Discover, Reuse and Share Knowledge on Service Oriented Architectures

Jesús Soto Carrión[1], Lei Shu[2], Elisa Garcia Gordo[3]
(1) Head of Artificial Intelligent Department, UPSAM University, Spain
(2) Specially Assigned Researcher, Osaka University, Japan
(3) Innovation consulting manager, ADIF (the Administrator of Railway Infrastructures), Spain

***Abstract* — Current Semantic Web frameworks provide a complete infrastructure to manage ontologies schemes easing information retrieval with inference support. Ideally, the use of their frameworks should be transparent and decoupled, avoiding direct dependencies either on the application logic or on the ontology language. Besides there are different logic models used by ontology languages (OWL- Description Logic, OpenCyc-FOL, ...) and query languages (RDQL, SPARQL, OWLQL, nRQL, etc..). These facts show integration and interoperability tasks between ontologies and applications are tedious on currently systems. This research provides a general ESB service engine design based on JBI that enables ontology query and reasoning capabilities thought an Enterprise Service Bus. An early prototype that shows how works our research ideas has been developed.**

***Keywords* — Service Oriented Architectures, Distributed Computing, Semantic**

I. INTRODUCTION

Recent advances in distributed computing have given rise a new philosophy of iteration between software components, called SOA. This new software architecture allows software components developed with different technologies can be "plug in" to an Enterprise Service Bus (ESB), that enable the interoperability scenario. Any component interface is described using WSDL (a open standard language using to publish the functionality provided by a service), in this way any software component can understand the operations provided by other components and establish the necessary communications to obtain a specific goal.

This paper shows the impact and utility provided by semantic web technologies "plug in" on ESB. Currently the main ESB manufactures (Oracle, IBM, BEA or Sun) lack of semantic web connectors (Chappell 2004), (Rademakers y Dirksen 2008) this fact forces to build components using a particular semantic web framework with following associated problems (Jesús Soto-Carrión 2008):

- Hinder development tasks: there not exist a common ontology access provider such as ADO or JDBC on data access, each semantic web framework (Jena, Protégé-OWL, Sesame or Redland) provided a specific application programming interface. Besides each framework has been developed with an specific programming language, this fact, joined to previous explained, causes an strongly dependency between application logic and semantic web framework.
- Coupled applications: common semantic web functionality implemented into different components.

On the other hand, component communication are exchanged messages that contain data, usually these data follow a fix structure (schema) without using flexible knowledge expressions provided by the semantic web emerging technology. Knowledge bases formalized with a sound logic model such as OpenCyc[12] or ontologies written in OWL-DL[1], should enhanced the interoperability scenarios between "plugged" components inside an ESB providing a rich semantic knowledge and inference operations.

The problems enunciated above broken the loose-coupling principle of service design[7]. For that reason this research has been focused on services interoperability using a general ontology reasoning connector, that provides a normalize interface to semantic functionality inside an enterprise service bus. A prototype that shows the semantic connector benefits has been developed. The functionality implemented using OpenESB technology to be able to carry out a semantic search on Google maps service using KML3 and a specific ontology to allow semantic annotations. An example of these type of search should be: "retrieve all religious building".

This paper has the following structure: firstly described a brief introduction related to SOA concepts, secondly presents the wide variety of query and knowledge representation semantic web languages, thirdly currently shortcomings in semantic web knowledge interoperability are exposed, fourthly the solution is exposed using the emerging ESB technologies to describe the general ontology reasoning connector, following presents the GORCON prototype, finally conclusions drawn for this work are explained.

II. SERVICE ORIENTED ARCHITECTURE

SOA is a form of technology architecture that adheres to the principles of service-orientation[10], it is an evolution of past platforms preserving successful characteristics of traditional distributed architectures, and bringing with it the interoperability among services that uses different technologies including legacy applications, databases and another types of backend systems. The main features provided by this

DOI: 10.9781/ijimai.2011.141

architecture are:

- Enterprise Application integration: enable the interoperability between new applications and legacy systems, neither risks and collateral effects.
- Loused-coupled architecture: based on services that can perform a delimited task, dependencies between services are modeled on a high level layer (choreography and orchestration).
- Business modeling: the business activities performed by a company can be modeled with a business language [18] that uses real human business language terms.
- Distributed technologies: necessaries to interconnect all different types of services. SOA deploy specifications SCA/SDO[2] and JBI[17] uses open standards in order to enable an interoperability scenario between all different services (DCOM, CORBA, Web Services, etc...).
- Abstraction: SOA abstracts programming language of services. SOA uses languages based on open standards (WSDL, SOAP, BPEL, ...) .

The SOA layers are showed in the figure 1, following are explained from low-level to higher-level:

- Low-Level services: this layer contains all services that perform delimited tasks. These services can be implemented with different languages and interact with information systems such as databases, legacy systems or embedded systems (sonar, radar, and etcetera).
- Middleware services: intermediate layer that enclosed all higher level services. These services uses low-level services in order to perform a specific task, i.e. obtain the best service provider (relative time, cost or effort).
- Business process: they are the more relevant entities inside SOA architecture. These entities work as mediators, they are invoked from an external request (can be origin from presentation layer) or an internal event. They are defined by orchestration and choreography languages[11][3].
- Presentation: represents the visual interfaces of one application or external systems that can invoke the business process to execute a business task.
- Security: vertical layer that contains all security technology artifacts used across all layers. SOA establish security service communication using contract policies [14], besides uses open standards to use a global identifier among different systems[5].
- Government: enclosed all mechanisms that establish a sound structure for decision making and planning. This vertical layer is focused on lifecycle services and optimize business process, analyzing how work SOA applications that uses company politics, procedures and standards (Brown et al. 2009).

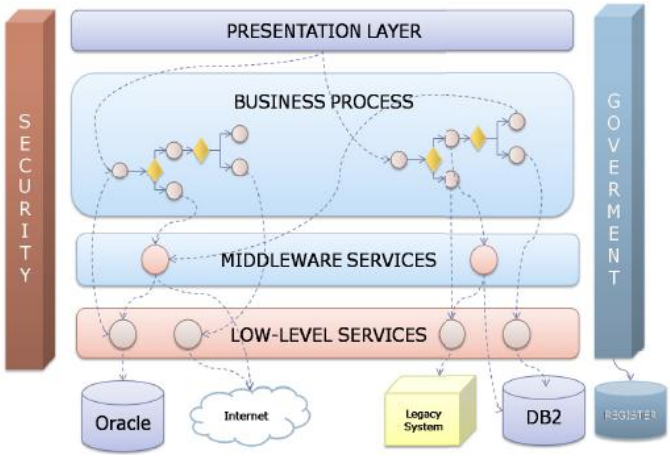

Figure. 1 SOA Layers

A. SOA SERVICE – The Basic UNIT

A service is not only a Web Service, commonly is usual confuse the concept with the technology. In SOA a Service can be developed with different technologies, the interfaces and security policies are described using a neutral open language. Thus the operations provided by a CORBA Servant or a DCOM object can be described in WSDL Language (instead of IDL or MIDL respectively). SOA provides the mechanisms that enable an interoperability scenario between services implemented with different distributed technologies (CORBA, DCOM, JMS,...), due to the use of open languages that facilitate understand the operations.

The interoperability concepts described by SOA architecture require of a robust design principles. There are several studies about the principles of service design (Oracle, IBM..) , mainly all converge in following set of principles annunciating by Thomas Erl[7]:

- Reusable: any service must be designed and developed keeping in mind reuse its operations in a application, company application domain or even for massive use in a public domain.
- Communication based on formal contract : services must provided a formal contract in which contained the narre of the service, access way, the operations implemented including in/out parameters description.
- Loose-coupling: services must be autonomous (such as LEGO puzzle piece), therefore may designed without relationship dependencies.
- Abstraction: services must hide logic and implementation issues from the outside world.
- Composition: any service must be designed in order to be used in higher-level services building.
- Stateless: a service implementation must not manage and store information about state.
- Discover ability: services may be found and assessed by some discover mechanism.

B. DEPLOYING SOA

A Service Oriented Architecture needs an infrastructure to

deploy services, process and applications that interoperate between them with different protocols and data schemes. The software infrastructure that supports SOA is called Enterprise Service Bus (ESB)[6].

1) ENTERPRISE SERVICE BUS (ESB)

An ESB provides a software infrastructure necessary to deploy SOA architecture. Among features provided by the most important suppliers (IBM, BEA, Oracle, Service Mix) worth mentioning (figure 2):

- Connectivity between any type of services: there are multitude of service technology that can be used inside a SOA architecture, ie. DCOM, CORBA, EJB, LDAP Servers, FTP, databases, JMS, MSMQ, SAP, CICS, among others.
- Neutral language: used into ESB to describe operations and interconnect services with a specific message exchange protocol (MEP). Any message transmitted inside the ESB can be enrouted.
- Data transformation mechanism: executed when two services, that uses different data schemes, needs translate data in order to establish a communication.
- BPM engine: interprets a business process language, executing actions following the flow defined, invoking services and receiving external request and messages.
- Security services: uses to provide a security layer to protect communications.
- Administration components: enable the components management installed on ESB, common operations that control the component lifecycle are "install", "uninstall", "stop" and "resume".

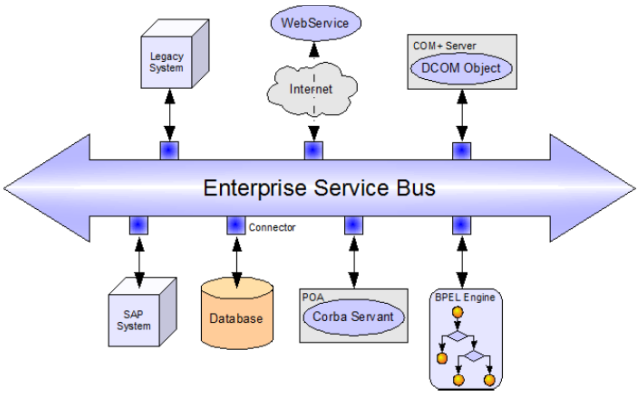

Figure. 2. Components "plugged" - Enterprise Services Bus

The internal architecture can be implemented with two principal approach, Service Component Architecture (SCA) & Service Data Objects (SDO) 4, and Java Business integration [17].

2) SCA/SDO

Service Component Architecture (SCA) is a set of specifications that describes an application building model on a service oriented architecture. SCA specifications are focused on component assembly, binding and implementation issues. The component is the basic piece that exposes a group of services using WSDL language. The assembly features provides the mechanism to build composite components describing the relationship structure with an XML languages. Following the principles of SOA, components can be implemented in different languages, for that reason its necessary specify the binding type (jms, soap, etc...).

The messages transmitted between components contain data necessary to execute the operations described on a service interface. Service Data Objects are a set of specifications (complementary to SCA specifications) that describes an simplify data model and an uniform access to heterogeneous data sets. SDO specifications are based on a disconnected data access model, is an alternative to DOM model since allow saving memory. SCA / SDO implementation examples are HydraSCA (Rogue Wave Software), IBM WebSphere (feature pack for SOA), BEA SCA for WebLogic, Oracle SOA/EDA and Active Matrix (TIBCO).

3) JBI

Java Business Integration specification[17] defines mediation architecture between heterogeneous services. The structure of JBI is composed of three components (see figure 3): Component Framework, Normalized Message Router (NMR) and Component Management:

Component Framework: describes all issues related to ESB components. JBI specification distinguishes two components types: "Service Engine" and "Binding Component". Service Engine (SE) components are internal services charge of main ESB execution functionalities, such as BPEL interpreter or data translation and transformation services. Binding Components (BC) enable service deploy over a SOA architecture. The internal design allow "plug in" and "unplug" components on an ESB (like a USB device). These features provides a flexible way to establish an enterprise application integration.

NMR provides a normalized message interchange mechanism between ESB "plugged" components. Each service (associate with a SE or BC component) exposes its interface operations using a WSDL descriptor. The operations described on WSDL interface establish the contract relationship with consumers, necessary on SOA architectures to integrate different components "plugged" on an ESB. Each normalized message routed into ESB contains metadata, payload (based on WSDL message structure) and attachments. These messages are translated from a specific protocol to normalized structured (and vice versa) by binding components, and enrouted by means of NMR from start point to end point using one of message exchange patterns (in-only, robust in-only, in-out or in optional-out).

Component 1→BC Start point→
NMR→BC End Point→ Component 2

Component Management enables the component lifecycle management based on JMX. These management components

provide operations to shutdown, stop, start, resume or paused binding or service engine component execution.

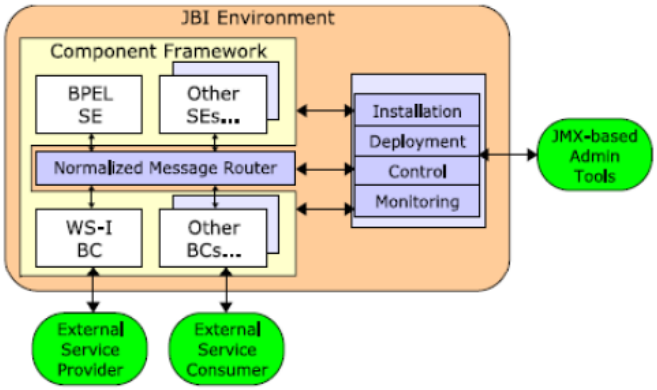

Figure. 3. JBI Components

III. SEMANTIC WEB LANGUAGES & FRAMEWORKS

Currently semantic web emerging technologies provides a wide range of frameworks that implement common functionalities, among which highlights Jena, Sesame or Redland. Each framework works with an specific set of languages (publish on standards or proprietary specifications) oriented to build and manage knowledge models. The ontology languages widely used are RDF (Resource Description Framework) and OWL (Ontology Web Language).

The general structure of a Semantic Web Framework has been represented in figure 4 (Ontology API):

- Schema API: functions set oriented both building and manipulating of ontology schema objects (class, relationships, properties and data types).
- Individual API: provides the main functionality to manage ontology individual objects.
- Inference API: include inference and reasoning mechanism which allow additional facts to be inferred from instance data and class descriptions. Besides it uses an internal or external reasoner (mainly thought DIG interface based on Description Logic Reasoners) to add check consistency, concept satisfiability, classification and realization operations.
- Query API: also influenced by Inference API, establishes the functionality to analyze and execute an specific ontology query language such as SPARQL or nRQL among others.
- Memory model: contains an ontology model on memory, usually in a graph structure, to carry out ontology API operations. A memory model can be serialized into an storage device using the persistent subsystem.
- Persistent Subsystem: provides the main functionality to work with a serialize ontology model upon a database or a file in a timely and transparent fashion.

There is a framework initiative that defines a general design to manage ontologies, called Protégé. In this research, Protégé structure has been analyzed against other frameworks (Jena, Sesame and Redland) to obtain software design ideas about general ontology management and structure issues. An in-deep explanation can be found in [9],[15],[16]. Based on CLOS MOP (Common Lisp Object System - MetaObject Protocol) and the Dynamic Object Model software design pattern, Protégé provides a set of abstract class and interfaces that allows execute ontology operations on different models (OWL or RDFS).

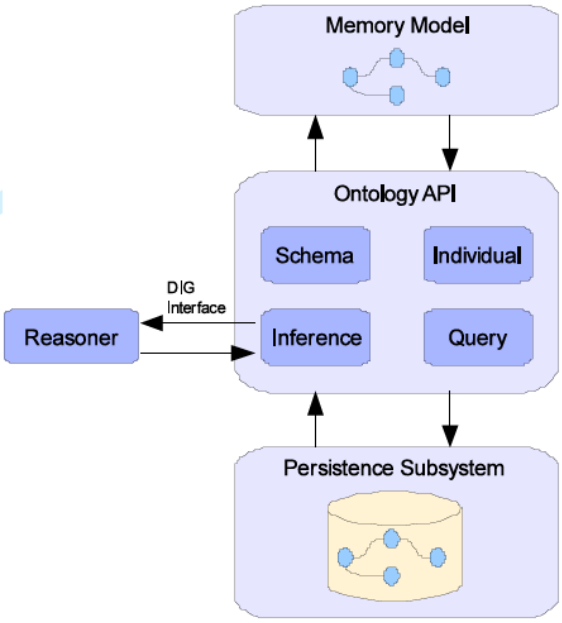

Figure. 4 SWF - general structure

IV. PROBLEM DESCRIPTION

Semantic frameworks provide a complete functionality focused to manage ontology models as had been previously mentioned, however nowadays there is no consensus aimed to resolve the strongly dependency between logic application and semantic/persistent layer. When a software architect decides change the semantic web framework underlying, just became aware that it is a tedious task because all code is strongly coupled [15].

Another motivation arises from the problem of distributed scenarios when different software components exchange information and need process common knowledge structures (called ontologies). In an Enterprise Service Bus there are binding components provided by third party manufactures that allow "plug in" different pieces of software developed with a vast variety of technology. Not all components can use semantic technologies because its underlying technology is older or not exits the way to create a binding.

Following an scenario is described in order to illustrate an example of these problems: imagine a CICS component that has been implemented using Cobol language and receives a set of messages that contains a sequence of medicine patient history based on OWL knowledge structured provided by open electronic health record ontology (OEHR)[19], COBOL language does not support a semantic library and the component needs some relevant operations such as check the consistency of data or retrieves all instances of one specific

class. In view of this situation, it is necessary developed a specific protocol between the CICS-COBOL component and one semantic framework.

This research is focused to resolve these problems, including SOA philosophy concepts, that suggest the possibility of extend a distributed scenario where several software components can take advantage of semantic functionality deployed on a service engine.

V.GENERAL ESB SERVICE ENGINE DESIGN

Using the technology offered by an Enterprise Service Bus, a general semantic service that resolves all problems enunciated in the previous section can be developed. Analyzing NMR behavior, external components should consume operations provided by a semantic web framework. This research uses only the following common operations:

- Check consistency: verify if an ontology is well defined, without inconsistencies between data types, duplicate entries, properties definitions, etcetera. Using this operation, component software can check the consistency of one or more individual received.
- Retrieve a specific individual.
- Retrieves individuals using SPARQL language.

Inspired by service engines and binding components provided by JBI developer's community and third party manufactures, a semantic service engine has been implemented. The general infrastructure that has been supported the development is showed in figure 5. The General Ontology Service Engine (GORSE) provides a general interface that supports all operations previously enunciated using Protégé OWL¬API as an underlying framework for ontology processing. GORSE can be deployed on a Enterprise Service Bus in accordance with JBI specification. Our prototype has been developed using OpenESB7.

The figure 5 shows how GORSE receives messages via Normalized Message Router. Using this system, different components implemented with different technologies can used semantic web functionality. For example, a IBM mainframe which contains a COBOL subroutine that needs process a XML file according an ontology schema instead of create a new specific program to do these semantic processing tasks. Another example can be a web service or some type of component (DCOM,CORBA, etc..) witch is deployed within an ESB but it is not possible uses a semantic web framework due to implementation constraints, or if the architecture design requirements establishes a decoupled semantic layer . GORSE has been developed following the ideas provided by OpenESB SQL Service Engine functionality[8]. The design structure is showed in figure 6.

Layer structure showed in figure 6 contains the following components explained from bottom to top:

- OpenESB: provides alI functionality related to build a SOA environment that interconnects heterogeneous services. GORSE has been built using libraries provided to create internal services.

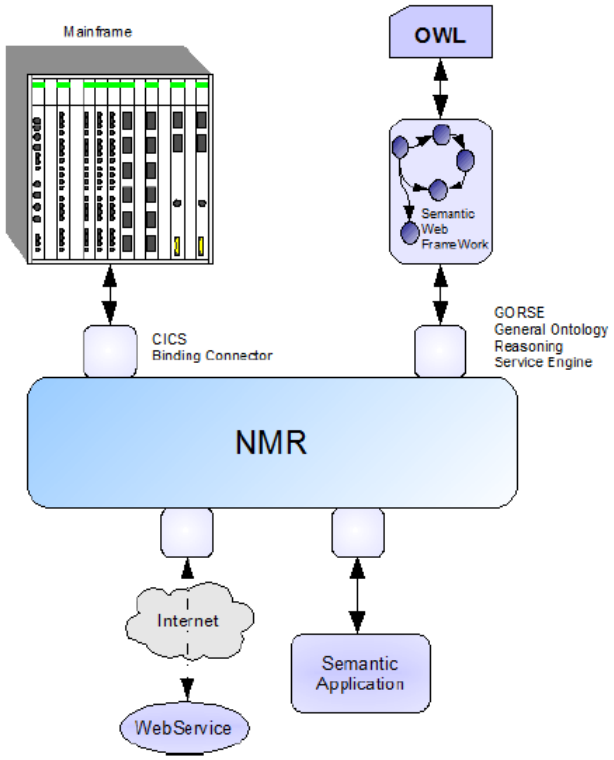

Figure. 5 Connection GORSE to NMR

- Interface Builder: used to develop a specific GORSE service which exposes an interface that contains management and query operations on a knowledge base (owl file or protégé database persistent subsystem) structured according to a
- conceptual model provided by an ontology. This component is an OpenESB - Netbeans plugin.
- Deploy services: a set of libraries that provides common functionality to deploy binding components or services engines. Plugin API uses deploy services to place and allocate resources into a SOA environment.
- Message Handler: is the highlight component focused to parse all operations received by NMR bus and launch suitable execution tasks. This component plays an important role into GORSE layer structure, uses top and bottom components functionality.
- Protege OWL-API: provides the main functionality to manage a knowledge base based structured according an ontology (classes, properties, instances and restrictions). This library contains all functions related to manage an ontology stored in a file or into a persistent subsystem.

GORSE: contains all specific tasks developed according the service interface created thought Interface Builder plugin.

Following we provided a detailed description of Interface Builder and GORSE components. All operations and messages received from NMR follow a schema provided by an auto-generated WSDL interface. The interface builder module has been developed to generate automatically the WSDL ontology

interface using specific parameters with are specified into gorse-settings.xml file, following we show a short example:

```
<connection>
  <database-url
  value=jdbc:mysql://localhost:3306/model>
  <knowledge-base value='ontomaps'/>
</connection>
```

This file contains key information about how GORSE gains access to the ontology persistent subsystem. The given example uses a short set of parameters, other ontology serialized representations can be specified, for example an OWL file instead of a relational database.

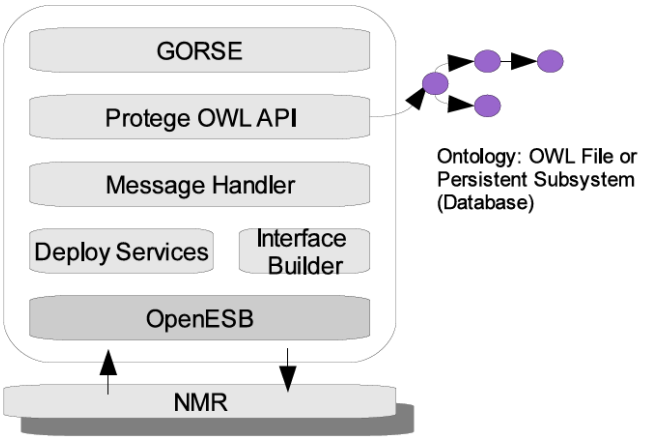

Figure. 6. GORSE Service Engine

Once the user has been configured these parameters, he can launch the build process with which will be created the WSDL interface. The interface builder model generates a WSDL interface using the following short set of rules:

- **For each OWL-CLASS**
  - **Create a XSD ComplexType - XSDOWL-CLASS.**
    - Into a Sequence (*):
      - Include an ontology ID element as xsd:anytype.
      - Mapping OWL Datatype properties - XSD 1 elements
      - Mapping OWL ObjectProperties - XSD ComplexTypes
        - Include references.
  - **Create Add-Operation**
    AddOnto[CLASS]lndividual and :
    - Input Message: InputMsg individual ns:XSDOWL-CLASS
    - Output Message - ResponseOperationMsg: resultcode xsd:int
    - Create Remove Operation
  - **RemoveOntolndividual[CLASS]:**
    - One-Way message: IDMsg
      - individuallD xsd:anyURl
    - Create Find Operation - Search[CLASS]
    - Input message - Find:
      - inputdata ns:XSDOWL-CLASS
    - Output message - FindResultsMsg:
      - result ns:LIST-XSDOWL-CLASS
  - **Add SPARQL Query operation:**
    - Input Message:
      - query xsd:string
    - Output Message:
      - LIST-XSDOWL-[CLASS]
  - **Create CheckConsistency Operation**
    - Input message - InputCheckConsistencyMsg:
      - rawXMLdata xsd:string
    - Output message
      - ResultCheckConsistencyMsg:
        - Resultxsd:Boolean

The above algorithm provides ontology control and management common operations inside ESB infrastructures. The service engine which implements WSDL interface is composed of different classes (ref), as we showed in the figure 7 "ProviderSEMessageHandler" is the mainly class focused to process all messages received from NMR message bus. This class inherits of "AbstractMessageHandler" class, a generic handler that includes relevant operations such as "send" or "processMessage". - The "processInMessageOnProvider" method declared in "ProviderSEMessageHandler" class, contains relevant code necessary to process all messages received from NMR Bus in accordance with ontology WSDL specification interface.

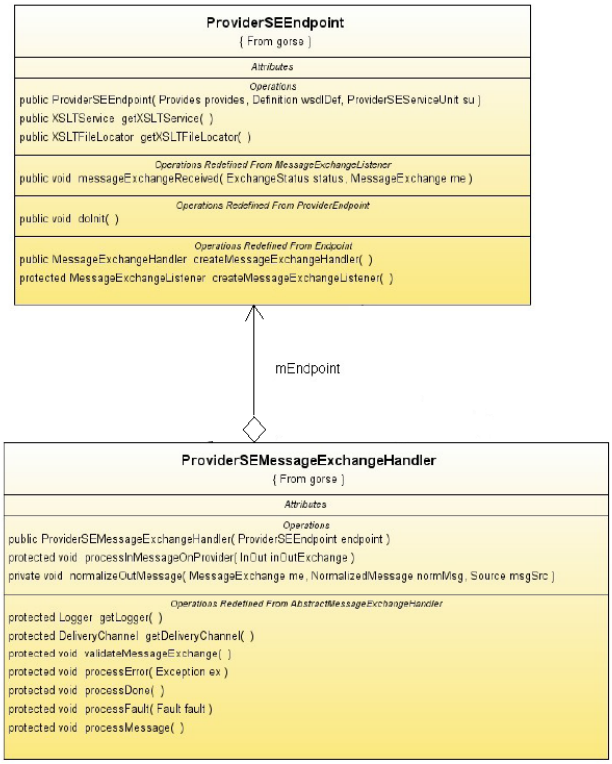

Figure. 7 Main Class Relationships

VI. PROTOTYPE

In order to illustrate how GORSE works, we have been developed an early prototype with uses and interconnect three services: google maps, a GIS coordination service and finally an ontology inside GlassFish OpenESB.

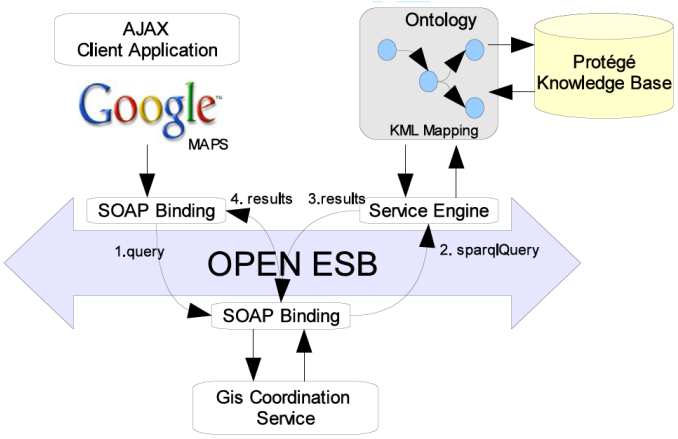

Figure. 8: Sequence of messages

Figure 8 depicts the SOA environment created using OpenESB to execute our test cases. All messages interchanged between different services have been labeled with a sequence number. The ontology has been plugged to OpenESB thought GORSE service. Following the sequence, firstly GIS coordination service receives a client request eg. searching buildings and places thought a web page, examples of this request are "religious buildings" or "has¬picture('Las Meninas')". Secondly, this service uses a SOAP proxy class created through GORSE WSDL interface to launch a request with an SPARQL input message enclosed. Thirdly GORSE service returns all results following a XSD schema. Finally GIS Coordination service decoupled knowledge information and KML data to merge into a Google Maps [13]. As an example, the following query has been executed:

```
author:DiegoVelázquez→ dc:creator[Oil-onCanvas]
```

GIS coordination service translates the previous query to SPARQL language as follows:

```
PREFIX
  ontoK:
  http://www.ijimai.org/2008/OntoKnowledgeBase.owl
SELECT ?resource ?coordinates WHERE{
    ?picture rdf:type ontoK:Oil-OnCanvas .
    ?picture dc:creator ontoK:DiegoVelazquez
    ?track ontoK:uri ?resource .
    ?track ontoK:coords ?coords .
```

"OntoK" prefix linking a limited ontology which have defined classes, properties and individuals in order to execute necessary case tests. This ontology is based on ATT Thesaurus and KML Google Schema, first used on historical-art scenarios and secondly necessary to work together with Google Maps. Our prototype works with a small knowledge base structured according to the ontology aforementioned. Using a wizard (Interface Builder, see figure 6) built following NetBeans philosophy, we have deployed a service on OpenESB that listen and executed all actions received though NMR Bus, such as "AddAuthor", "removeAuthor", "searchAuthor", "sea rchOil-onCanvasPictures", among others class (GORSE Service Engine showed on left side of figure 8). These actions are invoked by GIS coordination service in our case. Results data structure fulfill with an autogenerated XSD schema, and are transfered to GIS Coordination Service into a SOAP message. This service decoupled KML location information

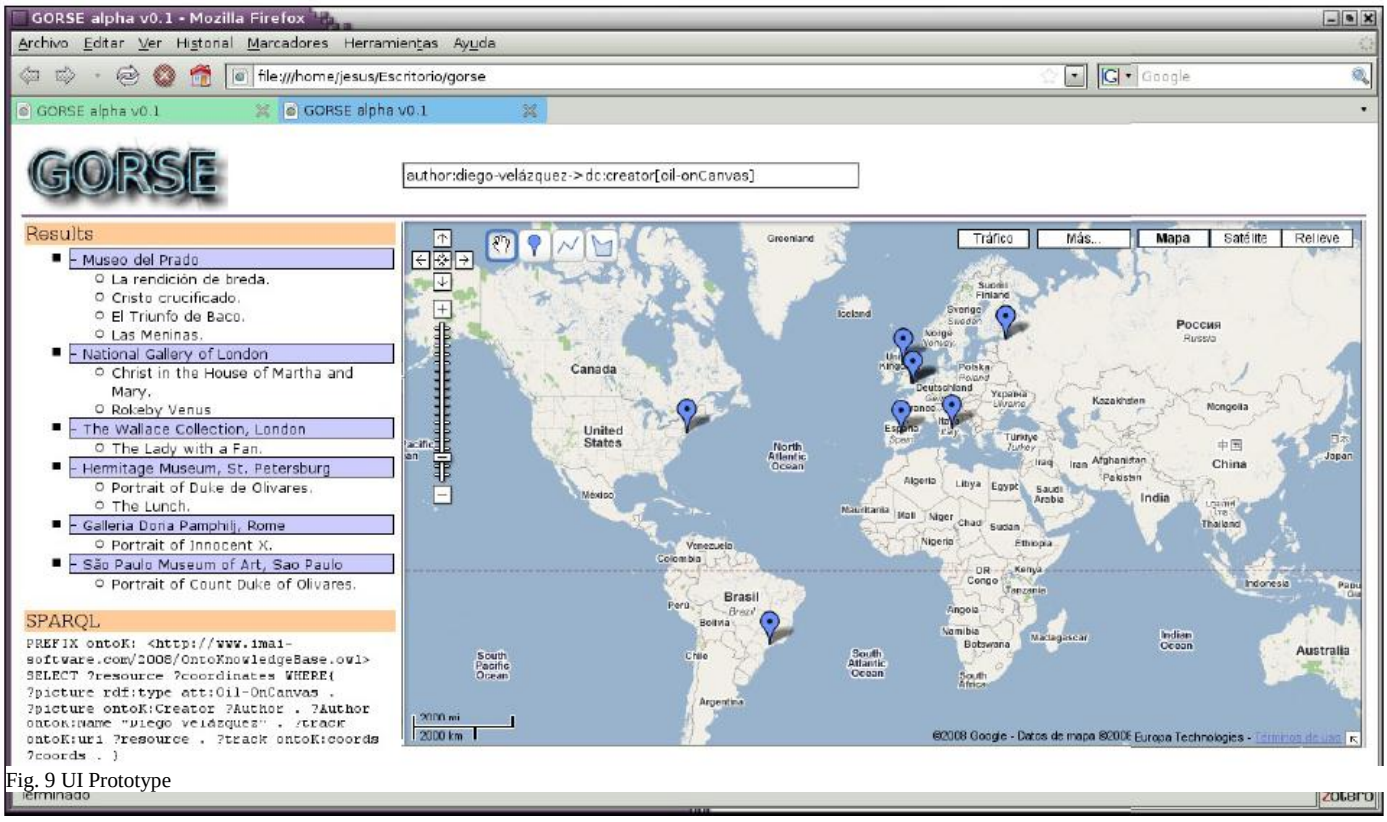

Fig. 9 UI Prototype

attached to individuals (stored into knowledge base) and individual structure to fit on Interface results. Firstly to adding a Google Maps overlay (right panel figure 9) and secondly to depict a resume of results (left panel figure 9). Knowledge structure can be used to create search filters that helps to launch more thorough searches. On right panel of prototype interface (see figure 9) a user can click on "Museo del Prado" element and application straight afterwards launch a pop-up window that shows the ontology structure. Therefore concepts like "religious-buildings" or "art-galleries" can be used to browse on knowledge base using GORSE service like common gateway of ontology query and management operations.

VII. CONCLUSIONS AND FURTHER WORK

SOA philosophy concepts provide new scenarios where interoperability of heterogeneous services is the key to reuse legacy systems. Using these powerful technologies in our research we have been suggest a new scenario where semantic web technologies play an important role. Legacy systems take advantage of all benefits provided by these technologies into a SOA environment. Further work will be focused to improve knowledge management and transport operations using semantic web services "plugged" on an ESB.

REFERENCES

[1] Baader, F., D. Calvanese, D. L McGuinness, D. Nardi, y P. F Patel-Schneider. 2002. The Description Logic Handbook. Cambridge University Press.

[2] Bieberstein, Norbert, Robert G. Laird, Keith iones, y Tilak Mitra. 2008. Executing SOA: A Practical Guide for the Service-Oriented Architect. 12 ed. IBM Press, Mayo 9.

[3] Bolie, Jeremy, Michael Cardella, Stany Blanvalet, et al. 2006. BPEL Cookbook: Best Practices for SOA-based integration and composite applications development. Packt Publishing, June 2000.

[4] Brown, William A., Robert G. Laird, Clive Gee, y Tilak Mitra. 2009. SOA Governance: Achieving and Sustaining Business and IT Agility. 12 ed. IBM, Enero 1.

[5] Cantor, S., J. Kemp, N. R. Philpott, y E. Maler. 2005. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2. 0. Committee Draft 4: 14.

[6] Chappell, David. 2004. Enterprise Service Bus. 1º ed. O'Reilly Media, Inc., Junio.

[7] Erl, Thomas. 2007. SOA principies of service design. Upper Saddle River, N.J. :: Prentice Hall,

[8] Jeff Stein.2007.open-esb: SQL Service Engine User's Guide. Septiembre. http://wiki.open-esb.java.net/Wiki.jsp?page=BuildSampleProjectSQLFileDatabaseWSDL

[9] Jesús Soto-Carrión. 2008. Semantic mechanisms oriented to flexibility of learning object metadata repositories. Universidad de Alcalá, Febrero 15.

[10] Josuttis, Nicolai. 2007. SOA in practice. Farnham: O'Reilly.

[11] Juric, Matjaz, B. 2006. Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition. 2º ed. Packt Publishing, Enero 9.

[12] Lenat, Douglas B. 1995. Cyc: A Large-Scale Investment in Knowledge Infrastructure. Communications of the ACM 38, no. 11: 33-38.

[13] Purvis, Michael, Jeffrey Sambells, y Cameron Turner. 2006. Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional. Apress, Agosto 14. Rademakers, Tijs, y Jos Dirksen. 2008. Open-Source ESBs in Action. Manning Publications, Septiembre 28.

[14] Rosenberg, J., y D. Remy. 2004. Securing Web Services with WS-Security: Demystifying WS-Security, WS-Policy, SAML, XML Signature, and XML Encryption. Pearson Higher Education.

[15] Soto-Carrión, Jesús, Oscar SanJuan Juan, y Luis Joyanes. 2006. Semantic Web Servers: A new approach to query on big datasets of metadata. WSEAS Transactions on Computers 5, no. 11: 2658-2662.

[16] Soto-Carrión, Jesús, Salvador Sánchez-Alonso, y Miguel-Ángel Sicilia. 2005. Semantic learning object repositories: flexibility and implementation issues.

[17] En .SUN JBI. 2005. Specification: JSR 208, Java Business Integration (JBI)("Specification"). Agosto 23. http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html.

[18] Weske, Mathias. 2007. Business Process Management: Concepts, Languages, Architectures. 1º ed. Springer, Noviembre 3.

[19] Peter J. Groen, Marc Wine.Virtual Medical Worlds – June 2009.Medical Semantics, Ontologies, Open Solutions and EHR Systems