

Project IST-FP6-026476 SEAMLESS
“Small Enterprises Accessing the Electronic Market of the Enlarged Europe by a Smart Service Infrastructure”
STREP – Information Society Technologies (IST)

Deliverable D3.4.2
eService Environment
Technical Specification

Workpackage WP3 – Technological Infrastructure
Task T3.4 – Service publication and execution

Abstract

The present document describes the work performed in the project task T3.4, which main aim is to test and check the DBE ExE functionalities.

This document describes the tests and their results, as well as the technologies tested as back up alternatives to the DBE solution.

This task specifies the SEAMLESS choices for publication and execution, as well as the software module interfaces that the SEAMLESS offers through the DBE technology.

Start date of project	Jan 1 st , 2006	Duration of project	30 months
Deliverable due date	Jul 21 st , 2007	Actual submission date	Jul 27 th , 2006
Dissemination level	PU	Revision status	Final
Responsible partner	ANTARA	Authors	T3.4 Participants

Table of contents

1	EXECUTIVE SUMMARY	4
2	INTRODUCTION	5
2.1	Purpose.....	5
2.2	Relationship	5
2.3	Document Structure	5
2.4	Audience and IPR considerations	6
3	STRATEGY FOR ADOPTING DBE RESULTS (FROM D01).....	7
4	REQUIREMENTS OF THE PUBLICATION ENVIRONMENT	8
4.1	Basic Principles.....	8
4.2	Technical Environment	9
4.3	Project Environment	10
5	MAIN BACKUP TECHNOLOGIES CONSIDERED	12
5.1	Web Services	12
5.2	The STIL project.....	14
5.3	ebXML ebMS / ebRR	15
6	TESTS OF DBE TECHNOLOGIES	17
6.1	Test generics	17
6.1.1	<i>Localization</i>	17
6.1.2	<i>Hardware / Operative Systems description</i>	17
6.1.3	<i>DBE ServEnt versions tested</i>	17
6.2	Test Cases	17
6.3	D01 pilots description	18
6.3.1	<i>Publish and execute Mediator SEAMLESS Services on DBE</i>	18
6.3.2	<i>Offering SRRN Services through DBE</i>	18
6.4	Pilot Test cases performed by the SEAMLESS team	19
6.4.1	<i>Basic test set</i>	19
6.4.2	<i>SEAMLESS-independent DBE ExE tests using BIAS™</i>	19
6.4.3	<i>SRRN through ServEnt</i>	20
6.4.4	<i>Publish and execute Mediator SEAMLESS Services on DBE</i>	20
6.5	Test results overview	21
6.5.1	<i>DBE Studio</i>	21
6.5.2	<i>DBE ServEnt</i>	21
6.6	Test conclusions.....	22
7	PUBLISHING SERVICES IN SEAMLESS	23
7.1	Reminder on the SRRN.....	23
7.2	Publishing a service on the SRRN	24
7.3	Additional features in respect to DBE.....	26
8	CONCLUSIONS	27
9	ANNEX: SEAMLESS MODULES EXPOSED VIA DBE.....	28
9.1	Semantic Translator	28
9.1.1	<i>storeExecutionData</i>	28
9.1.2	<i>beginSession</i>	29



9.1.3	<i>addSourceObject</i>	29
9.1.4	<i>translateObjects</i>	29
9.1.5	<i>isObjectTranslated</i>	29
9.1.6	<i>getTranslatedObject</i>	30
9.1.7	<i>closeSession</i>	30
9.1.8	<i>translate</i>	30
9.2	RRServices.....	31
9.2.1	<i>Lifecycle Interface</i>	31
9.2.2	<i>Query Interface</i>	34
9.3	Query Controller	40
9.3.1	<i>queryForMetadata</i>	40
9.3.2	<i>queryForMetadataAsync</i>	41
9.3.3	<i>getDetails</i>	42
9.3.4	<i>ping</i>	42
9.3.5	<i>getGLOBTranslationPath</i>	42



1 Executive Summary

Task T3.4 is a task that was not in the original SEAMLESS DoW (Description of Work), as it was included in the SEAMLESS project by the Contract Amendment. This Contract Amendment was generated after the conclusion of task T0.1, in order to reorganise the SEAMLESS project based on identified synergies with DE Cluster projects.

In T0.1, synergies between the SEAMLESS and the DBE project were identified, studied and some test cases were defined. These test cases were defined in order to maximize the benefits that SEAMLESS and DBE could offer each other.

The main issue at that stage was that the DBE project had not released useful results (in terms of software) yet. Therefore the task T3.4 was introduced in the DoW basically to follow-up the DBE progress, and to test and check the definitive versions of the DBE against the requirements that the SEAMLESS project demands to its service publication and execution environment. Apart from that, this task aimed to propose backup technologies in such a way (as introduced in D0.1) that the different SEAMLESS software components could offer a double interface (DBE Execution Environment plus other alternative technologies).

This deliverable introduces the experiences of the SEAMLESS development team testing the different versions of the DBE ServEnt technology, which is the base for the Prototype released under the SEAMLESS deliverable D3.4.1. The conclusions and description of these tests are also described in this document, as well as the interfaces of the software components under development by the SEAMLESS project using the DBE technology.

Some other technologies have also been tested in order to backup the DBE technology in case some problems could arise with it. The SEAMLESS project has decided since the very beginning to adopt DBE but always with a backup technology, in such a way that the software components have been developed basically under Web Services (with different implementation frameworks) and provided with a ServEnt Adapter so that they can be used from the DBE environments.



2 Introduction

2.1 Purpose

The Task T3.4 is aimed at facing and solving a critical problem of the SEAMLESS project: the choice of the publication and execution environment technology for its services. The SEAMLESS project is oriented to adopt the DBE project outcome but, at the same time, it must be ready to move to another environment if the DBE technology will not result reliable or efficient enough along the SEAMLESS project timetable.

For this reason, the task will keep testing the DBE technology while it is under development, and maintaining tight relations with other DE cluster projects. At the same time the team explores alternative solutions (e.g. Web services environments, STIL, ...) as possible backup technologies in case of unsolved DBE problems or SEAMLESS requirements being not completely covered by DBE.

This document introduces the experiences of the performed tests, as well as the description of the different software modules and interfaces that will be exposed through the DBE technology.

2.2 Relationship

Task T3.4 was added to the SEAMLESS project as one of the changes of the Contract Amendment resulting from Task T0.1. So this task will solve some issues enunciated at that task (the services publication and execution environment). It takes the inputs from Deliverables D0.1, where the issue were introduced as well as the test synergies with DBE ExE described there, in order to implement them as part of the test of the DBE technology.

The work of this task is also related to the development of the the entire infrastructure, semantic environment, and collaboration ,because the software modules to be tested under DBE technology have to be provided by these tasks, and the results of this task will influence the communication technologies between the modules developed there.

Finally, a special relationship between exists between Task T0.2 and Task T3.4. Task T3.4 performs the test implementations based on the information provided by Task T0.2 (Alignment on approaches and results) and provides this last one with the test results and software. By Task T0.2 the SEAMLESS project participated in the DBE final review meeting, with a demonstration of SEAMLESS components working under DBE. The software to perform this demonstration was developed and provided by Task T3.4.

2.3 Document Structure

This document is structured in the following way:

- First, an introduction to the collaboration between DBE and SEAMLESS project. A brief introduction to the DBE project, the synergies found and the strategies agreed for the implementation of these synergies.
- The SEAMLESS framework is composed of different software engines, with different requirements on the execution environment. In this second section, these modules and their requirements on the services publishing and execution environment are outlined.
- Apart from the DBE technologies, some other technologies have been studied and tested, so that they can act as backup of the DBE if necessary. They are described in the third section of this document.
- A section specifying how to publish a service in the SEAMLESS infrastructure, taking the SRRN as base technology.



- The fifth section of this document describes the different interfaces published by the SEAMLESS project by using the DBE ExE technology.
- The sixth section describes the tests performed by the T3.4 team on the DBE ExE technology.
- A section summarising the experience of the T3.4 team in the DBE adoption.

2.4 Audience and IPR considerations

This document is public.

It is released under the Creative Common license (Attribution-NonCommercial-ShareAlike), see <http://creativecommons.org/licenses/by-nc-sa/3.0/> for details.



The intended audience includes the following categories of possible readers:

- All the partners of the SEAMLESS consortium.
- All the projects of the DE cluster. The other projects of the DE cluster are expected to have much interest in knowing the intentions of the SEAMLESS project since these could have impact on the respective work plans. This document describes the SEAMLESS software components and interfaces that these projects could use through DBE.
- The DG INFOSO officers. The collaboration between projects acting in the same area is generally considered a significant added value of the research work.
- Technical staff of other projects, that can use the interfaces exposed in this document to build solutions on the DBE environment.



3 Strategy for adopting DBE results (from D01)

DBE is a 3 years-long 6FP Integrated Project, started on November 2003 and involving 20 organisations. The two main objectives of the DBE project are to provide Europe with a recognised advantage in innovative software application development by its small and medium-sized enterprises (software producer SMEs) and to achieve greater information and communication technology (ICT) adoption by SMEs in general.

The ultimate aim of the DBE is to create an ecosystem where applications within it behave like intelligent, interactive, and adaptive organisms, by organising the approach into the three layers:

- The business network, where buyers and suppliers establish business relationships.
- The services environment, where the companies exposes their services that can be executed and used by applications. It is necessary to split this layer into
 - Services Factory: Applications where the company's data and services are modeled and implemented.
 - Execution environment: Peer to Peer network (P2P) through which the companies (peers) expose the offered services, and executes demanded services.
- The evolutionary environment, where these company services are grouped into habitats, that interrelates and evolves.

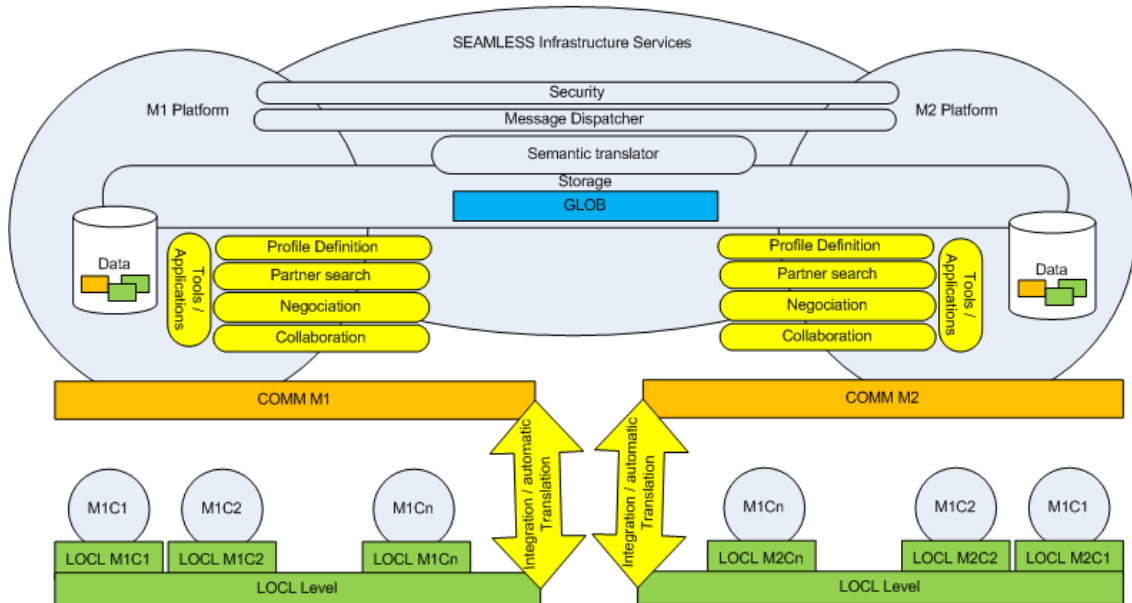
During the negotiation procedure the SEAMLESS project added a specific work package aimed at studying and aligning the possible synergies between SEAMLESS and the projects of the Digital Ecosystem Cluster (set of projects dealing with similar problems, addressing to a specific type of companies, and having in common a regional approach), putting a special attention in the DBE project (as the bigger project in the Digital Ecosystem Cluster).

Fruit of the collaboration between DBE and SEAMLESS, the SEAMLESS project concluded that there was a wide room for experimenting the adoption of the DBE infrastructure, and interesting opportunities to establish systematic exchanges of knowledge and results with other projects of the DE cluster. At that moment (June 2006) the main issue was the draft status of the DBE technology and some other open questions about the type of open source license under which the results were going to be distributed.

After a SWOT analysis (see deliverable D0.1), the SEAMLESS project decided to adopt and experiment with the DBE ExE (execution environment) technology, but under the following policy: The software modules being implemented and exposed through DBE have to be able of offering a double interface (access technology). In this way it is possible to have a backup execution environment if something goes wrong with DBE ExE. (See section 6.3.2 for more details).



4 Requirements of the publication environment



Picture 1: Architecture overview

4.1 Basic Principles

The SEAMLESS project needs to provide some primary technical features to support the business processes in the most effective way and to provide solutions for common problems faced in its domain. The following technical points are of key importance for the success of the whole project and therefore the architecture must fulfil these requirements. These primary features include:

- **Openness**
SEAMLESS should use existing developments and technology wherever possible. Thus developments should utilize open technologies and standards to emphasize the open structure of SEAMLESS. Furthermore, all technologies developed within the project should be based on existing, market recognized, or adopted standards in order to make them accessible as much as possible, independently from the technologies that the enterprises already use for their business.
- **Flexibility**
The impact of the architecture is not yet tested and therefore all possible applications of SEAMLESS cannot be implemented or even fully architected. Because of this, the system has to provide a flexible way to integrate new “modules” and to enhance the basic functionalities with more detailed and more complex features and use cases.
- **Integration**
A product can be good but fail in its acceptance by the market if it does not provide fair integration possibilities. Because of this, the SEAMLESS system should provide a simple interface for easily enabling access. Furthermore, a set of API functions should be provided to maximize the integration possibilities.
- **Security**
One of the most important issues in the project is security. Within this topic, SEAMLESS has to consider all relevant sub topics such as:

- Access control: the system has to make it as difficult as possible to manipulate and to read unauthorized data. The system has to make sure that it is almost impossible to access the system data with a foreign identity. Because critical company related information is being dealt with, the data has to be protected from being accessible by unauthorized agents. One possibility to protect this data is fully managed access control by the SEAMLESS infrastructure – i.e. the infrastructure should be able to provide only the data which is needed by the companies being involved in a process.
- Trust: it is most important to make sure that information, which is sent to a user, is received exactly in the same situation that it was offered. Furthermore, the user has to be sure that their data is kept private and so they can continue to trust the system. This can be endorsed by using open, or at least well documented, technologies and by providing a broad description of the system's architecture to its stakeholders. There should be a configurable confidence level between companies.
- **Performance**
The system will be used by companies with mission critical time-dependent applications, and performance is an important reason for using eBusiness solutions. One of the main features of SEAMLESS is the reduction of both time and work. Because of this, the system should provide a fast access to all necessary information in automated ways. It is desirable to provide the majority of information almost instantly.
- **Scalability**
Although SEAMLESS starts with a prototypes in textile and construction sectors, it has to be able to serve a much wider variety of users, also in terms of company size and requirements on volume. The system should therefore be scalable to support different usages. This also means that SEAMLESS should be able to deal with an increasing amount of users.
- **Reliability**
The user has to be sure that the system can be used whenever and wherever specific information is needed. Thus, since the architecture is used by companies in a global context, the SEAMLESS implementations in the market must be able to provide a 24*7 availability.

4.2 Technical Environment

The SEAMLESS environment is required to deliver services on a distributed geographical environment. On the technical side, the main requirement that a suitable environment should satisfy is the possibility of offering a distributed architectures.

This requirement has different implications both in terms of technologies for developing graphic user interfaces and in terms of business services. The scenario is conceived with the the idea that the companies should access the SEAMLESS applications by means of Web (HTML) interfaces and the business logic should be provided by the orchestration of Web Services. The selection of these technical choices is based on the following considerations:

One of the SEAMLESS objectives is to provide the companies with applications that should be perceived as simple means for performing difficult tasks. On this point of view, the choice of adopting a Web user interface represent a simple but powerful mechanism for interacting with the human user.

SEAMLESS should offer an open infrastructure for enabling company interactions. From this perspective, the decision of implementing the business logic by means of a Web Service architecture assures the possibility of reusing the same services in other contexts as well as the ease of expanding the SEAMLESS basic facilities.

These technologies represent the state-of-the-art of the distributed applications, thus guaranteeing the possibility of expanding the SEAMLESS platform in the next years.

From the point of view of the development environment, it should be open and independent from any commercial license. Furthermore, the possibility of working in a cross platform environment able to host both JAVA and Microsoft .Net technologies is considered absolutely the best choice.



4.3 Project Environment

On an overall point of view, the SEAMLESS environment can be represented as a collaborative framework where companies can access business services.

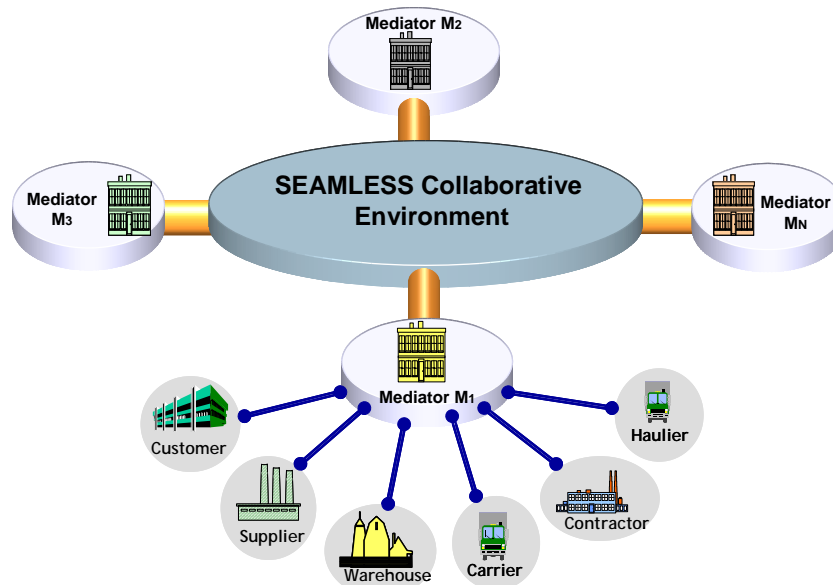


Figure 2: The SEAMLESS environment

The SEAMLESS framework (Figure 2) is formed by three main parts or levels, namely the collaborative environment itself, the mediators, and the SMEs associated to each mediator. Essentially, the SEAMLESS collaborative environment is the virtual space where SMEs will make e-business operations through the mediators. Each mediator acts as a kind of *facilitator* between its members and the collaborative environment.

Each interaction is considered a business operation that requires, indeed, a perfect understanding between the actors involved and which is supported by a specific information flow. Each SME has to be sure about the precise meaning of every single business interaction, not only at the operational level but specially at the semantic level.

On the technological viewpoint, the framework is realised by providing every mediator with an ICT infrastructure delivering the following functions:

- User Access Control. It is the function to register the mediator associated companies and to authorise their access to the mediator services. Trust and security are critical aspects since they impact on project results acceptance by the intended users.
- Mediator Basic Services. It is a suite of easy services the mediators makes available to its associated companies for enabling them to gain visibility in the electronic market, search for possible partners, negotiate and collaborate with them. These services are intended to play the role of basic functions embedded into the SEAMLESS infrastructure whose adoption by the target companies requires a minimum effort.

On the communication perspective, the environment should rely both on synchronous and asynchronous communication means. This requirement is based on the consideration that the infrastructure is supposed to support at the same time long-term executions with services characterised by immediate response times.

For instance, the possibility of dispatching and translating documents among companies is a typical example of long-term execution service since it is perfectly acceptable for companies that delivering might take a few minutes. Hence, for this kind of services the asynchronous approach should be adopted in order to avoid annoying waiting time for the final users. This also implies that a proper notification



mechanism should be conceived for informing the companies that the activated procedures have been terminated.

On the other hand, simplest processes like profile editing or partner search are expected to be executed in few seconds since companies might wish to iterate the action many times before finding a satisfactory result. In this context, a synchronous communications is necessary for presenting the results in a very quick way.



5 Main backup technologies considered

5.1 Web Services

The W3C defines a **Web service** as a software system designed to support interoperable machine to machine interaction over a network. Web services are frequently just Web APIs¹ that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services.

The W3C Web service definition encompasses many different systems, but in common usage the term refers to clients and servers that communicate using XML messages that follow the SOAP standard (an XML-based, extensible message envelope format, with "bindings" to underlying protocols).

The primary protocols are HTTP and HTTPS, although bindings for others, including SMTP and XMPP, have been written).

Common in both the field and the terminology is the assumption that there is also a machine readable description of the operations supported by the server, a description in the Web Services Description Language (WSDL). The latter is not a requirement of a SOAP *endpoint*, but it is a prerequisite for automated client-side code generation in the mainstream Java and .NET SOAP frameworks.



Basic functional scheme of a Web service

Some industry organizations mandate both SOAP and WSDL in their definition of a Web service.

For example, to improve interoperability of Web Services, the WS-I (Web Services Interoperability Organization) publishes profiles. A profile is a set of core specifications (SOAP, WSDL, ...) in a specific version (SOAP 1.1, UDDI 2, ...) with some additional requirements to restrict the use of the core specifications. The WS-I also publishes use cases and test tools to help deploying profile compliant Web Service.

Some other specifications have been developed or are currently being developed to extend Web Services capabilities. These specifications are generally referred to as WS-*. Here is a non exhaustive list of these WS-* specifications.

¹ An **application programming interface** (API) is a source code interface (defines the communication boundary between two entities, such as a piece of software) that a computer system or program library provides to support requests for services to be made of it by a computer program. An API differs from an application binary interface in that it is specified in terms of a programming language that can be compiled when an application is built, rather than an explicit low level description of how data is laid out in memory.

The software that provides the functionality described by an API is said to be an *implementation* of the API. The API itself is abstract, in that it specifies an interface and does not get involved with implementation details.

- **WS-Security**
Defines how to use XML Encryption and XML Signature in SOAP to secure message exchanges, as an alternative or extension to using HTTPS to secure the channel.
- **WS-Reliability**
An OASIS (Organization for the Advancement of Structured Information Standards) standard protocol for reliable messaging between two Web services.
- **WS-ReliableMessaging**
A protocol for reliable messaging between two Web services, issued by Microsoft, BEA and IBM it is currently being standardized by the OASIS organization.
- **WS-Addressing**
A way of describing the address of the recipient (and sender) of a message, inside the SOAP message itself.
- **WS-Transaction**
A way of handling transactions.

Apart from these specifications, an important point for considering the Web Services as a good technology for SEAMLESS is the fact of supporting both “synchronous” and “asynchronous” message exchange patterns.

Referring to the Web Services documentation at W3C: “[...] *In the context of Web services, the terms “synchronous” and “asynchronous” are used informally to describe certain message exchange patterns (MEPs)². In principle, MEPs may be arbitrarily complex, and may include various temporal relationships between messages. In practice, there is a small number of patterns for which the temporal relationships are well (if informally) understood. MEPs which describe temporally coupled or “lock-step” interactions are frequently referred to as “synchronous”. Examples include RPC-style request-response interactions and some kinds of transactional exchanges. Other MEPs allow messages to be sent without precise sequencing, and these are described as “asynchronous”. Examples include a flow of sensor event messages which need not be individually acknowledged, and an auction in which parties may submit bids at any time during the auction.*

The terms “synchronous” and “asynchronous” are descriptive, and do not correspond precisely to properties of MEPs. Occasionally the terms may be associated with particular message transport features, such as the re-use of a session. While specific implementations may support such notions, a dependency on such a feature would violate protocol independence, and therefore be problematic.

It is also worth noting that in some computing platforms or message transport systems the terms “synchronous” and “asynchronous” are perfectly well defined. For example many APIs include “asynchronous I/O” support, and certain message-oriented middleware systems offer synchronous and asynchronous notification and delivery modes. However, Web services are defined as platform- and transport- independent, and relying on implementation-specific terms is likely to result in confusion.

Many (most?) Web services do not use published MEP's, but instead rely on more or less informal patterns and techniques. In such cases, the terms “synchronous” and “asynchronous” are sometimes used to indicate the type of informal pattern being used. They may indicate whether or not coordination and synchronization techniques such as correlation data and particular transport bindings are to be used.

In view of the informal way that the terms are used, it is recommended that documents should not rely upon the use of “synchronous” or “asynchronous” in any normative specification. [...]

Some of the core SEAMLESS software modules, like the Semantic Translator (ST) service has been implemented using the Web services technology, so its technological context is the same as the previous W3C considerations. But, if we want to keep the use of the terms “synchronous” and “asynchronous”, despite discouraged, we can easily recognize what is, between them, the term that better fit this case. The

²In general the definition of a message exchange pattern:

- Is named by a URI.
- Describes the life cycle of a message exchange conforming to the pattern.
- Describes the temporal/causal relationships of multiple messages exchanged in conformance with the pattern.
- Describes the normal and abnormal termination of a message exchange conforming to the pattern.



Semantic Translator is “synchronous”. In fact its MEP describes interactions that are, all, regulated by strict temporal rules: first the client sends a request to the ST and then (and only then) the ST sends back the response to the client itself. The client **waits** until the ST has sent back the response prior to proceed in its execution.

The last sentence opens one more scenario for the use of the terms “synchronous” and “asynchronous”. In fact there could be a request-response interaction where the elaboration (needed by the service in order to provide the response to the client) lasts long, despite this is not the case of the ST. The client, in this case, should be kept alive and waiting until the elaboration has been terminated and the response has been sent back by the server. This imposes to waste some resources on the client side and to block the execution flux.

The second problem can be simply solved by uncoupling the execution of the main client process from the process that has to wait for the response coming from the server. Obviously, the waiting process needs resources (such as processor time, memory) to be kept alive, hence the second problem still exists. It can be faced at the interaction projecting-time or using (if possible) the technological support provided by the used message transport system implementation.

AXIS2, which has been used for publishing the first version of the Semantic Translator, provides support for both the described issues (resources wasting and execution blocking). Unfortunately none of them is optimal, since one does not completely solve the whole matter, while the other tends to invalidate the independence of the projecting phase from the implementation details and technological adopted solutions. DBE, as it was at the time of the tests, does not support such kind of solutions, since it provides just a RPC-like interaction pattern.

5.2 The STIL project

STIL (Telematic Tools for Enterprise Network Interoperability: Integrated Digital Logistics for Region Emilia-Romagna) is a project lasted 24 months (Feb 2005 – Jan 2007) partially funded by the Emilia-Romagna Region within the Initiative 1.1 of the Telematic Regional Plan.

STIL is finalised to the valorisation of current research activities in the regional area towards the realisation of a so-called Virtual Logistic Pole. By Virtual Logistic Pole (PLV) we mean a system of companies and networked logistic structures that use communication channels to coordinate logistics activities.

STIL grounded the birth of the PLV by realizing an ICT infrastructure supporting communication and collaboration between its members. In order to achieve this primary objective the project:

- found the basic technological layer for the creation of the PLV;
- tested the applicability of the concepts through the realisation of application prototypes;
- validated models and technologies adopted.

The applications developed within the project offers a first set of services to demonstrate the effectiveness of the PLV, in order for logistic or manufacturing enterprises to join the PLV or for ICT enterprises to develop other services on top of the infrastructure.

In conclusion the STIL objective is, first of all, to promote the realization of the so called “just in time communication” within the integrated logistics world, that is the exchange of information in a well-timed and safe manner.

In order to support the communication between the STIL services, a tailored layer was developed which is based on an agent-oriented (message) approach. It relies on two components respectively named SIMPA and SIMPA-WS.

SIMPA is an agent-oriented extension of the Java language, providing high-level abstractions for designing and developing complex and, in particular, concurrent applications for mainstream software engineering.

To tackle the complexity of modern and future software systems, SIMPA introduces high-level metaphors taken from human society and theories, namely agents analogous to humans, as executors of activities and



activities - and artefacts analogous of objects, resources, tools that are dynamically constructed, used, manipulated by humans to support / realise their individual and social activities.

This is called A&A (Agents and Artefacts) meta / conceptual model, and is based on interdisciplinary studies involving Activity Theory and Distributed Cognition as main conceptual background frameworks.

SIMPA-WS is a library for the SIMPA agent-oriented framework that makes it possible to easily develop applications that use Web Services or provide Web Services in a service-oriented architecture (SOA) fashion, but adopting agents and artefacts, that is, the basic abstractions provided by SIMPA, as basic high level building blocks.

Using SIMPA-WS both WS users and providers are modelled as SIMPA agents, as pro-active entities that, in the former case, need to access and use services in their working activities, encapsulating the user business logic of user applications, and in the latter case, process the requests for services, encapsulating the service business logic.

In both cases, artefacts are used as high-level mediating entities functioning as interfaces, encapsulating the technology needed to enable the interaction using WS-* standards. In particular: an artefact of type WSArtefact is provided on the user side to make it possible for SIMPA agents to access and use what ever existing Web Service, by simply creating and using instances of such an artefact.

Similarly, an artefact of type ServiceArtifact is provided on the service side to make it possible for SIMPA agents to process the messages related to a specific Web Service, again by simply creating and using instances of such an artefact.

The flexibility assured by the STIL project communication technology is maximum. In fact synchronous and asynchronous communications can be easily realised, together with the possibility to define special execution precedence policies for web services.

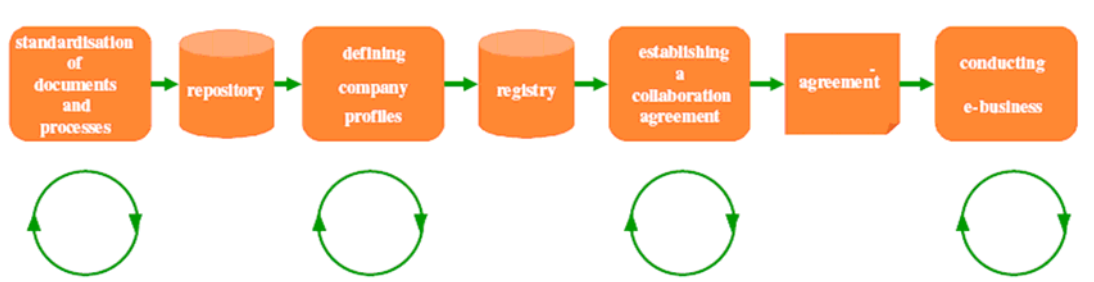
The only limit of this technology, that makes its use in SEAMLESS potentially risky, is that it is in an alpha development stage, and since the STIL project is concluded there is no certainty about that it will be developed and maintained in the future.

5.3 ebXML ebMS / ebRR

ebXML is a complete framework for B2B that will offer a complete set of standards to be used by businesses, organisations and authorities for their mutual electronic communication. ebXML comprises both technical standards for registries, repositories, protocols, profiles, agreements, etc. and standards for business modelling, information components and entities, registration procedures, etc. It is suggested to enable a global electronic marketplace where enterprises of any size and in any geographical location can meet and conduct business with each other through the exchange of XML-based messages.

It is jointly sponsored and coordinated by the United Nations (UN/CEFACT) and the Organisation for Structured Information Standards (OASIS), and is being developed through the participation of a large group of people, organisations, companies, and consortia from around the world.

ebXML is composed of four infrastructure components and several other efforts such as ones focused on document creation, business process definition, etc. The infrastructure components are orthogonal in design. They may be used together or separately in implementing an infrastructure.



The technical infrastructure is composed of the following major elements:

- A Messaging Service (ebMS)
- A Registry and Repository (ebXMLRR, compatible with SRRN).
- Trading Partner Information (ebXML CPP).
- Business Process Specification Schema (BPSS)

In this case, as the service publishing environment of ebXML (ebXMLRR) is compatible with SRRN, the component of main interest is the Messaging Service for the transmission of XML document in an asynchronous during the Negotiation and Collaboration stages.

The ebMS provides a standard way to exchange business messages between organisations. It provides means to exchange a payload (which may or may not be an XML business document) reliably and securely. It also provides means to route a payload to the appropriate internal application once an organisation has received it. The messaging service specification does not dictate any particular file transport mechanism (such as SMTP, HTTP, or FTP) or network for actually exchanging the data, but is instead protocol neutral.

For ebMS tests, HERMES has been used. Hermes is a Message Service Handler (MSH) or message gateway (Open Source, GNU General Public License Version 2) that provides a standardized, reliable, and secure infrastructure for enterprises to exchange business documents. It is in compliance with the OASIS ebXML Message Service (ebMS V2) standard. The latest version, Hermes 2 also supports Applicability Specification 2 (AS2), a common message transfer protocol used by retailers and manufacturers in supply chains. Hermes supports secure messaging functions through widely-adopted Internet security technologies, such as XML Signature, SSL (Secure Socket Layer) and S/MIME (Secure Multipurpose Internet Mail Extensions). It has also implemented reliable delivery features defined in ebMS and AS2 standards to ensure the exchanged message is received and intact. The feature list of Hermes includes message packaging, reliable messaging, message ordering, error handling, security, synchronous reply, message status service, and RDBMS persistent storage. Hermes also supports transport protocols, such as HTTP and SMTP, to suit different needs of large and small enterprises, and different business requirements.



6 Tests of DBE technologies

6.1 Test generics

6.1.1 Localization

The different tests have been performed by the technical staff of ANTARA, KELYAN and U MODENA, in their locations at Valencia (Spain) and Modena (Italy).

6.1.2 Hardware / Operative Systems description

Different configurations have been chosen for the test. The computers set that have participated in the experience are described in the following table. Although the main interest of the tests was the ServEnt component of DBE, some tests with the DBE Studio have also been realised by the team.

CPU	RAM	Operative System	Eclipse version
AMD Athlon 64	1 GB	Linux Ubuntu 6.06 LTS	Eclipse 3.2.2
PIV3000	512Mb	Windows XP	Eclipse 3.2.2
PIV1800	1GB	Windows 2003 Server	Eclipse 3.2.2
PIV1600	1GB	Linux Suse 9	Eclipse 3.2
Centrino 1.73	1GB	Windows XP	

Setup and basic tests were also performed under Windows Vista.

6.1.3 DBE ServEnt versions tested

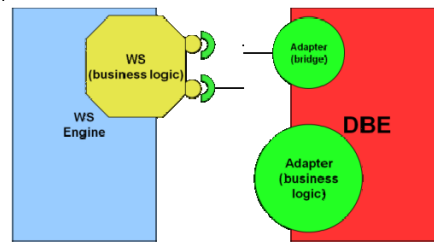
The versions tested by SEAMLESS teams are the following ones:

Version	Date released
0.3.8	2006-11-27
0.3.10	2007-01-10
0.3.12	2007-02-09
0.3.13	2007-03-12

6.2 Test Cases

As described in D0.1, the pilot cases implemented enable the SRRN and Mediator Services working with the DBE execution environment (ServEnt). In D0.1 the strategy of SEAMLESS for the adoption of the DBE execution environment was enunciated. It consists in being able of offer a double interface for these services being implemented under DBE. In this way it will be possible to have a backup execution environment. As this approach is very easy to implement because offering already implemented java services through the ExE is a matter of minutes (once the DBE technology is perfectly known, and the environment is perfectly set up), it don't introduces big deviations in the implementation of the services themselves, and allow the development team to make the implementations in a non dependent way of the execution environment to be used (this is true at least for the 90% of the services tested, see Conclusions).

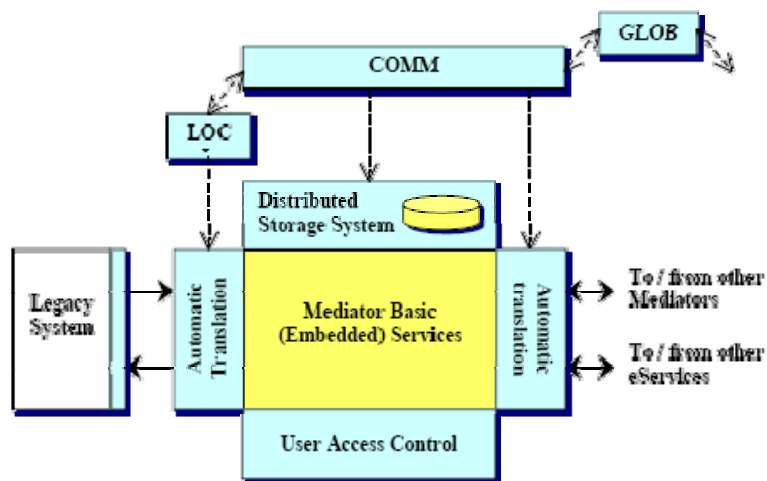




Picture 2: Diagram illustrating a WS interface being called from a DBE adapter

6.3 D01 pilots description

6.3.1 Publish and execute Mediator SEAMLESS Services on DBE



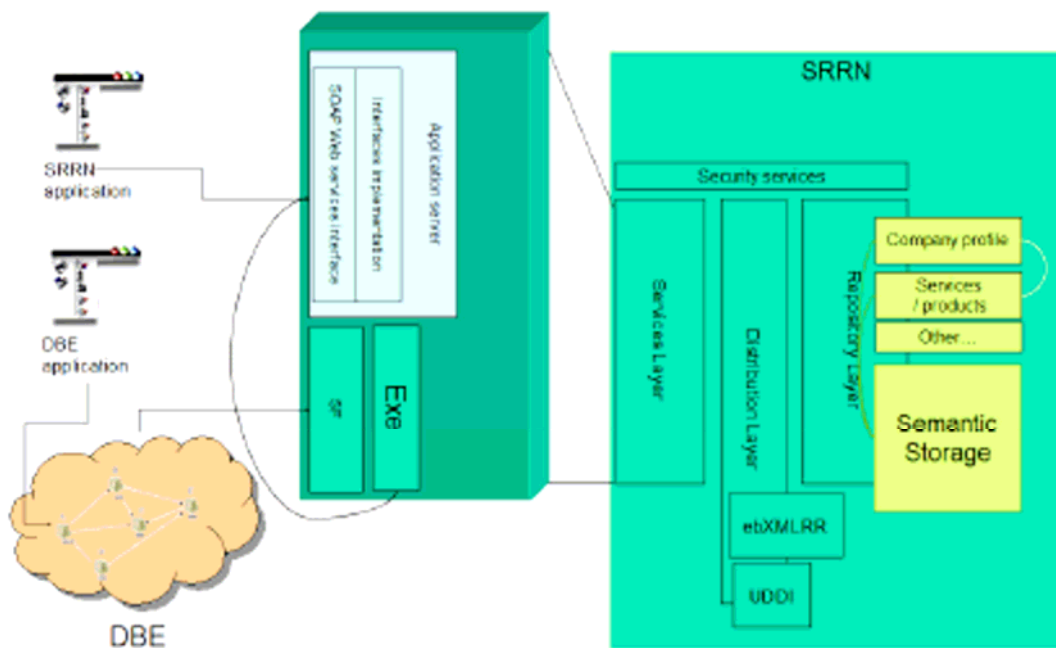
The main advantages of that approach is that it offers to the DBE infrastructure the advanced SEAMLESS semantic environment as service, while the SEAMLESS project, gets profit on the P2P network and publication environment of services of the DBE.

Once the complete SEAMLESS infrastructure will be implemented, this will allow SEAMLESS to offer through the DBE the following services:

- Configuration service (to select the specific product or service of interest).
- Demand modeling service (to represent the products and services to be bought).
- Demand configuration service (to select the specific product or service of interest).
- Quotation management service (to generate or answer to requests for quotation).
- Etc...

6.3.2 Offering SRRN Services through DBE.

The integrate the SRRN into the DBE offers different advantages both to SEAMLESS and DBE.



These mutual benefits can be enumerated in the following way:

- The SRRN could offer generic storage services to DBE (it can be a way to solve the Bootstrap problem that the DBE infrastructure don't offer) (See Conclusions chapter).
- The SRRN has less infrastructure constraints, because the services nodes will not need static IPs to be hosted, and will allow RRServices nodes to be localized in a dynamic way (with the constraints that this bootstrap issue introduces).
- It widens the scope of usage of the SRRN because in this case, each service Layer is both a node of SRRN and a node of DBE.
- Another (more) degree of freedom is introduced, because SEAMLESS Mediators can find each other information by joining /disconnecting to the global DBE group (once the bootstrap problem is solved).

6.4 Pilot Test cases performed by the SEAMLESS team

6.4.1 Basic test set

The basic test set (performed with every tested ServEnt version consists in:

- Setup and tests with the default configuration of the ServEnt.
 - HelloWorld test
- Configuration for localising fada nodes in the LAN.
 - Neighbour localisation tests
 - HelloWorld with the service running on different ServEnts (3 different nodes).
- Configuration for localising the installed Fada nodes from the internet.
 - Firewall and router configuration
 - Neighbour localisation tests
 - Testing the HelloWorld with the client pointing to the public IP of the ServEnt nodes.
 - Test of self-localisation to execute the HelloWorld.

6.4.2 SEAMLESS-independent DBE ExE tests using BIAS™

With the aim of testing the DBE ExE independently from the SEAMLESS environment, the team used BIAS™, which is a semantic based internet search engine developed by ANTARA. It offers different query and results retrieval through Web Services. The crawler engine is distributed using JXTA



technology. A proxy for the web services interface was implemented and tested with the different versions of the DBE ExE. A test implementation of the distributed crawler using DBE ExE was also done.

6.4.3 SRRN through ServEnt

For this test, the basic functionalities of the SRRN RRServices layer have been exposed through the DBE execution environment. These set of functionalities represents the different cases that can be found in the complete set of functionalities that the SRRN exposes to the applications, so it is considered that if the tests with this set of functionalities are successful, then the complete set of functionalities of the SRRN RRServices layer can be exposed through the DBE execution environment.

At the moment of the tests, the different RRServices layer functionalities were being exposed through Web Services, under an AXIS1 environment.

The interfaces implemented for testing purposes have been:

- RRServices objects lifecycle interface:
 - Store: functionality that allows the storage of metadata and attachments, well as defining the access rights for the stored information in one step. SOAP with attachments (MIME / DIME) is used by this functionality to transfer the file.
 - StoreMetadata: Functionality that allows to store an RDF description of one object (e.g. a company).
 - StoreRawDocument: Functionality that allows to store a document, while a default RDF is created and associated to the stored document. SOAP with attachments (MIME / DIME) is used by this functionality.
- RRServices objects query interface:
 - MegaPing: The megaping functionality consists of a recursive megaping that is spread through the SRRN infrastructure and allows to obtain information about the different nodes connected in the different SRRN layers.
 - GetLogsRecursive: The getLogsRecursive is a functionality that allows to retrieve the logs generated on the different nodes on the SRRN infrastructure having a transaction identifier. It contains different parameters that allow to configure the level of logs to be obtained (ERROR, DEBUB, WARNING, etc...) as well as the depth in layers to be queried (RRService, RRDistibution, RRRepository).
 - QueryMetadata: The queryMetadata functionality allows to retrieve the results of a RDQL , XML based query, or SparQL query on the metadata stored at the SRRN. Usually it returns a list of Identifiers that are used by the getMetadata to retrieve the RDF metadata itself.
 - getMetadata: Based on a metadata identifier, it returns the RDF file containing the description of the an object or document stored.
 - GetDocument: The getDocument functionality allows to retrieve a file (text or binary) stored at the SRRN. SOAP with attachments (MIME / DIME) is used by this functionality to transfer the file.

6.4.4 Publish and execute Mediator SEAMLESS Services on DBE

All the functionalities of the Semantic Translator layer have been exposed through the DBE execution environment for test purposes.

The exposure of those functionalities has been realized through the implementation of an additional layer placed on the top of the Semantic Translator original service layer interface. In “DBE words” that additional layer is called “adapter”.

Hence the adapter functions as a bridge between the clients and the Semantic Translator. It receives as input the data from the clients, elaborates (in order to “pack” it in the Semantic Translator interface recognizable form) and routes it to the Semantic Translator.



The exposure of the Semantic Translator succeeded, but a (obvious) time-loss (about the 20%) has been detected. In fact, it is just due to the transit of data through the adapter and the concerning elaboration, despite minimal.

At the moment of the tests, the Semantic Translator functionalities were being exposed through Web Services, under an AXIS2 environment.

6.5 Test results overview

6.5.1 DBE Studio

The DBE Studio is a very rich tool, plenty of features for *designing*, developing and deploying services. Unfortunately, during the tests the available documentation about the DBE Studio was revealed quite poor. The team had access to a short tutorial which explains how to get started, but it was clearly obsolete with respect to the version of the software being tested.

The DBE does manage to connect to the local ServEnt (as well as remote ones) but the interaction seems to be slow. The performed tests connecting to the DBE tests instances exposed by the DBE team were not always successful.

The user interface does not work properly in every case. It sometimes requires to be closed and reopened in order to start working again.

Additionally, some problems were found when deploying big .dar files through the DBE Studio.

6.5.2 DBE ServEnt

The core components of the ServEnt are perfectly installed by the setup application, some minor adjustments in the configuration files have to be done to have it working.

The FADA (p2p) node works at the first start. In ServEnt versions previous to 0.3.12, if the already cooked services (i.e. PortalService, SRservice) are installed, then a lot of exceptions (concerning the deserialization of some objects) are thrown in linux. Windows version does that well. It is very important to make the installation by using the required version of JDK (with previous (1.4.2) and further versions (1.6) it gives errors starting the services).

The DBE Portal was not complete even in the last versions under test. Some menu items point to fake “*lorem ipsum*” pages. Some others bring to *real* pages, like “Search & browse”, but the searching page does not find anything.

Again there is a very diffuse documentation, there are a lot of resources and not a clear place where find the solutions,

- Blogs
- Subprojects mailing lists
- Drivers sites,

Apart from that, once the correct versions of the JDK, etc... is installed, publishing a DBE service is quite easy:

- Define the **service interface** (that is, methods that are exposed to the *external world*);
- Realise the **adapter**, which is the implementation of that interface;
- the adapter could contain the service **business logic itself** or could be a **bridge** to an already existing service (exposed e.g. through SOAP Web Services),

but setting up a P2P network with neighbors localization is not well documented and the team have not succeeded in linking islands of neighbors. The samples given on this configuration simply did not work. Some minor performance problems were detected estimated in 20% of more time consuming than a SOAP Web Services interface.



Some problems regarding the implementation of the services have arisen between the different versions released (a DBE service developed under version 0.3.10 cannot be ran under 0.3.13).

A big problem arises from the fact that the P2P bootstrap mechanism is not working in the tested implementations of the ServEnt. It is working perfectly when talking about Local Area Networks (LAN), but services exposed trough DBE out of a LAN cannot be localized as neighbors, so in order to find them is necessary to have a register somewhere so that these entry points could be localized and asked for services by a client.

6.6 Test conclusions

The DBE approach is very innovative and they have developed interesting tools, although significant effort should be invested in enhancing reliability and quality of documentation. The SEAMLESS project identified in the very beginning the DBE developed technologies as with a good added value to the SEMALESS project , especially in the fields of execution environment and publication of services.

The ServEnt nodes bootstrap was one of the most expected results of the DBE from the seamless (just take a look to the section [6.3] D01 pilots description). This will allow dynamic nodes and services localization (without the help of a register), so allowing companies and mediator to dynamically enter the network and proceed to work. Unfortunately, the DBE project ServEnt final release have not implemented this feature in Wide Area Networks, which is the main environment in which the SEAMLESS project is working.

Nevertheless, considering that the DBE infrastructure will evolve and will be enabled with new features in the future, the SEAMLESS project decided to use and Expose some of the main intenal software components through the DBE, adding the Register approach for DBE sevice localization (through the SRRN). Mantaining thus a double interface implementation in these components (DBE + Web Services). Having the experience acquired by the technical team in this tasks, this will not imply a hard work, and this approach allow the team to mantain the possiblity of migrating entirely the infrastructure to DBE when the referred issues will be solved.

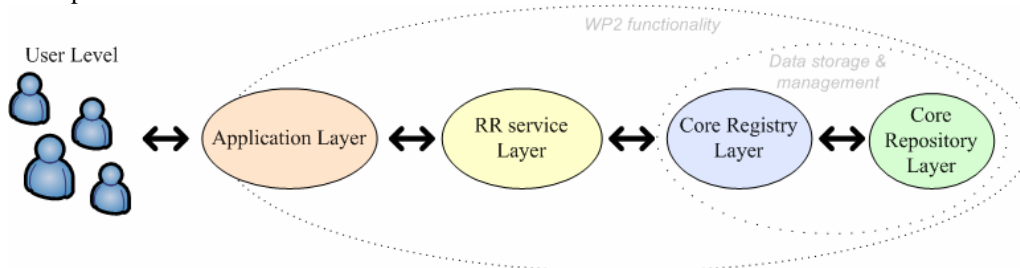


7 Publishing services in SEAMLESS

7.1 Reminder on the SRRN

The SRRN can be considered logically as data storage with access points to retrieve or store the data. These access points' nodes are characterised by the RRServices Layer, which offers to the applications different interfaces to perform operations on the SRRN.

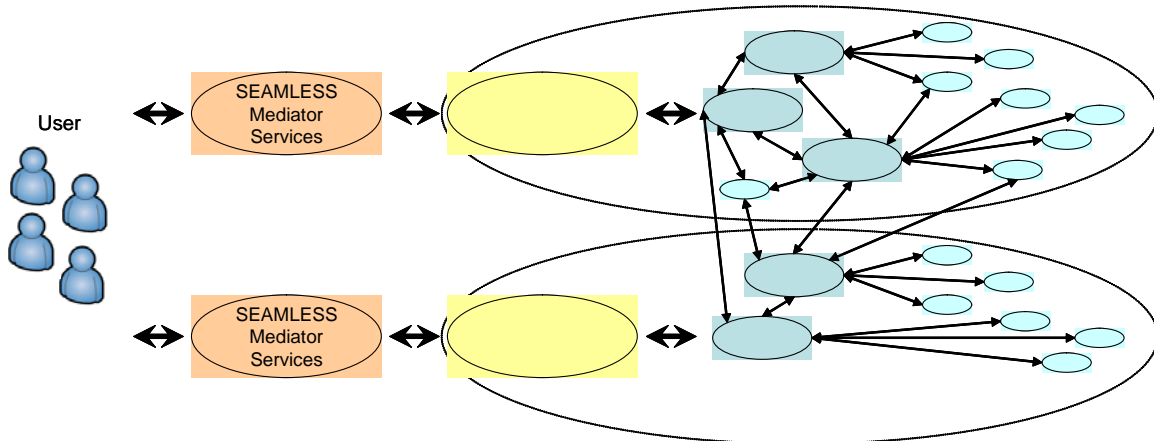
The SRRN Storage Service is a multilayered infrastructure, allowing sharing information regardless of where it is published.



- **Application Layer:** This layer represents all the different applications, that, in the SEEM environment, make usage of the storage services of the SRRN. Usually, this layer contains user oriented applications, or services offered to other applications (like in SEAMLESS).
- **RRServices Layer:** The Services Layer is in charge of managing the logic part of the Registry and Repository, to offer the services to the applications, but hiding the complexity of the layers below. It offers basic services of query and storage, as well as subscription services, version management, security and access control interfaces, traceability and log of transactions.
- **Distribution Layer;** It allows the transparent data distribution and retrieval from localizations that can be unknown to the applications. The distribution Layer is composed of a Peer to Peer network that allows the dynamic creation and organization of the infrastructure as well as its geographic distribution, but always given the sensation of a unique semantic database.
- **Repository Layer:** The Repository Layer allows the storage/ retrieval of the metadata and documents. It manages at a low level the information access rights.

The basic functionality principle of the SRRN relies on:

- SRRN implements a P2P philosophy, with federated repositories, and a multilayered structure.
- Gives support and uses standards and open specifications.
- Makes usage of security at the user level and nodes level, including usage of digital certificates, digital signature and access control.
- There are no constraints referred to the format and content of the information to be stored in it (neutral).



The main uses of the SRRN in the SEAMLESS project are:

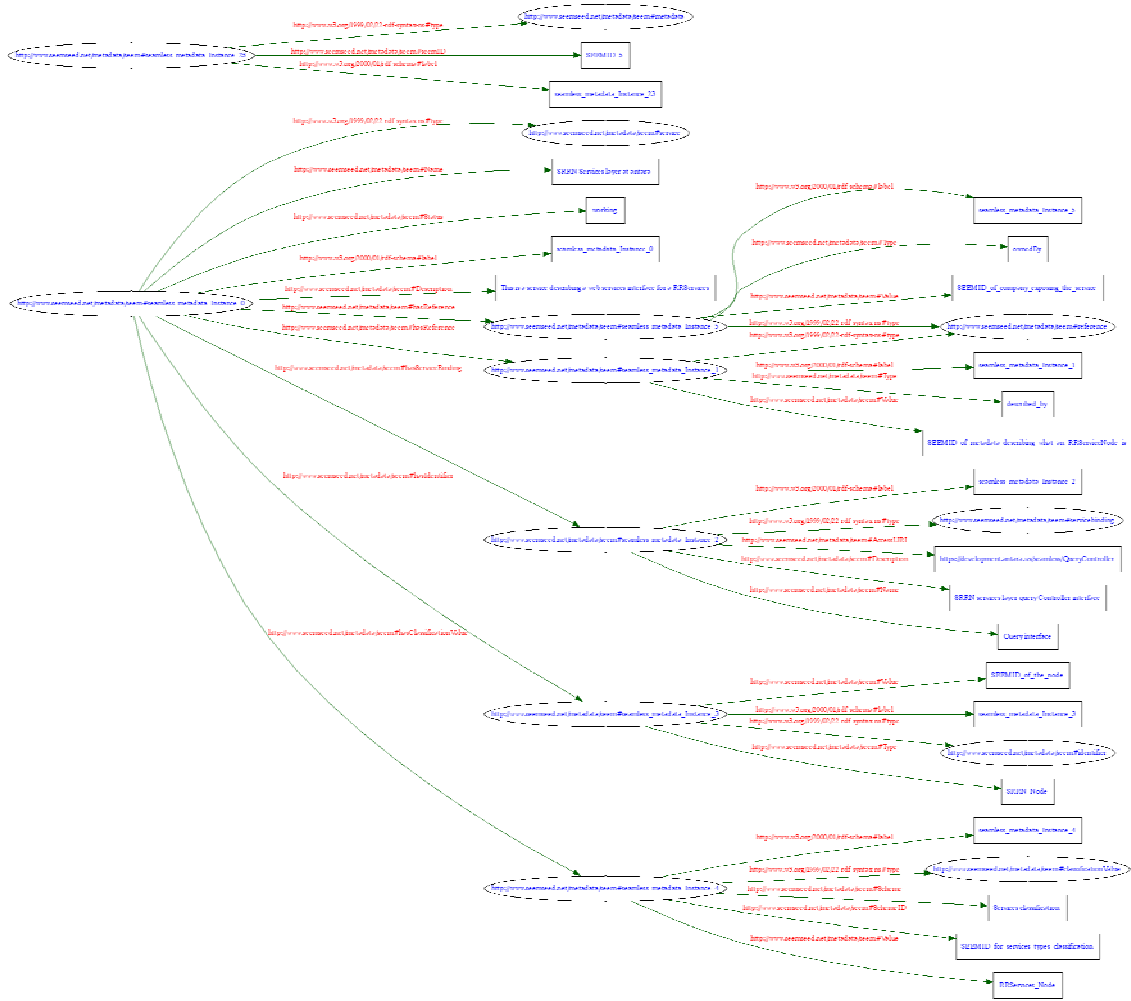
- Give support to the Semantic Services
 - Store and share the ontologies.
 - Store and share the mapping files between ontologies.
 - Subscription / notification on changes of these file types.
- Secure access to data
 - Provide access rights to the information through security framework
- Give support to the collaboration and negotiation processes
 - Store and share companies description
 - Store and share products/services demand / offer
 - etc...
- Give support to the query controller
 - Broadcast queries
 - Retrieve the ontologies mesh

And also storing and managing the services publishing procedures, following a strategy similar to “classic” services registries as they are ebXMLRR and UDDI, but with the advantage of storing the information using a semantic based approach (RDF).

For more detailed information on the SRRN, please refer to deliverable D3.3

7.2 Publishing a service on the SRRN

The SRRN contains metadata types so that the basic information types could be characterised. Inheriting the ebXMLRR and UDDI registry technologies, the SRRN defines some structures that can be used for storing and describing services and endpoints.



Picture 3: Graph describing a service with 3 endpoints at SRRN.

```

1: <?xml version='1.0' encoding='UTF-8'?>
2: <!DOCTYPE rdf:RDF [
3:   <!ENTITY rdf 'http://www.w3.org/1999/02/22-rdf-syntax-ns#'>
4:   <!ENTITY seem 'http://www.SEAMLESS.net/metadata/seem#'>
5:   <!ENTITY rdfs 'http://www.w3.org/2000/01/rdf-schema#'>
6: ]>
7: <rdf:RDF xmlns:rdf="&rdf;"
8:   xmlns:seem="&seem;"
9:   xmlns:rdfs="&rdfs;">
10: <seem:service rdf:about="&seem;seamless_metadata_Instance_0"
11:   seem:Name="SRRN Services layer at antara "
12:   seem:Status="working"
13:   rdfs:label="seamless_metadata_Instance_0">
14:   <seem:Description>This is a service describing a web services interface for a
RRServices</seem:Description>
15:   <seem:hasReference rdf:resource="&seem;seamless_metadata_Instance_1"/>
16:   <seem:hasServiceBinding rdf:resource="&seem;seamless_metadata_Instance_2"/>
17:   <seem:hasIdentifier rdf:resource="&seem;seamless_metadata_Instance_3"/>
18:   <seem:hasClassificationValue
rdf:resource="&seem;seamless_metadata_Instance_4"/>
19:   <seem:hasReference rdf:resource="&seem;seamless_metadata_Instance_5"/>
20: </seem:service>
21: <seem:reference rdf:about="&seem;seamless_metadata_Instance_1"
22:   seem:Type="described_by"
23:   seem:Value="SEEMID_of_metadata_describing_what_an_RRServiceNode_is"
24:   rdfs:label="seamless_metadata_Instance_1"/>
25: <seem:servicebinding rdf:about="&seem;seamless_metadata_Instance_2"
26:   seem:AccessURI="https://development.antara.ws/seamless/QueryController"
27:   seem:Description="SRRN services layer query Controller interface"

```

```

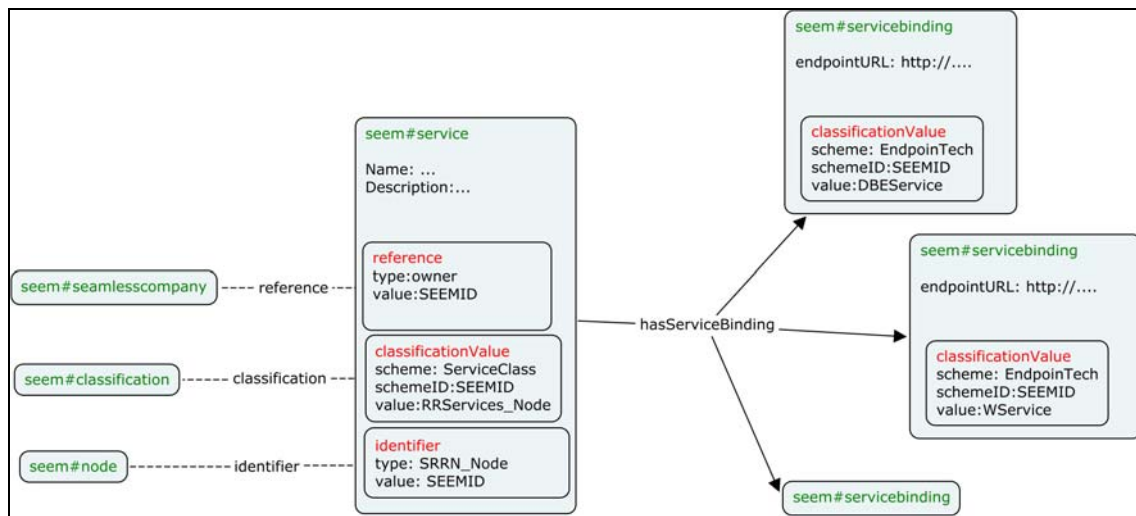
28:     seem:Name="Query interface"
29:     rdfs:label="seamless_metadata_Instance_2"/>
30: <seem:metadata rdf:about="&seem;seamless_metadata_Instance_23"
31:     seem:seemID="SEEMID_5"
32:     rdfs:label="seamless_metadata_Instance_23"/>
33: <seem:identifier rdf:about="&seem;seamless_metadata_Instance_3"
34:     seem:Type="SRRN_Node"
35:     seem:Value="SEEMID_of_the_node"
36:     rdfs:label="seamless_metadata_Instance_3"/>
37: <seem:classificationValue rdf:about="&seem;seamless_metadata_Instance_4"
38:     seem:Scheme="Services_classification "
39:     seem:SchemeID="SEEMID_for_services_types_classification"
40:     seem:Value="RRServices_Node"
41:     rdfs:label="seamless_metadata_Instance_4"/>
42: <seem:reference rdf:about="&seem;seamless_metadata_Instance_5"
43:     seem:Type="ownedBy"
44:     seem:Value="SEEMID_of_company_exposing_the_service"
45:     rdfs:label="seamless_metadata_Instance_5"/>
46: </rdf:RDF>
47:

```

4: RDF corresponding to the graph above.

The entities to be used in SRRN to define a service and its accesspoints are seem#service and seem#access point.

These entities contains information inherited from the parent data types, including the metadata mechanisms for logical classification, identification and linking information (classificationValue, identifier and reference) defined at the SRRN basic rdfs.



Picture 5: simplified representation of the definition of service and servicebindings

7.3 Additional features in respect to DBE

As it is depicted in Picture 5, it is very easy to characterise, store, and afterwards search for specific DBE interfaces. This mechanism will help to solve the limitations of the bootstrap functionality at the ServEnt.

To do that, a specific entry will be added to the classifications of types of servicebindings, so that these specific DBE interfaces could be localized and used by nodes that are not connected in the same LAN.



8 Conclusions

The SEAMLESS project, after testing results from DBE and other projects, has decided to follow the SOAP Web Services approach, exposing some of the core software component through the DBE. In this respect, DBE is not yet a mature technology (see conclusions on DBE technology), but due to the support that it will be given by their main developers, the very innovative approach supported, the easiness of transforming java based WS into DBE, and other positive reasons, the SEAMLESS project has decided to keep in the track of its adoption, once the issues that will give the SEAMLESS project real added value will be solved (e.g. the bootstrap problem).

Meanwhile, as already described, this core components (that have a pure synchronous behavior) will be exposed through a double interface, being available for the DBE community, and serving as tests of the DBE ExE technology for the SEAMLESS partners.

The eServices publication and localization environment, for both the SEAMLESS WS and DBE nodes, will be based on the SRRN, where the different services could be described, localized and executed afterwards.

Regarding the execution environment and the asynchronous communication pattern, in the SEAMLESS project the asynchronous case occurs in the negotiation and collaboration phases when communication between partner companies is basically constituted by exchanging business documents. The reply to a document (e.g. request for quotation or order) is yet another document (e.g. quotation, order confirmation) that can come much later. As written above (see WS section), it is possible to extend the client-server system in order to make it capable to support long-time interactions, using a request-ack pattern. This way to face the problem is obviously not priceless, since it makes the architecture and the interactions themselves more complex.

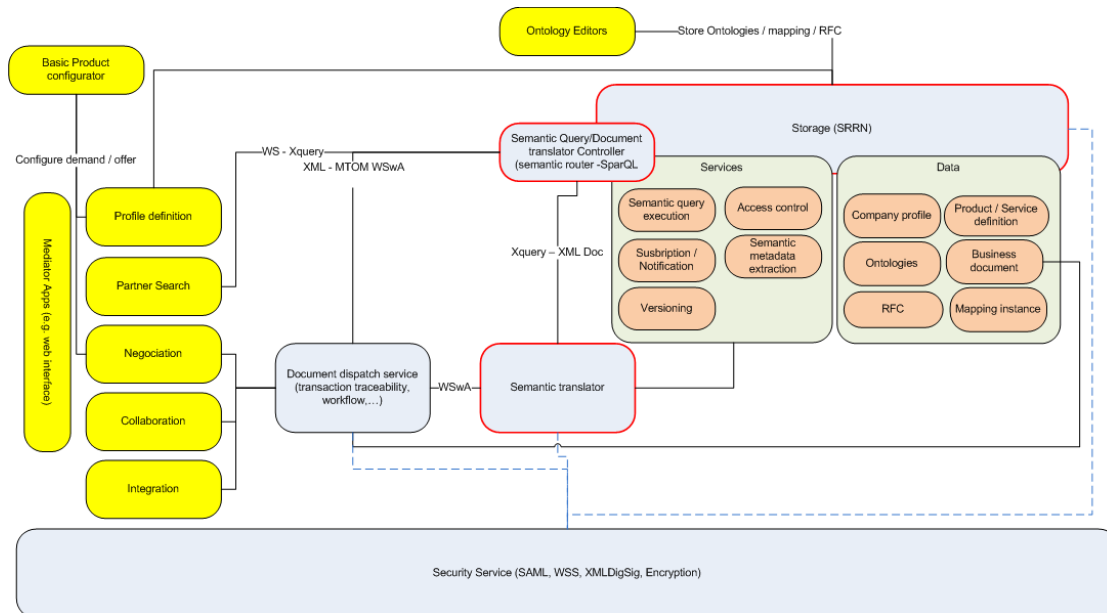
Suppose that the node A has some information that has to be computed in order to be used. Node A is not able to compute that information, while node B is. The elaboration is very complex and involves a human intervention, so the time needed by node B to perform it is long and not predictable *a priori*. The interaction could be decomposed into two request-response sub-interactions (the typical HTTP – which is the protocol used by the Web services – interaction). During the first one node A sends to node B the information needed by the latter to perform the elaboration. As soon as node B receives that information, it sends back to node A the acknowledgment (it functions as something like “ok I have received the data”). Then node A can continue its activity without dedicate a “special” process to wait for the response. After the elaboration has been completed (hours, days or months after), node B sends to node A the message which contains the elaborated data and node A responds back with another acknowledgment ending the compound transaction.

This is basically the policy adopted when implementing the SEAMLESS services and applications. Although presenting some drawbacks and making interactions more complex, it is quite homogeneous and simple to understand and implement. Furthermore, it is the most suited approach to move easily from the native AXIS2 environment to the DBE environment through the implementation of proper adapters. In fact it is perfectly matching the request/response protocol adopted by DBE.



9 Annex: SEAMLESS modules exposed via DBE

The picture below represents the SEAMLESS infrastructure, and the DBE artifacts are painted with a red line. As it can be observed, the services published at the DBE core services of the SEAMLESS infrastructure, and support all the semantic query and storing stuff.



The modules exposed via DBE are:

- **Semantic translator:** Software component responsible of the translation of queries and documents between two different ontologies (See Deliverable 2.2.2 for more details).
- **SRRN RRServices:** Entry point to the Registry and Repository artifact (see D3.3 for more details).
- **Query Controller:** Semantic router that controls the translation between ontologies (following the mesh obtained from the mapping files of the ontologies. It is able of translate queries (Xquery) or documents expressed following a GLOB (See Deliverable 3.3).

The interfaces offered by these component via SOAP Web Services, can be found in the deliverables described above. The DBE interfaces implemented are described in the following sections.

9.1 Semantic Translator

The interfaces that have been implemented are:

9.1.1 storeExecutionData

Stores ontology information or translation data in the Semantic Translator module along with proper metadata describing the provided data. This method feeds the Semantic Translator internal cache.

Input

- * *data*. The file to be stored.
- * *metadata*. Information describing the provided file.

Output



A return code 0 if the operation succeeded, otherwise a negative integer value representing an error code.

9.1.2 beginSession

Invokes the Semantic Translator for initialising the resources for a new translation session. The returned value is a XML string containing the session information.

Input

- * *username*. Username associated to the new session.
- * *password*. Password associated to the new session.
- * *src_ontology_meta*. Metadata identifying the source ontology.
- * *dest_ontology_meta*. Metadata identifying the destination ontology.
- * *execution_mode*. 0 – document translation; 1 – query translation ; 2 – terms translation
- * *data_model_type*. Identifies the ontology data model type. 0 – collaboration; 1- company; 2 – negotiation; 3 – product and service
- * *root_type_name*. Identifies the data model root type within the provided data_model_type.

Output

If the return code is 0, the return message property provides the session identifier to be used in the further invocations. Otherwise, an error return code.

9.1.3 addSourceObject

Adds a document, a query or term list as XML file to be translated to the provided session. The object structure has to be conform to the parameters provided to the beginSession method. Many documents/queries/terms can be translated within the same session.

Input

- * *username*. Username associated to the session.
- * *password*. Password associated to the session.
- * *session_id*. Session identifier.
- * *obj*. Object to be translated (document or query).

Output

The return message provides an object identifier if the operation succeeds. The methods authorise the request by the username, password and session_id values.

9.1.4 translateObjects

Launches the translation process on the provided objects.

Input

- * *username*. Username associated to the session.
- * *password*. Password associated to the session.
- * *session_id*. Session identifier.

9.1.5 isObjectTranslated

Returns the translation execution state for the given object within the given session.

Input

- * *username*. Username associated to the session.
- * *password*. Password associated to the session.
- * *session_id*. Session identifier.



- * *obj_id*. Object identifier.

Output

Return code is 0 if the translation has successfully completed, 1 if the translation is still running, otherwise a negative integer value representing an error code.

9.1.6 getTranslatedObject

Returns the translated object according the provided parameters.

Input

- * *username*. Username associated to the session.
- * *password*. Password associated to the session.
- * *session_id*. Session identifier.
- * *obj_id*. Object identifier.

Output

The translated object as attachment property if the process has been successfully executed, otherwise a negative return code.

9.1.7 closeSession

Closes the session, removing all the allocated information.

Input

- * *username*. Username associated to the session.
- * *password*. Password associated to the session.
- * *session_id*. Session identifier.

Output

Return code is 0 if the operation succeeded, otherwise a negative integer value representing an error code.

9.1.8 translate

Translates the provided content between two nodes of the SEAMLESS network, regardless their topology. No session is required for performing that kind of translation.

Input

- * *username*. Username for authentication.
- * *password*. Password for authentication.
- * *srcOntologyMeta*. Metadata identifying the source ontology.
- * *destOntologyMeta*. Metadata identifying the destination ontology.
- * *executionMode*. 0 – document translation; 1 – query translation ; 2 – terms translation
- * *srcDataModelType*. Identifies the ontology data model type. 0 – collaboration; 1- company; 2 – negotiation; 3 – product and service
- * *srcRootTypeName*. Identifies the data model root type within the provided *data_model_type*.
- * *translationContent*. The binary content to be translated according to what explained for the *addSourceObject* method
- * *isContentArchive*. If true, the provided binary content is considered a zipped archive of documents to be translated.

Output

The translated object (zipped according to the *isContentArchive* parameter) as attachment property if the process has been successfully executed, otherwise a negative return code.



9.2 RRServices

9.2.1 Lifecycle Interface

9.2.1.1 storeRawDocument

```
java.lang.String storeRawDocument(byte[] pBytDocument,  
    java.lang.String pStrVersion,  
    java.lang.String pStrDocumentName,  
    java.lang.String pStrDocumentType,  
    java.lang.String pAccessRights,  
    java.lang.String pStrTransactionID,  
    java.lang.String pStrUserID,  
    java.lang.String pStrSecurityToken,  
    java.lang.String pStrSourceGLOB)
```

The method stores a document in the repository. During storage metadata for the document is generated by the service. The metadata includes only basic information like identity of the creator, creation date...

Parameters:

`pBytDocument` - document (BASE64) to be stored in the repository.
`pStrVersion` - version Initial document version. If this parameter is null then the version information isn't added to document.
`pStrDocumentName` - Document name
`pStrDocumentType` - Type of document (for translation purposes)
`pAccessRights` - Permissions for document
`pStrTransactionID` - Transaction ID associated to the request
`pStrUserID` - User identifier
`pStrSecurityToken` - SAML token.
`pStrSourceGLOB` - Identifier of GLOB

Returns:

The unique identifier of the stored document.

9.2.1.2 store

```
java.lang.String store(byte[] pBytDocument,  
    java.lang.String pStrMetadataRDF,  
    java.lang.String pStrVersion,  
    java.lang.String pStrDocName,  
    java.lang.String pStrDocType,  
    java.lang.String pStrNewAccessRights,  
    java.lang.String pStrTransactionID,  
    java.lang.String pStrUserID,  
    java.lang.String pStrSecToken,  
    java.lang.String pStrSourceGLOB)
```

The method stores a document in the repository. The metadata may contain information on the document. The service enriches the metadata during the processing.

Parameters:

`pBytDocument` - document (BASE64) to be stored in the repository.
`pStrMetadataRDF` - Metadata for the document
`pStrVersion` - version Initial document version. If this parameter is null then the version information isn't added to document.
`pStrDocName` - Name of document
`pStrDocType` - Type of document (for translation purposes)
`pStrNewAccessRights` - Permissions for document



pStrTransactionID - Transaction ID associated to the request
pStrUserID - User identifier
pStrSecToken - SAML token.
pStrSourceGLOB - Identifier of GLOB

Returns:

The unique identifier of the stored document.

9.2.1.3 storeMetadata

```
java.lang.String storeMetadata(java.lang.String pStrMetadata,  
    java.lang.String pStrVersion,  
    java.lang.String pStrNewAccessRights,  
    java.lang.String pStrTransactionID,  
    java.lang.String pStrUserID,  
    java.lang.String pStrSecToken)
```

This method stores metadata entries in the repository following the RDF format defined for storage of metadata at the repositories. When creating the new entry, the repository generates a unique identifier for it.

Parameters:

pStrMetadata - a string containing the metadata that should be stored, in RDF format
pStrVersion - Version of the document. If version is null then no version is added to document. If version is "AUTO" then is generated automatically.
pStrNewAccessRights - - Permissions to this metadata.
pStrTransactionID - Transaction ID associated to the request
pStrUserID - User Identifier
pStrSecToken - SAML token

Returns:

The identifier of the stored metadata entry of the stored metadata if succeed.

9.2.1.4 updateMetadata

```
boolean updateMetadata(java.lang.String seemID,  
    java.lang.String metadata,  
    java.lang.String version,  
    java.lang.String transactionID,  
    java.lang.String userID,  
    java.lang.String securityToken)
```

Updates the metadata entry specified by the provided identifier.

Parameters:

seemID - Identifier of the metadata
metadata - New metadata to be stored
version - New document version. If this parameter is null then the metadata is updated but its version doesn't changes. If this parameter is "AUTO" then the metadata is updated and a new version tag is generated increasing by one the subversion value, example if last version is 1.3 the new version will be 1.4.
transactionID - Transaction ID associated to the request
userID - User identifier
securityToken - SAML token.

Returns:

The identifier of the stored metadata entry.

9.2.1.5 updateDocument

```
boolean updateDocument(java.lang.String pStrSeemID,
```



```
byte[] pBytDocument,  
java.lang.String pStrVersion,  
java.lang.String pStrDocumentName,  
java.lang.String pStrDocumentType,  
java.lang.String pStrTransactionID,  
java.lang.String pStrUserID,  
java.lang.String pStrSecurityToken,  
java.lang.String pStrSourceGLOB)
```

The method updates an existing document. The user has to provide the unique identifier of the document and the document itself.

Parameters:

pStrSeemID - Identifier of document to be updated
pBytDocument - New version of the document
pStrVersion - New document version. If this parameter is null then the metadata is updated but its version doesn't changes. If this parameter is "AUTO" then the metadata is updated and a new version tag is generated increasing by one the subversion value, example if last version is 1.3 the new version will be 1.4.
pStrDocumentName - Name of document
pStrDocumentType - Type of document
pStrTransactionID - Transaction ID associated to the request
pStrUserID - User identifier
pStrSecurityToken - SAML token.
pStrSourceGLOB - Identifier of GLOB

Returns:

The identifier of the updated document.

9.2.1.6 remove

```
java.lang.String remove(java.lang.String seemID,  
java.lang.String transactionID,  
java.lang.String userID,  
java.lang.String securityToken)
```

Removes the entry document and/or metadata specified by the provided identifier. In case of an document, the method deletes both document and metadata.

Parameters:

seemID - Id of the object in the repository to be deleted
transactionID - Transaction ID associated to the request
userID - User identifier
securityToken - SAML token.

Returns:

Id of the deleted element

9.2.1.7 setAccessRightsFor

```
boolean setAccessRightsFor(java.lang.String seemID,  
java.lang.String newAccessRights,  
java.lang.String transactionID,  
java.lang.String userID,  
java.lang.String securityToken)
```

Method that allows to set access rights to an object

Parameters:

seemID - Identifier of object in repository (seemID)
newAccessRights - New permissions
transactionID - Transaction ID associated to the request
userID - User identifier
securityToken - SAML token

Returns:

true in case the access rights had been changed, false otherwise.

9.2.1.8 setDefaultRightsFor

boolean **setDefaultRightsFor**(java.lang.String seemID,
java.lang.String transactionID,
java.lang.String userID,
java.lang.String securityToken)

Method that allows to set the default access rights to an object

Parameters:

seemID - Identifier of object in repository (seemID)
transactionID - Transaction ID associated to the request
userID - User identifier
securityToken - SAML token

Returns:

true in case the access rights had been changed, false otherwise

9.2.1.9 attachDocument

java.lang.String **attachDocument**(byte[] document,
java.lang.String seemIdOfMetadata,
java.lang.String documentName,
java.lang.String documentType,
java.lang.String document,
java.lang.String preDefinedNodeList,
java.lang.String transactionID,
java.lang.String userID,
java.lang.String securityToken)

method to attach an document to an existing metadata seemID.

Parameters:

seemIdOfMetadata - Identifier of metadata to attach document
documentName - Name of document
documentType - Type of document
document - Document to be stored
preDefinedNodeList - List of nodes to be used
transactionID - Transaction ID associated to the request
userID - User identifier
securityToken - SAML token

Returns:

new documentSeemID for this document

9.2.2 Query Interface**9.2.2.1 getAccessRightsFor**

java.lang.String **getAccessRightsFor**(java.lang.String seemID,
java.lang.String transactionID,
java.lang.String userID,
java.lang.String securityToken)



This method returns the access rights of an object in repository

Parameters:

seemID - The object's identifier
transactionID - Transaction id associated to request
userID - Identifier of user
securityToken - Parameter for user authentication

Returns:

An object ID with AccessPermissions.

9.2.2.2 getDocument

```
byte[] getDocument(java.lang.String seemID,  
                   java.lang.String transactionID,  
                   java.lang.String userID,  
                   java.lang.String securityToken)
```

Method to retrieve documents from the repositories

Parameters:

seemID - The identifier of the document to be retrieved
transactionID - Transaction id associated to request
userID - Identifier of user
securityToken - Parameter for user authentication

Returns:

The document is returned as byte array.

9.2.2.3 getMetadata

```
java.lang.String[] getMetadata(java.lang.String seemID,  
                               java.lang.String transactionID,  
                               java.lang.String userID,  
                               java.lang.String securityToken)
```

Get metadata allows to retrieve the metadata that describes the information at the SRRN for specific objects.

Parameters:

seemID - the seem identifier of the requested metadata entry, or an XML or RDQL query.
transactionID - Transaction id associated to request
userID - Identifier of user
securityToken - Parameter for user authentication

Returns:

A RDF string array containing the retrieved metadata entry, or an empty string if no metadata entry with the specified identifier could be found in the storage or access violates security rules.

9.2.2.4 megaPing

```
org.SEAMLESS.srn.datatypes.MegaPingResult megaPing(java.lang.String pStrTransactionID,  
                                                    java.lang.String pStrUserID,  
                                                    java.lang.String pStrSecurityToken)
```

Method that allow to know the status of all nodes.

Parameters:

pStrTransactionID - Transaction id associated to request
pStrUserID - Identifier of user



pStrSecurityToken - Parameter for user authentication

Returns:

Returns an object with status of nodes.

9.2.2.5 queryTest

org.SEAMLESS.srrn.datatypes.QueryTestResult[] **queryTest**(java.lang.String pStrXMLQueryTest,
java.lang.String pStrTransactionID,
java.lang.String pStrUserID,
java.lang.String pStrSecurityToken)

Method to perform an query test

Parameters:

pStrXMLQueryTest - Query in XQuery format

pStrTransactionID - Transaction identifier

pStrUserID - User identifier

pStrSecurityToken - Parameter for user authentication

Returns:

Object containing the tests results

9.2.2.6 removeAllDocumentVersions

boolean **removeAllDocumentVersions**(java.lang.String pStrDocumentSeemID,
java.lang.String pStrTransactionID,
java.lang.String pStrUserID,
java.lang.String pStrSecurityToken)

Method to remove all versions of one document from repository

Parameters:

pStrDocumentSeemID - ID of document one document to be removed

pStrTransactionID - Transaction id associated to request

pStrUserID - User identifier

pStrSecurityToken - Parameter for user authentication

Returns:

true if success.

9.2.2.7 removeAllMetadataVersions

boolean **removeAllMetadataVersions**(java.lang.String pStrSeemID,
java.lang.String pStrTransactionID,
java.lang.String pStrUserID,
java.lang.String pStrSecurityToken)

Method to remove all versions of metadata from repository

Parameters:

pStrSeemID - ID of metadata to be removed

pStrTransactionID - Transaction id associated to request

pStrUserID - User Identifier

pStrSecurityToken - Parameter for user authentication

Returns:

true if success.

9.2.2.8 getLastVersionDocument

javax.activation.DataHandler **getLastVersionDocument**(java.lang.String pStrDocumentSeemID,
java.lang.String pStrDocumentType,
java.lang.String pStrTransactionID,
java.lang.String pStrUserID,
java.lang.String pStrSecurityToken)

Method to retrieve last version of one document from the repositories by the Document seemID

Parameters:

pStrDocumentSeemID - ID of Document



pStrDocumentType - Type of document (for translation purposes)
pStrTransactionID - Transaction id associated to request
pStrUserID - User Identifier
pStrSecurityToken - Parameter for user authentication

Returns:

Datahandler with document.

9.2.2.9 getLastVersionMetadata

```
java.lang.String getLastVersionMetadata(java.lang.String pStrSeemID,  
                                       java.lang.String pStrTransactionID,  
                                       java.lang.String pStrUserID,  
                                       java.lang.String pStrSecurityToken)
```

Method to retrieve last version of metadata from the repositories by the seemID

Parameters:

pStrSeemID - ID of metadata
pStrTransactionID - Transaction id associated to request
pStrUserID - User Identifier
pStrSecurityToken - Parameter for user authentication

Returns:

string with metadata

9.2.2.10 getMetadataIDFromDocument

```
java.lang.String getMetadataIDFromDocument(java.lang.String pStrDocumentSeemID,  
                                           java.lang.String pStrTransactionID,  
                                           java.lang.String pStrUserID,  
                                           java.lang.String pStrSecurityToken)
```

Method to retrieve metadata which has associated the DocumentSeemID passed in parameters

Parameters:

pStrDocumentSeemID - ID of document to find its metadata.
pStrTransactionID - Transaction id associated to request
pStrUserID - User Identifier
pStrSecurityToken - Parameter for user authentication.

Returns:

string with metadata

9.2.2.11 queryMetadata

```
java.lang.String[] queryMetadata(java.lang.String pStrSrrnQuery,  
                                 java.lang.String pStrXMLQueryDescriptor,  
                                 java.lang.String pStrTransactionID,  
                                 java.lang.String pStrUserID,  
                                 java.lang.String pStrSecurityToken)
```

Method to query metadata from repositories

Parameters:

pStrSrrnQuery - Query to execute
pStrXMLQueryDescriptor - xml specifying some filters on the query
pStrTransactionID - Transaction id associated to request
pStrUserID - User Identifier
pStrSecurityToken - Parameter for user authentication.

Returns:

string array with metadata seemids founded.



9.2.2.12 ping

```
java.lang.String ping(java.lang.String echoed,  
    java.lang.String userID)
```

Method that allow to ping the service to know if it is running

Parameters:

userID - The user's Digital Signature
echoed - A String that is returned

Returns:

String with the input string and some information about the service.

9.2.2.13 getLogs

```
java.lang.String [] getLogs(java.lang.String transactionID2Find,  
    org.SEAMLESS.srrn.datatypes.EventLevelEnum level,  
    int depth,  
    java.lang.String transactionID,  
    java.lang.String userID,  
    java.lang.String securityToken)
```

Method to retrieve the information log generated by a transaction in a specific layer.

Parameters:

transactionID2Find - The transaction id which corresponds to the operation which we are looking for.
level - Info level of logs to retrieve: DEBUG, INFO, WARN or ERROR.
depth - Depth is the layers that are involved in the search. Values: 0: Application Layer 1: Service 2: Distribution 3: Repository
transactionID - Transaction id asociated to request
userID - User identifier
securityToken - Parameter for user authentication

Returns:

This method returns all log entries for a special transaction-ID.

9.2.2.14 getLogsRecursively

```
java.lang.String [] getLogsRecursively(java.lang.String transactionID2Find,  
    org.SEAMLESS.srrn.datatypes.EventLevelEnum level,  
    int depth,  
    java.lang.String transactionID,  
    java.lang.String userID,  
    java.lang.String securityToken)
```

Method to retrieve the information log generated by a transaction in a layer and all the layers that are involved with distinct levels of detail.

Parameters:

transactionID2Find - The transaction id to be executed
level - DEBUG, INFO, WARN or ERROR
depth - Depth is the number of layers that are involved. * Values: 0: Application Layer 1: Application, Service 2: Application, Service, Distribution 3: Application, Service, Distribution, Repository
transactionID - Transaction id asociated to request
userID - User Identifier
securityToken - Parameter for user authentication



Returns:

This method returns all log entries for a special transaction-ID.

9.2.2.15 querySyncReturnContent

```
java.lang.String[] querySyncReturnContent(java.lang.String query,  
                                           int start,  
                                           int number,  
                                           java.util.Calendar timestamp,  
                                           java.lang.String transactionID,  
                                           java.lang.String userID,  
                                           java.lang.String securityToken)
```

Method to retrieve rdf metadata in a synchronous way from distribution layer

Parameters:

query - The query to be executed
start - Number of the first result to be retrived
number - Number of results to be retrived from start result
timestamp - Max time to wait for results.
transactionID - Transaction id asociated to request
userID - User identifier
securityToken - Parameter for user autentication

Returns:

String array with metadata in RDF format.

9.2.2.16 getAllDocumentVersions

```
java.lang.String[] getAllDocumentVersions(java.lang.String pStrDocumentID,  
                                           java.lang.String pStrDocumentType,  
                                           java.lang.String pStrTransactionID,  
                                           java.lang.String pStrUserID,  
                                           java.lang.String pStrSecurityToken)
```

Method to retrieve all versions for a document

Parameters:

pStrDocumentID - ID of document to return.
pStrDocumentType - Type of document (for translation pourposes)
pStrTransactionID - Transaction id associated to request
pStrUserID - User identifier
pStrSecurityToken - Parameter for user autentication

Returns:

String array with Document seem Ids

9.2.2.17 getAllMetadataVersions

```
java.lang.String[] getAllMetadataVersions(java.lang.String pStrSeemID,  
                                           java.lang.String pStrTransactionID,  
                                           java.lang.String pStrUserID,  
                                           java.lang.String pStrSecurityToken)
```

Method to retrieve all versions for a metadata

Parameters:

pStrTransactionID - Transaction id associated to request
pStrSeemID - ID of metadata to return.
pStrUserID - Public user key of the user that executes the service



9.2.2.18 `getDocumentBySeemID`

```
byte[] getDocumentBySeemID(java.lang.String pStrSeemID,  
                             java.lang.String pStrType,  
                             java.lang.String pStrTransactionID,  
                             java.lang.String pStrUserID,  
                             java.lang.String pStrSecurityToken)
```

Retrieve document thru its document seemID

Parameters:

`pStrSeemID` - Seem Id of document
`pStrType` - Type of document (for translation pourpouses)
`pStrTransactionID` - Transaction id associated to request
`pStrUserID` - User Identifier
`pStrSecurityToken` - Parameter for user authentication

Returns:

Datahandler with document

9.2.2.19 `getDocumentIDFromMetadata`

```
java.lang.String getDocumentIDFromMetadata(java.lang.String pStrTransactionID,  
                                             java.lang.String pStrSeemID,  
                                             java.lang.String pStrUserID,  
                                             java.lang.String pStrSecurityToken)
```

Retrieve document thru its metadata seemID associated

Parameters:

`pStrTransactionID` - Transaction id associated to request
`pStrSeemID` - Seem Id of document
`pStrUserID` - User Identifier
`pStrSecurityToken` - Parameter for user authentication

Returns:

seemID of the document

9.2.2.20 `getDocumentInfoByMetadataSeemID`

```
String[] getDocumentInfoByMetadataSeemID(java.lang.String pStrSeemID,  
                                           java.lang.String pStrTransactionID,  
                                           java.lang.String pStrUserID,  
                                           java.lang.String pStrSecurityToken)
```

Method to retrieve the document information attached to metadata.

Parameters:

`pStrSeemID` - Metadata SeemId
`pStrTransactionID` - Transaction id associated to request
`pStrUserID` - User Identifier
`pStrSecurityToken` - Parameter for user authentication

Returns:

An array of information of documents attached to metadata.

9.3 Query Controller

9.3.1 `queryForMetadata`

```
java.lang.String[] queryForMetadata(java.lang.String pStrSRRNQuery,  
                                     java.lang.String pStrXMLQueryDesc,  
                                     java.lang.String pStrSrcOnt,
```



```
java.lang.String pStrSrcDataType,  
java.lang.String pStrRootType,  
int pIntScope,  
java.lang.String pStrUserID,  
java.lang.String pStrUserPWD,  
java.lang.String pStrSecToken)
```

Query the repository to obtain list of seemIDs

Parameters:

pStrSRRNQuery - Query in SparQL format
pStrXMLQueryDesc - T.B.D. (for translation services).
pStrSrcOnt - Source ontology of the user
pStrSrcDataType - Type of data (used in translation pourpouses)
pStrRootType - Root type of data (used in translation pourpouses)
pIntScope - T.B.D.
pStrUserID - User identifier
pStrUserPWD - User password
pStrSecToken - Parameter for user autentication

Returns:

A string array of metadata seemID founded.

9.3.2 queryForMetadataAsync

```
boolean queryForMetadataAsync(java.lang.String pStrSRRNQuery,  
java.lang.String pStrXMLQueryDesc,  
java.lang.String pStrSrcOnt,  
java.lang.String pStrSrcDataType,  
java.lang.String pStrRootType,  
int pIntScope,  
java.util.Calendar pCalTimeout,  
java.lang.String pStrFromEndpointURL,  
java.lang.String pStrFromServiceURN,  
java.lang.String pStrUserID,  
java.lang.String pStrUserPWD,  
java.lang.String pStrSecToken)
```

Query the repository to obtain list of seemIDs in asynchronous way

Parameters:

pStrSRRNQuery - Query in SparQL format
pStrXMLQueryDesc - T.B.D.
pStrSrcOnt - Source ontology of the user
pStrSrcDataType - Type of data (used in translation pourpouses)
pStrRootType - Root type of data (used in translation pourpouses)
pIntScope - T.B.D.
pCalTimeout - Timeout used in the query
pStrFromEndpointURL - URL used to return the results
pStrFromServiceURN - Service to be used to return the results
pStrUserID - User identifier
pStrUserPWD - USer password
pStrSecToken - Parameter for user autentication

Returns:

true if the query is successfully stored



9.3.3 getDetails

```
byte[] getDetails(java.lang.String pStrSeemID,  
                 java.lang.String pStrSrcDataType,  
                 java.lang.String pStrRootType,  
                 java.lang.String pStrVersion,  
                 java.lang.String pStrSrcOntID,  
                 int pIntScope,  
                 java.lang.String pStrUserID,  
                 java.lang.String pStrUserPWD,  
                 java.lang.String pStrSecurityToken)
```

Method to retrieve a document from the repository

Parameters:

pStrSeemID - Document identifier

pStrSrcDataType - Type of data (used in translation pourposes)

pStrRootType - Root type of data (used in translation pourposes)

pStrVersion - Version of the document

pStrSrcOntID - Source ontology (used in translation)

pIntScope - T.B.D.

pStrUserID - User identifier

pStrUserPWD - User password

pStrSecurityToken - Parameter for user authentication

Returns:

The document retrieved

9.3.4 ping

```
java.lang.String ping(java.lang.String pStrTransactionID,  
                     java.lang.String pStrSecToken)
```

Test method

Parameters:

pStrTransactionID - Transaction ID

pStrSecToken - Parameter for user authentication

Returns:

String with test characters.

9.3.5 getGLOBTranslationPath

```
java.lang.String[] getGLOBTranslationPath(java.lang.String srcGLOBMetadataSeemId,  
                                         java.lang.String destGLOBMetadataSeemId,  
                                         java.lang.String userID,  
                                         java.lang.String securityToken)
```

Method to retrieve a seemID list of mapping files used to convert from source to destination ontologies

Parameters:

srcGLOBMetadataSeemId - : ID of source ontology

destGLOBMetadataSeemId - : ID of destination ontology

userID - User identifier

securityToken - Parameter for user authentication

Returns:

String array with mapping files seemIDs

