-intensive tasks to be evenly distributed amongst the members. SRS Prisma is used to produce new indices and data sets on the “master” servers, and also to download new data/indices on each “slave” node. To monitor the state of the network, a cross-comparison tool has been developed using WSOjects to compare sizes and dates of libraries across the network of servers.

As new developments in SRS federation become available, either through SIMDAT or LION’s internal development, the SRSFed consortium provides a very useful technology tested by academic organizations that actively use SRS for their ongoing research.

2 Prototype Introduction

2.1 Prototype Architecture

The SIMDAT pharma prototype is a distributed application which provides middleware federation of SRS installations running on remote servers. The system consists of three main components:

- Directory Service (Semantic Broker),
- Node Broker (SRS Nodes),
- Federated Portal (User Interface).

Figure 2 shows an overview of the architecture. Although all components could exist multiple times, for our first prototype we assume only one instance of the directory service and the portal respectively. As the aim of the pharma prototype is the integration of multiple distributed SRS servers, we assume multiple node brokers and associated SRS installations.

In the next two sections, the node broker and the federated portal are described. The directory service consists of a number of subcomponents on which a substantial part of the pharma effort was spent. The individual subcomponents therefore deserve their own in-depth descriptions. The Semantic Broker is described in chapter 5, the Ontology Repository in chapter 4 and the Annotation Service in chapter 6. The security component is detailed in chapter 3.

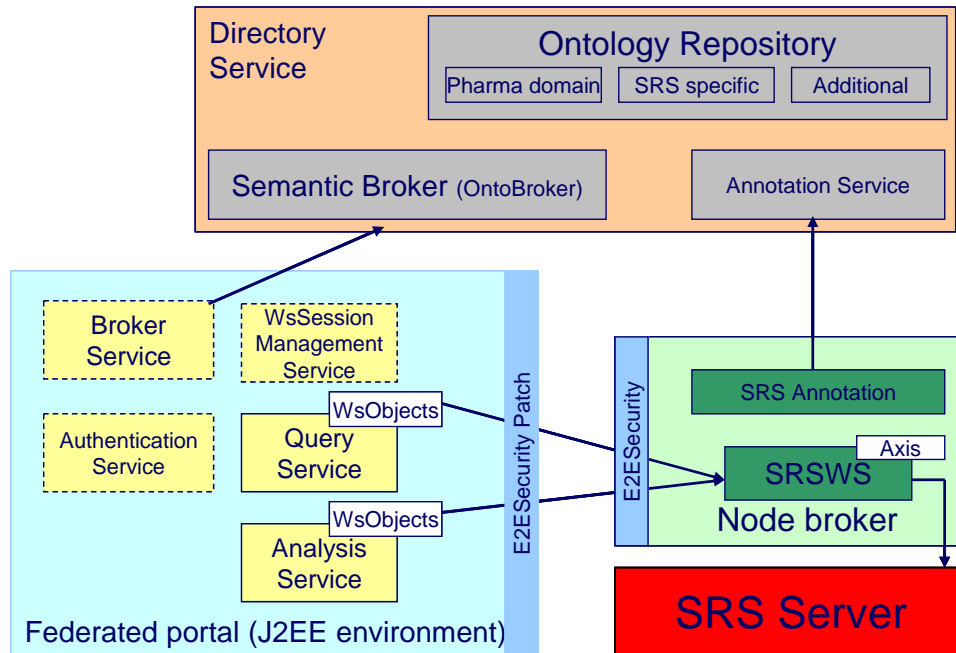


Figure 2: Federated Portal Overview

2.2 Node Broker

The Node Broker has two main objectives. Firstly, it is used to annotate the services available in an SRS installation. At the moment, this annotation is performed by a simple command line client. To define actors in the annotation file, we use a configuration file filled by the SRS server administrator. The client then connects to the SRS server via WsObjects to generate the full annotation XML file which is sent to a Web Service linked with the Service Directory.

Secondly, a node broker is used to Decrypt/Encrypt the communication between the portal and the SRSWS set up on the Axis proxy. The SRSWS on the Axis proxy forwards the decrypted message sent by the portal to the real SRSWS of the SRS installation (Figure 3). This security is provided by NEC's E2ESecurity application.

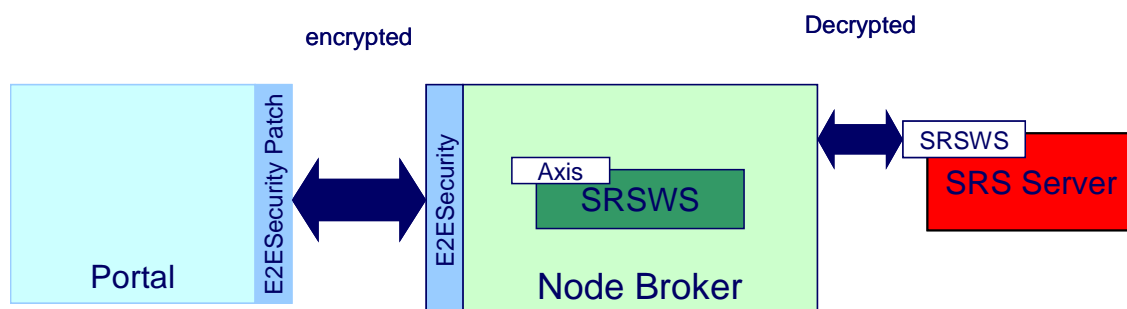


Figure 3: Integration of NEC's Security Component

2.3 Federated Portal

This is the environment the user accesses. It provides a local view on the services available in the system. It is a user-centred project-based environment with a simple interface to the query and analysis services and also the result management.

2.3.1 Description of the Services

Authentication Service

This service is used to authenticate the user. It's based on the JAAS security integrated in the J2EE environment.

WsSession Management Service

This service is used to handle the connections between the portal and the SRS nodes. The purpose of this service is to reduce the time consumed when establishing a connection with an SRS node.

Broker Service

This service offers some functionality to query the Discovery Service. It accesses the Discovery Service hosted by NEC and sends requests based on specific criteria.

Query Service

This service provides an interface for querying a specific SRS node and retrieving the results as SRS objects. This component uses the SRS based object retrieval.

Analysis Service

This service provides an interface for launching analysis tools from a specific SRS node and retrieving results as SRS objects. This component uses the SRS WsObjects web service client library interface with the SRS Application Launching functionality and result handling.

2.3.2 Interaction with the Portal

The Federated portal deals with two different servers:

- Semantic Broker
- Node Broker

The portal uses the semantic broker server to retrieve information necessary to the ontology browser.

Firstly, the portal retrieves the subjects from the Semantic broker via the Broker Service. Following this, the user can either browse a Subject or display the services available for the subject matching his requirements. This process enables the user to connect to the most relevant SRS server.

The portal uses the Node Broker to perform a full text search, an extended search or a similarity search on the selected SRS server. At this level, the portal only uses the WsSession Management Service, the Query Service, and the Analysis Service.

2.3.3 User Interface

The screenshots below show the user interface provided by the portal during different steps of a search:

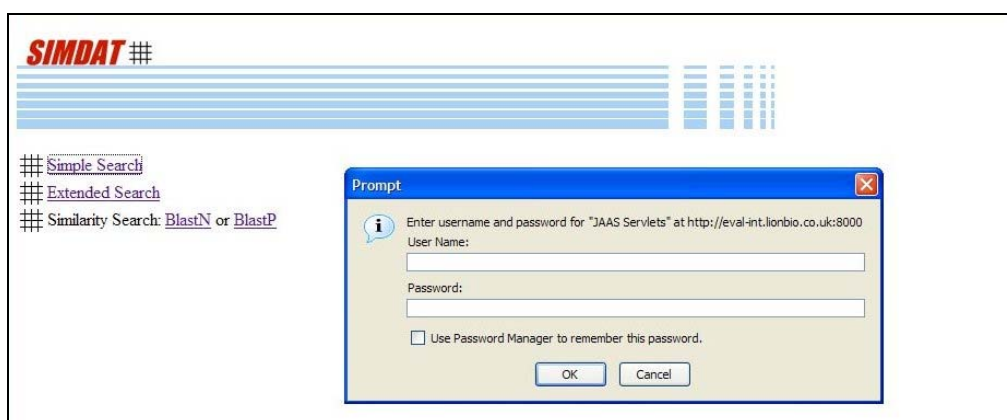


Figure 4: Authentication to Use the Services of the Portal

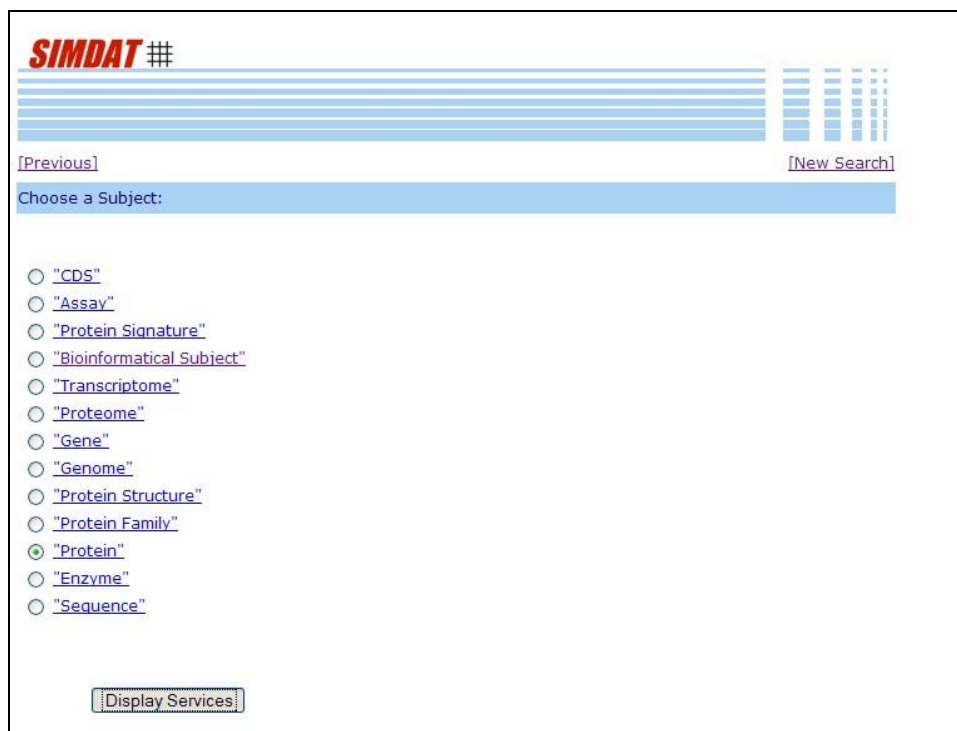


Figure 5: Browsing the SRS Ontology



Figure 6: Connection to the Most Relevant SRS Server for a Simple Search

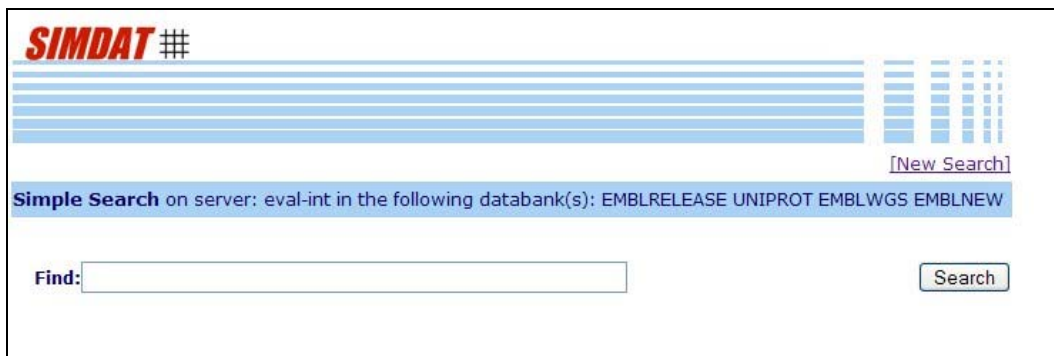


Figure 7: Simple Search on the Pre-selected Server

SIMDAT #

Primary Search

The query `[libs20={uniprot emblrelease emblnew emblwgs}-AllText:p53*]` retrieved 3708 entries

>[New Search] [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [Next]

Id	Identifiers	Description	SeqLength
UNIPROT:Q8NKM5	Q8NKM5_9CREN Q8NKM5	Putative fix B protein.	427
UNIPROT:Q977S2	Q977S2_9CREN Q977S2	Uncharacterized protein.	207
UNIPROT:O28333	O28333_ARCFU O28333	Cysteine proteinase, putative.	1088
UNIPROT:O29954	O29954_ARCFU O29954	Electron transfer flavoprotein, subunit alpha (EtfA).	317
UNIPROT:O29955	O29955_ARCFU	Electron transfer flavoprotein, subunit beta (EtfB).	247

Figure 8: Results Provided by the SRS Node

3 Security

3.1 Overview of NEC's Security Contributions for the Pharma Grid

The Pharma Grid coordinates pharmaceutical services, data and resources in order to support industrial and scientific pharmaceutical research. The Pharma Grid uses an untrustworthy network, the internet, for communication of highly sensitive data and to give access to services and resources. It is of paramount importance that access is granted only those who are authorized, that the data transmitted does not get eavesdropped, manipulated and is authentic. NEC's contribution to the security of Pharma Grids is twofold:

1. Provide a security mechanism that can ensure a high level of security of the data transmitted from one Grid endpoint to another, even across intermediaries.
2. Provide a dynamic access control mechanism that supports a variety of access models typical of Pharma collaboration scenarios. A dynamic mechanism is sought, that can be configured and can be adapted dynamically to changes in virtual organizations (VOs).

NEC's contributions for the first 12 months have focused primarily on Point 1. Nevertheless, regarding dynamic access control, specific Pharma requirements have been collected, and the main components of a Dynamic Access Control (DAC) module have been defined. Detailed specification work and implementation of DAC is planned from month 13.

3.2 End-to-End Security Properties for Pharma Grids

Pharma Grids must offer an adequate level of End-to-End Security, especially if they are to be deployed to form virtual organizations that include pharmaceutical companies. The main security properties that should be provided "end-to-end" should be the following:

-
- a. Strong Message Authenticity – not only the guarantee that the message received has been transmitted by the sender claimed in the message but also the assurance that it is indeed the message sent by the sender during that particular communication instance. Simple message authentication is not enough as it allows for replay attacks. Replay attacks can be particularly harmful for a Pharma Grid – a drug company may for example succeed in replaying old messages of a competing company causing outdated information to be regarded as fresh.
 - b. Message Integrity – the guarantee that the message was not tampered with during transmission.
 - c. Message Confidentiality – the guarantee that the message was not eavesdropped.
 - d. Non Repudiation and Accountability – in some cases claims contained in a message should not be deniable by their originator once the message is received.

The Pharma Grid-Middleware will be deployed in different organizations in different ways. Firewalls and proxies are likely to be in place to establish connections to highly protected security domains. Internal security policies will have to be observed when connecting resources and services to the Grid. It is important that the above security properties hold on messages from their endpoint of origin to their endpoint of destination. It is also important that end-to-end security can be achieved as independently as possible from internal security policies and network configurations.

3.3 Technology Uptake and PM1-PM12 Developments

During the FP5 project GEMSS [8], NEC has developed an end-to-end security framework. The framework has been deployed to secure interactions with medical simulation services which are grid enabled. The security properties required by GEMSS VOs are similar to those of the SIMDAT Pharma VOs discussed in the previous section. GEMSS grids process highly sensitive medical data – state of the art security technology had to be used by law to protect patient data and simulation results.

NEC's end-to-end security framework has been built to integrate with a bespoke web-services invocation mechanism provided by one of the partners in GEMSS. Most web and grid-service developments today, however, rely on standardized components which are not compatible with the invocation framework of GEMSS and early GRIA versions. For Java developments the JAX-RPC [5] (Java XML Remote Procedure Call) API by Sun is de facto a standard. SUN has a reference implementation of JAX-RPC, which it includes in its Web Services Development Pack [6]. Axis [7], a very popular open-source framework for the development of web services, implements JAX-RPC as well and it uses it as the basis for a more enhanced invocation engine.

One of the first tasks of NEC, in order to take up the GEMSS security technology for use in Pharma and possible other SIMDAT grids was to make the end-to-end security framework independent from bespoke web-service invocation mechanisms. The goal is to support JAX-RPC implementations and Axis engines. The goal has been achieved by NEC in the E2ESecurity framework version 1.0. A number of adjustments and modifications had to be made, as well as a series of tests. E2ESecurity is based on the GEMSS developments but is now a framework which can be deployed on any web and grid service client that uses JAX-

RPC implementations or Axis as well as Axis-enabled services. During the first 12 months of the SIMDAT project NEC made E2ESecurity a widely deployable framework which could also be used by other activities besides Pharma. E2ESecurity includes a number of improvements and additions to the GEMSS component. An important area of improvement is the definition of end-point security policies which is now more flexible and allows finer granularity. Another addition to the framework is a mechanism that allows for concurrent client invocations to the same service. This has been a necessary addition as portals that function as clients often need to establish concurrent service invocations. The framework is available on BSCW under the SIMDAT Contract and Consortium Agreement.

The E2ESecurity framework requires a public-key infrastructure (PKI) [9] to be in place. NEC set up an activity-wide PKI for SIMDAT Pharma on the basis of the GEMSS PKI experience. The SIMDAT Pharma PKI is at the moment used only for generating and distributing test certificates and for the PM12 prototype keystores.

3.4 E2ESecurity – Structure and Deployment Requirements

NEC's E2ESecurity is a framework for increasing the end-to-end security level of distributed applications, based on web-services. The framework can be used for end-to-end mutual authentication and shared key exchange as well as message-level security. E2ESecurity can be used to ensure a high level of security independently from network topologies of service providers – which may change from provider to provider and may not be disclosed to the outside world. The framework is particularly valuable when the underlying network topology includes intermediaries that act above the transport level. These intermediaries can range from fully fledged intermediate web-services to simple web servers and proxies in demilitarized zones.

E2ESecurity includes the following security mechanisms to achieve the properties we discussed earlier:

- Message-level signatures. SOAP [10] messages are signed to ensure end-to-end message integrity and message authentication. Signatures are included in soap messages and verified according to the WS-Security [11] specification.
- Mutual authentication of endpoints. An authentication protocol is executed above the SOAP layer between clients and a trust (authentication) service installed on SRS servers. WS-Trust [12] is used as a language of interaction between clients and the trust service.
- Shared-key exchange between endpoints. During the authentication protocol a symmetric-key is generated by the trust service and transmitted secretly to the client. This key is short lived and can be used for message-level encryption during a client/service session. The exchanged key forms a security context token which testifies the successful end-point authentication and the secrecy of the key. The token expires after some time and new authentication and secret key are re-negotiated. WS-SecureConversation [13] is used as a language for security context tokens.
- Message-level encryption. SOAP messages are encrypted with the symmetric key that has been exchanged between client and service. Encryption follows the WS-Security specification.

The above security measures are enabled on the client and on the server by setting corresponding security policies. Different combinations of basic policies are possible to build a security policy that guarantees the right level of protection.

3.4.1 Main E2ESecurity components

- *Policies*: They define security requirements that must be satisfied at a certain stage of a service invocation. Currently E2ESecurity includes two kinds of policies. Policies concerning request and response messages are:
 - encryption of a (SOAP) message and its (SOAP) attachments,
 - decryption of a (SOAP) message and its (SOAP) attachments,
 - signature of a (SOAP) message,
 - signature verification of a (SOAP) message.

Policies describing security contexts between client and invoked service are:

- X.509 [14] certificate exchange,
- mutual authentication and shared symmetric-key exchange.

Policies are configurable for individual services and their clients. The policy framework can easily be extended with additional policies and corresponding policy enforcers.

- *Policy Provider*: provides policy instances for endpoints. The current E2ESecurity module instantiates policies defined in client and server-side configuration files.
- *Policy Enforcers*: They enforce policies, by activating the appropriate security protocols (encryption/decryption of SOAP messages, mutual authentication, etc.). The objects of the security protocols are *Tokens* to which processors have access via a *Security Context*. Each policy requires a corresponding enforcer. Whenever a new policy is defined, it should come with an enforcer and an enforcer provider.
- *Enforcer Provider*: For each policy described by a policy descriptor, the enforcer provider provides an enforcer for the policy.
- *Tokens*: They are the main objects of security protocols. Tokens are not only keys and certificates but also context tokens which describe the properties of a multi-party security context. New tokens can be added if required by new policies and enforcers.
- *Security Context*: Tokens are accessed and managed through a security context. The E2ESecurity module implements a security context to add and retrieve tokens from a keystore, to generate new symmetric keys and set up context tokens, and to check if a token can be trusted.
- *Token Managers*: They are the initiators of security protocols used to set up a multi-party security context. Typically, a security context token is generated as result of a successful protocol execution. Currently E2ESecurity implements two token managers:
 - initiator role in an X.509 certificate exchange, and
 - initiator role in a protocol for mutual authentication and shared-key exchange.

Both use WS-Trust and WS-SecureConversation languages for their SOAP payloads.

- *Token Services*: They play the responder role in security protocols for multi-party security context tokens. Token services are typically stateful services as frequently steps in a protocol exchange depend on previous steps in the protocol. Currently we have implemented two token services:
 - responder role in an X.509 certificate exchange,
 - responder role in a protocol for mutual authentication and shared-key exchange.

Both are SOAP services and use WS-Trust and WS-SecureConversation languages for their SOAP payloads. These services act on behalf of the other services installed under the same security domain: they access the keystore of a service, exchange certificates and secret keys and also authenticate on its behalf. A registration mechanism ensures that token services can only act on behalf of registered services.

- *Sign-On*: Provides keystore location and password information to E2ESecurity components. This information can be specified in a configuration file. At start-up, the user is queried for a password input if the password is not set in the configuration file.
- *Security Handlers*: are the point of interaction between E2ESecurity and the environments into which it gets deployed. To enable advanced security, these handlers need to be installed on a service and on all clients that interact with the service. There are client-side handlers and service-side handlers. The current E2ESecurity module includes implementations for client-side and server-side SecurityHandlers for Axis in the package

`de.nece.ccr1.e2esec.transport.axis`

and class names:

- SecurityHandlerClientOut
- SecurityHandlerClientIn
- SecurityHandlerServiceOut
- SecurityHanderServiceIn

In addition to Axis, a client-side SecurityHandler for “pure” JAX-RPC implementations has been developed and included as part of the E2ESecurity module:

`de.nece.ccr1.e2esec.transport.jaxrpc.SecurityHandlerClient`

This way, in addition to Axis, E2ESecurity can be installed also on all web-service clients that perform service invocations through the JAX-RPC API. SRS clients, for example, were based initially on SUN’s JAX-RPC reference implementation instead of Axis.

3.4.2 General Requirements

- Java 1.5
- Services enabled through Axis 1.2
- Axis 1.2 or JAX-RPC-based clients

3.4.3 Installation and Configuration Requirements

a. Axis clients:

A Java system property with name `e2esec.config` must be set to point to the directory containing the E2ESecurity configuration files.

A Java keystore (JKS) is needed containing private key and X.509 certificate of the client as well as the corresponding CA certificate.

The security handlers are deployed via a client-side web-service deployment descriptor (WSDD).

b. JAX-RPC clients:

A Java system property with name `e2esec.config` must be set to point to the directory containing the E2ESecurity configuration files.

A Java keystore (JKS) is needed containing private key and X.509 certificate of the client as well as the corresponding CA certificate.

The class `de.nece.ccrl.e2esec.transport.jaxrpc.SecurityHandlerClient` can be deployed programmatically on a service port and added at the end of the handler chain as follows:

```
SecurityHandlerClient.addToHandlers(service,portname,session);
```

with `service` being a `javax.xml.rpc.Service` instance and `portname` the qualified name of a port of the service. If concurrent service objects are executed on a client, a session identifier should be passed to distinguish among sessions belonging to different service objects. Invocations that share the same session are always handled sequentially even if executed in concurrent threads.

c. Services:

A Java system property with name `e2esec.config` must be set to point to the directory containing the E2ESecurity configuration files.

A Java keystore (JKS) is needed containing private key and X.509 certificate of the service as well as the corresponding CA certificate. The service certificate must contain a prefix of the service endpoint as “Alternative Subject Name”. A certificate whose alternative subject name matches more than one service endpoint is understood as shared among all services with matching endpoint prefix. In this way, E2ESecurity allows to set up groups of services that share the same security tokens and security policies.

The security handlers and token services are deployed via the server-side web-service deployment descriptor (WSDD) (see [7]).

3.5 PM12 Prototype – E2ESecurity Integration and Deployment

The PM12 prototype of the Pharma activity represents a simple but common and useful grid scenario where five organizations join in a CA-centric VO to provide services and access to biological and pharmacological data. The prototype is based on SRS which is used as backbone technology for accessing data and services exposed by a Pharma VO provider node. Each node exposes its SRS services (hosted on SRS servers) through an Axis dispatcher service. Users are web-clients operating a browser, which connects to a federated portal. The federated portal is a gateway which in cooperation with a semantic broker selects and interacts with SRS services (through an Axis dispatcher) hosted by some of the VO provider nodes. Axis is used for client-service invocations of the federated portal with the Axis dispatchers of the provider nodes.

In the prototype we deploy E2ESecurity for securing the interactions between the portal and the nodes providing SRS services. The main advantage of using E2ESecurity in this case is that it helps achieving a high level of security for portal/provider connections independently from the way the service providers deploy their SRS servers on their network – they may install them in highly protected domains reachable only through a forwarding proxy in a DMZ. E2ESecurity in this case can cope better with local security policies, which would be very difficult to change in order to accommodate other security mechanisms. The prototype will not make use of WSOBjects based clients, however, if it were so, our integration work would allow end-to-end security between a WSOBject client connecting directly to a service provider. For the prototype, users use their browsers to connect to the portal. They are password authenticated at the portal and their connection is protected by SSL [15]. Since the protocol for user/portal interactions is not SOAP based, but HTML-HTTP, we can not deploy E2ESecurity for securing it at the moment. However, the username token that is authenticated at the portal is carried to a provider endpoint by E2ESecurity.

E2ESecurity between portal and service providers will allow dynamic access control mechanisms, planned for the next phase of SIMDAT, to transmit authorization tokens and attributes in a secure way.

E2ESecurity integration on the service provider side is done by installing the appropriate handlers on the dispatcher Axis services. Installation and configuration procedures have been outlined in the previous section.

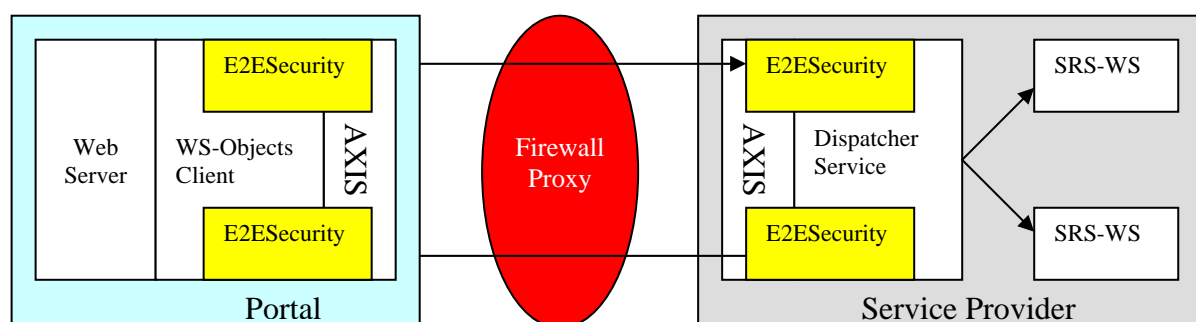


Figure 9: Deployment of E2ESecurity on the Pharma Prototype

A goal of our development was to make integrating the security component with the portal as easy as possible. The portal part into which we integrated E2ESecurity is basically a WSOBjects client based on automatically generated stubs. The Stubs have been modified in order to include the appropriate security handlers. E2ESecurity on the portal (and any other WSOBjects client) now gets activated by using the NEC_SRSSecurityManager class instead of the BasicProfileSecurityManger provided with WSOBjects.

4 Knowledge Model

The knowledge model is built from multiple connected ontologies. Ontologies provide a shared understanding of a domain of interest to support communication among human and computer agents (such as Web Services). Thus, ontologies serve as formal models of certain domains, helping to structure information sources in a comprehensible way and describe complex dependencies efficiently.

Ontologies allow setting up complete models rather than just attaching keywords to a service reference. This makes it possible to describe complex structures and encapsulate them with different levels of abstraction, taking advantage of the features that ontology-based approaches provide, such as hierarchical structures, inverse relations, etc.

In order to support professional users the description of services needs to consider domain specific concepts, e.g. regarding the specific algorithms a service can be based on. This is described in the following sections.

Once a model for the characterization of services and supporting models have been set up, concrete service instance needs to be annotated in order to link the formal description and the reference to the service. This is realized on the base of an annotation schema defined in XML syntax. The file can be created manually or with the help of an annotation tool (see chapter 6). Once the data has been created, the annotation file can be sent to a registry service where it is validated and stored (SRS node registration).

4.1 Service Model

The service ontology is conceptionally based on OWL-S: A service is described by one process, presents one or multiple profiles and supports one or multiple groundings.

The service profile provides all information needed for advertising and discovering services. The process model gives a detailed description of a service's operation. The grounding contains details on how to interoperate with a service.

The following section contains a brief description of the top-level concepts of OWL-S while the subsequent section explains how those OWL-S concepts are applied and extended to describe SRS services.

Alternatively the emerging WSMO model could have been used as a base. Both models do not yet have the status of a W3C standard and it is not foreseeable which of them will succeed. One of the coming activities in the area of service discovery will be the evaluation of WSMO as an alternative.

4.1.1 OWL-S

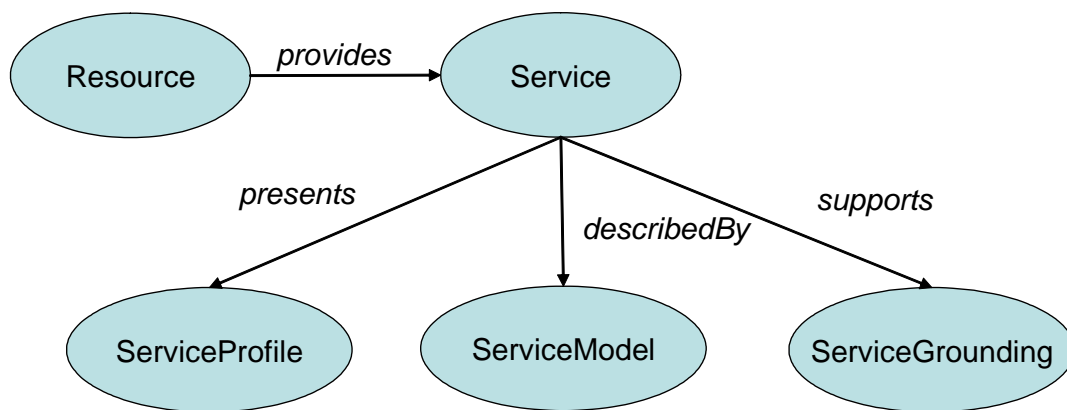


Figure 10:Top-level concepts of OWL-S

- **Service**

This concept directly corresponds to the actual service that is described semantically (every service that is described maps onto an instance of this concept);

- **Service Profile**

The service profile specifies the functionality of a service. The concept is the top-level starting point for customizations of the OWL-S model that support the retrieval of suitable services based on their semantic description. Within SIMDAT pharma the service profile has been used to describe the relations of services to data sources.

- **Service Model**

The service model specifies the how the functionality of the particular service is realized. This concept is currently not used within the semantic broker.

- **Service Grounding**

The service grounding specifies how the service has to be invoked (syntax of calls to the service; needs to be known for clients that want to use the service).

4.1.2 Application of OWL-S and extensions

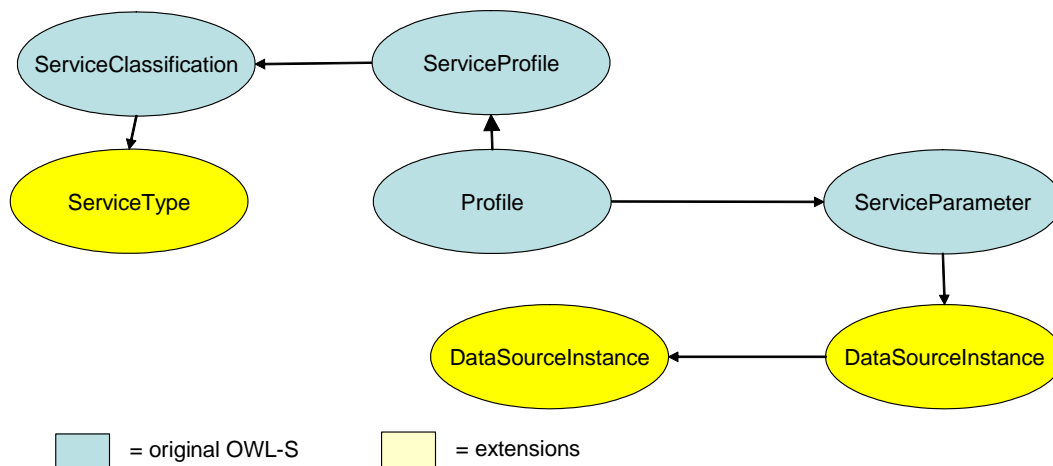


Figure 11: OWL-S extensions (top level)

4.1.2.1 Profile

An OWL-S service profile describes a service as a function of three basic types of information: what organization provides the service, what function the service computes, and a host of features that specify characteristics of the service.

The first information type is given by the contact information. For our purposes the actor class described in "<http://www.daml.org/services/owl-s/1.1/ActorDefault.owl>" is sufficient. For clarity a subclass "Database Provider" has been added to distinguish this special role.

The function of the service is described by a domain specific "Service Type" ontology, a classification system described in a separate chapter (see section 4.1.3). The available data sources for an SRS web service can be seen as the third type of information. This is the hardest part, as it needed to describe the primary entities, attributes, especially name, type, structure and semantic meaning of each attribute, but also the release / version of a public data source.

Both service type and data sources have been mapped to subclasses of *ServiceParameter*.

OWL-S provides three other possible parameters for classifying a service. But all three imply the usage of an existing classification system. The *service category* can be used with a taxonomy defined somewhere outside OWL-S and OWL. The *service classification* can be used to map to a OWL ontology of services, such as an OWL specification of NAICS and the service product to an OWL ontology of products, e.g. an OWL specification of UNSPSC. As all three ways are no good fit to the Service Type Ontology specified in F-Logic, it was decided not to use them.

4.1.2.2 Process

OWL-S is designed to describe fine-grained services, where each operation has a well-defined input and output parameters with atomic character. In contrast, the SRS architecture makes use of very flexible result sets (called "loaders") which are described by meta data. Therefore the meta data is not directly related to the process, but depends on the context, i.e. data sources or applications, in which the service operation is invoked.

The processes have very generic input and output parameters and have been designed to align with typical tasks performed with a SRS server. Typical examples are:

-
- Performing a full text search on one or multiple data sources (see process "<http://pharma.simdat.eu/process/#FulltextSearch> ")
 - Performing an extended search with constraints on multiple attributes (see process "<http://pharma.simdat.eu/process/#ExtendedSearch>")
 - Performing a similarity search with BLAST (see processes "<http://pharma.simdat.eu/process/#Blastn>" and "<http://pharma.simdat.eu/process/#Blastp>")

The OWL-S specification assumes that listing input and output parameters and their semantic meaning describes a service operation sufficiently. For our purposes this is only partly true. As the accessed data sources deal with various biological, chemical and medical aspects, the process can describe the operation only on a syntactic level. To “understand” the meaning, additional semantic meta data is needed for describing the underlying data source, their attributes and their hierarchical structure. For public data sources the concrete installed release must be known, too. All this information is part of meta data provided about the data source instances in the profile (see section 4.1.4).

Grounding

In OWL-S, both process and profile are thought as abstract representations of the service. Only the grounding deals with the concrete specification of the protocol and format. Usually this is mapped to a concrete web service with help of its WSDL.

For SRS this is complicated by the fact, that there is no WSDL (probably because of the complex structure uses in the SRS API). When using an SRS web service normally some proprietary client code is used to ease the usage of the flexible structure of the result sets. Therefore a proprietary grounding has been defined to sign this behaviour. It implies the usage of the special web service interface of SRS (WsObjects) and provides its URL address and protocol version.

This is an area which needs further investigation how SRS and its web service interface can be aligned to standards.

As the concrete binding to the WsObjects is out of scope for the PM12, the same SrsWebServiceGrounding can be used for all services of one server. In contrast, OWL-S normally restricts the grounding to one service.

4.1.3 Service Type Ontology

The Service Type Ontology represents a classification system for life science related services. The taxonomy has three starting points:

- Computation Service (services providing some computation)
- Data Service (services providing access to some data sources)
- Service by Scientific Domain (alternative starting point classifying by scientific domain)

The classification system is especially useful when goals are specified on a high level of abstraction. It allows distinguishing services on a very basic level. The Service Type Ontology is illustrated in the annex. It does not only cover the typical types frequently needed by SRS-users (such as BLAST) but also service outside the bioinformatics domain.

4.1.4 Data Source Ontology

Data sources have multiple aspects that are needed for our purposes: One has to distinguish at least between the abstract data source definition, which describes what information can be searched and retrieved, and the concrete data source instance on a node, which needs to be described by the release related information.

Additionally for dealing with sub- and supersets or derived datasets, information about such relations needs to be stored.

4.1.4.1 DataSourceDef

A “DataSourceDef” has a name, a description and may have alternative names (synonyms). Additionally provider, subject and restrictions about the species can be specified for a data source definition.

The provider property contains actor instances about the publisher and data curator for the given data source. The subject is a separate ontology describing the unit of interest of a data source entry, e.g. if it contains information about proteins, diseases, genes etc. See section 4.1.5 for more details. The NcbiTaxonomy property of a data source definition can be used to specify for which organisms the data source provides information. It is based on the NCBI Taxonomy (see <http://www.ncbi.nlm.nih.gov/Taxonomy/> for details).

There are several subclasses of DataSourceDef:

- DatabankDef is a concrete databank which has an atomic character, i.e. it is the unit released by the provider of the databank
- DataSourceGroup is a container to define a superset data source consisting of multiple member data sources
- RestrictedDbDef is a subset of a data source which is either restricted by its contained primary entities or its organisms
- BlastDbDef is a special data source needed for blast services and are normally generated/based on another data source

For each subtype of DataSourceDef some properties may be inferred with rules from the data sources, they are related to.

4.1.4.2 Supersets

The subclass DataSourceGroup is used to build supersets of data sources. A DataSourceGroup is meant to contain all data from its member data sources. In the SRS system this corresponds to “virtual libraries”.

4.1.4.3 Subsets

The subclass RestrictedDbDef is used to define a restricted set of a data source. Supported restrictions are the primary entity and the organism.

4.1.4.4 Derived Data Sources

The subclass BlastDbDef is used to define a data source which is prepared to be used as Blast databank. It describes from which data source containing sequence information the blast databank has been created.

4.1.5 Subject Ontology

This is an ontology for scientific terminology. It is used to specify the targeted kind of entity the information in a data source is describing. It contains concepts like “Biological Entity”, “Enzyme”, “Gene”, etc.

5 Semantic Broker

In this chapter we describe one of the technological key contributions of the Pharma application domain: the semantic broker. As mentioned in the introduction there is a requirement in pharmaceutical and other problem solving environments to publish and discover services also by their semantics, i.e., via an ontological description of the services. The knowledge model which has been introduced in the previous chapter builds the foundation for this description process. The semantic broker provides now functionality for both client (end user) and service provider to discover or publish services, respectively, according to a controlled vocabulary (knowledge model).

Basically, our approach can be understood as a semantic extension to the existing OASIS UDDI registry standard [16].

5.1 Semantic Broker: System Design

In the first working prototype, we implement the semantic broker as such a type of advanced directory service by means of the application of the state of the art SW (Semantic Web) and SWS (Semantic Web Services) technologies for automating SRS service discovery and selection process. In Figure 12 we illustrate the system architecture design of the semantic broker. The two kernel enabling technologies here are the OWL-DLP/F-Logic based bioinformatics ontology and OWL-S based service annotation and discovery/matchmaking.

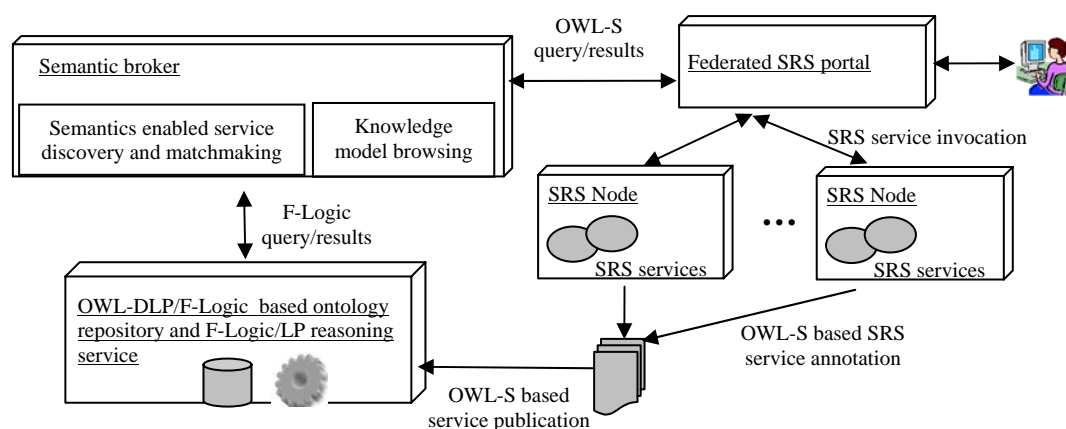


Figure 12: System Architecture Design of the Semantic Broker

5.1.1 OWL-DLP/F-Logic Based Bioinformatics Ontology and F-Logic/LP Reasoning Service

The bioinformatics ontology is the key enabling technology to the semantics enabled SRS service discovery and matchmaking in the semantic broker. We introduce some proven best practices popular in the SW and SWS community into the system design, and further adopt a standards-oriented design approach, taking system interoperability, reusability, and maintainability as the highest priorities.

First of all, we adopt a subset of the W3C OWL standard, i.e. OWL-DLP [1], as the technical basis of the application specific bioinformatics ontologies. As we use OntoStudio / Ontobroker, an industry strength LP product suite, in the project, we internally use F-Logic to represent OWL-DLP based bioinformatics ontologies, and further take advantage of Ontobroker's F-Logic/LP reasoning capability to get ontology reasoning support for the semantics enabled service discovery and matchmaking. Supported by OntoStudio / Ontobroker, the F-Logic represented bioinformatics ontologies can easily be transformed into the standard OWL-DLP/OWL represented ontologies, which can further be interpreted through standard OWL-DLP/OWL-DL reasoners such as Racer. As OWL-DLP is a common fragment of DL and LP, the semantic broker as well as its accompanying bioinformatics ontologies can easily be used with the DL based implementation/representation. In consequence, we can to the largest extent ensure semantic broker's interoperability with both DL and LP paradigms, in particular with two leading SWS standardization efforts: OWL/OWL-S and WSML/WSMO, as well as with some other DL based semantics enabled bioinformatics applications such as myGrid (<http://www.mygrid.org.uk/>).

Second, we adopt a distributed and modularized ontology structure, which is one of the key best practices in the SW community, but not yet fully acknowledged in the semantics enabled bioinformatics system design. While developing ontologies in the semantic broker, we clearly differentiate between the application specific SRS ontology, general bioinformatics domain ontology such as the NCBI taxonomy, and the service annotation ontology, i.e. OWL-S. In consequence, the reusability and maintainability of SRS bioinformatics ontologies are greatly promoted.

5.1.2 OWL-S-Based Service Annotation and Discovery/Matchmaking

In the first working prototype, each SRS services are annotated based on the DAML OWL-S 1.1 specification (also W3C Member Submission since Nov. 2004), which are then published onto the semantic broker through a service publication/registration Web service developed by Ontoprise. As the SRS service annotation is relatively complicated for the service provider, we further introduce a service annotation tool: SCAI TUAM to assist service providers in annotating SRS services using correct OWL-S and domain ontologies.

In terms of the current SRS service annotation requirements, we mainly focus on the OWL-S service profile and service grounding annotation for the SRS service discovery, matchmaking and grounding. In general, we choose following OWL-S service properties for the semantically rich SRS services annotation [2]:

- *profile:textDescription*: used to annotate general SRS service information.
- *profile:contactInformation*: used to annotate the SRS service provider.
- *profile:has_process and process:process* : used to annotate the SRS service process.
- *profile:serviceClassification*: used to annotate the SRS service classification.
- *profile:serviceParameter*: used to annotate SRS data sources.
- *profile:hasInput, profile:hasOutput, and profile:hasParameter*: used to annotate SRS service inputs/outputs.
- *service:ServiceGrounding*: used to annotate SRS service grounding.

In Figure 13, we illustrate the knowledge model used for the OWL-S based SRS service annotation and discovery/matchmaking.

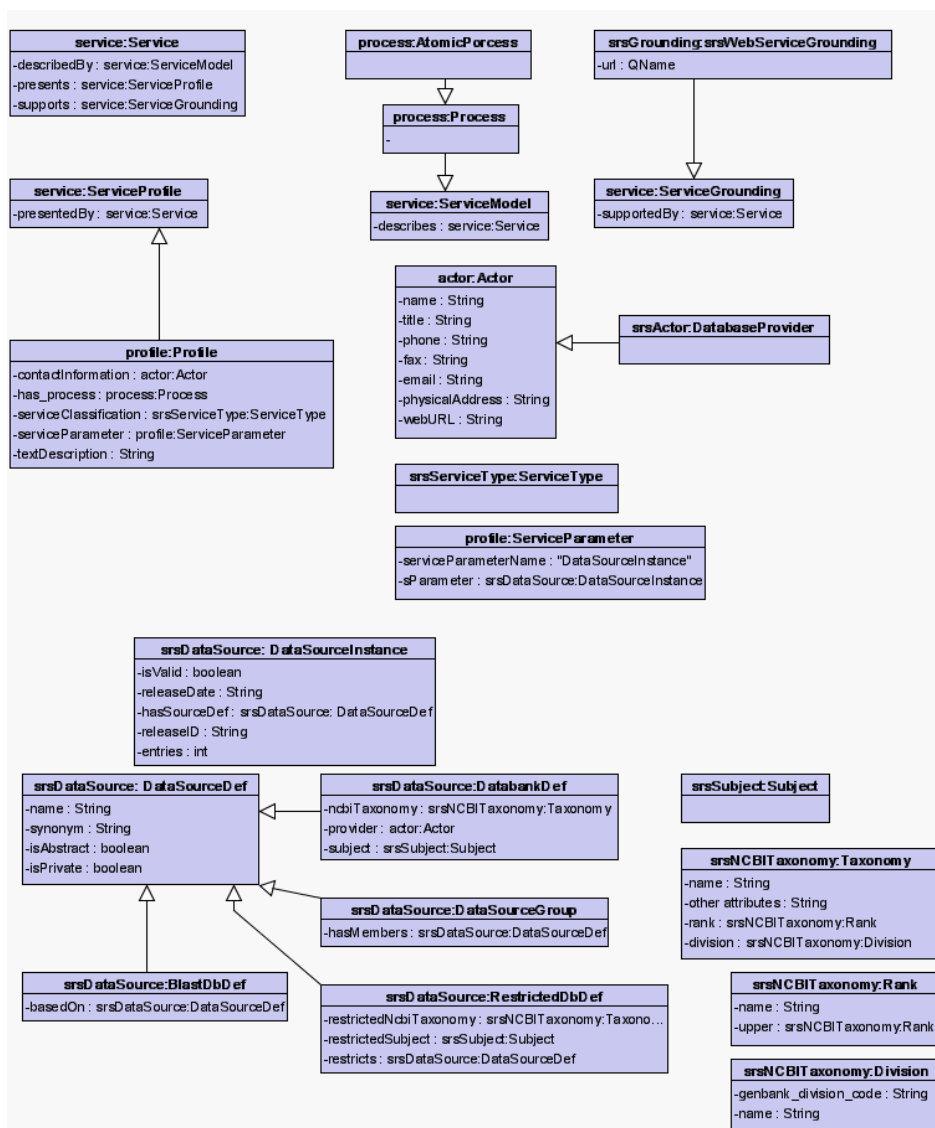


Figure 13: Knowledge Model Used for the OWL-S-Based SRS Service Annotation and Discovery/Matchmaking

In the first working prototype, we implement the semantics enabled service matchmaking principally based on the matchmaking algorithm proposed in [3]. This algorithm identifies four types of semantic “degree of match” between the user requirement and the SRS service annotation, i.e., exact, plugin, subsume, and fail. According to the users’ requirements, the semantics broker can calculate the matching degree of each SRS services and then recommend sets of candidate services to the federated portal together with respective matching degrees. The users can then choose recommended services via the federated portal to set up the *in-silico* experiment workflow.

5.2 Semantic Broker: Functionalities

In the first working prototype, the semantic broker implements two functionalities: the knowledge model browsing and semantics enabled service discovery and matchmaking.

5.2.1 Knowledge Model Browsing

The knowledge model browsing functionality is purposed to assist users in building semantic queries via the federated portal using controlled vocabularies. Technically, the federated SRS

portal builds the GUI for the users in virtue of the knowledge model browsing functionality. In the next project stage, the federated portal may take advantage of some advanced client-side ontology browsing tools in order to get more efficient support for the GUI construction.

So far the semantic broker implements the knowledge model browsing functionality through four standard Web service operations:

- *getRootConcepts*: return top level concepts in the knowledge model.
- *getSubconcepts* : return all direct or indirect sub-concepts of a given concept.
- *getInstances*: return all direct or indirect instances of a given concept together with all their properties.
- *getInstanceMatchings*: find specific instances according to the user required properties.

5.2.2 Semantics Enabled Service Discovery and Matchmaking

The semantic broker implements the semantics enabled service discovery and matchmaking functionality through the standard Web service operation: *getServiceMatchings*. In terms of the knowledge model illustrated in Figure 13, a semantic query can point to any of the data entries in the knowledge model, as well as to any combination of these data entries.

In the first working prototype, the semantics enabled matchmaking is mainly based on two ontology sets: the SRS service type ontology and SRS data source subject ontology. In addition to the semantics enabled matchmaking, the non-semantics-enabled matchmaking based on all other data entries such as the service provider (“exact” matchmaking based on name, title, email, phone, etc.) can also be included in the service matchmaking process. The semantic broker calculates the overall service matching degree taking into account both the semantics enabled and non-semantics-enabled matchmaking results.

5.3 Semantic Broker: Client-Side Query API

In order to hide the OWL-S query syntax from the federated portal and SRS users, we further develop the client side query API to wrap OWL-S specific query syntax. In Figure 14 we illustrate the client side API. Likewise, we also develop the query result wrapper API, which allows the federated portal to interpret OWL-S based query results. By means of these client side APIs, the federated portal can construct semantic queries and deal with the query results without the need of any OWL-S knowledge.

5.4 Semantic Broker: Deployment

For the first working prototype, we set up a semantic broker server at NEC. In Figure 15 we illustrate the semantic broker server architecture.

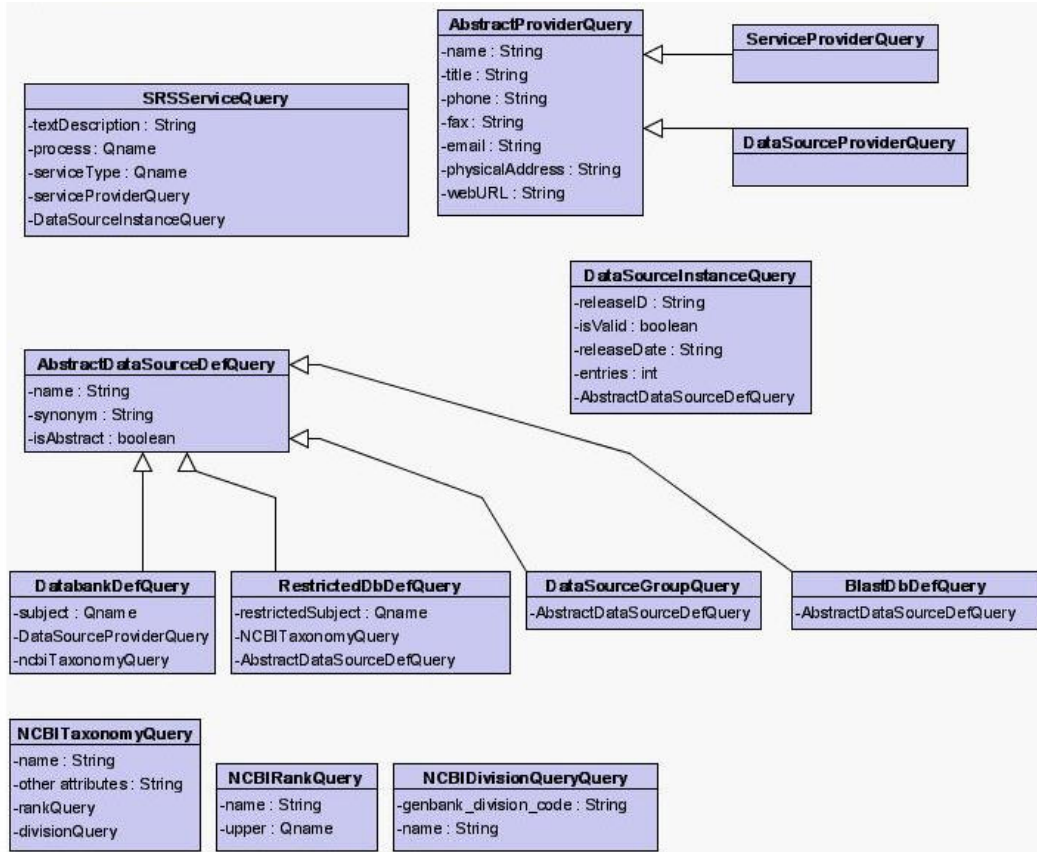


Figure 14: Client-Side Query API

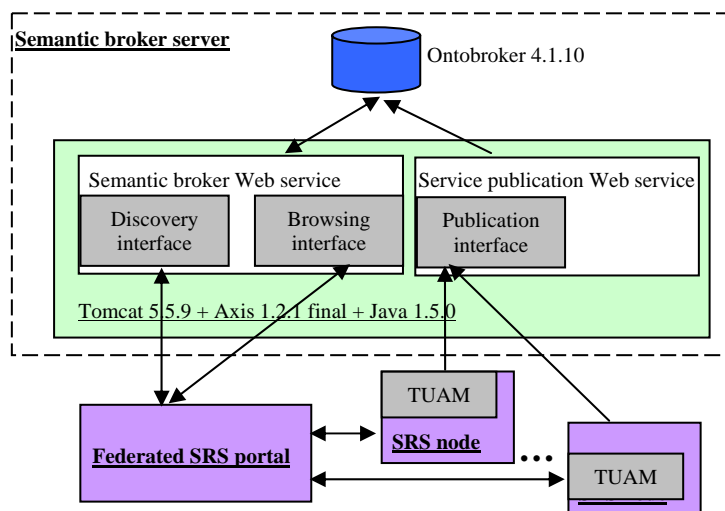


Figure 15: Semantic Broker Server Architecture

On the semantic broker server, the semantic broker itself is deployed as a standard Web service, which can directly be accessed by the federated portal for the knowledge model browsing or semantics enabled service discovery and matchmaking. Besides, we also deploy the SRS service publication Web service on the semantic broker for the SRS service publication/registration. On each SRS node, all SRS service descriptions are encapsulated in a single node description file, which can be published onto the semantic broker server through

standard Web service invocations and after that be updated by the service provider at any time.

5.5 Semantic Broker: Software Release

For the first working prototype, we release the semantic broker software for both server side and client side installations [4]. The server side semantic broker release is tested under Axis 1.2.1 final, Tomcat 5.5.9, Java 1.5.0_04, ANT 1.6.5, both on Windows XP and Fedora Core 3/4 Linux platforms. As a Tomcat Web application, the server side release can easily be deployed on any servers with Ontobroker pre-installed. The client side release includes a semantic broker test client and a service publication test client. It provides sufficient demos for semantic query construction and service publication. Supported by Ant scripts, the client side release can automatically generate classes based on WSDL documents for the dynamic binding to the server side Web services.

6 Service Annotation

6.1 Overview

As is well known from the Semantic Web paradigm, semantic annotations in the form of statement triples (e.g. RDF) [21] provide the basis for content-based search, instead of traditional string-matching approaches. The triples bind together subjects from the content part (e.g. a web page) and objects from an ontology (a controlled hierarchical terminology with relations and rules between the terms) via predicates, i.e. relationship types with a well-defined meaning.

As with static content in the semantic web, this kind of annotation also works with services on the net that offer platform-independent data access or compute facilities. Provided the web or rather grid-services are presented through one standardized interface, like WSDL (web service description language), semantic annotation can be applied to those base descriptions for semantic enrichment and disambiguation of service descriptions that are mostly restricted to arbitrary free text without any well-specified meaning.

SCAI-Bio has developed a tool for universal annotation and mediation (TUAM [20]) that attempts to solve the annotation problem – how can existing services on the grid be enhanced without modification of the service source documents – in the most general manner possible (see Figure 16). The implemented solution builds on the concept of semantic triples that are stored persistently independent of the source data (ontologies and grid service lists). The semantic connection is stored non-invasively as pointers to the respective data source along with a relationship type between the two data items.

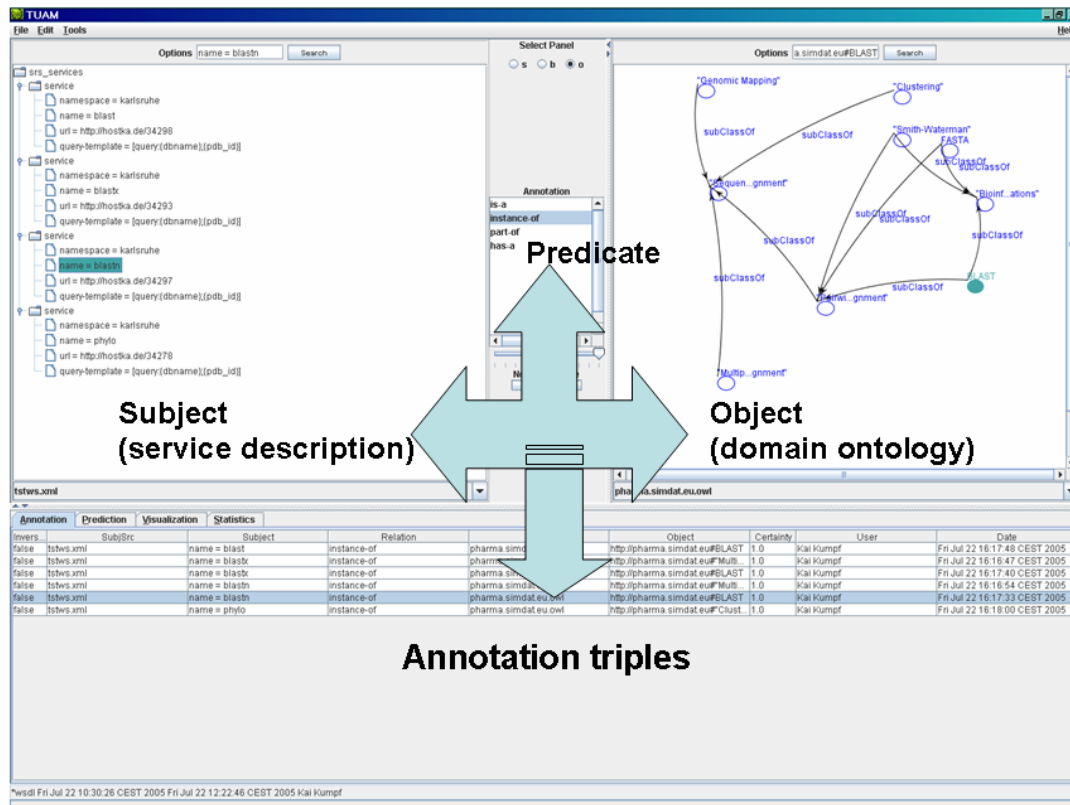


Figure 16: Annotation Triples in TUAM

The data structure resulting from the annotation process – basically a new semantic network graph – serves as a mediator for queries based on the semantically well described ontology part. First, classes are identified, that fulfil the query constraints, then instances of those classes in the annotations (pointers to grid services) are identified, and in a third step, the data those instances refer to (the services URLs) are retrieved.

The annotation triples can either be accessed through TUAM itself, through an annotation database view, or as RDF file export for further use. The Pharma-Grid solution rests on the database view, since this provides the most readily accessible and up-to-date view of the semantic annotations.

6.2 Terminology

- Annotation Refers to markup of data using other data
- Mediation is the process of retrieving information via a mediator – a data bit that binds together the query domain and the query range
- semantic annotation annotating data with semantically meaningful data, e.g. ontologies
- Triple the RDF triple <subject>,<predicate>,<object> forming an atomic statement, e.g. “service abc” “is-instance-of” “bioinformatics tool”.
- RDF [21] Resource Description Framework, an XML dialect for annotating web resource in the Semantic Web, supplying meta information

Semantic Web [23] The WWW plus meta information, allowing content-based search

Semantic Grid Semantic Web philosophy transferred to the Grid

triple store [22] Any storage facility that allows persistent storage of RDF or similar triple statements

6.3 TUAM Client

6.3.1 General Architecture and GUI

TUAM was designed to incorporate data from various sources and display them in plugin viewers implementing an interface that allows indexed access to the data items displayed. Indexed access means that we can identify unique keys for each datum selected. The keys of the annotated data will be stored alongside the datum values.

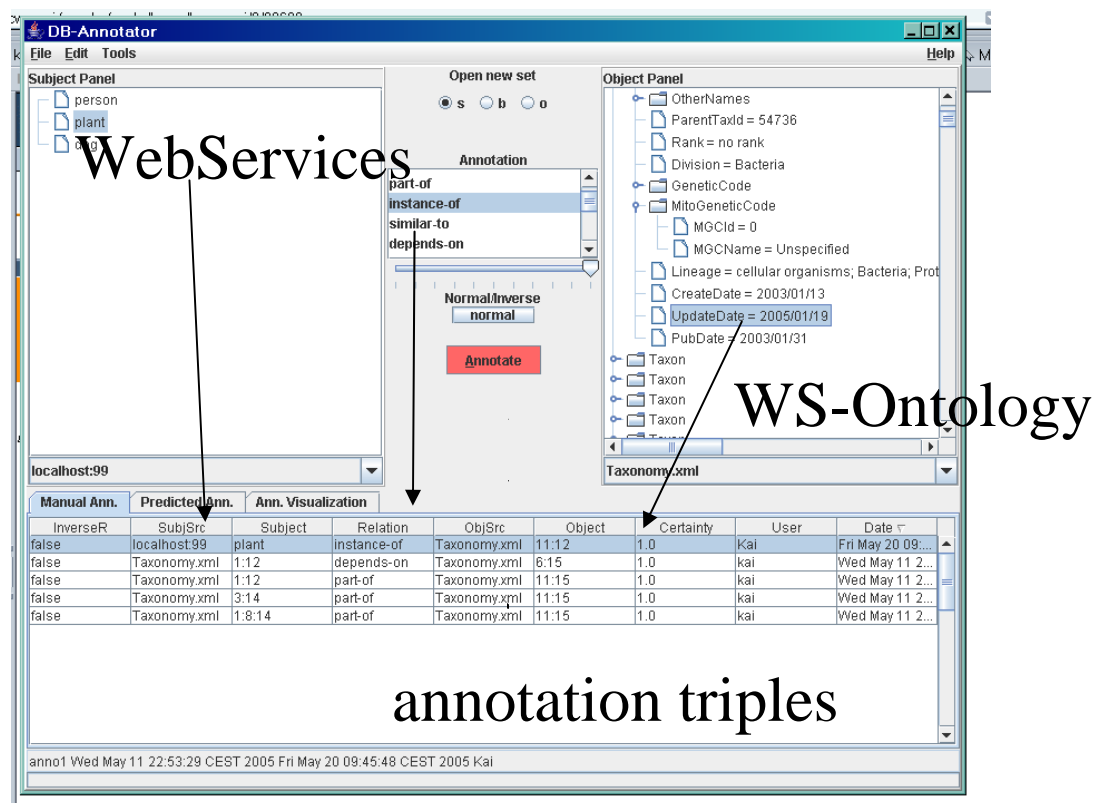


Figure 17: TUAM GUI Screenshot

For the purposes of the Pharma grid we have to deal with XML descriptions of grid services and a service domain ontology which is usually represented as an RDF graph structure. The annotation procedure consists of the following steps (c.f. Figure 17):

1. open both data sources (XML service descriptions and domain ontology) either from a file or from the internet via a URL into the subject and object panels respectively
2. choose a service items in the subject panel
3. choose a domain ontology class in the object panel
4. choose a relationship-type – the predicate from the centre list: we have designed the ontology in a way that the relationship type is instance-of, i.e. we are instantiating the chosen class

-
5. click “annotate”
 6. fill in non-optional class attributes that constrain the class instance

The annotation will be stored persistently in the relational database in extended triple form containing not only subject key / value, relationship type and object key / value but also the data source URLs, date of annotation, certainty, and user name. Persistence is implemented using the JPOX JDO 2.0 compliant package [24]. For export purposes, the Jena semantic web framework library was employed. Graph structures are displayed using the JUNG library [25].

6.3.2 Requirements

The TUAM software was written in Java 1.5.0 and requires a PostgreSQL (>7.4.) [26] or similar transactional relational database backend. For database (and Ontobroker) connectivity, the internet transport is SSL secured, which requires generation of a certificate for each client user.

6.3.3 Installation and Configuration

The triple store has to be identified in a tuam.ini file in the user’s home directory. Ontobroker connection parameters are set interactively during runtime. For database schema generation, JPOX schema tool must be invoked using the supplied ant build script once by the database administrator.

6.4 Data Sources and Formats Used

Service annotations have to be done locally by the SRS site administrators. The annotation will be done with classes from the central service ontology in Ontobroker whose interface to TUAM is transparent for the user. SRS services should minimally contain:

- 1 Site namespace URL (1)
- 2 Service name (1-n)
- 3 Service URL (WSObject URL) (1)

Numbers in parentheses give the cardinality for one site. The XML representation of these data must ensure that the URLs are unique for one site and the combination site:service. Annotations have to be applied to the Service Name, which uniquely identifies one service offered by one side.

The knowledge model for SRS and the bioinformatics services that uses some OWL-S parts is described in chapter 4.

7 Lessons Learned

Throughout the technical discussions it appeared that technical and domain experts are not always using the same *language* for describing the same technological entities. Hence, several misunderstandings occurred which caused considerable delay before a consolidated design of a main building block could be reached. After having experienced this for some time, we came to the conclusion to use simple examples throughout the individual design process in order to achieve a great amount of clarity.

In projects like SIMDAT, which have a strong industrial focus, commercial project partners are naturally interested in placing their software products at central positions of the architecture. Consequently, they are not tolerating the uptake of products from competitors either. Unfortunately, from a system design point of view this constraint does not always lead

to the most optimal solution. For example, in the Pharma prototype the selection ontology tool-suite from the corresponding commercial technology champion immediately resulted in the restriction to a simpler knowledge model for the service annotation. On the other hand, those dependencies and requirements arising from industrial partner underline the business relevance of the developed solutions.

From the integration work between software from two commercial partners (SRS from Lion and E2E Security from NEC) we learned that it is very important to provide clean interfaces. Not only does it make the integration much easier, it also helps to keep components and IPRs disjoint.

8 Conclusion

This document describes the technical details of the first SIMDAT Pharma prototype. The prototype shows what has been done during the first 12 months of the project within the Pharma activity. The development of the prototype also served as a learning environment for all partners to get an overview of and a feel for all the different components involved. The prototype will be developed further throughout the project to reflect the current state of the Pharma activity.

Within the first 12 months of the project we achieved:

- federation of multiple SRS installations,
- basic semantic description of databases and analytical tools,
- service discovery based on semantic queries,
- integration of end-to-end security between a client (here: the federated portal) and multiple SRS servers.

Ideally, the Pharma prototype has integrated technology coming from all technological areas. However, previously existing functionality of the pre-existing Pharma hosting environment in the areas of distributed data access and Grid infrastructure made it possible to postpone the uptake of them to the next project phase. Consequently, the Pharma consortium was able to concentrate more on the collaboration with other areas like ontology and VO which promised to have more impact on the PM12 prototype. Contrarily, the work tasks of the second phase are defined in a way that they are relying on a strong technology exchange with areas like workflow and integrated Grid infrastructure and thus demonstrating a balanced integration of the horizontal SIMDAT layers.

9 References

- [1] Grosz, B. N., Horrocks, R. Volz, and S. Decker, Description Logic Programs: Combining Logic Programs with Description Logic, in Proc. of the 12th International World Wide Web Conference (WWW 2003), Budapest, HUNGARY, May 2003.
- [2] Qu, C. and F. Zimmermann, Application of Standard Semantic Web Services and Workflow Technologies in the SIMDAT Pharma Grid, Position paper presented on W3C Workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria, June, 2005.
- [3] Paolucci, M., T. Kawamura, T. R. Payne, and K. Sycara, Semantic Matching of Web Services Capabilities, in Proc. of International Semantic Web Conference (ISWC), Sardinia, Italy, June 2002.

-
- [4] Qu, C. Semantic Broker User Manual, http://bscw.scai.fraunhofer.de/bscw/bscw.cgi/d28001/SB_Manual.pdf
- [5] Sun Microsystems, Java API for XML-Based RPC (JAX-RPC), <http://java.sun.com/webservices/jaxrpc/index.jsp>
- [6] Sun Microsystems, Java Web Services Developer Pack (Java WSDP), <http://java.sun.com/webservices/jwsdp/index.jsp>
- [7] The Apache Software Foundation, WebServices – Axis, <http://ws.apache.org/axis>
- [8] Grid-Enabled Medical Simulation Services, GEMSS, IST Project, IST-2001-37153, <http://www.gemss.de>
- [9] PKIX Working Group, IETF. <http://www.ietf.org/html.charters/pkix-charter.html>.
- [10] W3C, SOAP, <http://www.w3.org/TR/soap>
- [11] OASIS, Web Services Security: SOAP Message Security 1.0. OASIS Standard 2004401, March 2004.
- [12] S. Anderson et. al. Web Services Trust Language (WS-Trust), Specification, February 2005. <ftp://www6.software.ibm.com/software/developer/library/ws-trust.pdf>
- [13] S. Anderson et. al. Web Services Secure Conversation (WS-SecureConversation), Specification, February 2005. <ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf>
- [14] ITU-T Recommendation X.509. Information Technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks. March 2000.
- [15] A. O. Freier, et. al. The SSL Protocol Version 3.0. IETF Internet Draft, November 1996.
- [16] UDDI v3.0, Oasis standard, February 2005, http://www.oasis-open.org/news/oasis_news_02_03_05.pdf
- [17] Zdobnov, E. M., Lopez, R., Apweiler R., Etzold, T., *Using the Molecular Biology Data*. Biotechnology, vol. 5b: Genomics and Bioinformatics, C.W. Sernsen (eds); pp. 281-300, Wiley-VCH (2001).
- [18] Etzold, T., Harris, H., Beulah, S., *SRS: An Integration Platform for Databanks and Analysis Tools in Bioinformatics*. Bioinformatics: Managing Scientific Data, ISBN: 1-55860-829-X, pp109-145.
- [19] Public SRS servers and databases. <http://www.lionbio.co.uk/publicsrs.html>.
- [20] Kumpf, K., TUAM: A new tool for universal annotation and mediation, Journal of Web Semantics, 2005 (submitted)
- [21] RDF Semantics <http://www.w3.org/TR/rdf-mt/>
- [22] Lee, R: Triple Store comparisons <http://simile.mit.edu/reports/stores/>, July 26, 2004
- [23] W3C Semantic Web <http://www.w3.org/2001/sw/>
- [24] JPOX: Java Persistent Objects JDO <http://www.jpox.org/>
- [25] JUNG Java Graph library <http://jung.sourceforge.net/>
- [26] PostgreSQL- the world's most advanced database <http://www.postgresql.org/>