IRS-III: A Broker for Semantic Web Services based Applications

Liliana Cabral, John Domingue, Stefania Galizia, Alessio Gugliotta,
Vlad Tanasescu, Carlos Pedrinaci, Barry Norton

Knowledge Media Institute, The Open University, Milton Keynes, UK
{L.S.Cabral, J.B.Domingue}@open.ac.uk

Abstract. In this paper we describe IRS-III which takes a semantic broker based approach to creating applications from Semantic Web Services by mediating between a service requester and one or more service providers. Business organisations can view Semantic Web Services as the basic mechanism for integrating data and processes across applications on the Web. This paper extends previous publications on IRS by providing an overall description of our framework from the point of view of application development. More specifically, we describe the IRS-III methodology for building applications using Semantic Web Services and illustrate our approach through a use case on e-government.

1 Introduction

The integration of business applications on the Web became a far easier task with the advent of Web Services as part of a trend in XML-based distributed computing. Web Services enable companies to provide services by exposing process functionalities through a standard interface description, keeping intact their legacy implementation of computing systems. Thus, applications in diverse areas such as e-commerce and e-government can interoperate through Web Services implemented in heterogeneous platforms. For example, Google (http://www.google.com) has a Web Service interface to its search engine and Amazon (http://www.amazon.com) allows software developers to access product data through its Web Service platform.

A key problem with the use of standards for Web Service description (e.g. WSDL) and publishing (e.g. UDDI) is that the syntactic definitions used in these descriptions do not completely describe the capability of a service and cannot be understood by software programs. It requires a human to interpret the meaning of inputs, outputs and applicable constraints as well as the context in which services can be used.

Semantic Web Services (SWS) research aims to automate the development of Web Service based applications through Semantic Web technology. By providing formal representations based on ontologies we can facilitate the machine interpretation of Web Service descriptions. Thus, business organisations can view Semantic Web Services as the basic mechanism for integrating data and processes across applications on the Web.

In this paper we describe IRS-III (Internet Reasoning Service), a framework which takes a semantic broker based approach to creating applications from Semantic Web Services by mediating between a service requester and one or more service providers. This paper extends previous publications on IRS by providing an overall description of our framework from the point of view of application development. More specifically, we describe the IRS-III methodology for building applications using Semantic Web Services and illustrate our approach through a use case on e-government.

The rest of the paper is structured as follows: section 2 describes the overall approach and design principles of IRS-III; section 3 describes the IRS-III service ontology; in section 4 we present the framework including our approach for choreography, orchestration and mediation; section 5 describes how to develop applications using IRS-III followed by an example on e-government; finally, the last sections discuss related work and present our conclusions.

2 IRS-III Approach

The IRS project (http://kmi.open.ac.uk/projects/irs) has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the internet. IRS-I [3] supported the creation of knowledge intensive systems structured according to the UPML framework [10] and IRS-II [9] integrated the UPML framework with Web Service technology. IRS-III [5] has incorporated and extended the WSMO ontology [11] so that the implemented infrastructure allows the description, publication and execution of Semantic Web Services (SWS). The meta-model of WSMO describes four top level elements (in italics hence forth):

- *Ontologies*,
- *Goals*,
- *Web Services*, and
- *Mediator*s.

Ontologies provide the foundation for semantically describing data in order to achieve semantic interoperability and are used by the three other WSMO elements. *Goals* define the tasks that a service requester expects a *web service* to fulfil. In this sense they express the service requester's intent. *Web services* represent the functional behaviour of an existing deployed Web Service. The description also outlines how Web Services communicate (*choreography*) and how they are composed (*orchestration*). *Mediator*s describe the connections between the components above and represent the type of conceptual mismatches that can occur. In particular, WSMO provides four kinds of *mediator*s: *oo-mediators* link and map between heterogeneous ontologies; *ww-mediators* link *web services* to *web services*; *wg-mediators* connect *web services* to *goals*; *gg-mediators* link different *goals*.

IRS-III provides the representational and reasoning mechanisms for implementing the WSMO meta-model mentioned above in order to describe Web Services. Additionally, IRS-III provides a powerful execution environment which enables these descriptions to be associated to a deployed Web Service and instantiated during selection, composition, mediation and invocation activities.

The following describes the main application development activities supported by IRS-III when building Semantic Web Services:

- **Using domain ontologies** - The concepts and relations involved in the application scenario which are used to describe client requests and Web Service capability are provided in domain ontologies.
- **Describing client requests as *goals*** – The request for a service can be expressed from a business viewpoint and represented as a *goal*.
- **Semantically describing deployed Web Services** - The concepts defined in domain ontologies can be used in a *web service* description to represent the types of inputs and outputs of services and in logical expressions for expressing applied restrictions. This description can also include many other aspects such as *orchestration* and *choreography*.
- **Resolving conceptual mismatches** – *Mediator* descriptions can be used to declare which *mediation service* or *mapping rules* will provide conceptual alignment between *goals*, *web services* and domain *ontologies*.
- **Publishing and invoking semantically described Web Services** - Once a semantic description has been created for a deployed Web Service as above, it can be registered into IRS-III for *goal*-based invocation.

The IRS-III tooling consists of a Java API and a browser/editor which support developers in building applications out of Semantic Web Services. The IRS-III browser provides an easy to use graphical interface to support the creation of WSMO descriptions, to publish deployed Web Services against these descriptions and then to invoke the Web Services. The IRS-III Java API provides a data model for our WSMO implementation and remote access to the operations available from the IRS-III server. Recently, we have also developed a plug-in for WSMO Studio [4] for interoperability purposes, by aligning the IRS-III and WSMO4J (http://wsmo4j.sourceforge.net) APIs.

2.1 IRS-III Design Principles

The ever growing popularity of the Semantic Web is largely due to the extensive use of ontologies [7]. By providing an explicit formal model, ontologies facilitate knowledge sharing by machines and humans. The IRS-III approach is based on a set of design principles which use ontological metamodels as the means underlying selection, composition, mediation and invocation of Semantic Web Services as follows.

A) Semantic Descriptions as Knowledge Components –Within IRS-III, semantic descriptions of Web Services are provided as knowledge components representing the WSMO top-level elements. These knowledge components are executable ontological meta-models which are semantically linked and can be represented using our ontology representation language OCML [8].

B) Reasoning is ubiquitous – Reasoning is seen as an essential mechanism of all Semantic Web Service activities. IRS-III execution environment can easily invoke ontological queries over the underlying WSMO conceptual model as well as existing domain ontologies.

C) *Goal*-based invocation – A key feature of IRS-III is that Web Service invocation is capability driven. IRS-III supports this by providing a *goal*-centric invocation mechanism. A client application simply asks for a *goal* to be solved and IRS-III selects an appropriate *web service* invoking the associated Web Service.
D) *Goal*-based decomposition – In IRS-III a *web service* is either executable or composed. A composite *web service* expresses its functionality in terms of *goals*, following on the previous design principle for invocation.
E) Explicit mediation description – IRS-III uses the *mediator* description for two purposes. First, it can represent the role of a specific Web Service as a *mediation service*. Second, the different types of *mediators* can be associated with different mediation activities.
F) One-click Publishing – For supporting users who have an existing system which they would like to be made available for invocation through IRS-III, we provide 'one click' publishing mechanism of standalone code written in Java or Lisp in addition to the publishing of existing Web Services through WSDL descriptions.
G) Complete Descriptions - Within an ontological framework, it is easy to represent distinct aspects of a Web Service for different uses. The next section describes these aspects in more details.

3 The IRS-III Service Ontology

The IRS-III service ontology has originally been based on the UPML framework [10] [9], which forms the epistemological basis for IRS-III. This framework has been extended in order to incorporate the following main aspects specified by the WSMO conceptual model [11]:

- **Non-functional properties** – These properties are associated with every main WSMO element and can range from information about the provider such as organisation, to information about the service such as category, cost or trust, to execution requirements such as scalability, security or robustness.
- **Goal-related information** – a *goal* represents the user perspective of the required functional capabilities. It includes a description of the requested *web service capability*.
- **Web Service functional capabilities** – Represent the provider perspective of what the service does in terms of inputs, output, pre-conditions and post-conditions. Pre-conditions and post-conditions are expressed by logical expressions that constrain the state or the type of inputs and outputs.
- **Choreography** – The *choreography* specifies how to communicate with a Web Service. In WSMO this specification is formalized as Abstract State Machines.
- **Grounding** – The grounding is associated with the *web service choreography* and describes how the semantic declarations are mapped to a syntactic specification such as WSDL.
- **Orchestration** – The *orchestration* of a *web service* specifies the decomposition of its capability in terms of the functionality of other Web Services. In WSMO this specification is also formalized as Abstract State Machines.

- **Mediators** – In WSMO, a *mediator* defines which WSMO top elements are connected and which type of mismatches can be resolved between them.

The IRS-III implementation of the WSMO conceptual model has been extended in the following ways.

- **Explicit input and output role declaration** – IRS-III requires that *goals* and *web services* have input and output roles, which include a name and a semantic type. The declared types are imported from domain ontologies.
- ***Web Services* are linked to *Goals* via *mediators*** - If a wg-*mediator* associated with a *web service* has a *goal* as a source, then this *web service* is considered to solve that *goal*. An *assumption* expression can be introduced for further refining the applicability of the *web service*.
- **GG-mediators provide data-flow between sub-*goals*** – In IRS-III, *gg-mediators* are used to link sub-*goals* within an *orchestration*, and therefore they can provide dataflow and data mediation between the sub-*goals*.
- ***Web Service* can inherit from *Goal*s** - *Web services* which are linked to *goals* 'inherit' the *goal's* input and output roles. This means that input role declarations within a *web service* are not mandatory and can be used to either add extra input roles or to change an input role type.
- **Client Choreography** – The provider of a *web service* must describe the *choreography* from the viewpoint of the client. This means IRS-III can interpret the choreography in order to communicate with the deployed Web Service.
- **Mediation services are *goals*** – A *mediator* can declare a *goal* as the *mediation service* which can simply be invoked. The associated *web service* actually performs the necessary data transformation.

4 The IRS-III Framework

IRS-III is based on a distributed architecture composed of the IRS-III server, the publishing platforms and clients which communicate through the SOAP protocol, as shown in figure 1. The server handles ontology management and the execution of knowledge models defined for WSMO. The server also receives SOAP requests (through the API) from client applications for creating and editing WSMO descriptions of *goals*, *web services* and *mediators* as well as goal-based invocation. At the lowest level the IRS-III Server uses an HTTP server written in Lisp, which has been extended with a SOAP handler.

The publishing platforms allow providers of services to attach semantic descriptions to their deployed services and provide handlers to invoke services in a specific language or platform (Web Services WSDL, Lisp code, Java code, and Web applications). When a Web Service is published in IRS-III the information about the publishing platform (URL) is also associated with the *web service* description in order to be invoked. The IRS-III server is written in Lisp and is available as an executable file. The publishing platforms are delivered as Java Web applications; and client applications use the Java API.

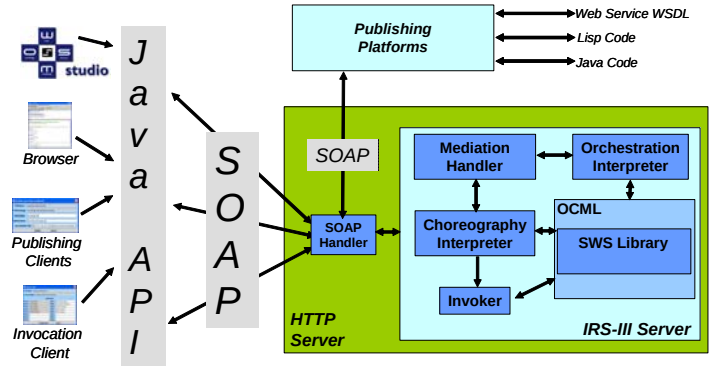

Fig. 1. The IRS-III framework

The main components of IRS-III are explained in the following:

- **SWS Library** – At the core of the IRS-III server is the SWS library where the semantic descriptions are stored using our representation language OCML [8]. The library is structured into knowledge models for *goals*, *web services* and *mediators*. Domain *ontologies* and knowledge bases (instances) are also available from the library.
- **Choreography Interpreter** – This component interprets the *grounding* and *guarded transitions* of the *choreography* description when requested by the mediation handler.
- **Orchestration Interpreter** – This component interprets the workflow of the orchestration description when requested by the mediation handler.
- **Mediation Handler** – The brokering activities of IRS-III including selection, composition and invocation are each supported by a specific mediation component within the mediation handler. These activities may involve executing a mediation service or mapping rules declared in a *mediator* description.
- **Invoker** – The invoker component of the server communicates with the publishing platform, sending the inputs from the client and bringing the result back to the client.

The following sections give more details of how choreography, orchestration and mediation of Semantic Web Services are implemented in IRS-III.

4.1 IRS-III Choreography

In IRS-III the choreography describes how to interact with a single deployed Web Service (client choreography). At the semantic level the choreography is represented by a set of forward-chaining rules and a grounding declaration expressed in OCML (see an example in listing 3). A rule executes actions based on communication primitives when the associated conditions (asserted facts) are satisfied. The *grounding* declares the operations involved in the invocation (communication primitives) and the

associated mappings to the implementation level. More specifically, each operation input and output is associated with a lifting or lowering function. The grounding also relates to information about the corresponding publishing platform.

This approach allows the functionality of a Web Service to be realized by calling one or more declared operations. The set of core communication primitives, which enables the exchange of messages between IRS-III and a deployed service, are listed below.

- ***init-choreography*** – The initial assertion made by IRS-III when the state of the choreography is initialized. IRS-III obtains the input values of operations from the *goal* invocation request.
- ***send-message*** - Calls a specific operation in the associated Web service.
- ***received-message*** - Contains the result of a successful *send-message* for a specific operation.
- ***received-error*** - If an operation generates an error then this primitive is used including the error message and the name of the operation causing it.
- ***end-choreography*** - Stops the choreography. No other rule will be executed.

More details about the formalization of IRS-III choreography, which is based on Abstract State Machines can be found in [6].

4.2 IRS-III Orchestration

In IRS-III the orchestration is used to describe a composed Web Service. At the semantic level the orchestration is represented by a workflow model expressed in OCML. The distinguishing characteristic of this model is that the basic unit within composition is a *goal*. Thus, the model provides control and data flow constructs over a set of *goals*. Further, dataflow and solving mismatches between *goals* are supported by *mediators*. An example of an orchestration description is given in listing 3. The set of control flow primitives which have been implemented so far in IRS-III are listed below.

- ***orch-sequence*** – Contains the list of *goals* to be invoked sequentially. A gg-*mediator* can optionally be declared between the *goals*, in which case the output of the source *goal* is transformed by the *mediation service* (if there is one) and used as input of the target *goal*.
- ***orch-if*** – Contains a condition and a body with one or more workflow primitives. The body part is executed if the declared condition is true.
- ***orch-repeat*** – Contains a condition and a body with one or more workflow primitives. The body part is repeated until the declared condition is false.
- ***orch-get-goal-value*** - Returns the result of the last invocation of the declared *goal* (used for example as part of a condition).
- ***orch-return*** – Returns the result of the current *goal* execution.

Further work is under specification in order to provide a three-layer orchestration model which integrates this semantic representation with a high-level (UML based) workflow representation and a low-level Abstract State Machine representation.

4.3 IRS-III Mediation

At the semantic level, IRS-III represents four basic types of conceptual mismatches that can occur when using Semantic Web Services. These types correspond to the WSMO models of *oo-mediator*, *wg-mediator*, *gg-mediator* and *ww-mediator* as described in section 2. In general there will be mismatches between the *goal* requests and available *web services* and between the *goals* themselves. The IRS-III mediation handler components are responsible for resolving the conceptual mismatches which may occur by reasoning over the given *goal*, *web service* and *mediator* descriptions. The mediation handler interprets each type of mediator accordingly during selection, invocation and orchestration.

Basically, a *mediator* declares a source component, a target component and either a *mediation service* or *mapping rules*. Hence, the *mediator* provides a semantic link between the source component and the target component, which enables *mediation services* or *mapping rules* to solve mismatches between the two. More details of mediation in IRS-III can be found in [1].

In this model, the *mediation service* is just another *goal*. As an example (see listing 3), the *mediation service* of a *wg-mediator* can transform input values coming from the source *goal* into an input value used by the target *web service*.

Mapping rules are used between two *ontologies* (source and target components). These mappings only concern to the concepts used during invocation and consist of three main mapping primitives:

- ***maps-to*** – relation created internally for every mapped instance.
- ***def-concept-mapping*** – generate the mappings (*maps-to* relation) between the instances of two concepts within an ontology.
- ***def-relation-mapping*** – generate a mapping between two relations using a rule definition within an ontology. As OCML represents concept attributes as relations, this primitive can be used to map between *input* and *output* descriptions.

5 Application Development with IRS-III

A Web application can invoke Semantic Web Services by sending “achieve-goal” requests to IRS-III with the input values from the user. IRS-III will then execute the appropriate deployed Web Services (see figure 2). This Semantic Web Service brokering scenario enables data and process integration across many business partners. The SWS provided can be shared or used to send common information to the diverse participating organisations.

In our methodology for developing applications using SWS with IRS-III we devise a customer team for creating *goal* descriptions according to user requests and a development team for creating *web service* descriptions for the available deployed Web Services. The application developer then creates *mediator* descriptions which connect domain *ontologies*, *goals* and *web services* and provide *mediation services* or *mapping rules* for solving mismatches between ontological elements.

Fig. 2. A simple SWS brokering scenario using IRS-III.

We created a generic application architecture which reflects our methodology for using IRS-III following on the steps described on section 2 as depicted in figure 3. Briefly, such architecture enables the functionality provided by existing legacy systems from the involved business partners to be exposed as Web Services, which are then semantically annotated and published using the SWS infrastructure. The architecture consists of four layers as explained next.

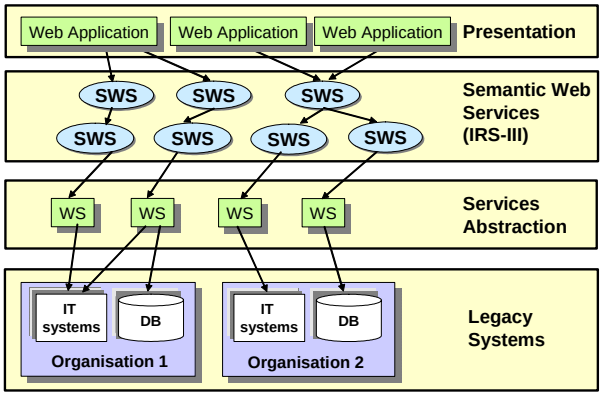

Fig. 3. A generic application architecture using IRS-III.

The legacy system layer consists of the existing data sources and information technology systems available from each organisation involved in the integrated application. The service abstraction layer enables the functionality of the legacy systems to be available as Web Services, abstracting from the implementation details. Current Enterprise Application Integration (EAI) software generally enables the easy creation of the necessary Web Services. Note that for the integration of standard databases the necessary functionality of the Web Services can simply be implemented as query (SQL) functions. The SWS layer is based on the Web Services provided by the service abstraction layer. The activities in this layer are mainly supported by the IRS-III infrastructure as outlined in section 2. Given a goal request, IRS-III will: a) discover a candidate set of *web services*; b) select the most appropriate one; c) resolve any mismatches at the ontological level; and d) invoke the relevant set of Web Services satisfying any data, control flow and invocation requirements. To achieve this, IRS-III, utilizes the set of Semantic Web Service descriptions which are composed of *goals*, *mediators*, and *web services*, supported by relevant domain ontologies. Finally, the presentation layer consists of the user interface, which is built on top of the SWS layer as a Web application accessible using a standard Web browser. G*oal* invocation

requests are generated with the data provided by the user through the user interface triggering the invocation of applicable SWS and as a result the execution of deployed Web Services in the service abstraction layer

In the next section we will further explain our methodology by mapping each architecture layer to the development activities related to a specific application in e-government. In the following we point out some generic considerations when using SWS as outlined in the architecture described above.

In general, during the requirements phase of application development, the stakeholders involved in the application scenario should provide information to ontology builders in order to create or reuse domain ontologies related to the application context. SWS make this process very simple and efficient because the only knowledge which must be modelled is related to the exposed functionality implemented by the Web Services. Developers do not need to model entire data sources or create class instances corresponding to thousands of database records; we only model the information used by Web Services.

By taking a top-down approach for semantically annotating services, IRS-III facilitates querying and reasoning about the capability of the service before its execution since the semantic relations between the descriptions used (*goal*, *web services*, *mediators* and domain *ontologies*) are well defined in the WSMO metamodel. The reasoning needed during the invocation of one service is efficient because it is limited to the scope of the invocation.

6 Application example on E-government

In the following we present relevant details of the prototype created for the case study on e-government within the DIP project (http://dip.semanticweb.org) for illustrating an application based on Semantic Web Services using IRS-III. The main requirement for applications in E-government relates to the interoperability of data and processes between services provided by different government agencies.

Our implemented scenario named “Change of Circumstances” involved two governmental agencies coordinated by Essex County Council (ECC) in UK. In this scenario a disabled mother moves into her daughter’s home and both are eligible to receive services and benefits – health and housing equipments – from service providing agencies. A case worker of the Community Care department helps a citizen to report her change of circumstance (e.g. address) to different agencies involved in the process.

Following from the architecture in Figure 3, at the presentation level we created an application user interface for the Change of Circumstances scenario. From the interface a case worker from Essex County Council has access to some functionality such as “update client details” and “create client assessment”. Behind each functionality there is one or more associated *goal* requests such as “update citizen address” or “find equipment”. A case worker can select a suitable functionality, fill in the required fields and then submit his request which will trigger the execution of the defined *goals*.

At the semantic level, we used IRS-III to provide WSMO descriptions to the deployed Web Services, including mediator descriptions for declaring the mappings between concepts not aligned. We then published the Web Services in IRS-III. The relevant integration aspect was the implementation of a composed *web service*, which accesses information from two different agencies. This composed service named "change-address-ws" will be explained in more details in the illustration of the semantic descriptions in the next section. This service is composed of two basic services. The first changes the address of the citizen within ECC, and the second service changes the address of the citizen within the agency providing services related to housing equipment.

At the service level, we developed a set of Web Services which performed basic operations on top of the databases of the two involved agencies. These Web services were deployed into an application server (SAP Exchange Infrastructure) provided by a partner at SAP in Germany and then published in IRS-III, running at the Open University in England. At the legacy systems level, we recreated anonymous content (due to privacy reasons) of the existing data sources for each agency involved.

6.1 Semantic Descriptions

In the following we present the domain ontologies and Semantic Web Service descriptions used in the application prototype. Each agency involved in the prototype development provided a domain ontology which represents its own information concerning the application scenario. A domain ontology can represent the viewpoint of the user and then be used to define *goals* or it can represent the viewpoint of a service provider and therefore be used for describing deployed Web Services. The ontologies were developed independently but both used a common upper-level ontology describing general concepts from the e-government domain (e.g. government-organisation, county-council, public-service, health-service).

Listing 1. Partial source code for concepts in the domain ontologies.

```
(def-class equipment ()
  ((has-product-code :type string)
   (has-description :type string)
   (has-cost :type string)
   (has-max-user-weight :type integer)
   (has-charging-value :type string)
   (has-product-widtht :type string)
   (has-product-height :type string)
   (has-product-seat-height :type string)))

(def-class citizen-address ()
  ((has-address-key :type integer)
   (has-postcode :type post-code-string)
   (has-premise-number :type integer)
   (has-premise-name :type string)
   (has-street :type string)
   (has-locality :type string)
   (has-town :type string)))
```

The two developed ontologies are as follow:

- Citizens ontology - Domain ontology created by Essex County Council describing information related to a citizen assessment for social benefits and ser-

vices. Contain classes defining for example: address, assessment, health problem, benefit, case worker and others.

- Equipment ontology – Domain ontology created by the Housing Department describing information related to ordering housing equipments. Contain classes defining for example: order, equipment, supplier, delivery descriptor and so on.

Listing 1 shows an excerpt of two concepts defined in the domain ontologies (attributes are self-explanatory). "Equipment" is used as output of the *goal* (listing 2) and "citizen-address" as input of one of the *web service*s. Instances of these classes can be created with the values of attributes provided through the user interface. Otherwise they can be lifted from the results of service invocations.

Listing 2 shows the definition of *goal* "find-equipment-goal". This instance of a *goal* defines 2 inputs ("has-input-role" slot) and one output ("has-output-role" slot). This *goal* takes the client weight and purpose and returns a list of suitable equipments.

Listing 2. Partial source code for the goal FIND-EQUIPMENT-GOAL.

```
(DEF-CLASS FIND-EQUIPMENT-GOAL (GOAL)?GOAL
      ((HAS-INPUT-ROLE
           :VALUE HAS-CLIENT-WEIGHT
           :VALUE HAS-CLIENT-PURPOSE)
       (HAS-OUTPUT-ROLE
           :VALUE HAS-SUITABLE-ITEMS-LIST)
       (HAS-CLIENT-WEIGHT :TYPE NUMBER)
       (HAS-CLIENT-PURPOSE :TYPE PURPOSE-DESCRIPTOR)
       (HAS-SUITABLE-ITEM-LIST :TYPE EQUIPMENT)
       (HAS-NON-FUNCTIONAL-PROPERTIES
           :VALUE E-GOV-ASSESS-ITEM-GOAL-NON-FUNCTIONAL-PROPERTIES)))
```

Listing 3 shows a partial definition of the *web service* "change-address-ws". This description declares a *capability* and an *interface* which are described in corresponding classes. The *interface* declares an *orchestration*, which is defined in another class. The "problem solving pattern" slot of the *orchestration* defines the workflow (sequence) for the composition of 2 sub-*goal*s. The choreography of one of the sub-*goal*s is defined by another class ("change-citizen-detatils-ws-choreography") which has a grounding and guarded transitions. The grounding includes information about the WSDL associated with the described service, the lowering of the inputs and lifting of the output; there is one rule in the guarded transitions which uses the operation "change-details-operation" defined.

Listing 3. Partial source code for the web service CHANGE-ADDRESS-WS.

```
(DEF-CLASS CHANGE-ADDRESS-WS (WEB-SERVICE) ?WEB-SERVICE
  ((HAS-CAPABILITY :VALUE CHANGE-ADDRESS-WS-WEB-SERVICE-CAPABILITY)
    (HAS-INTERFACE :VALUE CHANGE-ADDRESS-WS-WEB-SERVICE-INTERFACE)
    (HAS-NON-FUNCTIONAL-PROPERTIES
       :VALUE CHANGE-ADDRESS-WS-WEB-SERVICE-NON-FUNCTIONAL-POPERTIES)))

(DEF-CLASS CHANGE-ADDRESS-WS-WEB-SERVICE-INTERFACE (INTERFACE)?INTERFACE
   ((HAS-ORCHESTRATION :VALUE CHANGE-ADDRESS-WS-ORCHESTRATION)
    (HAS-NON-FUNCTIONAL-PROPERTIES
        :VALUE CHANGE-ADDRESS-WS-INTERFACE-NON-FUNCTIONAL-PROPERTIES)))

(DEF-CLASS CHANGE-ADDRESS-WS-ORCHESTRATION (ORCHESTRATION)
    ((HAS-PROBLEM-SOLVING-PATTERN
        :VALUE CHANGE-ADDRESS-WS-ORCHESTRATION-PROBLEM-SOLVING-PATTERN)))
```

```
(DEF-CLASS CHANGE-ADDRESS-WS-ORCHESTRATION-PROBLEM-SOLVING-PATTERN
   (PROBLEM-SOLVING-PATTERN)
      ((HAS-BODY :VALUE
          ((ORCH-SEQUENCE
               CHANGE-CITIZEN-DETAILS-GOAL
               REDIRECT-EQUIPMENT-GOAL)
           (ORCH-RETURN (ORCH-GET-GOAL-VALUE REDIRECT-EQUIPMENT-GOAL))))))

(DEF-CLASS CHANGE-CITIZEN-DETAILS-WS-CHOREOGRAPHY (CHOREOGRAPHY)
   ((HAS-GROUNDING :VALUE
       (GROUNDED-TO-WSDL CHANGE-DETAILS-OPERATION
          (http://changeDetails.wsdl "changeDetails" "changeDetailsPort"
           http://sap.com/research/dip/wp9/elmdb "AXIS")
           ((LOWER-TO HAS_CLIENT_ADDRESS "STRING"))
            (LIFT-TO HAS_ACKNOWLEDGMENT "STRING")))
        (HAS-GUARDED-TRANSITIONS :VALUE
             ((RULE1
                (INIT-CHOREOGRAPHY)
               THEN
                (SEND-MESSAGE 'CHANGE-DETAILS-OPERATION))))
```

Listing 4 shows the definition of mediator "address-mediator". This is an instance of a WSMO GG-mediator. It was used to transform “citizen-address” type to a string used by “redirect-equipment-goal”.

Listing 4. Partial source code for the ADDRESS-MEDIATOR mediator

```
(DEF-CLASS ADDRESS-MEDIATOR (GG-MEDIATOR) ?MEDIATOR
    ((HAS-SOURCE-COMPONENT :VALUE CHANGE-ADDRESS-GOAL)
     (HAS-TARGET-COMPONENT :VALUE REDIRECT-EQUIPMENT-GOAL)
     (HAS-MEDIATION-SERVICE
        :VALUE ADDRESS-MEDIATION-SERVICE-GOAL)
     (HAS-NON-FUNCTIONAL-PROPERTIES :VALUE
        ADDRESS-MEDIATOR-MEDIATOR-NON-FUNCTIONAL-PROPERTIES))))
```

7 Related Work and Conclusions

In this paper we have presented our approach to developing Semantic Web Services, supporting selection, composition, mediation and invocation of Web Services as well as our methodology for developing Web applications which use the IRS-III infrastructure. We have validated our approach in the context of a case study in e-government, which offers a motivating scenario for the use of Semantic Web Services with requirements and data provided by real users. In addition we use the case study to illustrate the semantic descriptions used by IRS-III.

Although a number of Semantic Web Service approaches now exist in addition to IRS-III and WSMO, including for example, OWL-S (http://www.w3.org/ Submission/OWL-S), SWSF (http://www.w3.org/Submission/SWSF) and WSDL-S (http://www.w3.org/Submission/WSDL-S); there are few frameworks which can comprehensively support the development of Semantic Web Services based applications. A more detailed comparison between approaches can be found in [2].

Overall, the work on IRS-III is more closely related to WSMX (http:// www.wsmx.org/) since both environments are based on WSMO. However, IRS-III is founded on a knowledge-based approach and infrastructure which introduces distinguishing design principles and semantic primitives for executing choreography, orchestration and mediation of Semantic Web Services. The SWS approaches listed

above share a number of common features with IRS-III; in particular, there are similarities between the ontological structures used for Web service functional descriptions. Additionally, these approaches enable grounding to WSDL. The main differences concern the behavioral aspects of service description; although a process-oriented abstraction could be constructed for orchestration, a state-based behavior is explicitly represented in our ontology. Moreover, IRS-III focuses on the problems that clients need to solve, providing for this reason a goal-centric invocation mechanism.

8 Acknowledgements

This work is supported by the DIP project (Data, Information and Process Integration with Semantic Web Services) (EU FP6 - 507483). The authors gratefully acknowledge the members of the DIP project and the WSMO working group for their insightful comments on our work. We also acknowledge the contribution of DIP members Mary Rowlatt, Robert Davies and Leticia Gutierrez from Essex County Council - UK.

References

1. Cabral, L. and Domingue, J.: Mediation of Semantic Web Services in IRS-III. In Workshop on Mediation in Semantic Web Services (MEDIATE 2005) in conjunction with the 3rd International Conference on Service Oriented Computing (ICSOC 2005), Amsterdam (2005)
2. Cabral, L., Domingue, J., Motta, E., Payne, T. and Hakimpour, F. (2004). Approaches to Semantic Web Services: An Overview and Comparisons. In proceedings of the First European Semantic Web Symposium, ESWS 2004, Heraklion, Crete, Greece. LNCS 3053
3. Crubezy, M., Motta, E., Lu, W. and Musen, M.: Configuring Online Problem-Solving Resources with the Internet Reasoning Service. IEEE Intelligent Systems, 2 (2003) 34-42
4. Dimitrov, M., Simov, A., Montchev, V. and Ognanov, D.: WSMO Studio: an Interfaced Service Environment for WSMO. In Workshop on WSMO Implementations (WIW 2005) Frankfurt, Germany. CEUR Workshop Proceedings, Vol. 134 (2005)
5. Domingue, J., Cabral, L., Hakimpour, F., Sell, D. and Motta, E.: IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. In Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany. CEUR Workshop Proceedings, Vol. 113 (2004)
6. Domingue, J., Galizia, S. and Cabral, L.: The Choreography Model for IRS-III. In proceedings of Hawaii International Conference on System Sciences (HICSS 2006), Hawaii (2006)
7. Gruber, T. R.: A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition, 5(2) (1993)
8. Motta, E.: Reusable Components for Knowledge Modelling. IOSPress, Amsterdam (1999)
9. Motta, E., Domingue, J., Cabral, L. and Gaspari, M.: IRS-II: A Framework and Infrastructure for Semantic Web Services. In proceeding of the 2nd International Semantic Web Conference (ISWC 2003). LNCS 2870 (2003)
10. Omelayenko, B., Crubezy, M., Fensel, D., Benjamins, R., Wielinga, B., Motta, E., Musen, M., Ding, Y.: UPML: The language and Tool Support for Making the Semantic Web Alive. In: Fensel, D. et al. (eds.): Spinning the Semantic Web: Bringing the WWW to its Full Potential. MIT Press (2003) 141–170
11. WSMO Working Group. Deliverable D2v1.2 Web Service Modeling Ontology (WSMO). http://www.wsmo.org/TR/d2/v1.2/ (2005)