Contract No. 004420

eu-DOMAIN - enabling **u**sers for
Distance-working & **O**rganizational **M**obility
using **A**mbient **I**ntelligence **N**etworks

D4.3 Prototype network intelligence pool with database, XML server

Specific Targeted Research or Innovation Project

Project start date 1st June 2004 **Duration 36 months**

Published by the eu-DOMAIN Consortium **May 2006 Version 1.2**
Lead Contractor Innova

Project co-funded by the European Commission
within the Sixth Framework Programme (2002 -2006)

Dissemination Level: Confidential

Document file: D4.3 Prototype network intelligence pool with database, XML server 1.2.doc

Work package: WP4 - Server side architecture

Task: T4.2 - Network intelligence pool and database management systems

Document owner: [Javier Cámara (Software AG Spain)]

Document history:

Version	Author(s)	Date	Changes made
1.0	Javier Cámara	18-01-2006	
1.1	Javier Cámara, Pablo Antolín	13-05-2006	Included description of Event Manager Update software features changed since 1.0
1.2	Javier Cámara, Pablo Antolín	20-05-2006	Minor QA updates. Final version submitted to the European Commission

Review history:

Reviewed by	Date	Validated

Index:

1. Executive summary 4
2. Where to get the software 5
3. The Data Server 6
4. The Domain Model concept and infrastructure 8
4.1 The Domain Model 8
4.2 Conceptual versus Operative Models 9
4.3 Implementation of the Conceptual Model concept in eu-DOMAIN 11
4.4 Modelling of events 12
5. The Domain Model Interpreter (DMI) 14
5.1 The Java DMI 14
6. The Network Intelligence Manager (NIM) and its portlet 18
7. The Event Manager 19
7.1 Sequence Diagram 20
7.2 Apache Pubscribe 20
8. Other components in the deliverable 22
8.1 The UDDI server 22
8.2 The WebDAV server 22
8.3 The Domain Model Browser (ex SOA Modeler) 23
9. The SOA tools: a byproduct 25
9.1 The UDDI registrar 25
9.2 The SOAP sender 26
9.3 The XML browser 26
10. Summary of Java packages 28
11. Integration tests 29
12. References 30

1. Executive summary

This document describes the results included in the deliverable *D4.3 Prototype network intelligence pool with database, XML server*, which are the following ones:

- The Data Server
- The Domain Model solution
- The Domain Model Interpreter
- The Network Intelligence Manager and associated portlet
- The UDDI Server
- The WebDAV Server
- The Domain Model Browser (previously known as the SOA Modeler)
- The Event Manager

Besides some byproduct tools for Service Oriented Architectures (SOAs).

All of them but the Data Server may be included in the “Network Intelligence” category.

The main goal of this deliverable is to release software to be used by other partners. Thus this document only describes these components and summarizes the design decisions taken about them, but does not include a full design or specification for them. It only makes references to their original specification documents.

Also, an integration test plan is included in the deliverable, although as a separate document [14] .

2. Where to get the software

D4.3 is supposed to deliver software, and this document is only a brief description of it. The actual software may be obtained from the following resources:

Resource	Location
Sources of the Event Manager software	Shared eu-Domain BCSW: http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d120696/EventManager.zip
Sources of the rest of the software of D4.3	Shared eu-Domain BCSW: http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d105605/sources.zip
Changes made to the ActiveBPEL used in D4.3	Shared eu-Domain BCSW: http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d107057/Changes%20made%20to%20ActiveBPEL.zip
Online demonstration of the Domain Model Interpreter	http://rtd.softwareag.es/dmiScenario/
Online demonstration of the Network Intelligence Manager	Instructions to access it were given in the eu-Domain technical forum at http://www.eu-domain.eu.com/forum/scripts/board-auth.cgi?lm=1135770845&file=/9/209.html
Online version of the SOA tools	http://rtd.softwareag.es/soaTools/

3. The Data Server

The original Data Server Specification is available in a separate document [1] . This document also describes the concrete Data Server hosted in the Software AG premises and gives directions on how to use it. Here, only a summary of this document is provided.

The Data Server is the infrastructure available for storing any persistent data that the server side of eu-DOMAIN may need for its operation. This means data *owned* by the server side, since when the data is controlled by a separate Content Provider, the server side will access it online through a web service interface provided by the Content Provider.

The capabilities of the Data Server are be provided by the Tamino XML Server product from Software AG [2] , a native XML database server capable to efficiently and securely store, index, query, retrieve and manage large amounts of XML documents.

While Relational Database Management Systems (RDBMSs) are nowadays the most widely used database technology, and for sure they could also be used in eu-DOMAIN, there are a number of arguments for using a XML database instead:

- The architecture of the elements of the server side is to be a service-oriented one, and in the end each one of these elements will publish and/or depend on interfaces built on top of XML data, defined using XML Schema. Thus, most (if not all) data used by the system is to have a XML representation, and thus it may be seen as natural to store it in an XML database.
- On top of the operative architecture of the system, a Conceptual Domain Model is to be defined, and again this definition is to be made in terms of XML and XML Schema (the Domain Model is described in section 4.1)
- It is easier to handle flexible structures (for example structures that are extended and customized upon deployment at a particular site) in a XML DBMS than in a RDBMS (see section 4.1 for the needs of flexibility in eu-DOMAIN)

Because of this, Tamino has been chosen as the technology supporting the Data Server. However it must be noted that the service-oriented architecture (SOA) of the eu-DOMAIN server side actually does not dictate any particular way of storing data. This architecture is composed of discrete services providing functionality through well defined interfaces, and it is up to each one of them to access a database or not to perform that functionality. Each element of the architecture depends on the availability of a number of such functionalities, but it is irrelevant who implements them, and how, as long as they are available and may be invoked. Elements of the SOA depend one from the other only by means of their published interfaces, each service inside the server side is responsible of implementing these in the best way it considers. For example, a Customer management portlet may depend in the existence of a service providing some *ListCustomers* operation, but for the portlet to be able to run, and for any element in the architecture, it is totally irrelevant whether this operation is implementing by accessing a RDBMS, an XML database, by combining the results of several databases, by accessing an external Content Provider or by any other means, as long as the contract defined by the operation is implemented as expected.

Combined with the fact that this architecture allows for defined operations to be implemented by external services (Content Provider), it means that the Data Server is not any more a critical component of the system. It is of course a useful facility for every service in the server side, but each Manager may choose to store its data using it, or in any other way. Even, if in a particular installation of eu-DOMAIN it is more proper to use a different database technology, it may be used – as far as the services using it are reimplemented to access the new database, of course, which may be not so trivial.

Actually, such freedom of each service is not fully complete, since although services are independent of each other, often its implementations will have to access related data that may be best stored together in the same database (e.g. in a *Patient Manager* and a *Treatment Manager*, it may be more

convenient to store treatments along with patient data). But at least for “service clusters” this freedom exists.

The software and licenses required to install and use a Data Server based on Tamino XML Server have been provided to the proper Consortium partners, and also a concrete Data Server is installed and available over the Internet in the Software AG premises (see [1] for details).

Currently, Tamino XML Server version 4.2 is being used.

4. The Domain Model concept and infrastructure

The concept of the Domain Model and the infrastructure that support it in eu-DOMAIN was first defined in a separate document [3] , which describes in large detail what is the eu-DOMAIN Domain Model and which is its purpose, along with the components to be deployed and implemented so that it becomes a reality.

Here, a summary of that document will be given, along with an update of the most relevant changes happened since it was written.

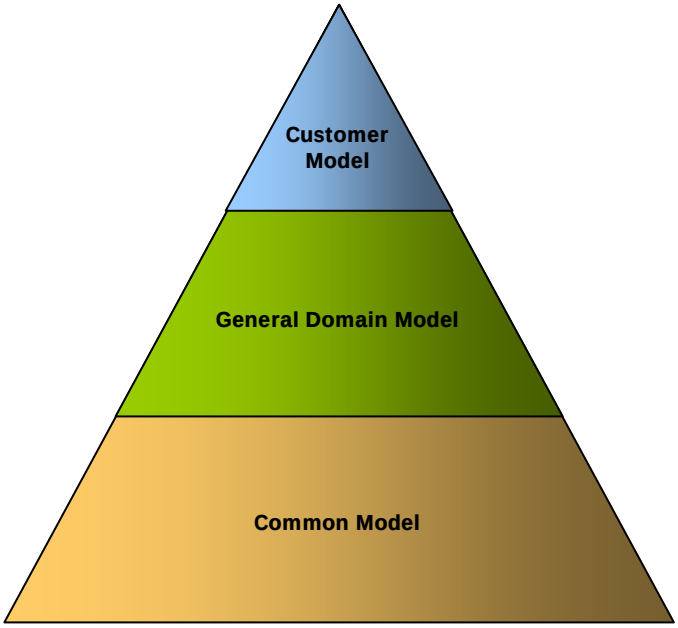
4.1 The Domain Model

A *Domain Model* is a suitable representation of the relevant features of a given Domain of activities (e.g. Health Care or Facility Management). The subjective criteria “suitable” and “relevant” are measured against the tasks that are to be achieved by using that model; e.g. a model to be interpreted by a human is very different from models to be interpreted by a machine or by both. Also, it is not required for them to represent more features than the ones needed for these concrete tasks, or do so in a way more complete than needed, since this “overmodeling” may be counter-productive.

The most widespread concept related with that of a Domain Model is the one of a *Data Model*. The main difference between them is that the only features from the Domain represented in a Data Model are the ones defining the data structure (entities, attributes, relationships and so on), leaving out other ones that are also highly relevant like the operations or processes to be performed with these data. Thus, while a Domain Model almost always includes a Data Model, usually a Data Model is not enough as a Domain Model.

Early [4] it was identified the need to make the implementation of eu-DOMAIN in such a way that it would be based to some extent on a dynamic Domain Model, instead of being built on a Model that is fixed at development time, and immutable after it without further development effort, as in most Information Technology (IT) systems. That dynamic Model could be adapted to different Domains and customers, and then eu-DOMAIN would be able to adapt its behaviour after it either on deploy or run time.

Otherwise, it seems clear that eu-DOMAIN would not be flexible enough to be adapted to the variety of Domains that it is intended to be used for (Health Care and Facility Management are the two first ones that it is focused on, but not the only ones), and probably not even to be used by different customers in a particular Domain. If eu-DOMAIN is not able of such flexibility it will not be a *product*, but just a *project* done for a particular scenario and not useable outside it without significant modification of the code base of its software.



Because of this, the eu-DOMAIN Domain Model has first a *Common Model* covering all the concepts that are common to most Domains; for example, the basic infrastructure of gateways and bundles, or standard business activities as invoicing.

On top and around this Common Model there is a *General Domain Model*, that adds all the concepts useful for most usages of eu-DOMAIN inside a given vertical Domain; for example, *Patient* for Health Care or *Technician* for Facility Management.

And yet complementing these two models there is a *Customer Model*, which would add to them all the particular needs of a specific installation of eu-DOMAIN for a given customer. For any such particular installation, besides allowing adding new concepts, the Domain Model infrastructure allows to *extend* existing ones too, so that the software components delivered are still able to work with them.

It is worth to note that, while it is not difficult to have such a definition of a Domain Model, having software components with such flexibility to be extended and adapted and at the same time performing useful functions for the Domain Users is not an easy task. All the decades of experience in software development show that making software adapt to different needs is still largely a matter of programming. Because of this, in eu-DOMAIN a pragmatic approach has been taken, implementing many of the features in a traditional hard-coded way, but also in such a way that they are extensible to handle new Domain concepts. The mechanisms to achieve that both at the Operative and at the Conceptual levels are described now.

4.2 Conceptual versus Operative Models

A Model is considered here to be *Conceptual* if it may be well understood and handled by people that are not Information Technology (IT) staff. Here, these people are considered to be *Domain Users*.

A Model is termed here as *Operative* when it describes the implementation of a system, entering into technical details such that may not understood by Domain Users; *and* provides enough information for software to actually access the system features (i.e. operate with it). There are usually many levels of Operative Models, e.g. from a WSDL to a class design to a RDBMS schema (even the source code of the software may be considered as a very model, very faithful), but in eu-DOMAIN the only Operative Model taken into account is the one defining the Service Oriented Architecture (SOA) that will be used to run eu-DOMAIN. Thus it includes the WSDL documents defining the services, the XML Schema documents related, the contents of the UDDI server, and so on.

The term Domain Model, although in [3] is considered to include both the Conceptual and the Operative Models, it more often used in the IT community to refer only to the Conceptual Model. However, here we will keep using the term to cover both the Conceptual and the Operative ones, and made explicit differentiations between them by using the terms Conceptual Model and Operative Model.

Usually there exists at least one Operative Model in every system (e.g. APIs or DB schemas), but most existing systems do not have a Conceptual Model that is up to date. If they exist at all, more often Conceptual Models are used only as a means of communicating IT developers with Domain Users, and once the Analysis phase is over they quickly become obsolete.

Approaches like Adaptive Object Models [16] advocate making the Conceptual Model equal to the runtime Operative Model, so that the software of the system is able to understand it and to base its operation into it. But as said before in eu-DOMAIN a pragmatic, mixed approach has been taken. The first requisite to be fulfilled by the eu-DOMAIN Domain Model is the one of flexibility stated above. It must be noted that the Operative Model of the eu-DOMAIN SOA already provides by itself with several facilities for achieving this goal:

- Reliance on a UDDI server allowing to redefine who implements any defined operation
- WSDL and XML Schema which have built-in mechanisms for extension that provide with great capabilities of adaptation

Thus, a system fully relying on a SOA already may have a great deal of flexibility[1]. However, the SOA model is still a technical one and Domain Users are *not* able to understand it. For example, Users of a Health Care Domain of course understand what the concept *Patient* from a Conceptual Model is, and what an operation of it called *setDeceasedStatus(alive or dead)* may do. But this *Patient* concept does not exist in a SOA Operative model. Instead, web services of a SOA may interchange different data structures containing pieces of patient data, like *Patient_Basic*, *Patient_For_Listing* or *Patient_Update_Request*; but there is no coherent structure called *Patient* because it is so large and complex that no real web service would receive or return it.

Because of this and of other technical features that usually Operative Models have (see [3] for more examples), the norm is that Domain Users are not able to understand these, and then they are not able to use it to instruct the system to adapt its operation. But the Domain Users of eu-DOMAIN must be able of performing at least the two following customizations: create **Notification templates**, and create/edit **Rules**. Since both of these cases will have to refer to data and operations which are defined in the Model (e.g. the name of a *Patient* or the phone number of a *Phisician*), the Domain Users will not be able to perform these customizations if they cannot express themselves in the terms of a Model that both and the software understand. Hence the need of a Conceptual Model.

For example, if the consequence of a rule is making a new annotation in a *Patient* record, a SOA Operative Model would require for the Domain User to locate which web service operation from among all the ones available in the UDDI server is to be called (e.g. there may be some 10 services/interfaces related to the Patient), and to figure how its technical parameters are to be fulfilled (e.g. including a *database version* for optimistic locks). On the other hand, a Conceptual Model could have nicely concentrated all the operations for the *Patient* under its class, and would use only parameters with a meaning in the Domain, automatically handling the technical ones.

Besides these cases of Notification templates and Rules, another similar requisite may be drawn from the need to have **flexible Domain applications**, i.e. applications that interact with Domain Users, offer functionality useful for a Domain, and that have to be customized to different Customer installations. For example, every Health Care installation of eu-DOMAIN will have a *Patient* and *Patient Record* concepts, and also some application to browse, search, view, create, edit, delete etc. patients. Although of course each Customer could create his own particular application to perform these capabilities, it is logical that a “Health Care edition” of eu-DOMAIN would include applications

[1] As said before, one thing is that the Model is flexible, and other that the software may actually keep working when this flexibility is applied.

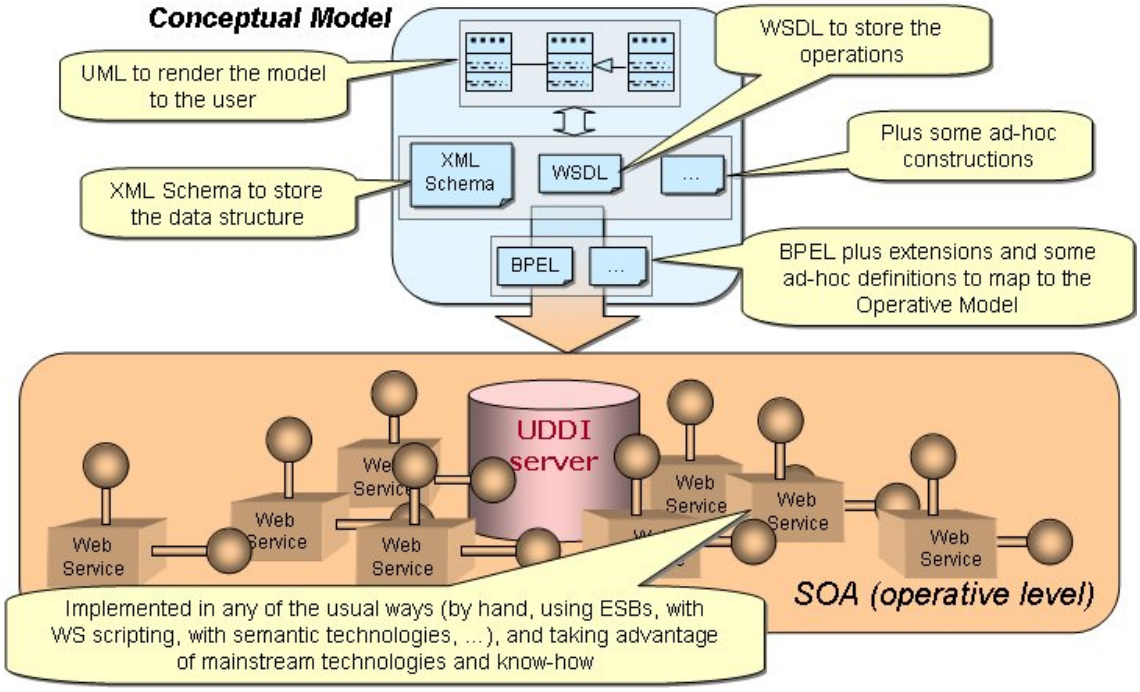
for that, just the same as it is going to provide server-side and client-side logic for Health Care features.

And these applications have to be adapted to the specific Model used in each installation. If this customization is intended to be done by IT staff at deploy time, then the applications may depend on the Operative Model; but if the applications are able to understand and exploit the Conceptual Model instead, then it may be Domain Users (maybe experts, but Domain anyway) who perform this customization. So this is not a strict requisite, but since it is usually good to allow Domain Users customizing as much as they can, it would be good to have it if possible.

To summarize, while the requisite for flexibility could be covered just by exploiting the Operative Model of the SOA, these other three requisites of the Notification manager, the Rule manager and the Domain applications mandate for having a Conceptual Model too.

4.3 Implementation of the Conceptual Model concept in eu-DOMAIN

As described before, the Operative Model used in eu-DOMAIN is the one commonly found in SOAs, so there is not much to decide about it. But for the Conceptual Model there are several options. After studying them in [3] , the solution taken is as follows:



- Both Domain Users and components depending on the Conceptual Model see it as an **Object-Oriented** one, i.e. mainly organized in classes, with inheritance and associations between them, containing each one data attributes and operations
- Internally, the Conceptual Model is represented using the same **technologies used in SOAs**, so that the compatibility between the Operative and Conceptual layers is greater:
    - XML Schema for data structures, including inheritance
    - WSDL for operations
    - A custom XML language for binding them (class = data + operations)

  (Also the WS-Topics language is used for events, as described later)
- The **mapping** between the Conceptual and the Operative layers is done by implementing conceptual operations by means of **BPEL** scripts:
    - Each conceptual operation may have one or more implementations, being each of them a BPEL process. When there is a need of choosing between these

implementations, the selection is done depending on the concrete needs to be achieved (e.g. which data is being retrieved)

- These operations have mappings between models. The input (Conceptual → Operative) and output (Operative → Conceptual) data mappings are implemented inside BPEL by using its own built-in facilities (i.e. the <assign> sentence). The Domain Model infrastructure analyzes these sentences automatically to find out about the mappings, e.g. for later selecting automatically which operation is to be called.

- **Fine-grained operations** (this is, Conceptual operations which are too detailed to be implemented as an operation of a regular web service) are implemented also by means of **BPEL** scripts, only that they do not call any web service during its operation.

In short, both the Conceptual and the Operative Model are defined to the system by means of a set of WSDL, XML Schemas, BPEL and other XML files. The Domain Model is dynamic because it may be changed just by changing these files, and flexible components have means to adapt to process the Model and adapt to these changes. But in the end the implementations of the Operative Model (definitions) i.e. the actual web services) must be in place according to its definitions for the system to work, and hence the mixed approach.

BPEL was chosen as the technology for the mappings from Conceptual to the Operative layer because it provides many advantages:

- It covers the needs of the mapping by providing a readable XML representation for copying data, and on top of that a rich processing model with XPath, conditions, loops and so on
- Although in most cases the Conceptual and the Operative layers will be much similar both in operations and in data, having available a fairly powerful language like BPEL would allow to adapt a given Conceptual Model to many different Operative Models, thus easing the reuse of Conceptual components over different Operative SOAs.
- It offers also a solution for implementing fine-grained Conceptual operations by providing scripting logic, XPath and other pluggable expression engines
- It perfectly integrates with WSDL and XML Schema, which are the basic means used to define both the Operative and the Conceptual Models

In general, it offers a ready implementation to handle Model definitions, implement mappings, evaluate expressions and invoke arbitrary web services[2]. Which besides is based on popular standards which have open source implementations available that may be counted on evolve with time.

4.4 Modelling of events

Besides regular classes, services and structures, the Domain Model must be also able to define *events*. Events are information messages that are distributed inside the eu-DOMAIN infrastructure as a whole and must be processed by it. For example, a gateway may send an alert to the server about the evolution of the heart rate of a patient not being conformant with an expected pattern previously configured by a physician.

These events must be modelled in order for humans to be able to customize the system upon them, and to administer the system. For example, the physician could establish a rule to handle such an event by e.g. sending a notification message to its mobile phone.

This means that events pertain to the Operative Model (because they are used in its actual operation), but also are visible in the Conceptual one (because Domain users should be able to establish rules upon them). Because of this, events in the Domain Model will be augmented with Conceptual features, like e.g. a method to access the whole *Patient* object associated to such a heart rate alert event.

[2] Which is not such a trivial operation because of complexities like Encoded vs Literal data representations as defined by WSDL 1

Just as the definition classes and services try use existing SOA standards whenever possible, events are defined by means of the WS-Topics language which is part of the WS-Notification specification [17] . Thus, WS-Topics documents may be directly included in the Domain Model definition along with WSDL and XML Schema, in order to be used by the Domain Model machinery and Conceptual components. For example, a rule creation wizard would be interested in finding about the defined events and the information available in e.g. the heart rate alert in order to help the physician to establish the mentioned rule.

5. The Domain Model Interpreter (DMI)

This component was first designed in [3] . Here only a summary and update is given.

Once the Conceptual Model is defined in the XML files described before, in order for a Conceptual component to be able to use it, it would have to do a number of operations:

- Parse the WSDL, XML Schema, BPEL and other definition documents
- Exploit the class structure defined by the Model
- Retrieve Conceptual data, and execute Conceptual operations, by processing the mappings to the Operative Model and invoking the proper BPEL processes
- Executing the proper BPEL processes

Obviously, these are non-trivial tasks that must be factored out into a common component to be reused by all other components at the Conceptual layer. This common component is the Domain Model Interpreter (DMI). Its main task is to allow introspection of the Domain Model definitions so that the Conceptual component can know about the Domain features and use them, and translate requests for this usage into operations of the Operative Model, i.e. web service calls.

Originally the technical interface (API) to call the own Domain Model Interpreter was not to be networked, but a local one, because the large Conceptual data structures are not suitable for network transmission. However, in the end, since only a Java DMI was built, in order for .Net components to be able to access it, a SOAP interface was also added.

5.1 The Java DMI

There is a Java DMI available as an API of classes offering:

- A representation in Java classes of the Metamodel underlying the Domain Model (i.e. with concepts like “Domain Model Class”, “Domain Model Operation”, etc), along with logic to create this metamodel objects from the XML documents defining the Model
- Additional classes to operate the Conceptual Model features, i.e. to retrieve Conceptual data and to invoke Conceptual operations

These classes are Domain-independent, this is they are valid for being used in every Domain. However, on top of them, Domain-specific classes may be built, so that they offer features (i.e. data and operations) specific of a given model. This would ease the development of Conceptual components relying on such features (e.g. a Patient Browser portlet), while at the same time not losing the ability to discover and invoke arbitrary, dynamically defined features added to the Model after these Conceptual components were developed.

The SOAP interface of the Java DMI allows to define instances of object classes defined in the Domain Model and to evaluate expressions against them, which should be enough for the envisioned .Net Conceptual Components.

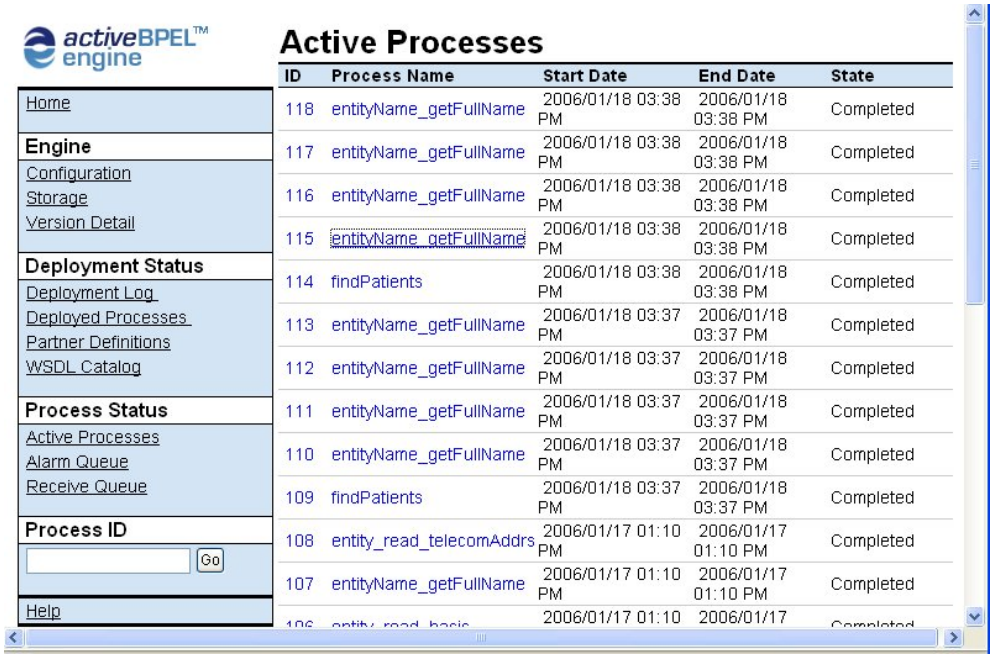
As described before, many of the DMI functionalities rely on the execution of BPEL scripts, which are best executed by an existing BPEL engine. Two open source engines were evaluated, FiveSight PXE [5] and Active Endpoints ActiveBPEL [6] , and the latter was chosen because of better maturity, performance, features and administration capabilities. Thus, the DMI Java classes handle the Model definition and decide when to execute which BPEL process, but for this they rely on the ActiveBPEL runtime.

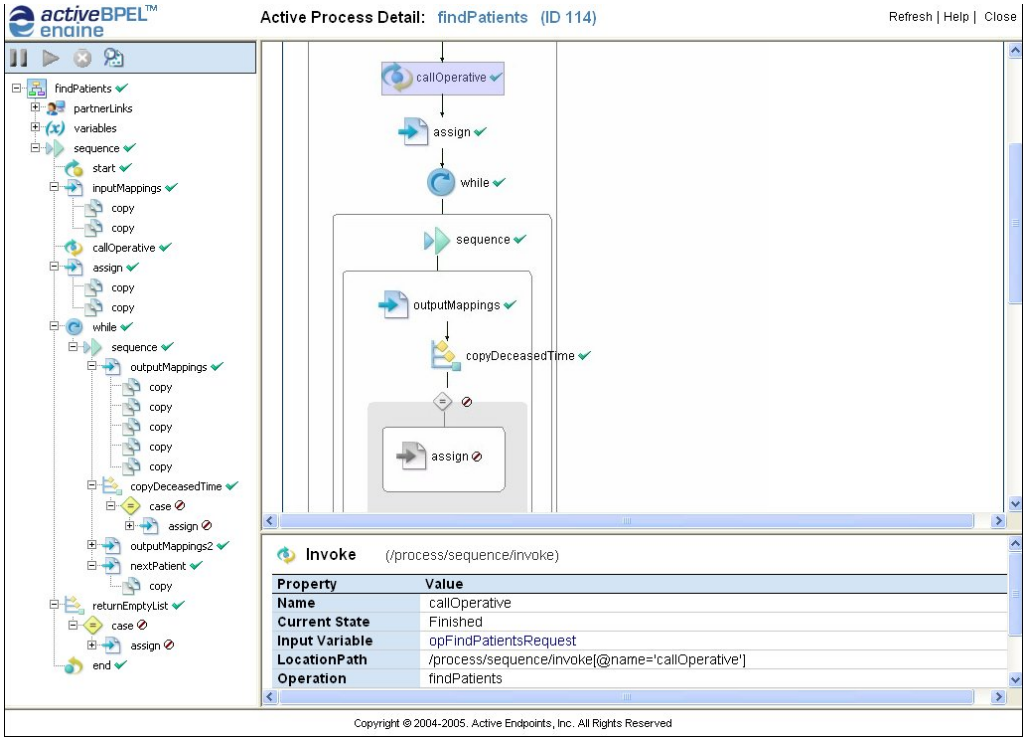
It must be noted that the BPEL process are invoked locally by directly calling the ActiveBPEL classes, i.e. that no SOAP call or network trip is involved in it.

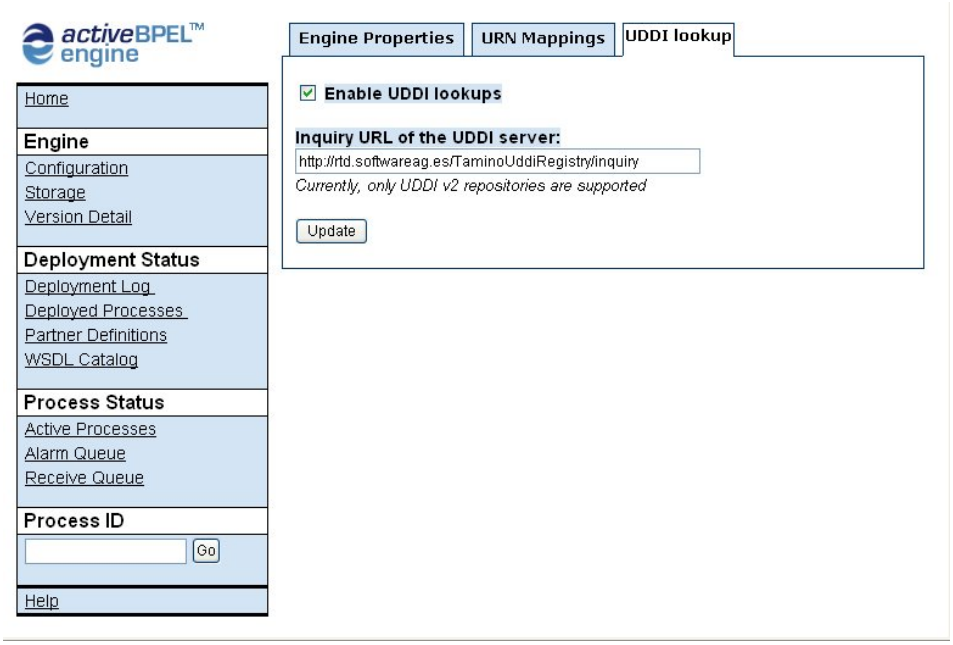
Since ActiveBPEL must run inside a web application server, and the DMI invokes ActiveBPEL locally, this means that the DMI must also be run inside this server. Because of this, Conceptual applications calling the DMI locally must also be run inside the server. But since all of the components in the

server side of eu-DOMAIN (portlets, Managers, and Interpreters) run inside some kind of web application server, this should not be a problem.

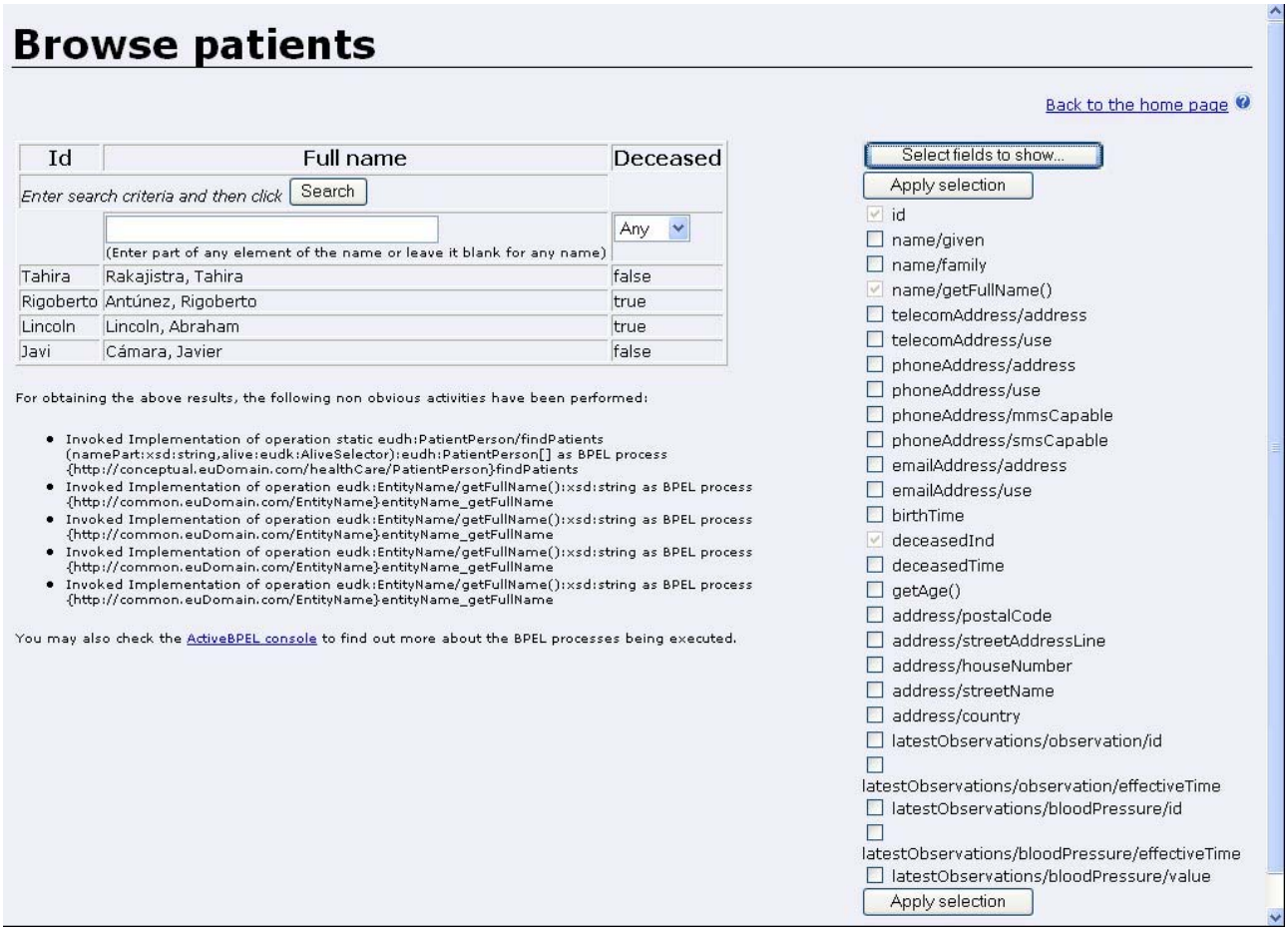
The open source approach has proven to be much useful, since it has allowed to better understand the inners of BPEL processing, patch a couple of bugs in the ActiveBPEL engine, and to add UDDI lookups to it (i.e. whenever a web service is to be invoked, its URL is looked up in the UDDI server by searching for the first implementation of its SOAP binding). All of these enhancements have been both contributed back to the open source community around ActiveBPEL and included in D4.3 .







The usefulness of the Java DMI has been tested by defining a scenario [7] which would provide verifiable, concrete requirements to be met. This scenario was drawn from the much bigger one "Patients as Customers". For example, this is a mock up of a Domain application with a User interface, which lists a fixed set of data items but allows to show additional ones dynamically obtained from the Domain Model:



While the current status of the Java DMI offers a significant set of capabilities allowing to work with realistic Domain Models, it must be noted that it still lacks the following features:

- ActiveBPEL currently has been run only inside Tomcat, and not inside JBoss, which is the chosen infrastructure
- Handling of Operative semantics like paging or lock parameters (as described in [3])
- Currently, when a data is retrieved which happens to have a *null* value, the next time it is requested it is not recognized as already available, but it is retrieved again
- Capability to select invoked services in terms of the gateway, device or bundle they are related to
- The algorithm the DMI uses to automatically select the best BPEL implementation to be called in order to retrieve a given set of data may be improved, e.g. by checking the currently available data and using better selection mechanisms
- Better handling of XML Schema substitution groups
- Handle XML Schema redefinitions, to ease customization of data definitions
- Allow reusable data mappings which are executed dynamically depending on the actual run time data type, instead of forcing all data mappings to be specified in each BPEL process
- Automatic deployment of BPEL processes from the Model into the engine (currently, all BPEL processes have to be manally deployed into the engine before the DMI tries to invoke them)
- It would be nice to have tools for automatically generating Domain-specific Java classes as described above (currently they are written by hand)

6. The Network Intelligence Manager (NIM) and its portlet

This Manager is the owner of the catalog of Gateways and provides the following interfaces (SOAP bindings):

- *GatewayCatalog* interface: allows to manage the information about existing Gateways, offering operations to list, add, update and delete gateways. It also offers a specific operation aimed to be used by gateways themselves: *update IP address*, so that a gateway with a variable IP address may easily communicate its new value to the server[3].

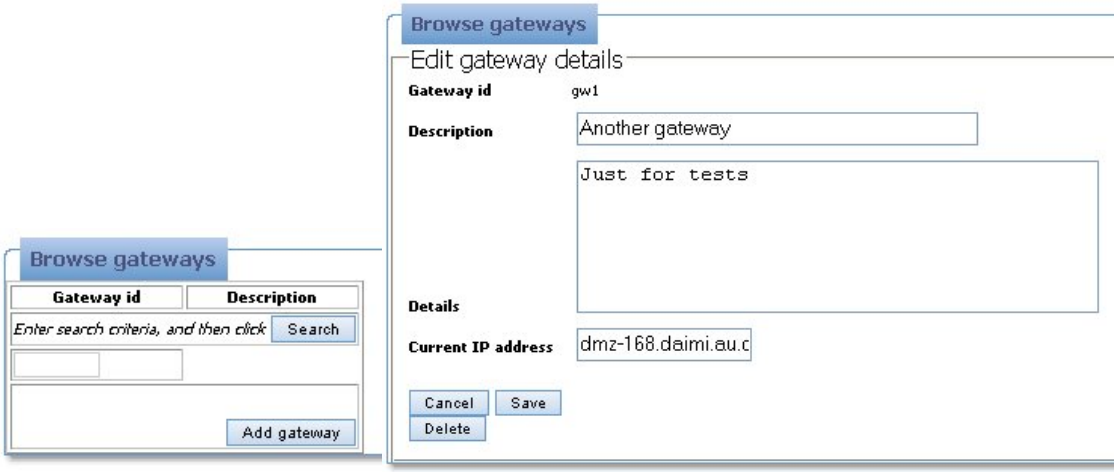
  Whenever the IP address of a gateway is changed, all of the web services that are registered in the Tamino UDDI server associated to that gateway are updated so that its URLs point to the new host address. After that, any eu-DOMAIN component wanting to contact the gateway may find the right address in the UDDI.

- *ServiceRegistry* interface: it offers one operation intended to be used by gateways to register a WSDL into the UDDI registry, e.g. when a new bundle is activated inside that gateway.

The Gateway catalog is stored in the Tamino Data Server. It also must be noted that the *update IP address* capability bypasses the UDDI APIs and directly accesses the internal Tamino database of the Tamino UDDI server (see section 8.1), so that the update of the service URLs is more efficient. This means that this feature of the NIM is tied to the concrete version of the Tamino UDDI server currently being used, not being possible to use it without modification on any other UDDI server.

Currently, this Manager has not the feature of allowing to extend the XML documents it handles. For example, the *GatewayData* structure which holds the information about a gateway may not be extended at a customer site to include new information, as described in section 4.

Managers have no Domain User interface, only SOAP interfaces, and thus they may not be directly used by Domain Users. But in many cases they offer useful features that Domain Users could use, and for this they need of a portlet to be run in the Interaction Server, as described in [4] . This is the case of the NIM, and thus associated to it there exists the NIM portlet, a JSR 168 [8] compliant portlet that allows Domain Users to manage the Gateway catalog: browse, add, modify and delete Gateways.

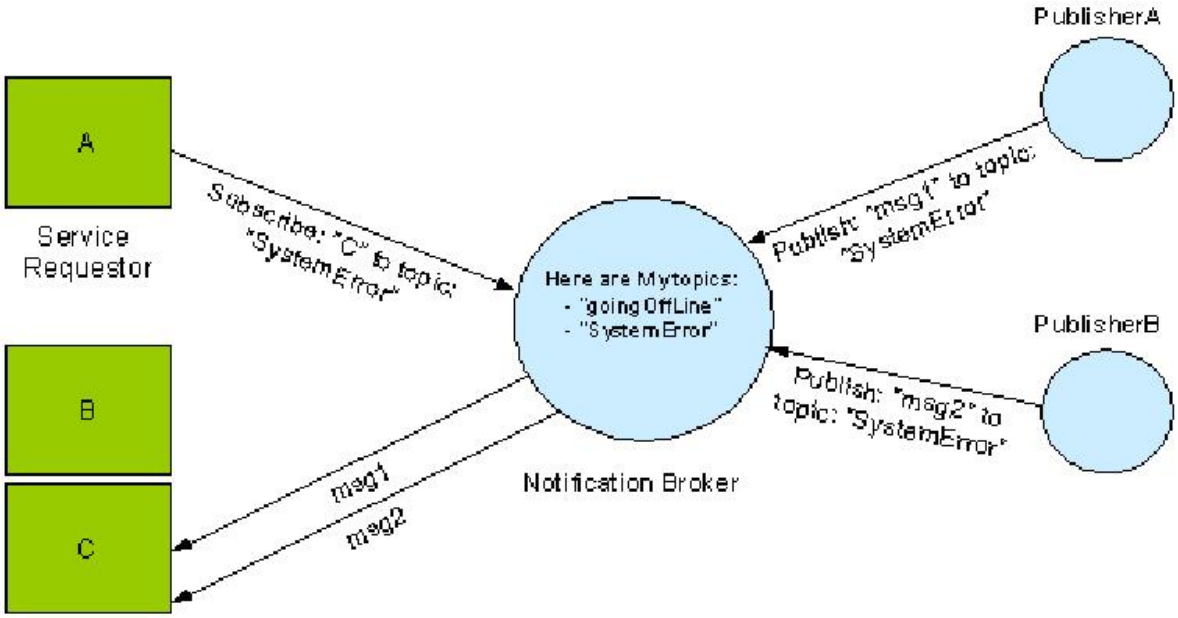


[3] When connected to the Internet by means like ADSL, GPRS or UMTS, gateways may not have fixed IP addresses, and thus eu-DOMAIN must keep track of the actual address of each gateway in order to be able to contact it. This is one of the purposes of the NIM.

7. The Event Manager

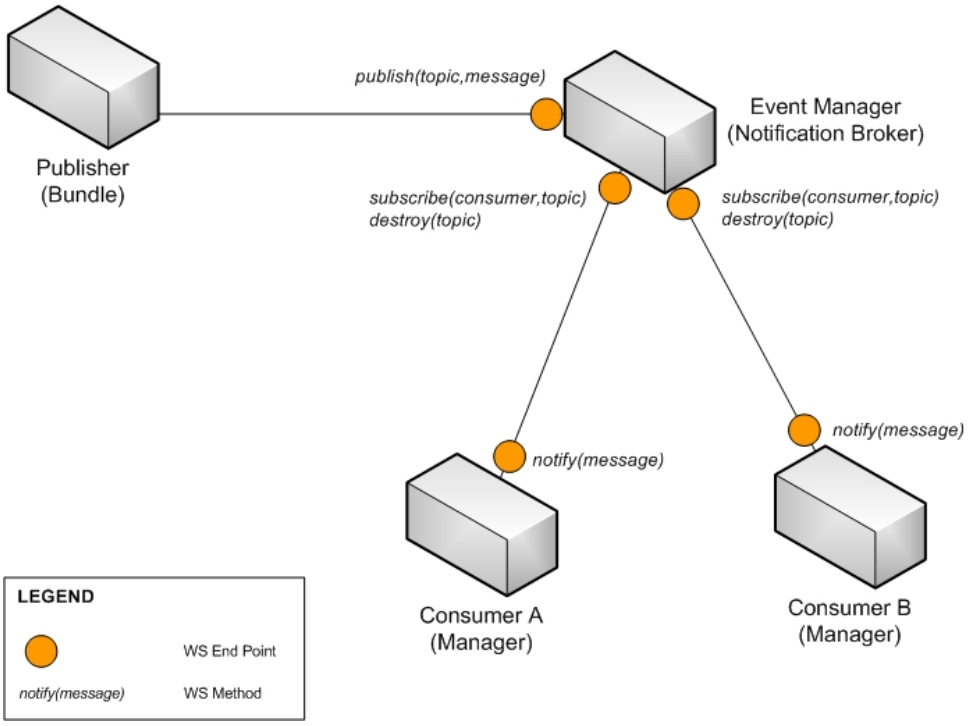
As described in [4] , will allow eu-DOMAIN components to communicate by means of a publish/subscribe mechanism fully based on SOA.

It will be based in the WS-Notification standard [17] (thus being fully compatible with the events defined in the Domain Model infrastructure defined before), and built after the open source Apache Pubscribe product. This is its general architecture:



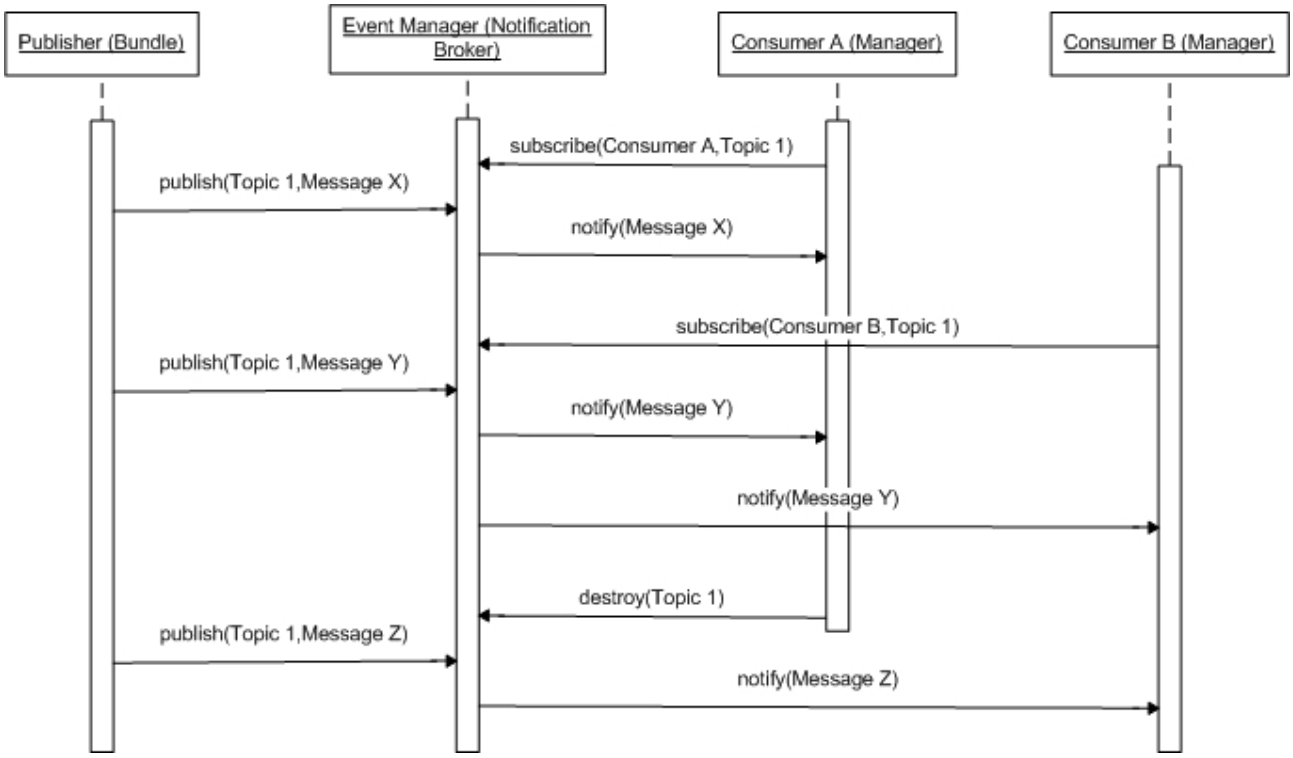
Server-side components will both publish and consume events. Client-side components will communicate with the server preferably by sending events to it, which will be published in the Event Manager and so distributed to any server-side component which has shown interest in it by means of a subscription, no matter of whether it is a conceptual or operative component.

The following image shows the architecture and services provided by the Event Manager, built upon the Pubscribe project.



7.1 Sequence Diagram

In order to clarify the normal use of the Event Manager, we provide the following sequence diagram, where a bundle (publisher) sends events to the Event Manager, which in charge of sending the notifications about the event to the managers (consumers) that are interested, let's say, subscribe, to that specific topic:



7.2 Apache Pubscribe

Apache Pubscribe [18] is a robust Java implementation of the Web Services Notification (WSN) family of specifications [17] . This specification tries to define the way Web services can interact

using the Notification pattern, which defines a way for consumers to subscribe to a producer for notifications whenever a particular event occurs.

WSN builds upon the WS-ResourceFramework family of specifications. Pubscribe leverages the Apache WSRF open source project [19] as its implementation of the WSRF foundation. At this moment, version 1.1 has been released.

8. Other components in the deliverable

8.1 The UDDI server

Most of the software of this deliverable which uses UDDI depends only on the UDDI v2 API[4], so any server complying with this specification may be used. The only exception to this is the NIM, that as described before has a dependency with the Tamino UDDI registry v2 [10] , which is the current technology used to implement this server.

In the future the software will be upgraded to use the UDDI v3 API, although there are no concrete plans for it. Software AG has already released a UDDI v3 server, although the v2 API will still be supported.

JAXR has been dismissed as an API to be used because eu-DOMAIN is not only Java (while JAXR is a Java-only standard), and because its implementations at the time[5] seem to be immature regarding UDDI support.

8.2 The WebDAV server

Currently, the XML documents defining the Domain Model are retrievable by the software described in this deliverable as long as they may be pointed by a URL, for example an http:// or file:// one. Thus, there are no dependencies of the software to any specific product used to store these documents.

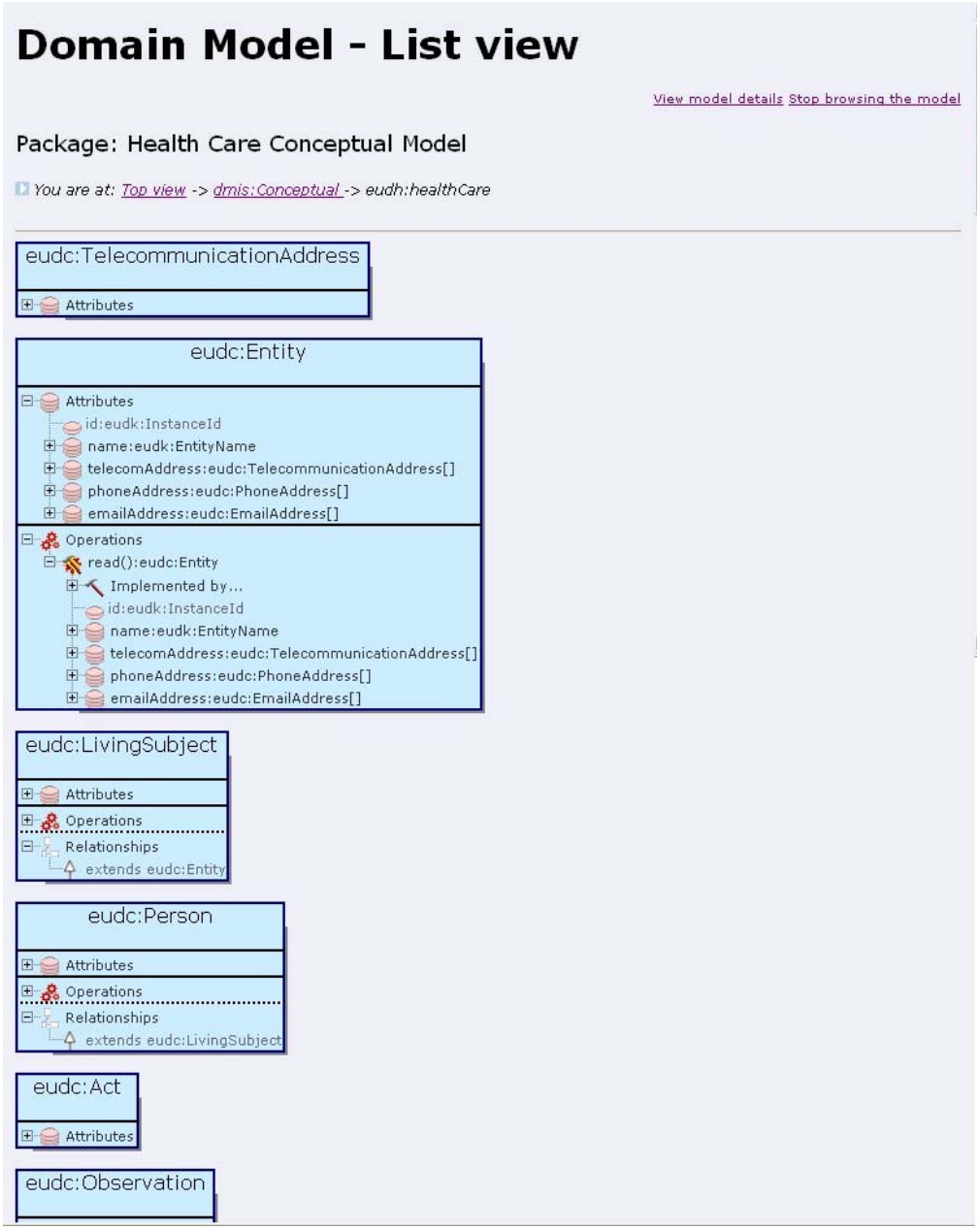
Therefore, a simple web server would suffice for providing access to them to all eu-DOMAIN components requiring it. But since these models have also to be created and maintained, a good option for storing them is to use a WebDAV server, which allows for collaborative editing.

Currently, the Tamino WebDAV server 4.2 [11] is being used, but as described above there are no dependencies at all from the software of the deliverable to this product. In a near future it will be upgraded to Tamino WebDAV server 4.4 .

[4] Access to UDDI is made through the UDDI4j library [9]
[5] The Sun Reference Implementation and Apache Scout were tested to access UDDI, with not good results

8.3 The Domain Model Browser (ex SOA Modeler)

In the original eu-DOMAIN architecture [4] , a *SOA Modeler* component was identified allowing to browse and edit the Domain Model. Besides, it was to be the basis for providing Domain Users with capabilities to select Domain Model elements. For example, when designing a Notification Manager template, the Domain User will have to select which data from the Model is to be included into it, and this selection is best done by browsing the Domain Model features and selecting from among them.

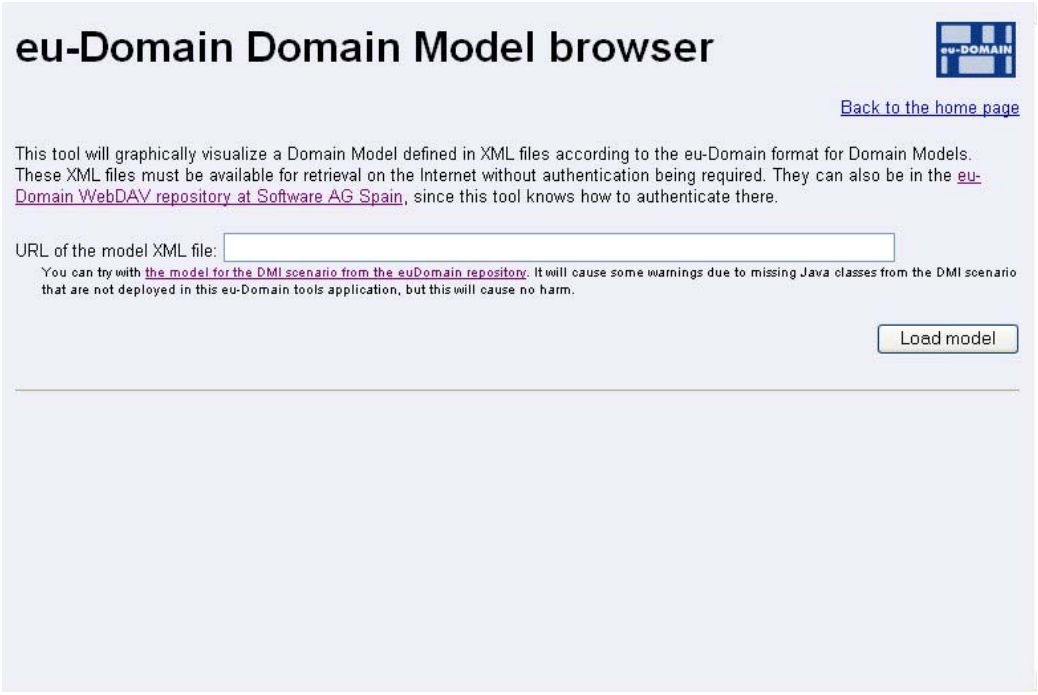


In general, the SOA Modeler is an essential component for humans, both from the Domain and IT, to visualize the Domain Model. Otherwise, they have to rely on text files or on third-party tools like [12] or [13] that, while useful to visualize the standards-based Domain Model, they show only partial views of it. Because of this an early version of it was developed soon, to help in comprehending the Domain Model concept and providing basic browsing capabilities.

However, it was agreed that for the operation of the system, the Domain Model Interpreter was critical, while the SOA Modeler was not, and thus the available resources were devoted to the (Java) Domain Model Interpreter.

Thus, the SOA Modeler is available only as that browser of the Domain Model, and indeed it is very useful for humans to visually grasp what the Domain Model contains. Thus, it has been renamed as the **Domain Model Browser**. It is a web application that renders the metamodel classes which represent the Model as UML-like class diagrams.

It is also have been made available as an independent tool at http://rtd.softwareag.es/eudTools/dmBrowser which allows to load and visualize any Domain Model from its XML definition. It is just a wrap-up of the SOA Modeler.



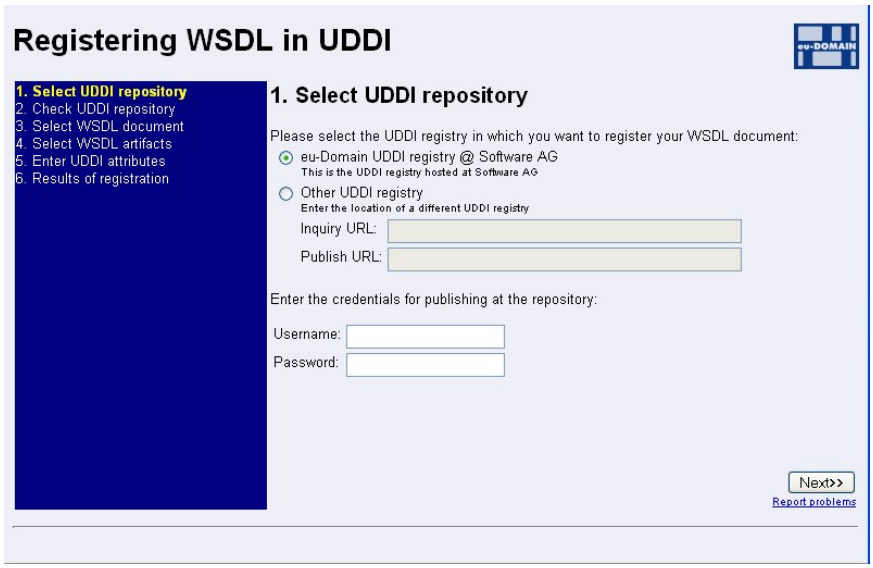
9. The SOA tools: a byproduct

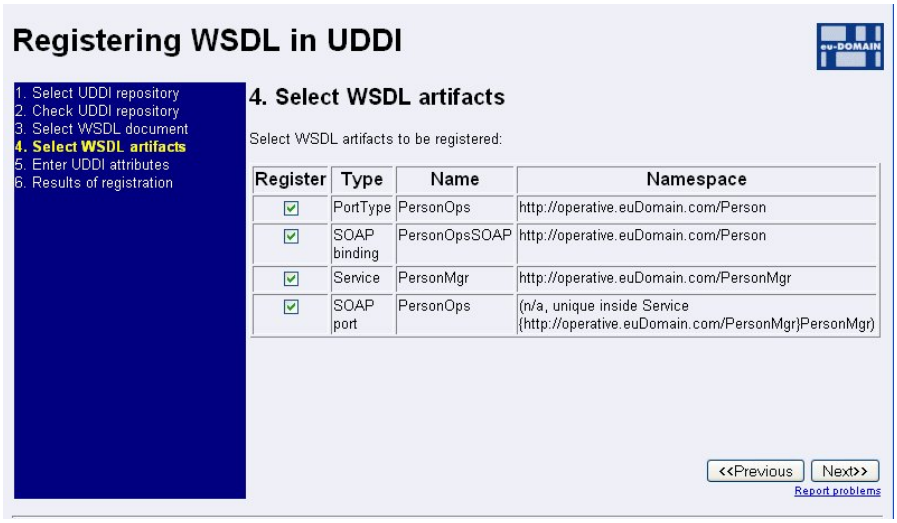
Being the Domain Model so closely related with SOA technologies, some of its components are useful not only for eu-DOMAIN, but also for many SOAs. Thus, for dissemination of the project, it was considered interesting to publicly publish over the Internet a set of “SOA tools” that could be useful for other people. These tools are available at http://rtd.softwareag.es/soaTools/, and are described now.

9.1 The UDDI registrar

As described before, the UDDI registry is a key element for the flexibility of the Operative layer of eu-DOMAIN. And of course there is a need to register services into it in order to be later found. However, being UDDI a fairly supported market standard, there were no plans to create a specific component to do, assuming that existing market tools would be available for that (e.g. see [13]).

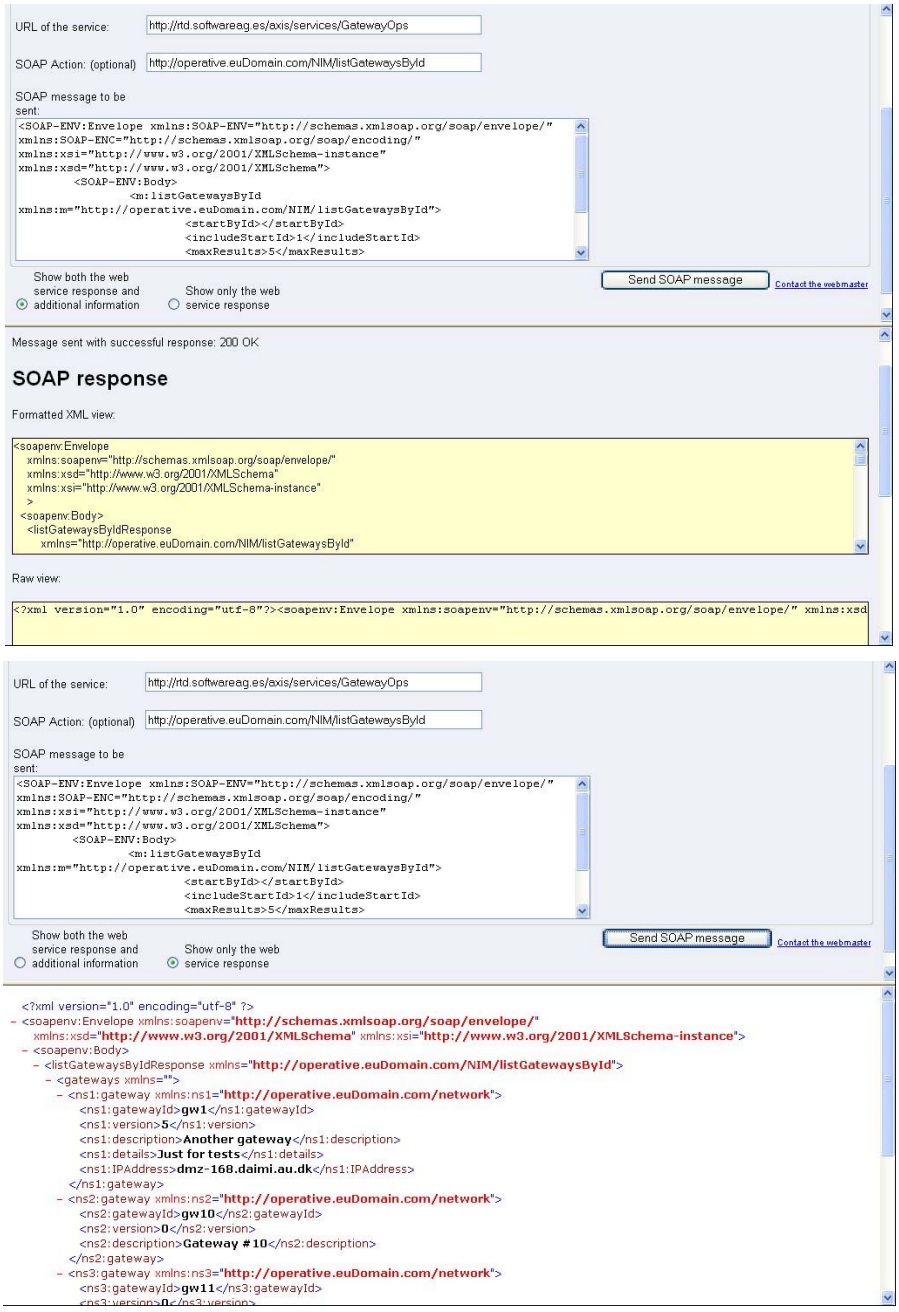
However, it was identified the need for gateway bundles to be able to perform such registration, and thus it was in place to provide capabilities to ease this task; in particular, the *ServiceRegistry* operations of the Network Intelligence Manager described before. And, in order to better build and test the logic underlying these operations, a UDDI registrar tool was also built. Thinking also in dissemination of eu-DOMAIN, it was also published over the Internet as a “SOA tool”.





9.2 The SOAP sender

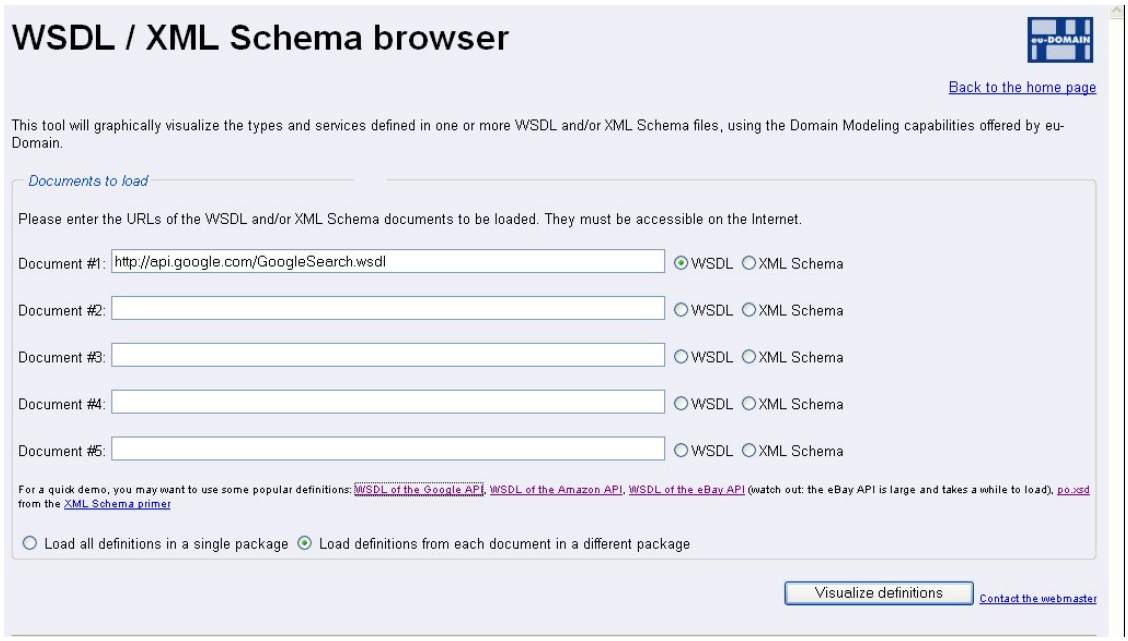
Often when one is involved in the usage or development of web services, one needs of a simple tool to send raw XML messages to arbitrary services, and to see its results. While there are many available tools for doing so in convenient ways (e.g. see [12] and [13]), having a concrete tool in eu-DOMAIN that could be used as a common reference for such task, available in every occasion and to every partner, was thought of interest. And so, in the SOA tools mentioned above, it was included such a SOAP sender.

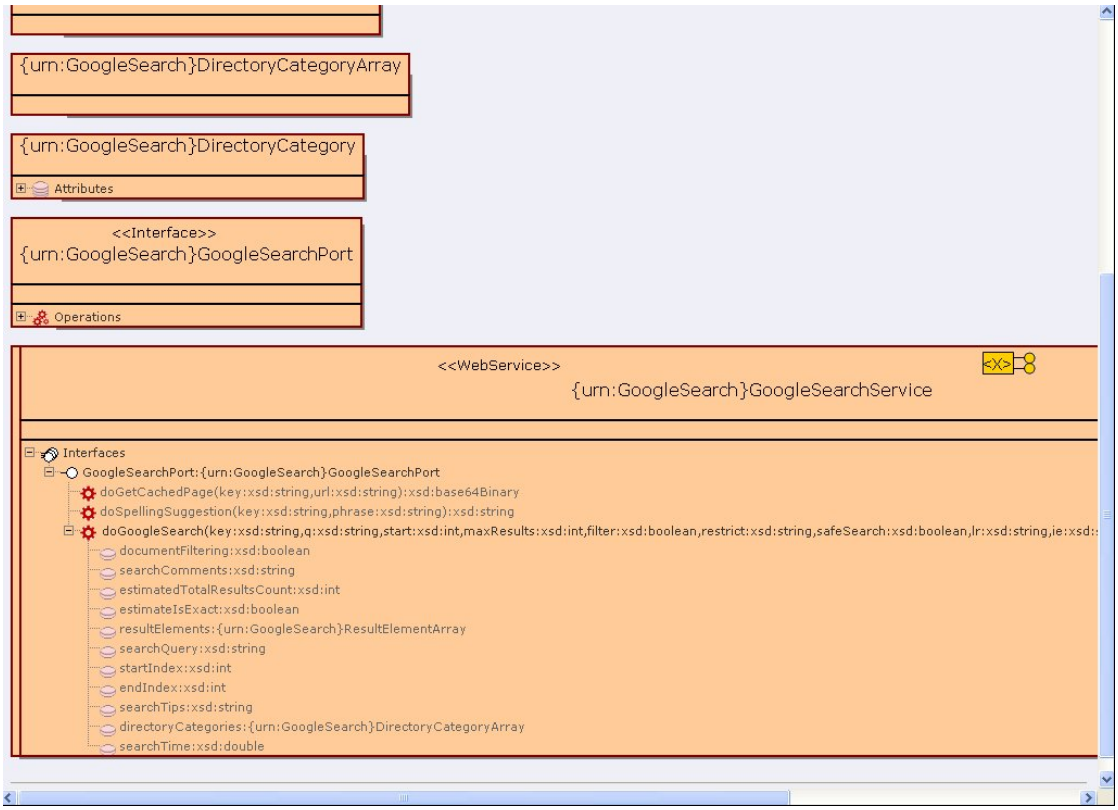


9.3 The XML browser

The Domain Model Browser described before allows browsing the Domain Model. Thanks to the fact that the very same mechanisms (i.e. XML Schema, WSDL and so) are used to represent both the Conceptual and the Operative layers, the same code is able equally to browse the Conceptual Model and the Operative Model. And, since the Operative Model is composed of the standard WSDL and

XML Schema documents used in any web services infrastructure, this browsing capability is useful to visualize them in any SOA context. And thus, again thinking in disseminating useful results of eu-DOMAIN, it was published as a SOA tool.





10. Summary of Java packages

This is a summary of the Java packages available in the deliverable, including other utility classes besides the one described until now:

Package	Description
com.euDomain.bpel	Handling of BPEL processes
com.euDomain.config	Access to simple configuration files
com.euDomain.dataserver	Access to the Data Server
com.euDomain.dmBrowser	The standalone Domain Model Browser
com.euDomain.explain	High-level logging
com.euDomain.metamodel	Classes representing the metamodel that is used to define Domain Models, and which load and give access to these Models
com.euDomain.nim	The Network Intelligence Manager
com.euDomain.nimPortlet	The NIM portlet
com.euDomain.sendSoap	The Send SOAP SOA Tool
com.euDomain.soaModeler	The SOA Modeler
com.euDomain.uddi	Access to UDDI repositories
com.euDomain.regToolWeb	The UDDI registrar SOA Tool
com.euDomain.xmlBrowser	The XML Browser SOA Tool

Package	Description
es.tid.eudomain.eventmng	Code generated with WSDL2Java for Event Manager
org.apache.ws.resource.example.eventmng	XBeans classes for Event Manager datatypes.
es.tid.eudomain.clients	Clients for Event Manager functions

The com.euDomain packages contain code which is specific of eu-DOMAIN, but other code has been reused or created which could be potentially used in other applications and thus have a different name:

Package	Description
com.softwareag.dataacc	Utilities for accessing data sources
com.softwareag.html	Utilities for handling HTML
com.softwareag.jsp	Utilities for creating JSPs
com.softwareag.lang	Core Java language utilities
com.softwareag.net	Networking utilities
com.softwareag.util	General purpose utilities
com.softwareag.view	Utilities related to MVC views in general
com.softwareag.xml	Utilities for handling XML in general
com.softwareag.xml.dom	Utilities on top of XML DOM
com.softwareag.xml.namespace	Utilities for XML namespaces
com.softwareag.xml.xmlSchema	Handling of XML Schema definitions

Also, the SOA Modeler depends on the external library *webctls.jar*, from Software AG, that provides a tree web control[6] .

And beside of these, the `test.*` packages include code used for testing. There are no formal unit test cases, however.

[6] Which in turn is based on the powerful dTree

11. Integration tests

There is a set of integration tests defined in the separate document [14] , which describes the resources needed for the tests, the tests themselves and also provides a template to leave a record of the tests performed.

Also, in [15] there is a record of the results of the tests as they were performed on January 2006.

12. References

[1] Javier Cámara, *Data Server Specification*, "D4.3 (fragment) Data Server specification_20050628.doc", June 2005

http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d75457/D4.3%20(fragment)%20Data%20Server%20specification_20050628.doc

[2] Software AG, *Tamino XML Server*, http://www.softwareag.com/Corporate/products/tamino/

[3] Javier Cámara, *Domain Model Specification*, "D4.3 (fragment) Domain Model specification_1.0.doc" (July 2005)

http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d77688/D4.3%20(fragment)%20Domain%20Model%20specification_1.0.doc

[4] eu-DOMAIN Consortium, *D3.1+4.1 Software Architecture Specification*, March 2005

http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d99333/euD033.1%2b4.1Software%20Architecture%20Specification%20V2.0%20final%20EC%20(read%20only).zip

[5] FiveSight, PXE BPEL process engine, http://www.fivesight.com/pxe.shtml

[6] ActiveBPEL LLC (ActiveEndpoints), ActiveBPEL process engine, http://www.activebpel.org/

[7] Javier Cámara, *Health Care Scenario for developing the DMI*, "Health Care Scenario for DMI_20050823 .zip"

http://rtd.softwareag.es/dmiScenario/

[8] Java Community Process, *JSR 168: Portlet specification*, http://www.jcp.org/ja/jsr/detail?id=168

[9] SourceForge, *UDDI4j library*, http://uddi4j.sourceforge.net/

[10] Software AG, *Tamino UDDI registry v2*, http://developer.softwareag.com/tamino/uddi/download.htm

[11] Software AG, *Tamino WebDAV server v4.2*, http://developer.softwareag.com/tamino/webdav/download.htm

[12] Altova, XML Spy, http://www.altova.com/products_ide.html

[13] Eclipse Project, *Eclipse Web Tools Project*, http://www.eclipse.org/webtools/

[14] Javier Cámara, *D4.3 Integration tests*, "D4.3 Integration tests.doc"

http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d106692/D4.3%20Integration%20tests.doc

[15] Javier Cámara, *Record of D4.3 Integration tests performed on January 12, 2006*, "D4.3 Integration tests @ 20060112.doc"

http://bscw.isis.alexandra.dk/bscw/bscw.cgi/d106696/D4.3%20Integration%20tests%20@%20020060112.doc

[16] Adaptive Object Models , http://www.adaptiveobjectmodel.com/

[17] For Web Services Notification (WS-Notification or WSN), see:

- OASIS, Web Services Notification Technical Committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn
- IBM, *WS-Notification specification*, http://www-128.ibm.com/developerworks/library/specification/ws-notification/

[18] Apache Software Foundation, *Apache Pubscribe project*, http://ws.apache.org/pubscribe/

[19] Apache Software Foundation , *Apache WSRF project*, http://ws.apache.org/wsrf/

[20] For Web Services Resource Framework, see:

- OASIS Web Services Resource Framework committee, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf
- Apache Software Foundation, *Web Service Resource Framework*, http://ws.apache.org/wsrf/wsrf.html