Università degli Studi di Trento

Dipartimento di Ingegneria e Scienza dell'Informazione

Professional Master of Technologies for E-Government

Testing Domain Gate in the industrial context

Thesis work by: Siarhei Bykau

University Supervisor: Fabio Casati

Company Supervisor: Pietro Braghieri

Academic Year 2007/08

Contents

List of Figures:

1. Introduction

The following document intends to describe the work that has been done during the stage in DetlaDator Company. The main aim is to introduce the problem which was defined for the stage, explain its context, possible solutions and how it was solved.

The stage started on February 18 and finished on July 4. The work was divided into two parts: research and development phases. The first phase was around two month and included research on the features and abilities of OpenSPCoop and Sirv-Interop frameworks. Several communication scenarios have been developed and run under these frameworks.

The second phase was the development of the use case which was provided by DeltaDator in the middle of May. At the end of this phase the data model and prototype for the use case has been implemented and tested.

The document is organized as follows: Section 2 describes the problem of stage as a global theoretical challenge in an interoperability area; Section 3 introduces the possible architectural solutions for solving this problem. Section 4 considers three implementations of architectures – ServiceMix, OpenSPCoop and Sirv-Interop. Section 5 talks about the practical use cases which have been processed during the stage. Section 6 concludes the document.

2. Interoperability

Modern information systems can’t work in isolation and aspire to communicate with each other in order to provide their clients with a better quality of service. This relates and to public administrations: integration helps them to exchange information and thus broaden the possible service functionality. For example, medical insurance office can automatically request information about person fiscal code and doesn’t wait until person gives this information manually.

According to ISO/IEC 2382-01, Information Technology Vocabulary, Fundamental Terms, interoperability is defined as follows: "The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units".[1] We are going to consider a specific area in interoperability: how to connect public services of different administrations or even administrations of different states. It is an actual problem nowadays because there is a tendency to couple services of different countries not only in EU but in the world scale.

In June 2002, European heads of state adopted the eEurope Action Plan 2005 at the Seville summit. It calls on the European Commission “to issue an agreed interoperability framework to support the delivery of pan-European eGovernment services to citizens and enterprises”. This framework would address information content and recommend technical policies and specifications to help connect public administration information systems across the EU.

The eEurope Action Plan 2005 as well as the Decisions of the European Parliament, the Council and the Commission quoted above have adopted and promote a set of general principles which should be respected for any eGovernment services set up at a pan-European level.

The main principles of interoperability which were stated are [2]:

- **ORGANISATIONAL INTEROPERABILITY.** This aspect of interoperability is concerned with defining business goals, modeling business processes and bringing about the collaboration of administrations that wish to exchange information and may have different internal structures and processes. Moreover, organizational interoperability aims at addressing the requirements of the user community by making services available, easily identifiable, accessible and user-oriented.

- **SEMANTIC INTEROPERABILITY.** This aspect of interoperability is concerned with ensuring that the precise meaning of exchanged information is understandable by any other application that was not initially developed for this purpose. Semantic interoperability enables systems to combine received information with other information resources and to process it in a meaningful manner. Semantic interoperability is therefore aprerequisite for the front-end multilingual delivery of services to the user.
- **TECHNICAL INTEROPERABILITY.** This aspect of interoperability covers the technical issues of linking computer systems and services. It includes key aspects such as open interfaces, interconnection services, data integration and middleware, data presentation and exchange, accessibility and security services.

The main focus of this work is on technical aspects of interoperability, because semantic and organizational integration are another parts of research in interoperability and the stage task didn't include any research in it. We just refer to them in order to make comparison of several technologies in their ability to support semantic and organizational features. The next step is to consider different technological architectures of interoperability which exist nowadays. Section 3 gives the description of Java Business Integration (Section 3.1) platform and SPCoop (Section 3.2) – Italian specification for public administration interconnection.

3. Architectures

2.1. Java Business Integration

JBI defines an architecture that allows the construction of integration systems from plug-in components that interoperate through the method of mediated message exchange. The message exchange model is based on the web services description language 2.0 [3].

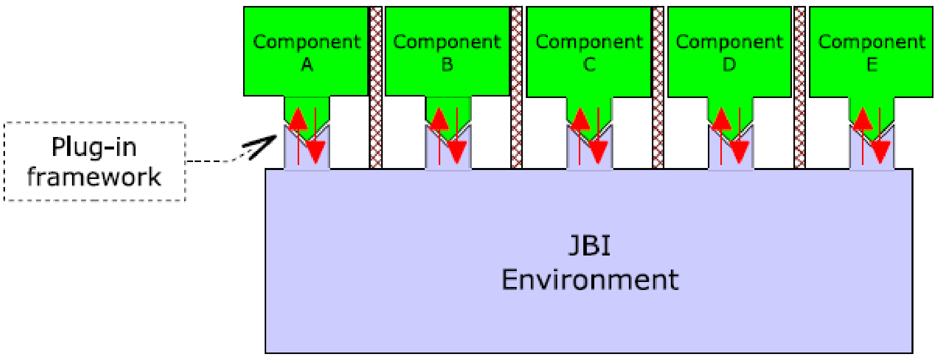


Figure 1 Plug-in construction of JBI

Figure 1 illustrates, at a high level, the JBI concept of plug-in components. JBI provides specific interfaces for use by a plug-in, while the plug-in provides specific interfaces for use by JBI. Components do not interact with each other directly. JBI provides for interoperation between plug-in components by means of message-based service invocation, described using a standard service description language. This provides a consistent model of services that the components can provide and consume.

All messages which components use to exchange information are described using Web Service Description Language version 1.1 or 2.0 [4]. WSDL provides a declarative model of message-based services on two levels:

- Abstract service model. A service is defined using an abstract messaging model, without reference to a particular protocol or wire encoding.

- Concrete (bound) model. An abstract service that is bound to a particular protocol and communications end point.

JBI uses the abstract service model as the main basis of component interactions. Components play one of two roles in such interactions:

- Service Provider. The component that performs the given service (either directly or as a proxy for an external provider).

- Service Consumer. The component that invokes a given service (either directly or as a proxy for a remote consumer).

There are several implementations which follow this standard: OpenESB, Apache ServiceMix (see Section 4.1), Mule, OW2 PEtALS.

2.2. SPCoop

In 1999 Italian Public Administration (PA) were completely isolated. Each PA specifies its own communication standards and it was a problem to involve a new PA into work. Italian government had realized this problem and in 2001 carried out the reform of Italian Constitution which attributed new possibilities for action to local authorities. Since then, the right to pass laws autonomously represented an increasingly effective means for decentralization with respect to administrative, organizational and also technical aspects. But from an ICT management point of view, this decentralization generated different points of decision, possibly leading to different ICT choices as well as different organizational processes. From the technical point of view there was a necessity to develop a common solution which should fulfill both organizational and technical needs. As a result CNIPA (Centro Nazionale per l'Informatica nella Pubblica Amministrazione) has proposed SPCoop framework [5]. SPCoop is not only a software framework, but also a technical and organizational platform whose aim is to create the conditions for a long-lived legally valid cooperation among administrations.

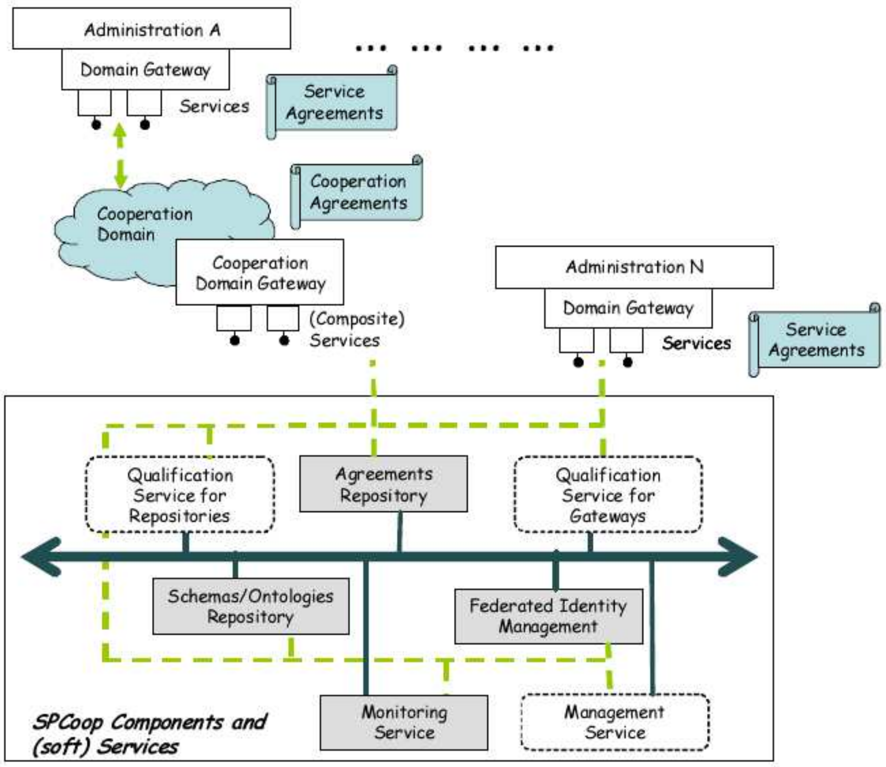


Figure 2 Components of SPCoop

The model proposed for SPCoop is based on the following principles [5]:

- The PAs cooperate through the supply and the use of application services; these services are offered by the single administration through a unique (logic) element belonging to its own information system called Domain Gateway. In this way the complete autonomy of the single administration is guaranteed, as far as it concerns the implementation and management of the provided application services, as they can be based on any application platform, being it pre-existent or new, as long as they are supplied through the Domain Gateway. The fruition of the application services is carried out through the exchange of messages, whose format is formally specified in the Italian standard referred to as e-Gov Envelop. Such a standard is basically an extension of SOAP;
- A service works on the basis of an agreement among at least two subjects (supplier and client); such agreements have a technical basis and an institutional/jurisdictional basis. These agreements should be formalized in order to support the development and the life-cycle of

services in a (semi-)automatic way. The agreement specification is called Service Agreement and is based on the XML language.

- Sets of administrations which need to cooperate in order to provide composite application services form a Cooperation Domain; the services supplied by such a domain are externally described through Service Agreements, and internally by a specification describing how the different PAs concur to compose the final service, referred to as Cooperation Agreement.

In order to support these principles SPCoop has following components:

- **Agreements Repository** is the software component used to register and to maintain the Cooperation/Service Agreements. It can be considered as the “database” of the cooperation. This component offers functionalities for the registration, the access, the update and the search of the agreements. The UDDI standard is the core of this component; however this standard does not offer all the required functionalities, therefore it has been extended.

- **Schemas/Ontologies Repository** is the software component offering functionalities to deal with the service and information semantics, in order to find out services that are more suitable to provide required functionalities. This component acts as a structure to store ontologies and conceptual schemas, offering functionalities of registration, access, update and reasoning on them.

- **Federated Identity Management** is used to authorize and control the access to application services over SPCoop; the federation is needed to reuse the already in-place identity management systems of regional and national authorities. Integration is done through specific interfaces supporting SAML v2.0.

- **Monitoring Service** is in charge of monitoring the respect, by the different services, of the Service Level Agreements (SLAs) declared in the Service Agreements. Its development is planned for the future (i.e., it has not been included in the currently active tender), as standards and technologies for the definition and the enforcement of SLAs (e.g., WSLA or WS-Agreement) are not yet considered mature.

A service agreement is a well-specified XML document that regulates the relationships of an application service between a supplier and a client in the following aspects: (i) service interface, (ii) conversations admitted by the service, (iii) access points, (v) Service Level Agreements (SLAs), (v) security characteristics and (vi) descriptions of the semantics of the service. The formal and well specified nature of the service agreement has been done to support the development and the life-cycle of services in a (semi-)automatic way. Moreover, the public nature of the service agreement makes easier the establishment of domain ontologies that allows to aggregate services with similar semantics. Finally, in the context of a set of public administrations (i.e., a Cooperation Domain), services can be composed and orchestrated, thus generating other services described in turn by service agreements.

A Service Agreement describes a 2-party collaboration/cooperation, with a subject offering a SPCoop application service and another subject using such a service. A lot of administrative processes do not concern only a single administration, but they involve different subjects. The Cooperation Domain is the formalization of the wish of different subjects to join in order to cooperate for the automation of administrative processes. Inside the Cooperation Domain, a responsible coordinator should be identified; it assures the organizational and technical effectiveness and the coordination of all involved subjects and of the set of composite application services supplied outward by the Cooperation Domain. The Cooperation Domain is seen outward as a service supplier acting like a normal domain of a single administration; the main difference is in the way its services are designed and deployed: in the Cooperation Domain they are built by composing and integrating simple services offered by the involved administrations; whereas for the single domain the supply of a service is related to applications that are fully under the responsibility of the single administration.

4. Implementations

4.1. *Servicemix*

Apache ServiceMix is an open source distributed ESB built from the ground up on the Java Business Integration (JBI) specification JSR 208 and released under the Apache license. The goal of JBI is to allow components and services to be integrated in a vendor independent way, allowing users and vendors to plug and play [6].

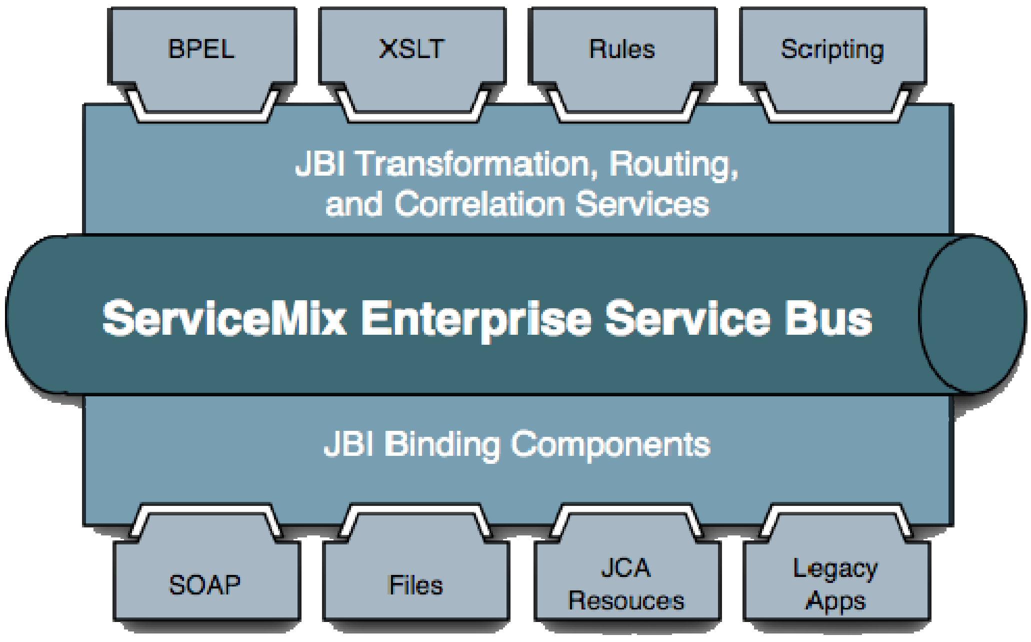

BPEL
XSLT
Rules
Scripting
JBI Transformation, Routing, and Correlation Services
ServiceMix Enterprise Service Bus
JBI Binding Components
SOAP
Files
JCA Resouces
Legacy Apps

Figure 3 JBI and ServiceMix

ServiceMix includes a complete JBI container that supports all parts of the JBI specification including:

- A normalized message service and router
- JBI management beans (MBeans)
- Ant tasks for management and installation of components
- Full support for the JBI deployment units with hot-deployment of JBI components

ServiceMix also includes the following components and services:

- Service components
    - Rules-based routing via the Drools rule engine
    - A client API for working with JBI components and services
    - An implementation of Web Services Notification
    - Business Process Execution Language (BPEL) support for Web Services BPEL via PXE (preboot execution environment)
    - Support for caching service invocations using a Map cache or a JCache provider
    - Support for the Java Connector Architecture
    - Timer integration via the Quartz library
    - Scripting support, allowing any Java Specification Request 223-compliant scripting engine to be used to create a component, perform a transformation, or act as an expression language
    - Transformation using Extensible Stylesheet Language Transformations
    - Schema validation of documents using the Java API for XML Processing 1.3 and XML Schema or RelaxNG
    - XSQL for working with SQL and XML via Oracle's XSQL library
- SOAP bindings
    - Support for a SOAP stack based on the Streaming API for XML (StAX) via ActiveSOAP
    - Support for the Java API for XML-based Web Services to make a Web services client invocation or host a Java-based Web service and expose it over multiple protocols
    - Reflection to allow plain-old Java objects (POJOs) to be deployed in ServiceMix
    - SOAP with Attachments API for Java and Apache Axis support
    - Integration with POJOs via the XFire SOAP stack
    - Integration with the Apache Web Service Invocation Framework (WSIF)
- Transport bindings
    - Email support via JavaMail

- File-based components for writing messages to files and polling directories and sending files into the JBI
- FTP support via the Jakarta Commons Net library
- HTTP support for client-side and server-side processing
- Bindings to the Jabber network via the Extensible Messaging and Presence Protocol
- JMS support via ActiveMQ
- RSS support via the Rome library for accessing and processing RSS feeds
- VFS (virtual filesystem switch) via the Jakarta Commons Net library, which provides access to file systems, jar/zip/bzip2 temporary files, World Wide Web Distributed Authoring and Versioning, Samba (Common Internet File System), HTTP, HTTPS, FTP, SFTP, and others

From this list we can see that ServiceMix supports many protocols, open specification and etc. Due to the fact that it is developed under open source license ServiceMix becomes one of the most suitable solutions for enterprise integration and correspondingly it can be applied for the case of public administration interconnection. The main disadvantage of this framework is that it doesn't support organizational and semantic interoperability. The main reason is that developers of ServiceMix didn't have the aim to support this thus ServiceMix covers only technical interoperability (see Section 2).

4.2. OpenSPCoop

OpenSPCoop [7] is an open source implementation of SPCoop specification. The project has been started in 2004 by University of Pisa and now it has active status (current version is 1.0). The main objectives of OpenSPCoop are:

- *Interoperability*. OpenSPCoop intends to allow interconnection between public administration by implementing SPCoop specification;
- *Security.* It should support mechanisms for security interconnection;
- *Simple, common way of usage.* If the way of exposing services becomes simple and unified it allows increase the number of such services, their costs;
- *Innovations*. Open source implementation is a good way to adapt new standards and specifications.

OpenSPCoop has several subprojects which have been developed in the scope of OpenSPCoop:

- The basis library org.openspcoop.egov which contains all necessary classes to work with eGov envelope [8];
- Domain Gateway for OpenSPCoop – software implementation of domain gateway of SPCoop (it bases on org.openspcoop.egov package);
- Service Register – implementation of the third part which has configuration of services and their agreements;
- Service for Event Management – software module which supports EDA (Event Driven Architecture) for OpenSPCoop. For example, it can maintain publisher-subscriber pattern.

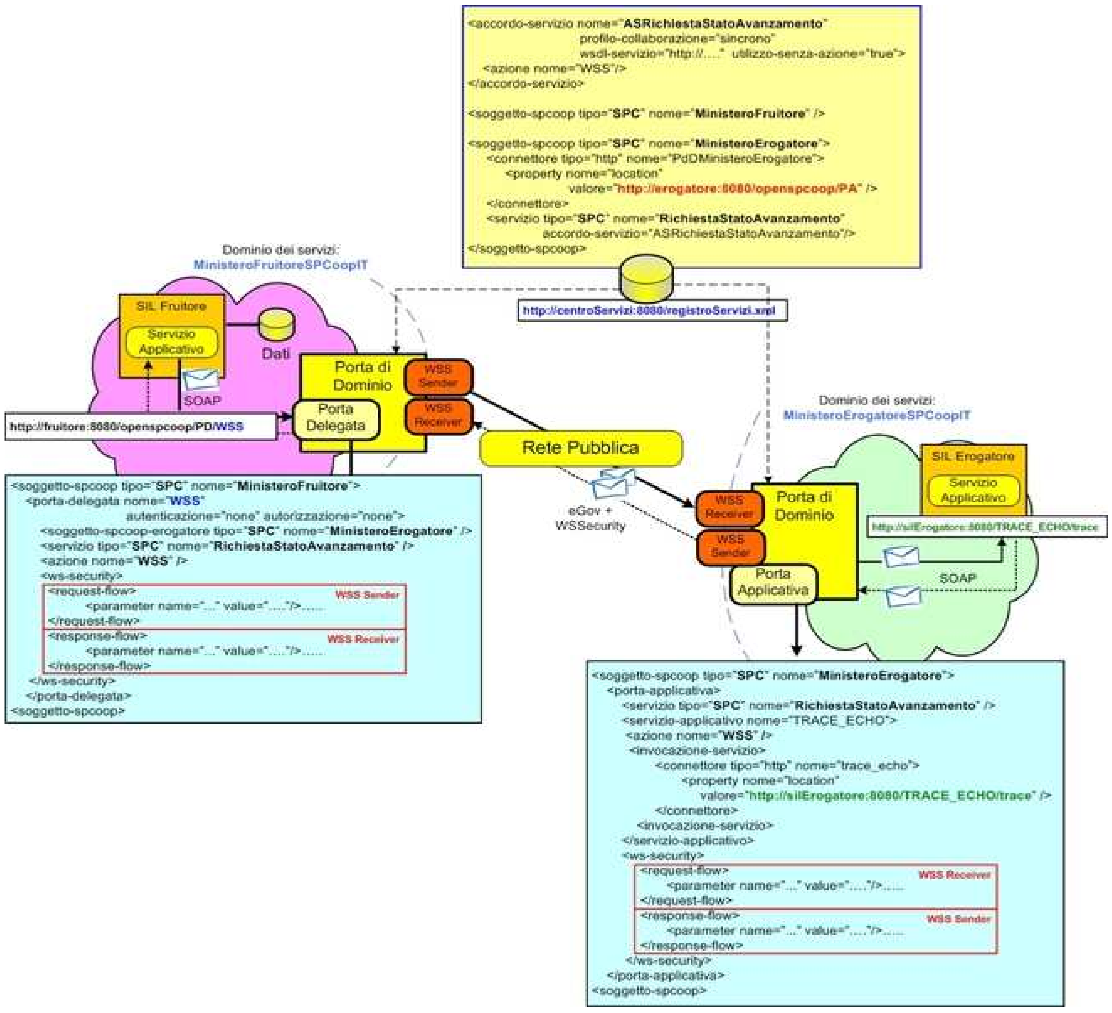
Domain Gateway in OpenSPCoop has one feature which developers was keeping in mind during it creation. They designed the generic service configuration. It means that in most cases it is not necessary to implement any wrapper class in order to expose or consume public service. All configuration can be made on run time and cover basic communication scenarios.

From the technological point of view OpenSPCoop uses following technologies:

- Application Server: Jboss/Geronimo
- SOAP Engine: Axis
- UDDI Server: Apache jUDDI
- UDDI Java API: UDDI4j
- Security: WSS4J

As an illustration of this principle the basic scenario of in-out communication was exposed through OpenSPCoop environment (see Section 5.1). The result of this experiment became a complex web services system which has OpenSPCoop as an intermediate layer for all messages.

OpenSPCoop supports WS-Security standard as security mechanism for message delivery. WS-Security describes how to attach signatures and encryption headers to SOAP messages. In addition, it describes how to attach security tokens, including binary security tokens such as X.509 certificates and Kerberos tickets, to messages. The security communication in OpenSPCoop can be illustrated Figure 4:



```
<accordo-servizio nome="ASRichiestaStatoAvanzamento"
                  profilo-collaborazione="sincrono"
                  wsdl-servizio="http://...."  utilizzo-senza-azione="true">
   <azione nome="WSS"/>
</accordo-servizio>

<soggetto-spcoop tipo="SPC" nome="MinisteroFruitore" />

<soggetto-spcoop tipo="SPC" nome="MinisteroErogatore">
    <connettore tipo="http" nome="PdDMinisteroErogatore">
        <property nome="location"
                  valore="http://erogatore:8080/openspcoop/PA" />
    </connettore>
    <servizio tipo="SPC" nome="RichiestaStatoAvanzamento"
              accordo-servizio="ASRichiestaStatoAvanzamento"/>
</soggetto-spcoop>
```

```
<soggetto-spcoop tipo="SPC" nome="MinisteroFruitore">
   <porta-delegata nome="WSS"
                   autenticazione="none" autorizzazione="none">
      <soggetto-spcoop-erogatore tipo="SPC" nome="MinisteroErogatore" />
      <servizio tipo="SPC" nome="RichiestaStatoAvanzamento" />
      <azione nome="WSS" />
      <ws-security>
       <request-flow>                                    WSS Sender
             <parameter name="..." value="...."/>......
       </request-flow>
       <response-flow>                                   WSS Receiver
             <parameter name="..." value="...."/>......
       </response-flow>
    </ws-security>
   </porta-delegata>
<soggetto-spcoop>
```

```
<soggetto-spcoop tipo="SPC" nome="MinisteroErogatore">
   <porta-applicativa>
      <servizio tipo="SPC" nome="RichiestaStatoAvanzamento" />
      <servizio-applicativo nome="TRACE_ECHO">
       <azione nome="WSS" />
       <invocazione-servizio>
            <connettore tipo="http" nome="trace_echo">
                 <property nome="location"
                     valore="http://silErogatore:8080/TRACE_ECHO/trace" />
            </connettore>
         <invocazione-servizio>
      </servizio-applicativo>
      <ws-security>
        <request-flow>                                  WSS Receiver
             <parameter name="..." value="...."/>......
        </request-flow>
        <response-flow>                                 WSS Sender
             <parameter name="..." value="...."/>......
        </response-flow>
       </ws-security>
   </porta-applicativa>
<soggetto-spcoop>
```

Figure 4 WSS in OpenSPCoop

4.3. Sirv-Interop

Sirv-Interop[9] is the second implementation which was considered during the stage. Sirv-Interop was developed before SPCoop became an official standard. On the one hand it causes the problem with interoperability between different implementation (for example with OpenSPCoop), but on the other hand Sirv-Interop team paid a lot of attention on security issues and organization model.

Sirv-Interop requires:

- Java 1.4 or later;
- Apache Tomcat 4.1.24 or later;

It structure can be illustrated on Figure 5:

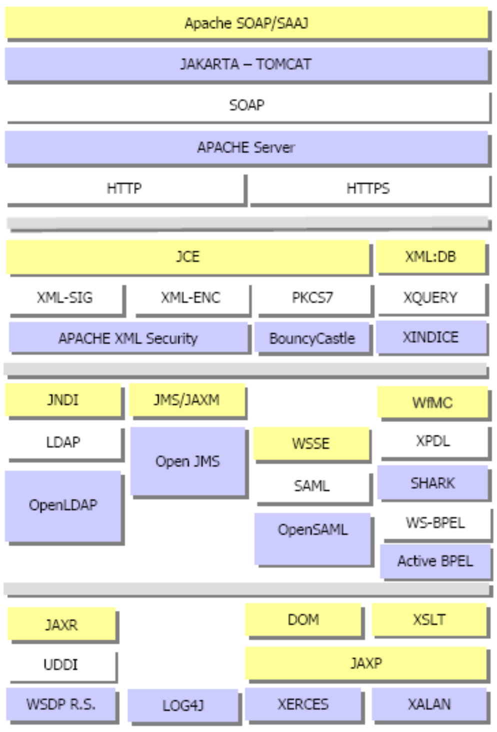


Figure 5 Sirv-Interop Technologies

Let consider the steps of configuration application and delegation domains in order to understand the features of the framework. The first thing is to write a wrapper class which plays a role of connector. For instance, we have a service, names MySevice and we'd like to expose it through application gateway.

Wrapper class should extend PortaApplicativa class and implement Wrapper interface:

```
public class MyService extends PortaApplicativa implements Wrapper
```

There is a method public `Messagio applica(Messagio messagio)` which should be redefined in order to implement functionality of message handling. Then it is necessary to

1. Get income message, parse it, get input data;
2. Define how to call MyService(what protocol to use, what are the input parameters etc);
3. Call MyService, get results;
4. Enrich output message with output data and send it.

The next step is the configuration of application gateway:

```
<?xml version="1.0" encoding="UTF-8"?>
<deployment name="ListaServizi"
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
     <service name="MyService" style="message">
          <parameter name="className"
     value="package.of.myservice.MyService"/>
     <parameter name="allowedMethods" value="esegui" />
     </service>
</deployment>
```

One of the parameters points on java class of our wrapper, thus we bind virtual Sirv-Interop service with wrapper implementation.

After these steps MyService becomes available as Sirv-Interop service and can be invoked by delegation gateway. The advantages of this approach are that we can (i) call through any protocol (SOAP, RMI …), (ii) do any transformation with messages (serialize, convert to xml …), (iii) handle exception manually. The main disadvantage is necessity to write wrapper (the cost of exposing a new service).

From security point of view Sirv-Interop framework has a very important feature – federated authentication. If we are dealing with many PA, which can produce a great amount of connection between them, we have to solve the problem of credential management. There should be a way how to store, validate, get certificates, passwords etc. Sirv-Interop uses Security Assertion Markup Language (SAML) for these purposes. The concepts of SAML standard can be illustrated by Figure 6:

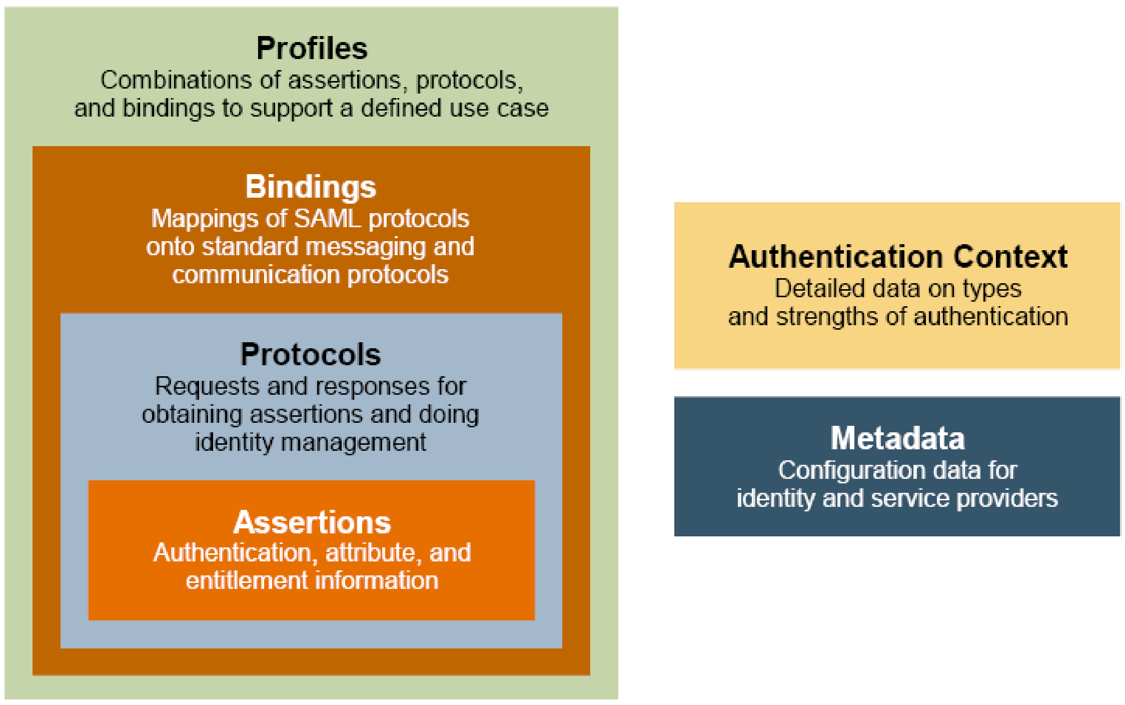

Profiles
Combinations of assertions, protocols, and bindings to support a defined use case
Bindings
Mappings of SAML protocols onto standard messaging and communication protocols
Protocols
Requests and responses for obtaining assertions and doing identity management
Assertions
Authentication, attribute, and entitlement information
Authentication Context
Detailed data on types and strengths of authentication
Metadata
Configuration data for identity and service providers

Figure 6 SAML Concepts

SAML defines XML-based assertions and protocols, bindings, and profiles. A SAML assertion contains a packet of security information. SAML assertions are usually transferred from identity providers to service providers. Assertions contain statements that service providers use to make access control decisions. A SAML protocol describes how certain SAML elements (including assertions) are packaged within SAML request and response elements, and gives the processing rules that SAML entities must follow when producing or consuming these elements. A SAML binding is a mapping of a SAML protocol message onto standard messaging formats and/or communications protocols. A SAML profile describes in detail how SAML assertions, protocols, and bindings combine to support a defined use case. The most important SAML profile is the Web Browser SSO Profile.

5. Use Cases

5.1. OpenSPCoop Prototype

In order to carry out an experiment with OpenSPCoop framework the project for Application Integration and Business Process Management course was chosen as an infrastructure for SOA. JobBank project has the structure which has enough complexity for verifying the ability of OpenSPCoop to expose generic services. It has the following architecture (Figure 7):

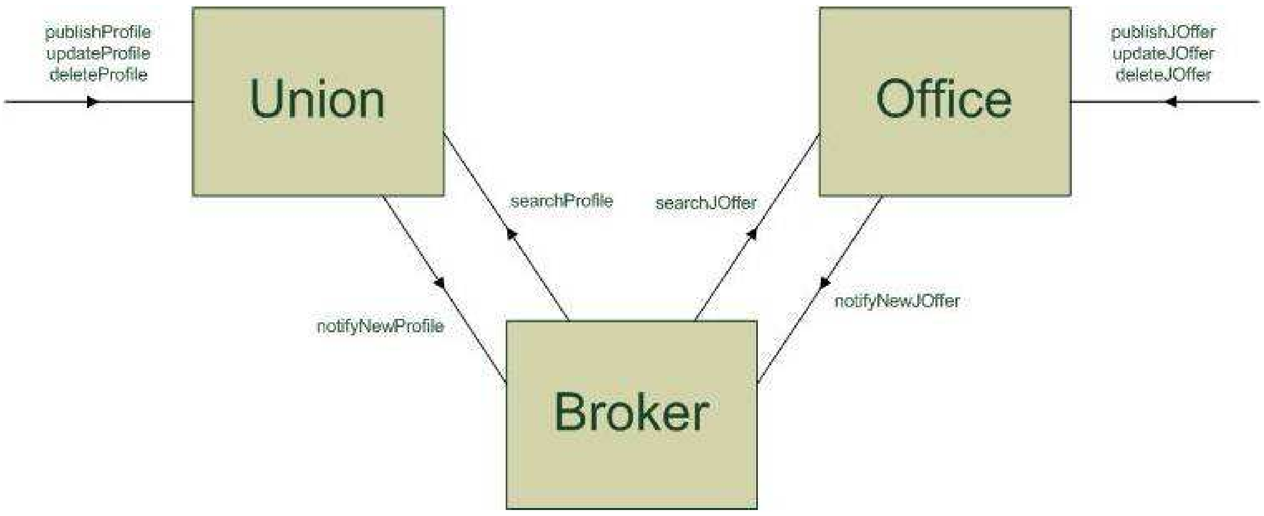


Figure 7 JobBank SOA Architecture

Union is a service which allows publishing, updating and deleting profiles for workers, in turn Office gives opportunity to publish, update, and delete offers for companies. Broker is a mediator which can bind union and office services together by means of notifying methods. The basic communication scenario of this system can be illustrated by sequence diagram (Figure 8):

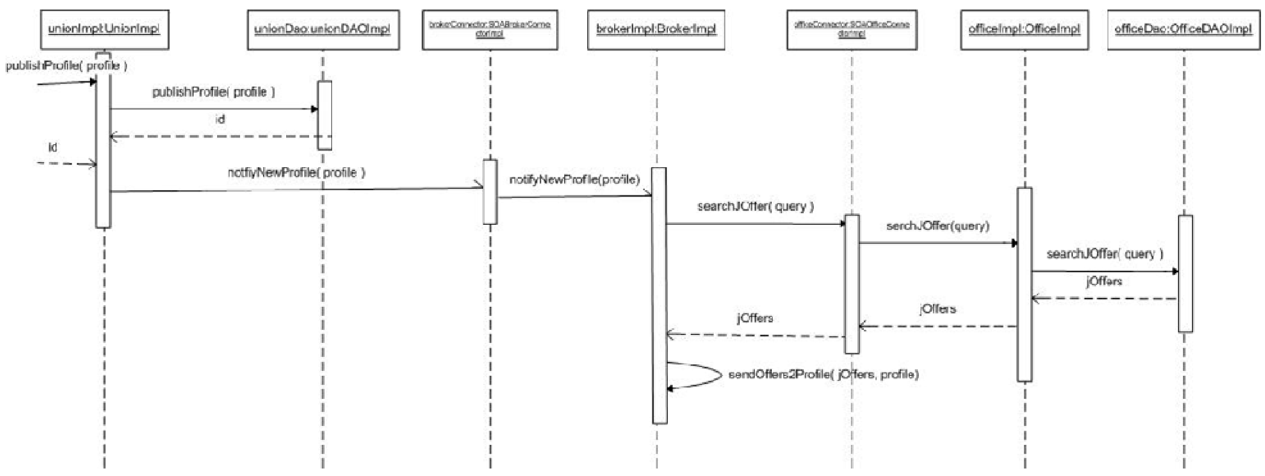

unionImpl:UnionImpl
unionDao:unionDAOImpl
brokerConnector:SOABrokerConnectorImpl
brokerImpl:BrokerImpl
officeConnector:SOAOfficeConnectorImpl
officeImpl:OfficeImpl
officeDao:OfficeDAOImpl
publishProfile(profile)
publishProfile(profile)
id
id
notfiyNewProfile(profile)
notifyNewProfile(profile)
searchJOffer(query)
serchJOffer(query)
searchJOffer(query)
jOffers
jOffers
jOffers
sendOffers2Profile(jOffers, profile)

Figure 8 Sequence Diagram for JobBank Project

We can see from sequence diagram the following example: worker publish his profile in union service, service saves this profile in its database and notifies broker about a new one. Broker performs searching in office service using data in this profile. Office tries to find matches and if they are sends result to worker via email. In JobBank project all messages are SOAP messages, and all services – web services. The idea was to expose each service through OpenSPCoop, so that emulate the situation of different public administrations (PAs) interconnection (PA for Worker Union and PA for Recruiting Office).

The generic way of request-response communication configuration as follows:

1. Configure the service register of OpenSPCoop;
2. Configure MinisteroErogatoreSPCoopIT domain inside OpenSPCoop;
3. Configure MinisteroFruitoreSPCoopIT domain inside OpenSPCoop.

1) Configure the service register of OpenSPCoop

Add service agreement and subjects to registroServizi.xml:

```
<registro-servizi xmlns="http://www.openspcoop.org/dao/registry"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openspcoop.org/dao/registry
registroServizi.xsd">
```

```
   <accordo-servizio nome="ASRichiestaStatoAvanzamento" profilo-
collaborazione="sincrono" utilizzo-senza-azione="true"  />
   <soggetto-spcoop tipo="SPC" nome="MinisteroFruitore">
      <connettore tipo="http" nome="PdDMinisteroFruitore">
         <property nome="location"
valore="http://localhost:8080/openspcoop/PA" />
      </connettore>
       <servizio tipo="SPC" nome="RichiestaStatoAvanzamento"
accordo-servizio="ASRichiestaStatoAvanzamento">
          <fruitore tipo="SPC" nome="MinisteroFruitore" />
       </servizio>
   </soggetto-spcoop>
   <soggetto-spcoop tipo="SPC" nome="MinisteroErogatore">
      <connettore tipo="http" nome="PdDMinisteroErogatore">
           <property nome="location"
valore="http://localhost:8080/openspcoop/PA" />
      </connettore>
      <servizio tipo="SPC" nome="ComunicazioneVariazione" accordo-
servizio="ASComunicazioneVariazione">
          <fruitore tipo="SPC" nome="MinisteroFruitore" />
      </servizio>
  </soggetto-spcoop>
</registro-servizi>
```

2) Configurate MinisteroErogatoreSPCoopIT domain inside OpenSPCoop

Add configuration of MinisteroErogatoreSPCoopIT to config.xml:

```
<openspcoop xmlns="http://www.openspcoop.org/dao/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openspcoop.org/dao/config
config.xsd">
   <soggetto-spcoop tipo="SPC" nome="MinisteroErogatore" >
      <porta-applicativa nome="PA_esempioSincrono">
         <servizio tipo="SPC" nome="RichiestaStatoAvanzamento" />
         <servizio-applicativo nome="PROVIDER" >
            <invocazione-servizio>
               <connettore tipo="http" nome="provider">
                  <property nome="location"
valore="<provider_webservice_url>" />
               </connettore>
            </invocazione-servizio>
         </servizio-applicativo>
      </porta-applicativa>
   </soggetto-spcoop>
   <configurazione>
```

```
      <accesso-registro>
         <registro nome="RegistroXML" tipo="xml"
location="/etc/openspcoop/registroServizi.xml" />
      </accesso-registro>
      <inoltro-buste-non-riscontrate cadenza="1" />
      <messaggi-diagnostici spcoop="infoSpcoop"
openspcoop="infoOpenspcoop" />
   </configurazione>
</openspcoop>
```

3) Configure MinisteroFruitoreSPCoopIT domain inside OpenSPCoop Add configuration of MinisteroFruitoreSPCoopIT to config.xml:

```
<openspcoop xmlns="http://www.openspcoop.org/dao/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.openspcoop.org/dao/config
config.xsd">
   <soggetto-spcoop tipo="SPC" nome="MinisteroFruitore" >
      <porta-delegata nome="CooperazioneSincrona"
autenticazione="none" autorizzazione="none">
         <soggetto-spcoop-erogatore tipo="SPC"
nome="MinisteroErogatore" />
         <servizio tipo="SPC" nome="RichiestaStatoAvanzamento" />
      </porta-delegata>
   </soggetto-spcoop>
   <configurazione>
      <accesso-registro>
         <registro nome="RegistroXML" tipo="xml"
location="/etc/openspcoop/registroServizi.xml" />
      </accesso-registro>
      <inoltro-buste-non-riscontrate cadenza="1" />
      <messaggi-diagnostici spcoop="infoSpcoop"
openspcoop="infoOpenspcoop" />
   </configurazione>
</openspcoop>
```

The results of this experiment with OpenSPCoop are following:

- We have constructed a configuration which allows us to expose web service through OpenSPCoop environment;
- All configuration has been done without any implementation;
- Web services can be implemented in any language and deployed on any platform;

- We can easy force user to use credentials in order to authenticate himself against OpenSPCoop.

5.2. Sirv-Interop Prototype

DeltaDator is a software provider for solutions in public sector in Italy. Civilia [10] is ERP system which allows public administrations manage their resources. There are two components PrometeoWeb which allows managing budget and Contabilità Finanziaria which check the budget. The problem is to allow them communicate with each other even if they work in different PAs. For instance, one public administration can generate budget and another should check it.

We can present communication between PrometeoWeb and Contabilità Finanziaria (two components of Civilia) using sequence diagram Figure 9:

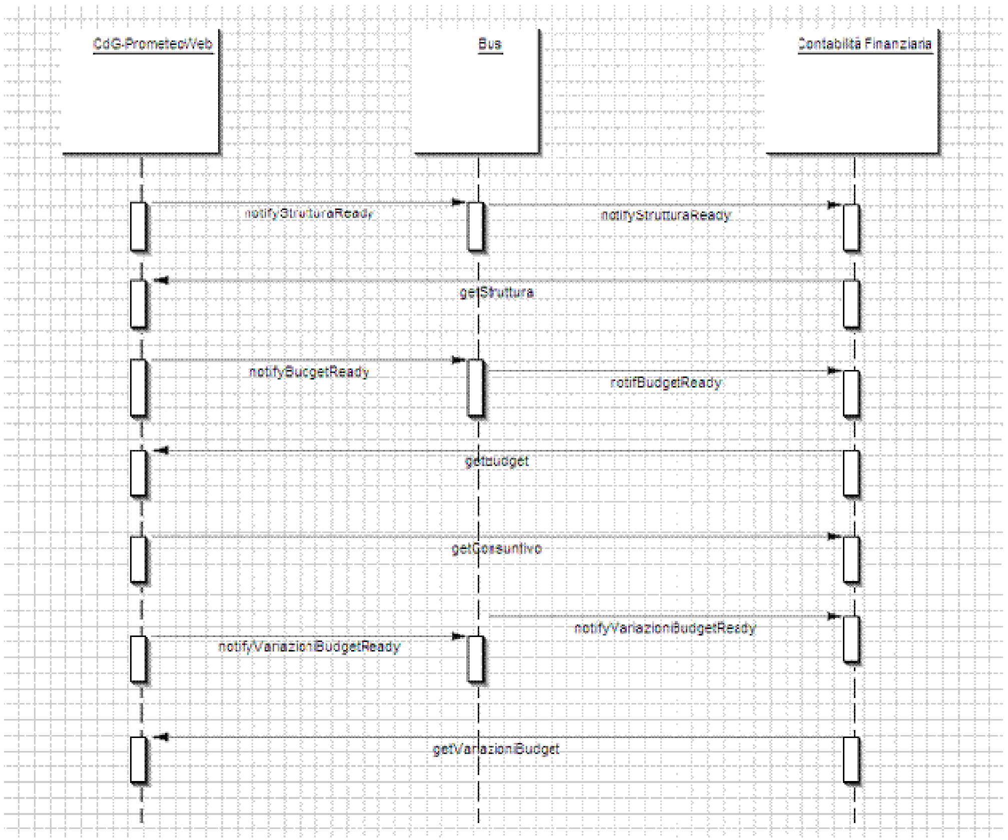


Figure 9 Communication between PrometeoWeb and Contabilita Finanziaria

From the diagram it follows that PrometeoWeb exposes three methods for Contabilità Finanziaria: *getStruttura*, *getBudget*, *getVariazioniBudget*. *getStruttura* returns the structure of the budget, after

Contabilità Finanziaria receives the structure it can call for the budget using method *getBudget* and, finally, it also can ask for variances of budget (*getVariazioniBudget*).

DeltaDator has provided the data structure which describes all entities within the scope of the project.

Balance consists of two parts Entrate (revenue) and Uscita (expenditure). Entrate includes:

- Titoli
- Categorie
- Risorse

Uscita consists of:

- Titoli
- Funzioni
- Servizi
- Interventi

The relations between them can be illustrated using this Figure 10:

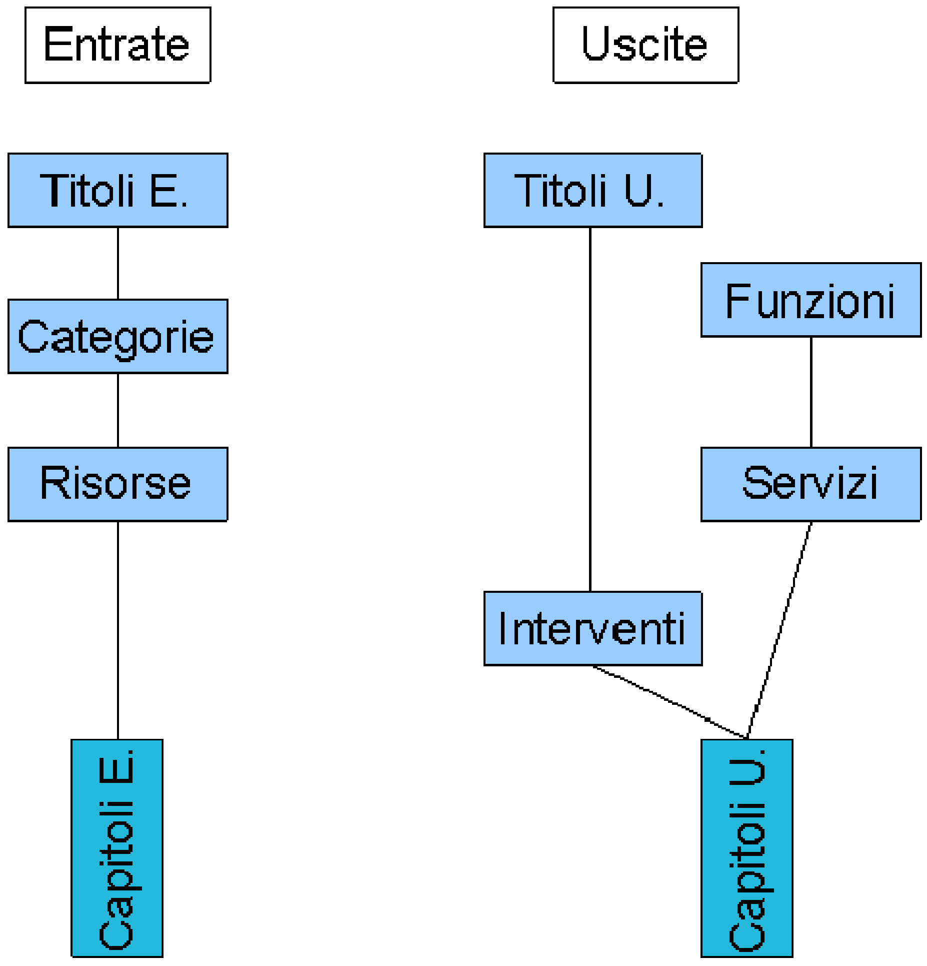


Figure 10 Entrate and Uscite structures

Each Entrate consists of a set of Titoli Entrate. Titili Entrate can contain several Categorie which can hold Risorse. Finally, Capitoli Entrate are kept by Risorse.

Uscite has more complex structure because each Capitolo Uscita must be related to Intervento and Servizio at the same time. Also DeltaDator site gave the internal structure of the data kept by the PrometeoWeb and Contabilità Finanziaria:

Nome	Tipo	Dimensione	Obbligatorietà
ID			

Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 1 Titolo Entrata

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_TITOLO			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 2 Categoria

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_CATEGORIA			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 3 Risorsa

Nome	Tipo	Dimensione	Obbligatorietà
ID			

FK_RISORSA			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 4 Capitolo Entrata

Nome	Tipo	Dimensione	Obbligatorietà
ID			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 5 Titolo Spesa

Nome	Tipo	Dimensione	Obbligatorietà
ID			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 6 Funzione

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_FUNZIONE			

Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 7 Servizio

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_INTERVENTO			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required
Descrizione Corte Conti	VARCHAR2	100	Not Required

Table 8 Intervento

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_SERVIZIO			
FK_INTERVENTO			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 9 Capitolo Spesa

Nome	Tipo	Dimensione	Obbligatorietà

ID			
FK_DESTINAZ_PADRE			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 10 Destinazioni

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_NATURA_PADRE			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 11 Nature

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_PROGRAM_PADRE			
Codice	VARCHAR2	20	Required
Descrizione	VARCHAR2	250	Required

Table 12 Programmi

Nome	Tipo	Dimensione	Obbligatorietà
ID			

ANNO	NUMBER	4	Required
FK_CAPITOLO_SPESA			Required (se manca fk_spesa)
FK_CAPITOLO_ENTRATA			Required (se manca fk_entrata)
FK_NATURA			Required
FK_DESTINAZIONE			Required
FK_PROGRAMMA			

Table 13 Budget

Nome	Tipo	Dimensione	Obbligatorietà
ID			
ANNO_PLURIENNALE	NUMBER	4	Required
FK_BUDGET			Required
IMPORTO	NUMBER	20,6	Required

Table 14 Valore Budget

Nome	Tipo	Dimensione	Obbligatorietà
ID			
ANNO	NUMBER	4	Required
DATA	DATE		Required
NUMERO	NUMBER	12	Required
NOTE	VARCHAR2	500	

Table 15 Variazione Contabile

Nome	Tipo	Dimensione	Obbligatorietà
ID			
FK_VARIAZIONE_CONT			
FK_CAPITOLO_SPESA			Required (se manca fk_spesa)
FK_CAPITOLO_ENTRATA			Required (se manca fk_entrata)
FK_NATURA			Required
FK_DESTINAZIONE			Required
FK_PROGRAMMA			

Table 16 Variazione Budget

Nome	Tipo	Dimensione	Obbligatorietà
ID			
ANNO_PLURIENNALE	NUMBER	4	Required
FK_VAR_BUDGET			Required
IMPORTO	NUMBER	20,6	Required

Table 17 Valore Variazione Budget

Nome	Tipo	Dimensione	Note
Anno	NUMBER	4	
Data Registrazione			
Destinazione			

Programma			
Natura			
Saldo			True = a singola fattura
Numero Documento			Numero Fattura (presente solo se Saldo = true)
Data Documento			Data Fattura (presente solo se Saldo = true)
Data Competenza Da			
Data Competenza a			
Note			
Tipo Movimento			Ordinata o Fatturata
Note			

Table 18 Consuntivo

Use case implementation has been consisted of:

1. Data model proposal in WSDL format;
2. Implementation of the prototype which allows exposing all these services through Sirv-Interop framework.

Actually, the first task can be describes as a serialization of relational data into xml. The data model includes the definition of two services: PrometeoWeb and Contabilita Finanziaria. For Destanzione, Nature and Programmi entities there were chosen the way of recursive data representation:

```
<xs:element name="Destinazione">
 <xs:complexType>
  <xs:sequence>
```

```
   <xs:element ref="tns:Destinazione" minOccurs="0"
maxOccurs="unbounded"/>
   <xs:element name="Descrizione" type="xs:string"/>
  </xs:sequence>
  <xs:attribute name="Codice" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
<xs:element name="Destinazioni">
 <xs:complexType>
  <xs:sequence>
   <xs:element ref="tns:Destinazione" minOccurs="0"
maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

The feature of this approach is that the full data structure is send during the data transmission and thus we can guarantee reference integrity on transport layer. The same situation is with Nature and Programmi.

Capitoli Entrata and Spese were serialized in inverse way (comparing with Figure 10):

```
<xs:element name="CapitoloSpesa" >
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Intervento">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Descrizione" type="xs:string"/>
      <xs:element   name="DescrizioneCorteConti"   type="xs:string"
minOccurs="0"/>
       <xs:element name="TitoloSpesa">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Descrizione" type="xs:string"/>
          <xs:element name="DescrizioneCorteConti" type="xs:string"
minOccurs="0"/>
         </xs:sequence>
         <xs:attribute        name="Codice"        type="xs:string"
use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="Codice" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
    <xs:element name="Servizio">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Descrizione" type="xs:string"/>
       <xs:element name="DescrizioneCorteConti" type="xs:string"
minOccurs="0"/>
       <xs:element name="Funzione">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Descrizione" type="xs:string"/>
          <xs:element name="DescrizioneCorteConti" type="xs:string"
minOccurs="0"/>
         </xs:sequence>
         <xs:attribute name="Codice" type="xs:string"
use="required"/>
        </xs:complexType>
       </xs:element>
      </xs:sequence>
      <xs:attribute name="Codice" type="xs:string" use="required"/>
     </xs:complexType>
    </xs:element>
   </xs:sequence>

  </xs:complexType>

 </xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
```

It was serialized in this way because there are cases when Codice are not unique for Capitolo and thus there are situations when desterilizer will not be able to get original relation data without discords.

According to the requirements for the prototype the following architecture has been chosen:

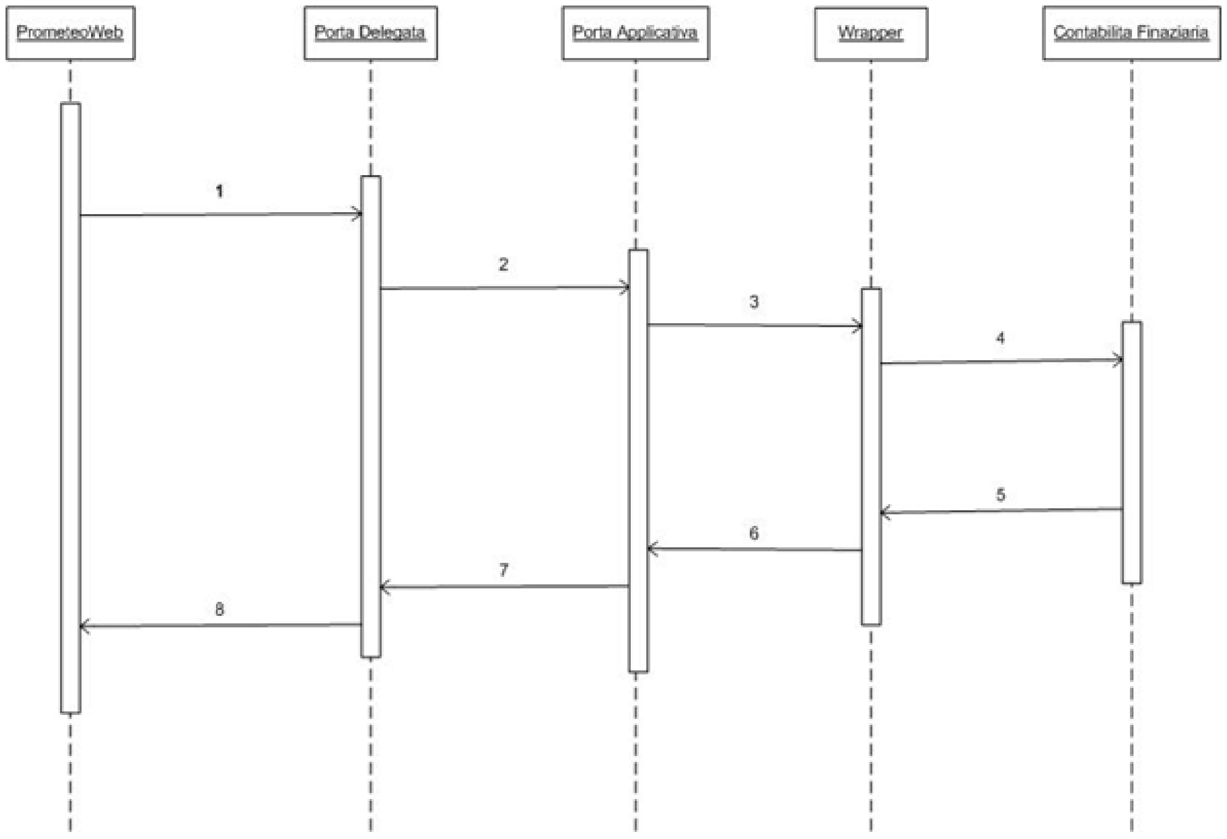


Figure 11 Sequence diagram for PrometeoWeb

From the Figure 111 we can describe the architecture by explaining the sequence of calls for PrometeoWeb (for example, *getConsuntivo* method).

1. PrometeoWeb needs to call *getConsuntivo* method of Contabilita Finaziaria. The call goes through the Sirv-Intreop framework. PrometeoWeb sends request to Porta Delegata by using pddinterface.ws.client.CallWebServicePortaDelegata class (it is inbuilt in Sirv-Interop framework class for PD calling);

2. Porta Delegata transforms request into eGovernment envelope (Busta di dGov) and sends message to corresponding service of Porta Applicativa;

3. Porta Applicativa receives message and redirects it to the wrapper of Contabilita Finanziaria service;

4. Wrapper contains business logic of calling Contabilita Finanziaria (it can be SOAP, RMI or whatever). It sends the message to original service;

5. Contabilita Finaziaria replies to Wrapper;

6. Wrapper sends returns result of the call to Porta Applicativa;

7. Porta Applicativa encapsulates it to envelope and resends it to Porta Delegata;

8. Porta Delegata returns result to PrometeoWeb.

The following architecture applies for all calls within the scope of the Prototype (getNature, getProgrammi, getDestinzioni, getBudget, getVariazioneContabile are exposed by PrometeoWeb; getConsuntivo is exposed by Contabilita Finanziaria).

The core package contains two subfolders: lego.prometeoweb and lego.contabilitafinaziaria which holds wrappers for PrometeoWeb and Contabilita Finanziaria accordingly. Because of the similarity between wrappers there exists the parent class PrometeoWeb.java which extends PortaApplicativa and implements Wrapper interface (necessary condition to create wrapper for custom service). It contains the basic functionality: logging, error handling, message handling etc. The standard example for Sirv-Interop was used as a template for this class. For each method of PrometeoWeb there are different classes which implement custom functional of calling. In our case it is just logic of taking data from file and sending it as a reply:

```
package lego.prometeoweb;

import com.insiel.lz.eccezione.ErroreXML;
import porte.Messaggio;

import java.io.FileInputStream;

public class Budget extends PrometeoWeb{

    protected Messaggio doStuff(Messaggio messaggio) throws
ErroreXML{
         Messaggio msgOut = messaggio.creaMessaggioRisposta();

        //read response from xml file
        try{
            FileInputStream fis = new
FileInputStream("Budget.xml");
            int x= fis.available();
            byte b[]= new byte[x];
            fis.read(b);
            String risposta = new String(b);
            msgOut.setAllegato(this.nomeAllegatoRisposta,
risposta.getBytes());
        } catch(Exception e){
            log.error(e.getLocalizedMessage());
            throw new ErroreXML(e.getLocalizedMessage());
        }
     return msgOut;

   }
}
```

The same situation in lego.contabilitafinanziaria package, there is just only one file because we need to expose only one method of this application.

The result of the call is taken from xml file which conforms xsd (PrometeoWeb.wsdl and ContabilitaFinziaria.wsdl). All xml files are placed in folder xml.

In order to work with this Prototype it is necessary to configure Sirv-Interop components: Porta Applicativa and Porta Delegata.

The configuration of Porta Applicativa needs adding this to servizi.wsdd:

```
<service name="Budget" style="message">
        <parameter name="className"
value="lego.prometeoweb.Budget"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="Nature" style="message">
        <parameter name="className"
value="lego.prometeoweb.Nature"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="Programmi" style="message">
        <parameter name="className"
value="lego.prometeoweb.Programmi"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="CapitoliEntrate" style="message">
```

```
        <parameter name="className"
value="lego.prometeoweb.CapitoliEntrate"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="CapitoliSpese" style="message">
        <parameter name="className"
value="lego.prometeoweb.CapitoliSpese"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="Destinazioni" style="message">
        <parameter name="className"
value="lego.prometeoweb.Destinazioni"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="VariazioneContabile" style="message">
        <parameter name="className"
value="lego.prometeoweb.VariazioneContabile"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
    <service name="Consuntivo" style="message">
        <parameter name="className"
value="lego.contabilitafinanziaria.Consuntivo"/>
        <parameter name="allowedMethods" value="esegui"/>
    </service>
```

This configuration binds the service in Porta Applicativa with its wrapper.

Porta Delegata requires the definition of services in ListaServizi.xml:

```
<Parte Identificativo="PrometeoWeb" desc="PrometeoWeb">
        <Servizio codiceServizio="Budget">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="Nature">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="VariazioneContabile">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="Programmi">
```

```
<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="CapitoliEntrate">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="CapitoliSpese">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
        <Servizio codiceServizio="Destinazioni">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
    </Parte>
    <Parte Identificativo="ContabilitaFinanziaria"
desc="ContabilitaFinanziaria">
        <Servizio codiceServizio="Consuntivo">

<indirizzoPorta>http://localhost:8080/portaapplicativa/servlet/spas
aaj </indirizzoPorta>
        </Servizio>
    </Parte>
```

6. Conclusion

This document provides the full description of the work has been made during the stage in DeltaDator Company. The problem of interoperability between public administrations, which was set by DeltaDator site, has been successfully solved, using the following steps:

- **Theoretical research of the problem**. European Interoperability Framework was chosen as high-level solution for the problem;

- **Analysis of the available possible implementation**. Initially we have chosen two architectures for service interoperability: JBI – open source java standard and SPCoop – Italian framework for integration which includes both technical and organizational aspects. After that we've considered the most known implementation of the mentioned architectures: ServiceMix for JBI, OpenSPCoop and Sirv-Interop for SPCoop. The comparison of them can be illustrated by the table:

Feature	**ServiceMix**	**OpenSPCoop**	**Sirv-Interop**
Open source	Yes	Yes	No
WS-Security	No	Yes	Yes
Federated Authentication	No	No	Yes
Organizational interoperability	No	Yes	Yes
Semantic interoperability	No	Yes	Yes

Table 19 Technology comparison Table

- **Implementations of prototypes for all platforms**. These prototypes are based on the "Application Integration and Business Process Management" class project and they illustrate how effectively ServiceMix and OpenSPCoop can play role of intermediate layer for application messages;

- **Implementation of the DeltaDator use case.** The implementation includes development of data model and programmable environment. The choice of Sirv-Interop framework as a primary one was forced by DeltaDator site and can be explained by internal reasons of company.

References

1 Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.

2 European Interoperability Framework for Pan-European eGovernment Services, http://europa.eu.int/idabc

3 JSR-000208 Java Business Integration 1.0, http://jcp.org/aboutJava/communityprocess/final/jsr208/index.html

4 Web Services Description Language (WSDL) 1.1 , http://www.w3.org/TR/wsdl

5 Roberto Baldoni1, Stefano Fuligni, Massimo Mecella1, and Francesco Tortorelli. The Italian e-Government Service Oriented Architecture. Strategic Vision and Technical Solutions.

6 ServiceMix, http://servicemix.apache.org/home.html

7 Adnrea Corradini, Tito Flagella, Andrea Poli. OpenSpCoop: un'Implementazione della Specifica di Cooperazione Applicativa per la Pubblica Amministrazione Italiana.

8 SPC, "Specifiche della Busta di e-Government, Edizione 1.0", CNIPA, 21 Aprile 2004

9 Progetto SIRV-INTEROP SERVIZI DI INTEROPERABILITA' PER ENTI ED AMMINISTRAZIONI DELLA REGIONE VENETO, http://interopcms.regione.veneto.it/

10 DeltaDator s.p.a., http://www.deltadator.it/