Title:

IRS-III: A Broker-based Approach to Semantic Web Services

Authors:

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

Address:

Knowledge Media Institute
Open University
Milton Keynes, MK7 6AA, UK

Fax number: +44 (0) 1908 653169

E-Mails:
{ j.b.domingue, l.s.cabral, s.galizia, v.tanasescu a.gugliotta, b.j.norton, c.pedrinaci}@open.ac.uk

IRS-III: A Broker-based Approach to Semantic Web Services

JOHN DOMINGUE, LILIANA CABRAL, STEFANIA GALIZIA,
VLAD TANASESCU, ALESSIO GUGLIOTTA, BARRY NORTON, and
CARLOS PEDRINACI
Knowledge Media Institute, The Open University

A factor limiting the take up of Web services is that all tasks associated with the creation of an application, for example, finding, composing, and resolving mismatches between Web services have to be carried out by a software developer. Semantic Web services is a combination of semantic Web and Web service technologies that promise to alleviate these problems. In this paper we describe IRS-III, a framework for creating and executing semantic Web services, which takes a semantic broker based approach to mediating between service requesters and service providers. We describe the overall approach and the components of IRS-III from an ontological and architectural viewpoint. We then illustrate our approach through an application in the eGovernment domain.

Categories and Subject Descriptors: H.3.5 [Information Storage and Retrieval]: Online Information Services—Web-based services I.2.5 [Artificial Intelligence] - Programming Languages and Software
General Terms: Design, Languages, Management
Additional Key Words and Phrases: Semantic Web, Semantic Web Services, Ontology, WSMO, IRS-III

---

1. INTRODUCTION

A factor limiting the take up of Web services is that all tasks associated with the creation of an application, for example, finding, composing, and resolving mismatches between Web services have to be carried out by a software developer. Semantic Web services is a combination of semantic Web and Web service technologies which promise to alleviate these problems.

From a business perspective a key feature of Web services is that they can be viewed as implementations of business services. Commercial organizations can thus use Web services technology to expose elements of their business processes. For example, Google [Google, 2005] has a Web service interface to its search engine and Amazon allows software developers to directly access their technology platform and product data [Amazon, 2006].

From an information technology viewpoint the two important features of Web Services are that: a) they are accessible over the Internet using standard XML-based protocols; and, b) the interface of the Web service is independent of its

actual implementation. The first feature gives Web services high availability whereas the second feature facilitates reusability and interoperability.

Three main technologies are currently used to implement Web services: SOAP [SOAP, 2003], WSDL [WSDL, 2001] and UDDI [UDDI, 2003]. SOAP is an XML based, stateless, one-way message exchange protocol for interacting with Web services over HTTP. WSDL is an XML based format for describing Web services as collections of network endpoints or ports. UDDI is a standard for defining a registry, which allows clients to find Web services through descriptions of business entities, business services or via predefined business categories.

A key problem with the above technologies is that they are purely syntactic. They thus rely on software developers to understand the intended meaning of the descriptions and to carry out the activities related to Web service usage. Semantic Web services (SWS) research aims to automate the development of Web service based applications through semantic Web technology. By providing formal descriptions with well defined semantics we facilitate the machine interpretation of Web service descriptions.

The Semantic Web [Berners-Lee et al., 2001] is an extension of the current Web, where documents incorporate machine executable meaning. The overall semantic Web vision is that one day it will be possible to delegate non-trivial tasks, such as booking a holiday, to software programs able to locate and reason over relevant heterogeneous online resources. One of the key building blocks for the semantic Web is the notion of an ontology [Gruber, 1993]. An ontology is an explicit formal shared conceptualization of a domain of discourse. More specifically, an ontology facilitates semantic interoperability by capturing the main concepts and relations that a community shares over a particular domain.

In this paper we describe IRS-III (Internet Reasoning Service), a framework for creating and executing semantic Web services, which takes a semantic broker based approach to mediating between service requesters and service providers [Cabral et al., 2006; Domingue et al., 2004; Domingue et al., 2005a; Domingue et al., 2005b; Tanasescu et al. 2007]. More specifically, we have extended the

core epistemological framework of our previous IRS-II framework [Motta et al., 2003] (see more details in Section 2) and incorporated the Web Services Modelling Ontology [WSMO, 2007; Fensel et al., 2006] conceptual model (see more details in Section 4) into the IRS-III framework.

A core design principle for IRS-III is to support capability-based invocation (we give a full list of our design principles in Section 3). A client sends a request which captures a desired outcome or goal and, using a set of semantic Web service descriptions, IRS-III will: a) discover potentially relevant Web services; b) select the set of Web services which best fit the incoming request; c) mediate any mismatches at the conceptual level; and d) invoke the selected Web services whilst adhering to any data, control flow and Web service invocation constraints.

In Sections 4, 5 and 6 we describe the general architecture for IRS-III and then describe in detail how IRS-III handles: choreography, the interaction rules for invoking a single Web service; orchestration, the control and data flow for composite Web services; and, mediation, the resolving of mismatches at the conceptual level.

Over the past few years we have been using IRS-III to develop SWS applications in real world contexts within large collaborative national [MIAKT, 2002] and European funded projects [DIP, 2004]. In Section 7 we outline how SWS based systems can be successfully developed and deployed using IRS-III and in Section 8 we illustrate our approach through an eGovernment application we created to support emergency planning. Section 9 discusses the benefits of our approach. The final two sections of the paper contain an overview of related work, conclusions and future work.

2 THE IRS PROJECT OVERVIEW

The IRS project has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the Internet. IRS-I supported the creation of knowledge intensive systems structured according to the UPML (Unified Problem-solving Method Development Language)

framework [Fensel and Motta, 2001]; IRS-II [Motta et al., 2003] integrated the UPML framework with Web service technologies.

The UPML framework partitions knowledge into domain models, task models and problem solving methods (PSMs), which are connected via bridges. Each knowledge model type (task, PSM and domain model) is supported by appropriate ontologies, as follows.

- *Domain models* - domain models describe the domain of an application (e.g. vehicles, a medical disease);
- *Task models* – a generic description of the task to be solved, specifying the input and output types, the goal to be achieved and pre and post-conditions;
- *Problem Solving Methods* – a description of the generic reasoning process to be applied, for example, heuristic classification or propose and revise;
- *Bridges* – contain mappings between the different model components within an application.

Within IRS-III we have now extended this framework and incorporated the WSMO conceptual model, thus generating the new main epistemology entities: goals, Web services and mediators. Additionally, we provide a set of tools to support the SWS developer at design time in creating, editing and managing a library of semantic descriptions as well as publishing and invoking semantic Web services.

3. PRINCIPLES UNDERLYING IRS-III

The IRS-III design principles listed below outline how a broker based platform can support the creation and execution of semantic Web services. As such, the principles take into account knowledge level, operational, and usability criteria.

a. *Brokering role* – we exploit the benefits of semantic technologies in a brokering context where IRS-III mediates between a client and a service provider. We use ontologies to separately capture the client and the provider context and use reasoning over the client and provider viewpoints to support interoperability and collaboration.

b. *Underpinned by ontological representations* - the ever-growing popularity of the semantic Web is in a large part due to the extensive use of ontologies. By providing an explicit formal model, ontologies facilitate knowledge sharing by machines and humans. Within IRS-III we use our own ontology representation language, OCML [Motta, 1998]. More details on OCML are provided in Section 4.1.

c. *Clean ontological separation of user and Web service contexts* - the starting point of our approach is the representation of client requests. A client will exist in its own context which should be modelled explicitly as part of the semantic descriptions. This context will be quite different from that of the Web service. For example, an end user may request a holiday with a preference for a certain climate, located near particular cultural artefacts and amenable to children within a specific age range. The required flights and hotel booking Web services will be described using concepts such as 'city' and 'available date'. Our view is that distinct ontological structures are required to describe potential users and Web services.

d. *Capability based invocation* – a key feature we want to provide based on the previous principle is the ability to invoke services based on the client request. In general, clients will not be interested in the details of service functionality so we support invocation via a desired capability expressed as a WSMO goal. IRS-III then acts as a broker: finding, composing and invoking appropriate Web services in order to fulfil the request. This principle supports principle (a) but of course a semantic broker also requires additional features such as ontology support, separation of user and provider concerns, and support for publishing as set out in the other principles in this section.

e. *Single representation language* – in IRS-III we encode all semantic descriptions with our single representation language OCML. That is, OCML is uniformly used for representing service models (including meta-modelling), rules and logical expressions. This is not the case for several of the other major SWS initiatives. For example, within the OWL-S approach [OWL-S, 2006], due the lack of representational power in OWL-DL, the

modeller is free to choose his or her own favourite language for representing pre and post-conditions. Consequently, language specific reasoners are required to reason over OWL-S descriptions including pre or post-conditions. Within the WSMO approach, the WSML family of languages [WSML, 2005] embeds WSMO concepts within the grammar of the language. For example, the concepts goal, mediator and Web service are defined using inbuilt keywords. Whilst, reducing the syntax burden for the SWS developer this design decision restricts the scope and power of WSML as WSMO concepts are not available as first class citizens. A single representation language provides a number of benefits for IRS-III. Firstly, we can use WSMO concepts in other ontologies and we can subclass WSMO concepts. For example, one can define new types of goals for a particular domain or application or one can define that the type of an attribute is a goal. The second benefit is that we can utilise the principle of self-hosting[1], that is, we can define internal components of IRS-III in OCML using our service ontology. We expand on this in later sections.

f. *Ease of use* – creating SWS based applications is a very complex task and it is thus essential that the support platform is as easy to use as possible for SWS application developers. Within IRS-III, for example, the browser hides some of the complexity of the underlying service ontology by bundling up related class definitions into a single tabbed dialog window. While we have not attempted to measure the ease of use of IRS-III, we have been able to use IRS-III successfully in a number of tutorials as well as project use cases involving real users in industrial contexts.

g. *Seamless publishing of services* - users quite often will have an existing system functionality, which they would like to be made available as a service, but have no knowledge of the tools and processes involved in turning a stand-alone program into a Web service. We therefore created IRS-III so that it supports ‘one click’ publishing of stand-alone code (currently Java and

[1] http://en.wikipedia.org/wiki/Self-hosting

Lisp). We also publish Web services from the given WSDL description or from the URI (i.e. a HTTP GET request) of a web application.

h. *Inspectability* - in many parts of the life cycle of any software system, it is important that developers are able to understand the design and behaviour of the software being constructed. This is also true for SWS applications. This principle is concerned with making the semantic descriptions accessible in a human readable form. In IRS-III, we make the service models available from libraries and inspectable within a purpose built browsing and editing environment.

i. *Interoperable with SWS frameworks and platforms* - one of the main aims for Web services is to enable the interoperability of programs over the Internet. A reasonable extension of this is that, as far as possible, SWS frameworks and platforms also should be interoperable. For this reason, IRS-III has an OWL-S import mechanism [Hakimpour et al., 2004] and is interoperable with WSMO implementations (e.g. WSMO Studio (www.wsmostudio.org); and WSMX - www.wsmx.org) through a common standard API (http://www.oasis-open.org/committees/semantic-ex).

j. *Executable semantic descriptions* - the semantic representations should be executable directly or should be able to be compilable to a runnable representation. This principle follows on the operational ability of our underlying language OCML [Motta, 1998] to execute functions, rules and semantic definitions as part of the reasoning system. This makes IRS-III partially self-hosting – a number of components are implemented in OCML using the IRS-III SWS descriptions.

Given that IRS-III and WSMO share the UPML framework as a common ancestor (including common authors), it is not surprising that a number of principles are common to both approaches. In particular, principles (b) and (c), related to the use of ontologies and separating the descriptions of Web service consumer and Web service provider, are shared.

4. THE IRS-III FRAMEWORK

The IRS-III framework has been designed to fulfil the design principles outlined in Section 3. In this section we describe in detail the elements of the IRS-III framework as motivated from the principles. We start with a description of the IRS-III underlying language and reasoning system. We then describe the IRS-III service ontology, which includes commonalities and differences with the WSMO conceptual model. Finally, we describe the main components of the IRS-III framework including the *IRS-III Server,* the *IRS-III Publishing Platform* and a number of clients, according to Figure 1.

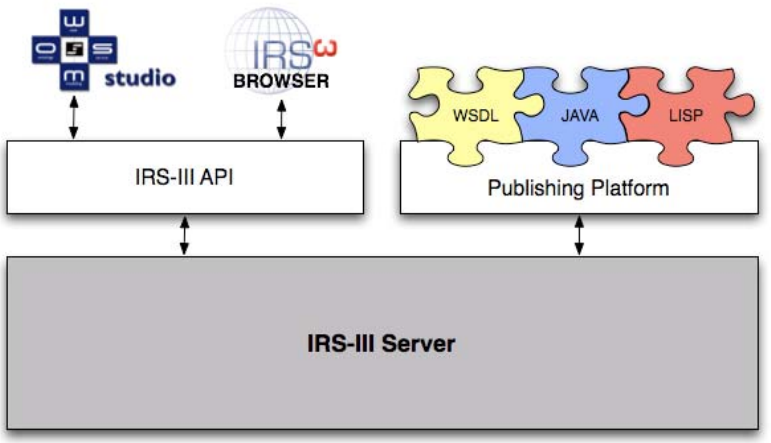

Figure 1. The IRS-III Framework.

4.1. IRS-III Representation Language and Reasoning

As mentioned earlier IRS-III uses OCML [Motta, 1999] for internal representation. The OCML language combines a frame system with a tightly integrated forward and backward chaining rule system and includes constructs for defining: classes, instances, relations, functions, procedures and rules. Additionally, procedures and functions can be attached to Lisp code. This feature allows ontologies related to service descriptions to be attached to our service invocation mechanism thus enabling inferred values to reflect the state of a deployed service (e.g. to retrieve a current exchange rate). The constituent constructs of OCML are tightly integrated. Classes are unary relations and class

attributes are binary relations. Moreover, relations in OCML can be defined using forward or backward chaining rules. The operational semantics of the forward chaining system are equivalent to OPS5 [Forgy, 1981] and the backward chaining rule system has equivalent operational semantics to Prolog [Clocksin and Mellish, 1984]. OCML has been used in a wide variety of projects covering, for example: knowledge management [Domingue and Motta, 2000], online shopping [Domingue et al., 2003] and semantic Web browsing [Dzbor et al., 2007].

The OCML environment incorporates a reasoner, which is central to the IRS-III server (as can be seen in Figure 2). All the major server components use the OCML reasoner and are partly defined using the IRS-III service ontology.

OCML contains import/export facilities to RDF(S) [RDF, 2004; RDF Schema, 2004] and import facilities for OWL [OWL, 2004]. The RDF import and export makes use of a Lisp implementation of Wilbur [Lassila, 2001] and covers all RDF constructs. We are still testing our OWL import facility but it is currently powerful enough to seamlessly import the whole of the DOLCE ontology [Gangemi et al., 2002]. Additionally, the IRS-III API contains a module for translating between OCML based SWS descriptions and WSML based SWS descriptions. This module is able to cope with basic concepts, attributes and instances and future work will investigate the translation of logical expressions. Details on OCML including the semantics of the interpreter and the OCML proof system are given in [Motta, 1999].

4.2. The IRS-III Service Ontology

The IRS-III service ontology, as partially shown in appendices I, II and III, forms the epistemological basis for IRS-III and provides semantic links between the knowledge level components describing SWS and the conditions related to its use. These descriptions are interpreted by the OCML reasoner described in Section 4.1. We describe the commonalities and differences between the service ontology and WSMO and then describe how the service ontology is used within IRS-III.

4.2.1 Commonalities between the IRS-III Service Ontology and WSMO

Motivated by design principles (b) and (c), the IRS-III service ontology contains the following main items, which are also part of the Web Services Modelling Ontology [WSMO, 2007; Fensel et al., 2007].

- *Non-functional properties* – these properties are associated with every main component model and can range from information about the provider such as the organisation's legal address, to information about the service such as category, cost and quality of service, 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* – specifies how to communicate with a Web service. More on our choreography can be found in Section 5.2.
- *Grounding* – associated with the Web service choreography, a grounding describes how the semantic declarations are associated with 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. More on our orchestration work can be found in Section 5.3.
- *Mediators* – a mediator specifies which top elements are connected and which type of mismatches can be resolved between them. Mediators are described in Section 6.

4.2.2 Differences between the IRS-III Service Ontology and WSMO

The differences between our ontology and WSMO are strongly influenced by principles (*d*), (*f*) and (*h*) as described below:

- *Meta-classes for the top-level SWS concepts* – meta-class definitions for goal, mediator and Web service have been defined (see Appendix 1). These classes

have a ‘meta-’ extension (e.g. meta-goal) and enable the IRS-III components to reason over the top-level concepts within the service ontology as first class entities.

- *SWS user definitions as classes* – following from the previous item, we enable users to define the required goals, mediators and Web services as subclasses of the corresponding WSMO concepts rather than as instances. In our view a class better captures the concept of a reusable service description and taxonomic structures can be used to capture the constitution of a particular domain. For example, goals for booking flights may have sub-goals for booking European flights and for booking long-haul flights. A proposal for extending WSMO with goal templates, similar to our goal classes, has been suggested recently [Stollberg and Norton, 2007].
- *SWS invocation contexts as instances* – we reserve instances for invocation. When IRS-III receives a client request, instances of relevant goals, mediators and Web services are created to capture the current invocation context.
- *Explicit input and output role declaration* – in the interests of simplifying the definition of goals and Web services (principle (f)), our ontology incorporates explicit input and output role declarations. The declared input and output types are imported from domain ontologies. This feature enables SWS developers to view goals and Web services as ‘one-shot’ thus minimising the need to consider complex interaction when appropriate. Some of the definitions related to input and output roles are contained in appendices I and II. Section 5.2 describes how IRS-III deals with Web services which have non trivial communication requirements.
- *Orchestration and choreography language* – the representation of our orchestration and choreography, described in Section 5, are defined within the service ontology.
- *Using SWS descriptions for implementing internal components* – following from design principles (b), (e), (f), (h) and (j) we implement several IRS-III internal components using the service ontology and OCML. Our assumption is that IRS-III components described through goals, mediators, and Web

services and through ontological concepts and relations are easier to understand and maintain than if they were implemented purely in a programming language. A small part of the ontology related to describing and implementing IRS-III internal components is contained in appendices II and III.

For illustration purposes, we provide an example of a goal definition below (in listing 1) as specified in IRS-III, for calculating the exchange rate between two currencies. The goal has two input roles: `has-currency-1` and `has-currency-2`, which represent the two currencies for which an exchange rate is required. For each input role within IRS-III we specify the type of values allowed, which can be inherited by Web services linked through a wg-mediator. The output, `has-exchange-rate` is of type `float`. The post-condition represented by an anonymous OCML relation (called a *kappa expression*) states that the output is the current exchange rate between the two currencies.

```
Exchange-rate-goal
  "This goal returns the exchange rate between two currencies."
  Input Role
    has-currency-1 currency "string"
    has-currency-2 currency "string"
  Output Role
    has-exchange-rate float "float"
  Post Condition
    (kappa (goal)
      (== (has-role-value goal has-exchange-rate)
          (the-current-exchange-rate
            (has-role-value goal has-currency-1)
            (has-role-value goal has-currency-2))))
```

Listing 1. An example goal as specified in IRS-III - the `exchange-rate-goal`.

4.2.3 Using the Service Ontology

Before we describe the IRS-III server and its components we first highlight the main ways in which the service ontology is used to implement the core functionalities.

- *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. This feature supports principle (*d*).

- *GG-mediators provide data-flow between sub-goals* – in IRS-III, gg-mediators are used to link sub-goals within an orchestration and so they also provide dataflow between the sub-goals.
- *Web services can inherit from goals* - 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. A small number of the relations related to this feature are shown in Appendix II.
- *Client choreography* – the provider of a Web service must describe the choreography from the viewpoint of the client. Within WSMO the choreography expresses a number of constraints which should not be violated when a deployed service is invoked. Within the IRS-III we evaluate the client choreography in order to interact with the deployed Web service.

Mediation services are goals – a mediator declares a goal as the mediation service which can simply be invoked. The required data transformation is performed by the associated Web service.

4.3 IRS-III Server

As can be seen in Figure 2, the *IRS-III Server* builds upon an *HTTP Server* written in Lisp [Riva and Ramoni, 1996] which has been extended with a SOAP handler. As mentioned above we aim to make IRS-III self-hosting by implementing the main components as far as possible using the service ontology above and OCML (see appendices II and III).

At the heart of the server is the *SWS Library,* where the semantic descriptions associated with Web services are stored using our representation language OCML [Motta, 1998]. The library is structured into domain ontologies and knowledge models for goals, Web services and mediators as described previously (see Section 4.2). Typically our applications consist of mediator models importing from relevant goal and Web service models. Following our design

principle of inspectability (*h*), all information relevant to a Web service is stored explicitly within the library.

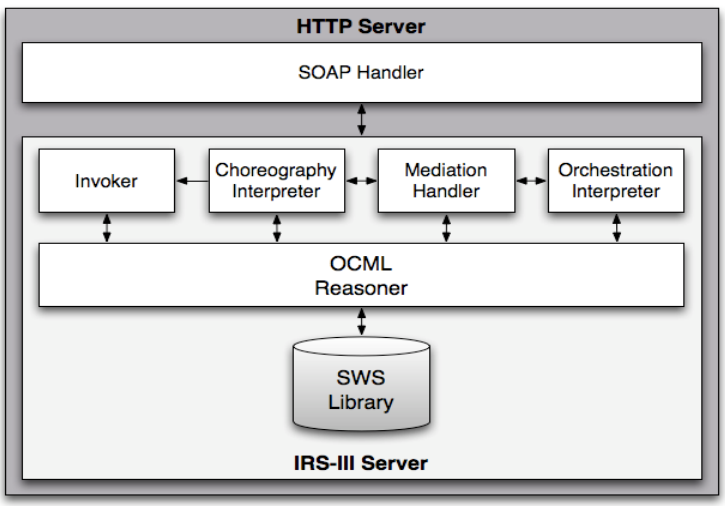

Figure 2. The IRS-III Server.

Within IRS-III, a Web service is associated with an orchestration and choreography definitions. Orchestration specifies the control and data flow of a composite Web service, whereas, choreography specifies how to interact with a single Web service. The choreography component communicates with an invocation module able to generate the required messages in a SOAP format. The functionality of the choreography and orchestration interpreters are described in sections 5.2 and 5.3 of the paper.

A *Mediation Handler* provides functionality to interpret mediator descriptions including running data mediation rules, invoking mediation services and connecting goals and Web services. The mediation component is described in more detail in Section 6.

4.4 The IRS-III Publishing Platforms

Publishing with IRS-III entails associating a deployed Web service with an IRS-III Web service description. When a Web service is published all of the information necessary to call the service - the host, port and path - is stored within the choreography associated with the Web service. Additionally, updates are made to the appropriate *Publishing Platform.* IRS-III contains publishing

platforms to support grounding to stand-alone Java and Lisp code and to Web services. Web applications accessible as HTTP GET requests are handled internally by the IRS-III server.

4.5 The IRS-III Clients and IRS-III API

IRS-III was designed for ease of use (principle f) and, as mentioned earlier, a key feature of IRS-III is that Web service invocation is capability driven. The *IRS-III Browser* supports this by providing a goal-centric invocation mechanism. An IRS-III user simply asks for a goal to be solved and the IRS-III broker locates appropriate Web service semantic descriptions, using the wg-mediators (as described in Section 6) and then invokes the underlying deployed Web services.

The IRS-III API facilitates the integration of our platform with other SWS infrastructures thus supporting design principle (i). Additionally, the IRS-III client and publishing platforms interact with the IRS-III server through the API. Recent work has seen our API aligned with the API of the Semantic Execution Environment standard being defined within OASIS [OASIS, 2006].

5. CHOREOGRAPHY AND ORCHESTRATION IN IRS-III

In this section, we describe in detail how the choreography and orchestration of semantic Web services is implemented in IRS-III. We present a specification for service interaction which formalizes how the functionality of a deployed Web service is achieved, through choreography and orchestration. Choreography focuses on facilitating Web service invocation in a manner which conforms to pre-specified interaction constraints. Within orchestration, the focus is on decomposing a Web Service functionality, into sub-goals which can potentially match against available Web services. At specific points in the brokering of a user request the IRS-III will need to be able to invoke relevant deployed Web services.

5.1 IRS-III Service Interaction Specification

Our overall view is that goal achievement consists of a number of discrete steps, in which, at any given point of the execution, the next action performed will depend upon the current state. Given the above, we adopt the Abstract State

Machine (ASMs) [Börger, 1998] formalism to represent the IRS-III interaction with a client (choreography) or providing Web Services (orchestration).

Abstract State Machines are a mathematical model which provide a parallel action based formal language capable of describing a single agent and multiple agents collaborating in an asynchronous fashion. Moreover, ASMs add the following advantages:

- *Minimal ontological commitment* [Gruber and Olsen, 1994] - the use of ASMs minimizes the ontological commitment for the developer. This principle leads to an increasing of ontological reuse and sharing, particularly useful in our context and related with the IRS-III design principle (b).
- *Comprehensiveness* - ASMs are expressive enough to model all aspects associated with dynamic computation. According with IRS-III principle (h), comprehensiveness alleviates the difficulties arising when information needs to be computed on-the-fly.
- *Formality* - ASMs provide a formal framework for expressing dynamics. Adopting ASMs, for representing service communication and cooperation, meets the need of an explicit formal model claimed in the IRS-III design principle (b).

We describe our service interaction model as the tuple $\langle E, S, C, T \rangle$, where:

- E a finite set of events;
- S the (possibly infinite) set of states;
- C the (possibly infinite) set of conditions;
- T represents the (possibly infinite) set of transitions rules.

The events represent actions performed during the interface execution. The subset of events from E which can occur in choreography and orchestration differ. Specifically, $E = Ec \cup Eo$: where Ec is the set of choreography events; and Eo is the set of orchestration events. In more detail, $Ec = \{$*obtain, present, provide, receive, obtain-initiative, present-initiative*$\}$ [Galizia and Domingue, 2004; Domingue et al. 2005b]. Every choreography event maps to an operation (either WSDL operation, Lisp function or Java method) during the conversation viewed from the IRS-III perspective. Similarly, the set of possible orchestration

events are *Eo* = {*invoke-goal, invoke-mediator, find-mediator, evaluate-logical-expression, return-output*}. While the choreography events are derived from the communication model proposed in KADS [De Greef and Breuker, 1992], orchestration events meet the general goal composition requirements (see the following section).

The main ASM components are states and transition rules. The notion of state is formalised as a classical mathematical abstract structure, where, data are abstract objects, characterized by a signature, comprised of universes, functions, and relations. The universe or domain is a set of data containing partial functions and predicates (attributes and relations) which encapsulate the universe.

Transition rules, in ASMs, are local functions which update the abstract states, and can be expressed as follows:

$f(t1, t2, ..., tn) := t$

where f is an n-ary function, t is a function name, and $t_1, t_2, .., t_n$ are terms.

In our interface model, given a transition step T_i, a state $s_i \in S$ is a non-empty set of ontologies that define a state signature over which transition rules are executed. Optional mediators are used to solve ontology or data mismatches.

The parameterized choreography state is a set of instances, concerning message exchange patterns and the choreography execution. Every state includes a constant subset, which identifies the Web service host, port, and location, which is invariant whenever the same Web service is invoked, and the event instantiation $e \in Ec$, dependent on the event which occurred at step Ti.

The orchestration states characterize the phases of the workflow process during goal composition. Given a transition step Ti, an orchestration state contains a description of the triggering-event, the control flow step identifier, and the result - the output of the achieved sub-goal.

A condition $c \in C$ (also called guard) depicts a situation occurring during interface execution. Every constraint within the condition has to be verified before the next event is triggered.

Transition rules express changes of state by modifying a set of instances within the state signature. In particular, a transition rule, $t \in T$, updates the state

after the occurrence of an event, $e \in E$, and consists of a function, $t : \left(S, 2^C\right) \underset{E}{\rightarrow} S$, that associates a pair $(s, \{c_1, .., c_n\})$ to s', where s and $s' \in S$, and every $c_i \in C \ (1 \leq i \leq n)$.

5.2 Choreography Implementation

Choreography addresses the problem of communication between a client and a Web service. Since the IRS-III acts as a broker the focus of our choreography work is between the IRS-III and the relevant deployed Web services. We assume that IRS-III clients are able to formulate their request as a goal instance. This means that we only require choreographies between the IRS-III and the deployed Web services. Our choreography descriptions are therefore written from the perspective of the IRS-III as a client of the Web service.

A choreography is described in IRS-III by the declaration of a *grounding* and a set of *guarded transitions*. The grounding specifies the conceptual representation of the operations involved in the invocation of a Web service and their mapping to the implementation level. More specifically, the grounding definitions include *operation-name*, *input-roles-soap-binding*, and *output-role-soap-binding*. Guarded transitions can be seen as ASM transition rules as above with two specific restrictions: a) '***If***' rules do not chain and are of the form "***If*** *condition* ***then Fire*** *Event*"; and b) conditions are mutually exclusive so only one rule can fire at a time. These represent the interaction between IRS-III and the Web service and are applied when executing the choreography. This model is executed at a semantic level when IRS-III receives a request to achieve a goal.

Our overall view is that any message sent by IRS-III to a Web service will depend on its current state, which will include a representation of the messages received during the current conversation.

As mentioned earlier we classify communication in IRS-III choreography according to two dimensions, following the system-client cooperation model proposed in KADS [De Greef and Breuker, 1992], namely: 1) the initiative in the communication; and 2) the direction of the communication [Galizia and Domingue, 2004]. The initiative dimension expresses which actor, either the

IRS-III or the Web service, is responsible for starting the communication, while the direction represents the communication route, which can be from the system to the client or vice-versa.

5.2.1 Choreography primitives

As mentioned in Section 4.2.2 a set of primitives have been included within IRS-III service ontology; among them, we have defined a set of choreography specific primitives, specialising the events listed above, which can be used in transition rules. Our primitives provide an easy to use interface to control a conversation between IRS-III and a Web service. Developers are also able to include any relation defined with the imported ontologies within guarded transition specifications.

Init-choreography. Initializes the state of the choreography. This primitive runs before a Web service is invoked by IRS-III. At this point IRS-III has the initiative and is ready to start the communication.

Send-message. Calls a specific operation in the Web service. If no inputs are explicitly given, IRS-III uses the input values from the original goal invocation.

The type of event which occurs with send-message is *present* (see Section 5.1) since IRS-III holds the initiative and the communication direction is from IRS-III to the Web service.

Send-suspend. Suspends the communication between IRS-III and the Web service, without stopping the choreography execution. This action will occur, for example, when the IRS-III lacks some data required by a Web service. Executing this primitive suspends the dialog and stores the current state so that communication can be resumed later. The event associated to send-suspend is *present* since communication direction is from IRS-III to the Web service and the IRS-III has (and keeps) the initiative.

Received-suspend. The communication is suspended by the Web service, when for some reason it is not able to respond to an invocation. As with send-suspend the choreography execution is put on hold. The Web service is free to resume the dialog when conditions allow. The event occurring here is *receive*, because the

Web service has taken the initiative from IRS-III and the communication direction is from the Web service to IRS-III.

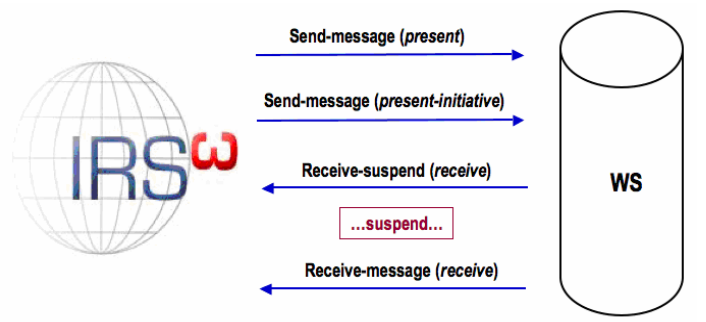

Figure 3. The choreography primitives and events arising when a Web service suspends communication.

Figure 3 shows choreography primitives and events which occur when a Web service suspends communication. Initially IRS-III has initiative, but it is handed over to the Web service, by executing the primitive send-message and respectively triggering the event *present-initiative*. The Web service suspends the communication through the event *receive*. The corresponding IRS-III primitive is receive-suspend. When the Web service resumes the dialog the associated event is *receive* again, because the Web service still has the initiative, and the executed primitive is receive-message. Both the primitives send-suspend and receive-suspend can generate a time-out error received from the Web service. That is managed by the primitive receive-error, described below. IRS-III does not have its own time-out management system.

Received-message. Contains the result of a successful send-message for a specific operation. In the general case the trigged event is *obtain*, if however the Web service had previously suspended the communication it will be *receive* (see Figure 3). In both situations the message direction is from the Web service to IRS-III, but in the former, IRS-III has the initiative, and in the latter the Web service has control of the dialog.

Received-error. If the execution of a Web service causes an error, then the received-error primitive is used. The parameters of received-error include the error message and the type of error which occurred. Time-out is a possible error

which can occur. In a fashion similar to received-message, described above, the event taking place is either *obtain* or *receive*.

End-choreography. Stops the choreography. No other guarded transitions will be executed.

5.2.2 Choreography execution

The IRS-III uses the OCML forward-chaining-rule engine (Section 4.1) to execute a choreography. This means that rules belonging to a choreography are fired according to the state. One important feature of the execution environment of IRS-III is that it allows the scope of the choreography to be defined for the set of ontologies involved in the Web service description.

The IRS-III server carries out inferences at an ontological level. As mentioned earlier, the IRS-III components are specified as combination of an ontological meta layer and WSMO definitions. During communication with a Web service, the ontological level descriptions need to be mapped to the XML based representations used by the specific Web service invoked. We provide two mechanisms which map: a) from the ontological level to XML (lower); and, b) from XML to the ontological level (lift).

The *Lift* construct lifts an XML string to an ontological relation, represented in OCML. A generic version of this relation is defined within the IRS-III ontology. SWS developers are free to overwrite this relation inline with the relationship between the results of Web service calls and the ontologies used. The lift primitive has the following input parameters: class-name, web-service-class, xml-string and produces an instance of class-name as output. The semantic developer can thus customize how XML is parsed according to the classes within the underlying ontology and the particular Web services selected. In order to cope with input in XML format the lift primitive utilizes an inbuilt SAX based XML parser.

The *Lower* construct lowers ontological elements to XML. The input parameters to lower are: instance-name and a Web service class. The output is xml-string. As for the lift primitive, the XML generated can be customized according to classes within the ontology and the Web service class. For example,

the XML generated for instances of a person class may include a full name for one Web service and only a family name for another.

The lifting and lowering definitions are created manually at design time. Our mechanisms overlap with XSLT transformations in that we have defined a number of XML access and XML query functions which are accessible from the IRS-III service ontology. The main difference is based upon the fact that our transformations can make use of the semantic Web service descriptions. For example, the type and structure of instances created from XML in a lifting phase, can be based upon rules. Also, as mentioned above, the serialization produced in the lowering phase depends on the class of the target instance.

5.2.3 Choreography Example

Our choreography example is based on the application illustrated in Section 8. Particularly, we refer to the choreography description of the `Gis-Filter-WS` Web service depicted in Figure 9. This service receives as input a list of possible accommodation centres for an emergency (e.g. hospitals, inns, rest centres, etc.), and it returns a filtered list according to the specific situation. For each kind of accommodation, a specific filtering operation is available.

`Gis-Filter-Web-Service-Interface-Choreography`

`Grounding:`

```
  grounded-to-lisp normal gis-filter
                   has-gis-data "sexpr"
                   "sexpr"
  grounded-to-lisp filter-hospitals gis-internal-filter-hospitals
                   has-gis-data "sexpr"
                   "sexpr"
  grounded-to-lisp filter-inns gis-internal-filter-inns
                   has-gis-data "sexpr"
                   "sexpr"

                   ... omitted ...

  grounded-to-lisp acknowledge-error acknowledge-error-message
                   has-acknowledgement "int"
                   "string"
```

`Guarded-transitions:`

```
  start
    init-choreography
   then
    send-message 'normal

  exec-filter-hospitals
    received-message normal ?result
```

```
   hospital ?result
  then
   send-message 'filter-hospitals
   end-choreography

exec-filter-inns
   received-message normal ?result
   inn ?result
  then
   send-message 'filter-inns
   end-choreography

    ... omitted ...

gis-data-error-transition
   received-error normal ?error-message ?error-type
   gis-data-type-error ?error-type
  then
   send-message-with-new-input-role-pairs
        'acknowledge-error has-acknowledgement 0
   end-choreography
```

Listing 2. The `Gis-Filter-Web-Service-Interface-Choreography` example.

In Listing 2 we show a portion of the grounding definition. After the operation name the next part of the grounding description contains the name of the implementing component. In this case it is the name of the Lisp function within the Lisp publishing platform. For a standard Web service one would use the name of the operation within the WSDL file, and for a Java implementation it would be the name of the Java class and method. The soap bindings for the inputs and output are then specified.

The second part of the choreography contains the set of guarded transitions. Above we show four guarded transitions. `Start` initializes the choreography session and then invokes the deployed service by sending the message associated with the `normal` operation. `Send-message` takes the values of the input roles from the associated goal instance, transforms the values to an XML representation (using *lower*), and then invokes the Web service. `Exec-filter-hospitals` and `exec-filter-inns` use the choreography specific `received-message` relation. Responses from a Web service invocation are first transformed into an ontological representation, using the relation *lift*, and then asserted as (`received-message <operation-name> <lifted-invocation-response>`). The following expressions in the condition check whether the result of the invocation is the expected accommodation type (e.g. a

list of either hospitals or inns). The executive part of the guarded transition sends a message related to the respective operations and ends the choreography.
The final guarded transition shown, `gis-data-error-transition`, handles data type errors. When this is the case the acknowledge-error operation is invoked. Every guarded transition execution updates the choreography state.

5.3 Orchestration Implementation

An orchestration formalism and supporting architecture components should support a range of tasks related to the definition of service control and data flow. These tasks will include: the creation of an orchestration by a developer; the execution of an orchestration; visualizing an orchestration definition; reasoning about behaviour; and conformance testing (against for example a Web service choreography). Our work up until now has concentrated on the first three tasks whilst ongoing work, [Norton, 2007], is investigating the remaining tasks.

In IRS-III, the orchestration is used to describe the model of 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 the resolution of mismatches between goals is supported by mediators.

We have found that our work on orchestration has led to a number of composition specific requirements which we list below.

- *Goal centric composition* - following from the top level design principle of capability based invocation (d). The basic unit within composition is a goal. We thus provide control and data flow constructs over sets of goals.
- *Invocation is one-shot* - currently, we assume that when a goal is invoked the result is returned and there is no direct interaction between any of the underlying Web services involved. A corollary of this principle is that all communication is mediated by IRS-III. Thus a dialog between two Web services becomes a combination of an appropriate loop construct and a pair of IRS-III to Web service choreographies.

- *Orchestration is layered* - no one representation can fully support the full range of tasks for which an orchestration will be used. Within the IRS-III we are thus creating a number of layers each of which support a specific set of activities.

We provide design-time compositional support through a simple goal discovery tool. A simple form enables goals to be found according to a variety of properties including the type of input or output role and non functional properties.

5.3.1 Orchestration primitives

Layered on top of ASMs, we provide a set of control flow primitives which have been implemented so far in IRS-III as 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 argument given as the result of the current context or orchestration.

5.3.2 Orchestration Example

A full example which includes orchestration is given in Section 8, with the semantic descriptions described in Section 8.2.4. We refer to Figure 9 which shows how an orchestration is formed therein over three subgoals. The orch-sequence primitive, described above, is used to link the three subgoals in a control flow, and four gg-mediators are used to define their control flow:

```
Get-polygon-gis-data-with-filter-ws-interface-orchestartion
  has-problem-solving-pattern
```

```
    get-polygon-gis-data-with-filter-ws-interface-orchestartion-psp

Get-polygon-gis-data-with-filter-ws-interface-orchestartion-psp
  (problem-solving-pattern)
  has-body
    orch-seq
      convert-polygon-points-goal
      get-circle-gis-goal
      gis-filter-goal

Polygon-to-circle-radius-gg-mediator
  (gg-mediator)
  has-source-component convert-polygon-points-goal
  has-target-component get-circle-gis-data-goal
  has-mediation-service polygon-to-circle-radius-goal
```

Listing 3. An orchestration example - `get-polygon-gis-data-with-filter-ws-interface-orchestration` - taken from the eGovernment example in Section 8.

6. MEDIATION

Our mediation approach consists of modelling specialized mediators which provide a mediation service or declarative mappings for solving different types of conceptual mismatches. The mediation handler interprets each type of mediator accordingly during selection, invocation and orchestration. The mediator models are created at design time and used at runtime.

6.1. Meditation Specification

In the following we present a specification for mediation based on the example in Figure 4. Here, we will refer to Web service and mediator descriptions independently from their implementations.

Mediator descriptions *M* between ontologies *O*, Web service descriptions *WS* and goal descriptions *G* can be defined according to the following cases:

a. Two different Web services, say WS_1 and WS_3, can access different data, say *x* and *y*, denoted by different concepts, say *Customer* and *Client*, through different ontologies, say O_1 and O_3. We view this as a standard case in the context of the semantic Web.

b. Two different Web services, say WS_1 and WS_2, can access different data, say *x* and *y*, but denote the same concept, say *Customer*, by sharing the same ontology, say O_1.

c. Two different Web services, say WS_2 and WS_3, can access the same data, say *y*, but denote two different concepts, say *Customer* and *Client*, by using different ontologies, say O_1 and O_3.

d. A Web service, say WS_1 with an input/output parameter of an arbitrary type, say *Customer* may be able to achieve a goal, say G_1, with an input/output parameter of arbitrary type, say *Citizen*.

e. Two Web services, say WS_2 and WS_3, may be combined (e.g. through a sequence workflow construct) to provide a composed Web service functionality, which may be able to achieve a goal, say G_1.

f. Two goals (not shown) may also be combined to provide a composed Web service functionality, which may be able to achieve a goal, say G_1.

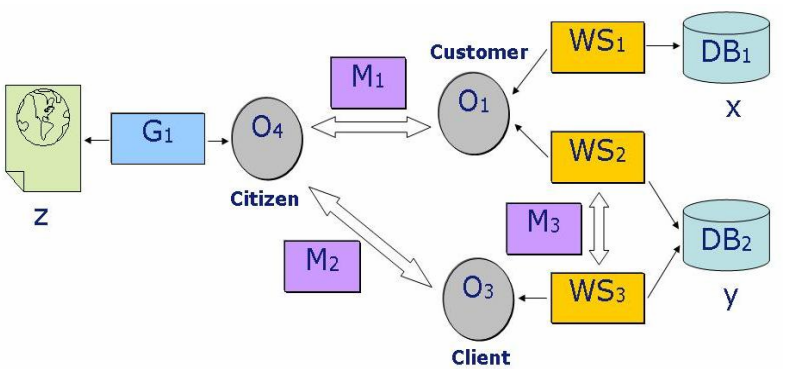

Figure 4. An example of mediation in the context of SWS

Our solution to the above would be comprised of five types of mediators of the form ${}_{S}M_{T}$, where *S* is the source of the mediator and T the target of the mediator as follows:

1. ${}_{Op}M_{Oq}$ mediates between instances of Op and instances of Oq, which can be associated with a goal or Web service.

2. $_{Gp}M_{WSq}$ mediates between input/request elements of Gp denoted by G_p^{IN} and input/request elements of WSq, denoted by WS_q^{IN}.
3. $_{WSq}M_{Gp}$ mediates between output/response elements of WSq denoted by WS_q^{OUT} and output/response elements of Gp, denoted by G_p^{OUT}.
4. $_{Gp}M_{Gq}$ mediates between output/response elements of Gp, denoted by G_p^{OUT}, and input/request elements of Gq, denoted by G_q^{IN}.
5. $_{WSp}M_{WSq}$ mediates between output/response elements of WSp, denoted by WS_p^{OUT} and input/request elements of WSq, denoted by WS_q^{IN}.

We can associate a relation mapping (*MAP*) to the $_{Op}M_{Oq}$ mediator above in order to map between instances of ontologies *Op* and *Oq*. *MAP* can be modelled as logical rules, which can be generated with the support of a design-time tool. The remaining types of mediators can be associated with a mediation function (MF), which can be modelled just as a standard semantic Web service in order to perform transformations between inputs and outputs. As indicated above, these mediators can provide mediated data-flow between a goal and a Web service, and between Web services or goals. Thus, *MF* and *MAP* are executable components to be used at runtime.

6.2. Mediation Implementation in IRS-III

In IRS-III we represent the types of mediators defined above using the WSMO-based models of oo-mediator, wg-mediator, gg-mediator and ww-mediator (as shown in Appendix I). As we stated in Section 4.2, our model includes meta-classes for the top-level components, which also includes meta classes of the main mediators. 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 resolve mismatches between the two. In this model, the mediation service is just another goal. For example, a mediation service in a wg-mediator transforms the input

values coming from the source goal into an input value used by the target Web service (see also [Cabral and Domingue, 2005]).

6.2.1. Mapping rules

Mapping rules (MAP in the specification above) are used between two ontologies (source and target components). These mappings target the concepts used during invocation and consist of three main mapping primitives:

- **Maps-to**. A relation created internally for every mapped instance.
- **Def-concept-mapping**. Generates the mappings, specified with the *maps-to* relation, between two ontological concepts.
- **Def-relation-mapping**. Generates 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.

Listing 3 shows an example of a mapping rule and how it has been used to link the slots of classes in two different ontologies. More specifically, the definitions below link the `has-citizen-name` slot of class `citizen` in the source ontology to the *`has-client-name`* slot of class *`client`* in the target ontology. The `def-concept-mapping` construct associates each instance of the *`citizen`* class to a newly created instance of the *`client`* class and links them by generating instances of the relation `maps-to` internally. The `def-relation-mapping` construct uses the generated `maps-to` relation instances within a rule which asserts the value of the mapped citizen name to the value of the client name.

IRS-III executes the mapping rules within a temporary ontology created by merging the source and target ontologies. The temporary ontology is then discarded after the Web service invocation.

```
def-concept-mapping citizen client

def-relation-mapping citizen-client-name-mapping
  (has-client-name ?client ?value)
 if
  (maps-to ?client ?citizen)
  (has-citizen-name ?citizen ?value)
```

Listing 3. An example of a mapping rule mapping between citizen and client concepts.

6.2.2. Mediation Services

Wg-mediators, gg-mediators and ww-mediators have a data mediation capacity for transforming inputs between source and target components by using mediation services (MF, in the specification above) and have different roles within the process mediation as explained next.

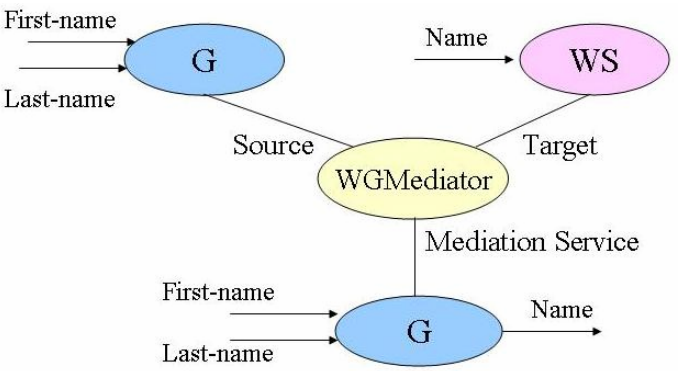

Figure 5. Mediation between a goal and a Web service. Two inputs of a goal are transformed into one input of the target Web service

Figure 5 shows a graphical illustration of mediation taking place between a goal and a Web service via a wg-mediator. In this example, the goal requested by the application takes two inputs (first and last names), which are transformed by the mediation service into one input (name) used by the target Web service. A specific example of a gg-mediator in the context of orchestration can be see in section 5.3.3.

6.2.3. The mediation handler

The overall design goal for IRS-III is to act as a semantic broker between a client application and deployed Web services available at large on the Internet. This brokering activity can be seen as mediation itself, which within IRS-III is further broken down into data, goal and process mediation, each supported by a specific module in the architecture. The IRS-III mediation handler and its relationship to the other main IRS-III components and semantic descriptions are shown in Figure 6.

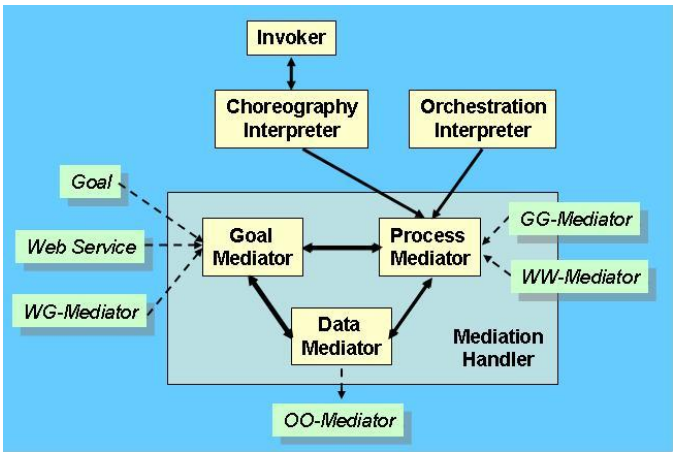

Figure 6. The IRS-III mediation hander. Components are labelled in bold and semantic descriptions are labelled in italics.

The steps below describe the overall sequence of mediation activities taking place during the selection, composition and invocation of semantic Web services.

1. The *Goal Mediator* component searches for wg-mediators whose source component matches the current goal request from a client application. It selects a Web service which matches the requested capabilities (input types, preconditions, assumptions, non-functional properties etc). The types of mismatches that can occur are: a) the input types of a goal are different from the input types of the target Web service; and b) Web services have more inputs than the goal.
2. The *Process Mediator* component establishes an interaction with a deployed Web service by executing its Web service client choreography. The *Process Mediator* creates the communication messages corresponding to the choreography communication primitives. Additionally, the *Process Mediator* keeps the state of the communication throughout the sequence of operation calls executed by the Invoker component.
3. The *Process Mediator* can also execute the orchestration of a composite Web service. Here the *Process Mediator* keeps the state of the orchestration (control and data flow) between the invocations of sub-goals. Additionally, the *Process Mediator* searches for gg-mediators connecting sub-goals in the orchestration. The presence of gg-mediators indicate that dataflow will occur between the

sub-goals with mismatches resolved by the declared mediation service. The types of mismatches which can occur are: a) output types of a sub-goal are different from the input types of the target sub-goal; b) output values of a sub-goal are in a different order from the inputs of the target sub-goal; and, c) the output of a sub-goal has to be split or concatenated into the inputs of the target sub-goals.

4. The *Data Mediator* component is used by the *Goal Mediator* and by the *Process Mediator* to map data across domain ontologies. Mapping rules declared within oo-mediators are executed to achieve the desired mapping.

7. CREATING SEMANTIC WEB SERVICE BASED APPLICATIONS

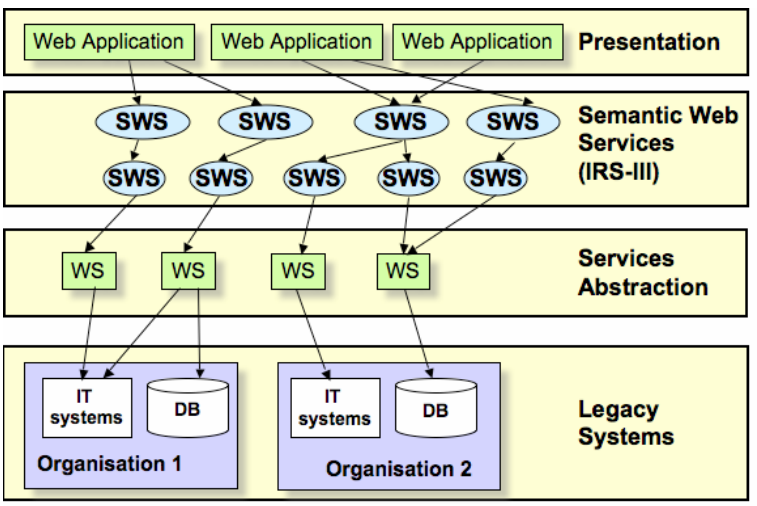

Figure 7. The generic architecture used when creating IRS-III based applications.

Our generic application architecture is depicted in Figure 7. As can be seen, the architecture is composed of four layers and enables collaboration between one or more stakeholders in a distributed fashion. In particular, our approach 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 IRS-III infrastructure. From the bottom up the four application layers are:

- *Legacy system layer* - consists of the existing data sources and IT systems available from each of the parties involved in the integrated application.

- *Service abstraction layer* - exposes the (micro-)functionality of the legacy systems as Web services, abstracting from the hardware and software platforms. In general existing Enterprise Application Integration (EAI) software will facilitate the creation of the required Web services. Note that for standard databases the necessary functionality of the Web services can simply be implemented as SQL query functions.
- *Semantic Web services layer* – given a goal request, this layer, implemented in IRS-III, will: a) discover a candidate set of services; b) select the most appropriate; c) resolve any mismatches at the data, ontological or process level; and d) invoke the relevant set of Web services satisfying any data, control flow and invocation requirements. To achieve this, IRS-III, utilises the set of semantic Web service descriptions which are composed of goals, mediators, and Web services, supported by relevant ontologies.
- *Presentation layer* – a Web application accessible through a standard Web browser which is built upon the semantic Web services layer. The goals defined within the semantic Web services layer are reflected in the structure of the interface and can be invoked either through the IRS-III API or as an HTTP GET request. We should emphasise that the presentation layer may be comprised of a set of Web applications to support different user communities. In this case each community would be represented by a set of goals supported by community related ontologies.

In order to successfully create applications from semantic Web services as depicted in Figure 7 above four key activities need to be carried out as follows:

1. *Requirements capture* – the requirements for the overall application are captured using standard software engineering methodologies and tools. We do not advocate any particular requirements capture method, but envisage that the resulting documents describe the stakeholders, the main users and roles, any potential providers for Web services, and any requirements on the deployed infrastructure and interfaces.
2. *Goal description* – using the requirements documents above, relevant goals are identified and described in IRS-III. During this process any required

supporting domain ontologies will either be created from scratch or existing ontologies will be re-used.

3. *Web service description* – descriptions of relevant Web services are created within IRS-III. Again, any domain ontologies required to support the Web service descriptions are either defined or re-used as necessary.
4. *Mediator description* – mismatches between the ontologies used, and mismatches within and between the formal goal and Web service descriptions are identified and appropriate mediators created (see Section 6).

All of the above steps are carried out by the *SWS application developer.* The first two steps are *user/client* centric and therefore involve discussions with the relevant client stakeholders, whereas Step 3 will require dialogue with the Web service providers. Steps 2 and 3 are mostly independent and in the future we expect libraries of goals and Web services to become generally available to support re-use.

In the next section we illustrate how we used the general application development approach to build an eGovernment application for emergency planning.

8. AN IRS-III EGOVERNMENT APPLICATION

In general, governmental agencies will operate autonomously under different tiers of government with no central control. For example, the scenario below involves national, county and district level agencies. IT based collaboration in these circumstances where distributed heterogeneous software platforms need to interoperate, in terms of data and processes, without a central control regime provides a natural application area for SWS technology. Additionally, government agencies will each have their own distinct viewpoints which will differ again from the general citizen. The ability to aggregate and re-use diverse information resources relevant to a given situation and further to make this available as a basis for transparent interaction between community partner organisations and individual citizens is a key benefit that SWS technology can provide.

In the last few years a number of projects have applied SWS technology in the eGovernment domain, but only a few among them show reusability and composability in a real usage scenario [Medjahed and Bouguettaya, 2005; Medjahed, 2005]. Our application, developed within the context of the EU funded DIP project [DIP, 2004], integrates a diverse set of Web services into an easy-to-use Web based interface.

8.1 Application Scenario and requirements

The overall context of the application is Essex County Council (ECC). ECC is a large local authority in South East England (UK) comprised of 13 boroughs and containing a population of 1.3M. Following a number of initial interviews with a variety of stakeholders holders in ECC it was decided to focus the scenario on the ECC Emergency Planning department, and more concretely to focus on emergencies which arise from extreme weather conditions. The Emergency Management System (EMS) application, called eMerges [Tanasescu et al. 2007], is a decision support system which assists an Emergency Planning Officer (EPO), in gathering information related to an extreme weather emergency.

8.2 Architecture

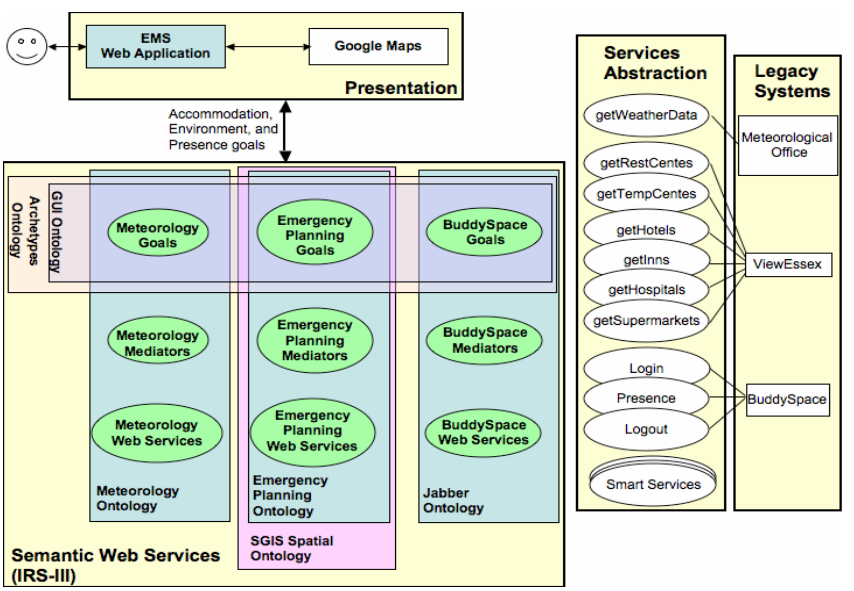

Figure 8. The architecture for the Emergency Planning Management System which follows the 4-layered approach described in Section 8.

As shown in the Figure 8, based upon the generic application framework introduced in the previous section, we developed an application architecture comprised of the following four layers.

8.2.1 Legacy system layer

The EMS aggregates data and functionalities from three different sources:

- *Meteorological (MET) Office* – a national UK organisation which provides environmental resources and in particular weather forecast data.
- *ViewEssex* - a collaboration between ECC and British Telecom (BT) which has created a single corporate spatial data warehouse. As can be expected ViewEssex contains a wide range of data including data for roads, administrative boundaries, buildings, and Ordnance Survey maps, as well as environmental and social care data. Within the application we used building related data to support searches for suitable rest centres.
- *BuddySpace* - an instant messaging client facilitating lightweight communication, collaboration, and presence management [Eisenstadt et al., 2003] built on top of the Jabber instant messaging protocol [Jabber, 2006]. The BuddySpace client can be accessed on standard PCs, as well as on PDAs and on mobile phones, which in an emergency situation may be the only hardware device available.

8.2.2 Service abstraction layer

We distinguish between two classes of services: *data* and *smart*. The former refer to the three data sources introduced above, and are exposed by means of Web services:

- *Meteorological service* – this service provides weather information (e.g. snowfall) over a specific rectangular spatial area.
- *ECC Emergency Planning services* – using the ViewEssex data each service in this set returns detailed information on a specific type of rest centre within a given circular area. For example, the 'getHospitals' Web service returns a list of relevant hospitals.
- *BuddySpace services* – these services allow presence information for online users to be accessed.

Smart services represent specific emergency planning reasoning and operations on the data provided by the data services. They are implemented in a mixture of OCML and Lisp and make use of the EMS ontologies. In particular, we created a number of *filter services* which select the GIS data according to emergency-specific requirements (e.g. rest centres with heating system, hotels with at least 40 beds, easy accessible hospital, etc.). The criteria used were gained from our discussions with the EPOs and therefore mean that users will only receive information relevant to the specific situation.

8.2.3 Domain Ontologies for the Semantic Web services layer

As we stated in Section 7, the semantic Web services layer is comprised of SWS descriptions within IRS-III. The goals, mediator and Web service definitions use ontologies which reflect the client and provider domains. The following ontologies were developed to support the SWS descriptions.

- *GUI ontology* - is composed of user-interface concepts and is used to display the results of invocations to the IRS-III. The ontology also allows us to abstract over the particular interface that is used. For the EMS application we instantiated the ontology to reflect the Google Maps API.
- *Archetypes ontology* – based on the work of Mark [Mark, 1989], this ontology provides a cognitively plausible description of geographical objects. For example, in addition to being a place where ill people are treated a hospital can be viewed as a 'container' of people and provider of 'shelter' since it shares features with the *archetypal* 'house' concept. It is assumed that any client, whilst maybe lacking the specific knowledge for domain specific concepts, will be familiar with the archetypes contained in this ontology.
- *SGIS spatial ontology* - describes the concepts commonly found in GIS, such as points, spatial objects with attributes, polygons and fields. A field denotes an object with no clear boundary, such as a flood area.
- *Meteorology, ECC Emergency Planning and Jabber domain ontologies* – these ontologies represent the concepts used within the services attached to the data sources, such as 'snow' and 'rain' for the Met Office, 'hospitals' and

'supermarkets' for ECC Emergency Planning, and 'session' and 'presence' for the Jabber services.

The existence of several domain ontologies reflects our principles (b) and (c) (see Section 3), where the different actor viewpoints/terminologies (user and three data providers) are independently represented ontologically.

8.2.4 Semantic Web Service Descriptions

As prescribed in Section 7, the goals, mediators, and Web service descriptions of our application link the Met Office, ECC Emergency Planning, and BuddySpace Web services to the user interface. Correspondingly, the Web service goal descriptions use the SGIS spatial, meteorology, ECC Emergency Planning and Jabber domain ontologies whilst the goal encodings additionally rely on the GUI and archetypes ontologies. Mismatches are resolved by the defined mediators.

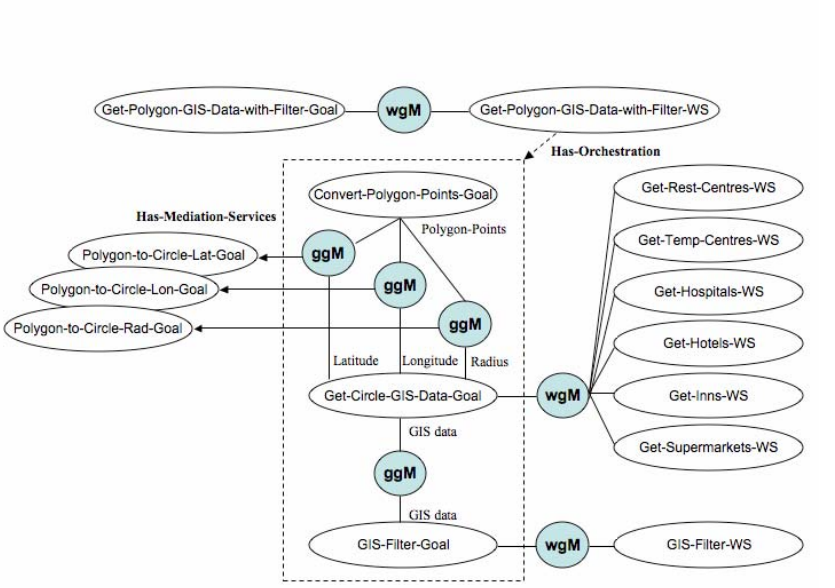

Figure 9. A portion of the WSMO descriptions for the EMS application.

A small portion of the SWS descriptions are shown in Figure 9. `Get-Polygon-GIS-data-with-Filter-Goal` represents a request for available shelters within a delimited area. The user specifies the requirements as a target area, a sequence of at least three points (a polygon), and a shelter type (e.g. hospital, inn or hotel). As mentioned above the set of ECC Emergency Planning

Web services each return potential shelters of a specific type with a circular query area. The obtained results need to be filtered in order to return only shelters correlated to emergency-specific requirements (for example a snowstorm). From an SWS point of view the problems to be solved by this particular portion of the SWS layer included: (a) *selecting* the appropriate ECC Emergency Planning Web service; (b) *meditating* the difference in area representations (polygon vs. circular) between the goal and Web services; and (c) *orchestrating* the retrieve and filter data operations. Below we outline how the SWS representations in Figure 9 addresses these problems.

- *Web service selection* - each SWS description of an ECC Emergency Planning service defines, in its capability, the specific class of shelter that the service provides. Each definition is linked to the `Get-Circle-GIS-Data-Goal` by means of a unique wg-mediator (shown as wgM). The inputs of the goal specify the class of shelter, and the circular query area. At invocation IRS-III discovers through the wg-mediator all associated Web services, and selects one on the basis of the specific class of shelter described in the Web service capability.
- *Area mediation and orchestration* – the `Get-Polygon-GIS-data-with-Filter-Goal` is associated with a unique Web service that orchestrates by simply invoking three sub-goals in sequence. The first gets the list of polygon points from the input; the second is `Get-Circle-GIS-Data-Goal` described above; finally, the third invokes the smart service that filters the list of GIS data. The first two sub-goals are linked by means of three gg-mediators (depicted as ggM) that return the centre, as a latitude and longitude, and radius of the smallest circle which circumscribes the given polygon. To accomplish this, we created three mediation services invoked through: `Polygon-to-Circle-Lat-Goal`, `Polygon-to-Circle-Lon-Goal`, and `Polygon-to-Circle-Rad-Goal` (the related wg-mediator and Web service ovals were omitted to avoid cluttering the diagram). The results of the mediation services and the class of shelter required are provided as inputs to the second sub-goal. A unique gg-mediator connects the output of

the second to the input of the third sub-goal. In this case no mediation service is necessary.

8.2.5 Presentation layer

The prototype implementation is a Web interface using Google Maps for the spatial representation part of the application. The interface is built using the Google Web Toolkit[2], using AJAX techniques on the client to communicate with a Java servlet, which itself connects to IRS-III through its Java API. The most significant component of the interface is a central map, supporting spatial objects. A spatial object can have an area based location, in which case it is displayed as a polygon, or be point based, in which case it is displayed as a symbol. All objects present the same interface, with affordances and features, displayed in a pop up window or in a hovering transparent region.

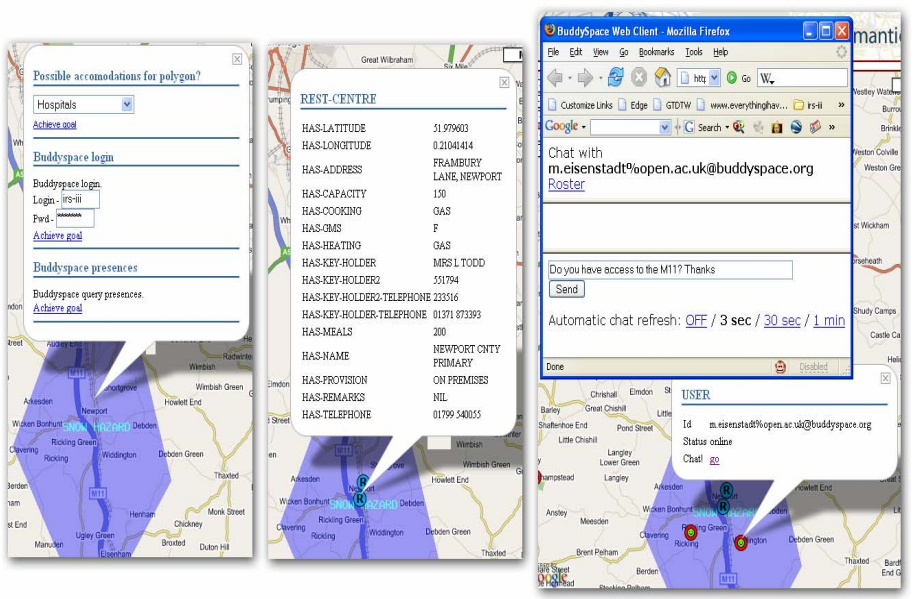

Figure 10. Showing three screenshots of our application in use. 10a) Goals available for the snow hazard, 10b) obtaining detailed information for a specific rest centre, 10c) initiating a discussion with an online emergency worker.

Imagine that an EPO would like to know which rest centres are available within a particular area and then to use this information to contact relevant local agency staff. To achieve this, the EPO would carry out the steps below:

[2] http://code.google.com/webtoolkit/

1. Based on external information the EPO draws a polygon on the map, then, assigns a hazard type to the region. In this case our EPO has selected *snow storm*.
2. The EPO clicks within the displayed hazard region to bring up a menu of available goals. In this case (Figure 10a) three goals are available: show available shelters, login to BuddySpace and get the presence information for related staff.
3. The EPO asks for the available Rest Centres inside the region, and then inspects the detailed attributes for the Rest Centre returned (Figure 10b).
4. The EPO requests to see the presence status for all staff within the region and then starts a discussion the closest online agency worker (Figure 10c).

Since IRS-III SWS integration allows the description of any XML data source available on the Web, the data source integration approach presents notable advantages compared to approaches based on standards such as the one demonstrated in the OWS-3 initiative[3]. The advantages can be summarized as follows:

- *Framework openness* - standards make integration easier but are not mandatory; any other schema can be integrated into the system.
- *High level service support* - all the benefits of the underlying IRS-III SWS platform, such as discovery and composition etc. are immediately available; in other solutions support for discovery and composition is embedded within the syntactic standards themselves, which implies a specific format and adds ad hoc reasoning capabilities to standard software applications, which is time consuming and error prone.

The eMerges Web application is available at http://irs-test.open.ac.uk:8080/EMerges/.

[3] http://www.opengeospatial.org/projects/initiatives/ows-3

9. BENEFITS OF OUR APPROACH

The combination of an SWS-based approach, a meta-model completely captured within an ontology, a comprehensive infrastructure and adherence to agreed APIs provides a number of benefits for our approach.

By adopting SWS, we capture the knowledge associated with the background context together with the requested and provided capabilities of services, and hence support automatic reasoning and reuse. In this way, service invocation, discovery, composition, and mediation can be automated by adopting the best available solutions for a specific request increasing the flexibility, scalability, and maintainability of an application. Particularly, in IRS-III, the actual execution sequence of services is not hard-coded, but it is dynamically created using goal-based discovery and invocation. Several Web services may be associated with a goal, and only the most applicable will be discovered and invoked at runtime (late binding). If a new service becomes available, the developers simply need to describe and then link it to an existing goal. If a service is altered, the specific semantic description will be affected only, and not the whole business process.

The definition of the internal IRS-III mechanisms as a combination of an ontological meta-layer and WSMO definitions provides a non-ambiguous understandable operational semantic definition. Additionally, the substitution of IRS-III components (e.g. for orchestration or mediation) by external services becomes feasible.

Creating and managing ontologies is a time consuming expensive activity which involves: understanding a domain, acquiring and representing knowledge, and populating with instances. Maintaining consistency when a target domain or related resources are altered adds further complications. For example, in complex domains such as eGovernment, centralized ontologies would require an unrealistic development effort with no guarantee of satisfactory results in terms of comprehensively capturing domain knowledge. Moreover, eGovernment domains by their very nature have no central control as multiple agencies stationed at different levels of government, from local district to the national level, are involved. IRS-III provides a comprehensive infrastructure which has

been used in a number of real world settings including eGovernment (described in this paper) and medical imaging [Dupplaw et al., 2004]. The proposed methodology makes the knowledge capture and maintenance process simpler and more efficient in two ways. Firstly, only knowledge directly related to the exposed functionality need be modelled. This minimalist approach also improves the management of the ontology evolution and maintenance phases. Secondly, the knowledge capture effort can be distributed among all of the stakeholders: each party describes – and it is responsible for – its particular domain; in this way, several viewpoints can be independently and concurrently described. Involved parties can also reuse already existing ontologies. Our mediators are able to resolve mismatches among the several viewpoints without the need to alter any parties existing code or service. As a result, we obtain a model that addresses the required lack of central control.

Finally, our multiple publishing platforms introduce two further benefits: a) they significantly lower the barrier in moving from stand alone code to a SWS, and b) they ensure that IRS-III is independent of any particular communication protocol.

10. RELATED WORK

There are a number of existing Semantic Web Services approaches, which we outline in the following, with the common objective of automating the tasks of Web service discovery, selection, composition, mediation and invocation. IRS-III distinguishes itself from these approaches mainly because it relies on a knowledge-based integrated environment for modelling, reasoning and execution. We take advantage of the representational and reasoning power of OCML (see Section 4.1), not only for modelling ontologically the semantic descriptions of goals, Web Services and mediators, but also for implementing (with the support of Lisp) many of the components that perform selection, choreography, orchestration and mediation. More specifically, IRS-III comprises a combination of: a) an explicit representation of the IRS-III Service Ontology; b) an OCML meta-layer supporting reasoning over goal, mediator and Web

service classes (see Section 4.2.2 and appendices I and II; and c) OCML relations supporting the description and execution of internal components (see Appendix III).

IRS-III is an extension of the previous IRS-II framework [Motta et al., 2003], which is based on an early approach to providing reusable components over the Internet, namely the UPML framework [Fensel and Motta, 2001], funded by the European Commission within the IBROW project [Benjamins et al., 1998]. The UPML framework supported the semi-automatic construction of knowledge intensive applications by structuring reusable knowledge components into tasks, problem solving methods and domain models. Bridges were used to connect knowledge models of different types. In IRS-I [Crubezy et al., 2002], the UPML framework was used to broker between task based requests and a library of problem solving methods, and within IRS-II problem solving methods were linked to deployed Web services. In IRS-III we have adapted our platform to be WSMO compliant and focused on Web service specific issues.

The work most closely related to our approach is WSMX [WSMX, 2005], the reference implementation of WSMO with which we share a common API (currently standardised through OASIS). WSMX is an open source service oriented architecture, which uses a de-coupled reasoning service. Unlike IRS-III, the WSMO conceptual model, which is defined using the OMG Meta-Object Facility (MOF) [OMG, 2002], is not contained within WSMX, and thus can not be combined with ontology instances represented in the WSMO reference language WSML [WSML, 2005].

The OWL-S approach [OWL-S, 2006] defines an upper ontology for semantically describing Web services and is comprised of three top-level elements: service profile, service model and service grounding. The core functional description of services in OWL-S is contained in the service profile. A service is described mainly in terms of its functional parameters: inputs, outputs, preconditions and effects (IOPEs). The OWL-S service model describes services behaviourally through a process model which divides processes into two types: atomic processes and composite process. Composite processes are specified

through a pre-defined set of control structures. The main differences between the OWL-S and IRS-III approaches are mainly derived from the differences between OWL-S and WSMO. The OWL-S profile is comparable to WSMO capabilities and non-functional properties and the OWL-S process model is equivalent to orchestration in WSMO. However, OWL-S does not define Web service choreographies and has no explicit notion of a goal or mediator. Within OWL-S, mediation is considered to be handled during discovery and decomposition through architectural components and mediation services are treated as 'standard' Web services. In contrast WSMO based approaches regard the mediation role as a first class citizen. In addition, OWL-S differs from IRS-III by not having an integrated architecture. However, a number of OWL-S related tools have been implemented such as the OWL-S plug-in for Protégé [OWL-S Tools, 2006], the OWL-S Web service Matcher[4] (OWLSM), and an OWL-S Plug-in for Axis[5]. The core OWL-S framework is the OWL-Virtual Machine (former DAML-S VM) [Paolucci et al., 2003], a general purpose Web service client which relies on the OWL-S process model and grounding to support interaction between OWL-S Web services.

Another related SWS approach is called WSDL-S and which was developed within the METEOR-S project [METEOR-S, 2006]. WSDL-S mainly differs from the previous mentioned frameworks because it follows a bottom-up approach for semantically describing Web services. This approach defines WSDL extensions for, representing preconditions, effects and data mappings by linking WSDL elements to external ontologies. Part of this work is now taking place with the Semantic Annotations for Web services Description Language W3C working group [SAWSDL, 2006]. Following on the bottom-up approach within the project, a composition framework MWSCF [Sivashanmugam et al., 2005] has been developed, which uses the semantic descriptions for fulfilling workflow-based processes automatically.

[4] http://owlsm.projects.semwebcentral.org/
[5] http://ivs.tu-berlin.de/Projekte/owlsplugin/

Regarding SWS choreography and orchestration, similarly to WSMO we follow the ASM formalism, but differ in the way we represent and handle the semantic descriptions. IRS-III makes use of a forward-chaining-rule engine to execute a choreography. The orchestration handler uses a Lisp interpreter to execute the workflow-based orchestration. Other known standards for Web services choreography are not easily comparable with our model because the descriptions are syntactic. W3C glossary [W3C, 2004b], states simply that a Web service choreography concerns the interaction of services with their users. The Web service choreography Description Language (WS-CDL) describes the behaviour observable from an external point of view, emphasizing collaboration amongst interested parties, where the communication progresses only when jointly agreed ordering rules are satisfied [Kavantzas, 2004]. Dijman and Dumas [Dijman and Dumas, 2005] depict both static and dynamic aspects of the global communication among heterogeneous Web services using Petri Nets.

11. CONCLUSIONS AND FUTURE WORK

Semantic Web services research has the overall vision of bringing the Web to its full potential by enabling applications to be created automatically from available Web services in order to satisfy user goals. Fulfilling this vision will radically change the character of all online interaction including the nature of e-Commerce, e-Science, e-Learning, and eGovernment. Key to achieving this vision is the provision of SWS platforms able to support the development and use of online libraries of re-usable software components indexed through generic and domain specific ontologies. In this paper we have presented our SWS platform IRS-III, which contains a suite of tools to enable the development and management of semantic descriptions. Using the semantic Web service descriptions, IRS-III, through orchestration, mediation and choreography components, can broker between incoming goal requests and applicable Web services. Also, IRS-III is to a large extent self-descriptive from the ability to use semantic relations and internal goals during the brokering process.

Further work is under way [Norton et al., 2007] in order to extend our orchestration representation and integrate this with UML Activity Diagram based workflows to facilitate automatic orchestration generation and take-up by the general software developer community.

IRS-III also forms a significant input to the OASIS Semantic Execution Environment standardisation process. Ongoing work involves both WSMX and IRS-III and is focused on creating an OASIS standard execution environment for semantic Web services [OASIS SEE TC, 2006].

Recently we have developed a plug-in which integrates IRS-III into WSMO Studio [WSMO Studio, 2006], a WSMO compliant semantic Web service editor available as a set of Eclipse plug-ins which facilitates reusability and extension by third parties. The plug-in enables WSMO entities to be transparently translated between WSML and OCML. Additionally, the plug-in allows users to achieve goals through IRS-III using a simple point-and-click interface.

Over the past few years we have used IRS-III to create a number of SWS based applications and we described our overall approach in this paper through an application within the eGovernment domain which we created to support emergency planning. Our application provides a simple interface, based on Google Maps, and integrates GIS, meteorological, presence and ontology based Web services.

Within a number of new EU funded projects we are currently creating applications in the areas of: business process modelling, linking IRS-III to a BPEL engine [BEA Systems et al., 2002] [SUPER, 2006]; e-learning, integrating IRS-III with a learning object repository [LUISA, 2006]; and, bio-informatics, describing Grid services related to the human musculo-skeletal system [LHDL, 2006]. The diversity of the domains in which we are able to deploy IRS-III is evidence of the utility and robustness of our overall approach, and, we fully expect to gain further valuable insights into the overall requirements for semantic Web services during the deployment process. In this respect we welcome external parties to use our platform - the IRS-III API and browser for can be downloaded from the IRS-III Web site at http://kmi.open.ac.uk/projects/irs/.

ACKNOWLEDGEMENTS

This work was supported by the DIP (Data, Information and Process Integration with Semantic Web Services) project. DIP (FP6 - 507483) was an Integrated Project funded under the European Union's IST programme.

The authors gratefully acknowledge the members of the DIP project and the WSMO working group for their insightful comments on our work.

REFERENCES

AMAZON. 2006. Available from http://www.amazon.com/gp/browse.html/104-6906496-9857523?%5Fencoding=UTF8&node=3435361/.

BEA SYSTEMS, IBM CORPORATION, MICROSOFT CORPORATION., SAP AG, SIEBEL SYSTEMS. 2002. Business Process Execution Language for Web Services. http://www.ibm.com/developerworks/webservices/library/ws-bpel

BENJAMINS, V.R., PLAZA, E., MOTTA, E., FENSEL, D., STUDER, R., WIELINGA, B., SCHREIBER, G., AND ZDRAHAL, Z. 1998. IBROW3 - An Intelligent Brokering Service for Knowledge-Component Reuse on the World Wide Web. *In Proceedings of the 11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*, Banff, Canada, April 1998.

BERNERS-LEE, T., HENDLER, J. AND LASSILA, O. 2001. The Semantic Web. *Scientific American*, 284 (4), 34-43.

BÖRGER, E. 1998. High Level System Design and Analysis Using Abstract State Machines. *In proceedings of the International Workshop on Current Trends in Applied Formal Method*: *Applied Formal Methods*, 1-43.

BURSTEIN, M. BUSSLER, C., ZAREMBA, M., FININ, T., HUHNS, M.N., PAOLUCCI, M., SHETH, A.P., WILLIAMS, S. 2005. A Semantic Web Services Architecture. IEEE Internet Computing, Vol. 9 , 5, 72 – 81.

CABRAL, L. AND DOMINGUE, J. 2005. Mediation of Semantic Web Services in IRS-III. *In Proceedings of the International Conference on Service Oriented Computing* (ICSOC 2005), Amsterdam, The Netherlands.

CABRAL, L., DOMINGUE, J., GALIZIA, S., GUGLIOTTA, A., NORTON, B., TANASESCU, V., PEDRINACI, C. 2006. IRS-III: A Broker for Semantic Web Services based Applications. *In Proceedings of the 5th International Semantic Web Conference*, Athens, USA, November, 2006.

CLOCKSIN, W.F. AND MELLISH, C.S. 1984. – Programming in Prolog. Springer-Verlag New York, Inc. New York, NY, USA.

CRUBEZY, M., MOTTA, E., LU, W. AND MUSEN, M. 2002. Configuring Online Problem-Solving Resources with the Internet Reasoning Service. *IEEE Intelligent Systems*, 2, 34-42.

DE GREEF, P. AND BREUKER, J. 1992. Analysing system-user cooperation in KADS. *Knowledge Acquisition*, Special issue: The KADS approach to knowledge engineering 4(1), 89-108.

DIJKMAN, R. AND DUMAS, M. 2004. Service-Oriented Design: A Multi-Viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4): 337-368, 2004.

DIP. 2004. Data, Information, and Process Integration with Semantic Web Services, http://dip.semanticweb.org/.

DOMINGUE, J., CABRAL, L., GALIZIA, S., AND MOTTA, E. 2005a. A Comprehensive Approach to Creating and Using Semantic Web Services, *In Proceedings of the* W3C *Workshop on Frameworks for Semantics in Web Service,* Innsbruck, Austria, June 9-10, 2005.

DOMINGUE, J., CABRAL, L., HAKIMPOUR, F., SELL D., AND MOTTA, E. 2004. IRS-III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. *In Proceedings of the Workshop on WSMO Implementations (WIW 2004),* Frankfurt, Germany, September 29-30, 2004, CEUR Workshop Proceedings, ISSN 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-113/paper3.pdf.

DOMINGUE, J., GALIZIA, S., AND CABRAL, L. 2005b. Choreography in IRS-III- Coping with Heterogeneous Interaction Patterns in Web Services, *In Proceedings of the 4th International Semantic Web Conference (ISWC 2005)*, November 6-10, 2005, Galway, Ireland.

DOMINGUE, J. AND MOTTA, E. 2000. Planet-Onto: From News Publishing to Integrated Knowledge Management Support. *IEEE Intelligent Systems Special Issue on Knowledge Management and Knowledge Distribution over the Internet*, (26-32).

DOMINGUE, J., ROMAN, D., AND STOLLBERG, M. 2005c. Web Service Modeling Ontology (WSMO) - An Ontology for Semantic Web Services, *Position paper at the W3C Workshop on Frameworks for Semantics in Web Services*, June 9-10, 2005, Innsbruck, Austria.

DOMINGUE, J., STUTT, A., C MARTINS, M., TAN, J., PERTUSSON, H., MOTTA, E. 2003. Supporting Online Shopping through a Combination of Ontologies and Interface Metaphors. *International Journal of Human Computer Studies*, Vol.59, 5, (699 723).

DZBOR, M., MOTTA, E., DOMINGUE, J. 2007. Magpie: Experiences with supporting Semantic Web browsing. *Journal of Web Semantics*, 2007.

DUPPLAW, D., DASMAHAPATRA, S., HU, B., LEWIS, P. AND SHADBOLT., N. 2004. Multimedia Distributed Knowledge Management in MIAKT. *In Proceedings of the Workshop on Knowledge Markup and Semantic Annotation*. , in conjunction with *ISWC 2004,* November 2004, Hiroshima, Japan.

FENSEL, D. AND BUSSLER, C. 2002. The web service modeling framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.

FENSEL, D., LAUSEN, H., POLLERES, A., DE BRUIJN, J., STOLLBERG, M., ROMAN, D., DOMINGUE, J. 2006. Enabling Semantic Web Services: Web Service Modeling Ontology. Springer, 2006.

FENSEL, D. AND MOTTA, E. 2001. Structured Development of Problem Solving Methods*, IEEE Transactions on Knowledge and Data Engineering,* 13(6), 9131-932.

FORGY, C.L. 1981. OPS5 User's Manual, Technical Report CMU-CS-81-135, Carnegie Mellon University, 1981.

EISENSTADT, M., KOMZAK, J. AND DZBOR, M. 2003. Instant messaging + maps = powerful collaboration tools for distance learning. *In Proceedings of TelEdu '03,* Havana, Cuba, 17-22 May 2003.

GALIZIA, S., AND DOMINGUE, J. 2004. Towards a Choreography for IRS-III*. In Proceedings of the Workshop on WSMO Implementations (WIW 2004),* Frankfurt, Germany, September 29-30, 2004, CEUR Workshop Proceedings, ISSN 1613-0073. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-113/paper7.pdf.

GANGEMI, A., GUARINO, N., MASOLO, C., OLTRAMARI, A., SCHNEIDER, L. 2002. Sweetening ontologies with DOLCE. *In Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*, Siguenza, Spain, October, 2002.

GOOGLE . 2005. Google Web APIs http://www.google.com/apis/index.html/.

GRUBER, T. R. 1993. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2).

GRUBER, T. R., AND OLSEN, G. R. 1994. An ontology for engineering mathematics. *In Proceedings of Fourth International Conference on Principles of Knowledge Representation and Reasoning*, Gustav Stresemann Institut, Bonn, Germany, May 24-27, 1994.

HAGEL, J., DURCHSLAG, S. AND SEELY BROWN, J. 2002. Orchestrationg Loosely Coupled Business Processes: The Secret to Successful Collaboration. http://www.johnhagel.com/paper_orchestratingcollaboration.pdf

HAKIMPOUR, F., DOMINGUE, J., MOTTA, E., CABRAL, L. AND LEI, Y. 2004. Integration of OWL-S into IRS-III, *In Proceedings of the first AKT Workshop on Semantic Web Services*.

HAVANTZAS, G., BURDETT, D., RITZINGER, G., FLETCHER, T. AND LAFON, Y., BARRETO, C. 2005. Web Service Choreography Description Language Version 1.0. W3C Working Draft 17 December 2004. (Available at http://www.w3.org/TR/ws-cdl-10/).

KELLER, G., NÜTTGENS, M., AND SCHEER, A. W. 1992. Semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Prozessketten (EPK). Technical Report Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWi), Heft 89. Universität des Saarlandes, January 1992.

LASSILA, O. 2001. Enabling Semantic Web Programming by Integrating RDF and Common Lisp. *In Proceedings of the First Semantic Web Working Symposium*. Stanford University, 2001.

LHDL. 2006. Living Human Digital Library, http://www.livinghuman.org/.

LUISA. 2006. Learning Content Management System Using Innovative Semantic Web Services Architecture, http://luisa.atosorigin.es/.

JABBER. 2006. Jabber SoftwareFoundation. http://www.jabber.org/.

MARK., D. 1989. Cognitive Image-Schemata for Geographic Information: Relations to User Views and GIS Interfaces. *In Proceedings of GIS/LIS '89* (pp. 551–560), Orlando, Florida, 1989.

MEDJAHED, B. 2005. A Multilevel Composability Model for Semantic Web Services*. IEEE Transactions on Knowledge and Data Engineering*, 17(7), 954 – 968, July 2005.

MEDJAHED, B. AND BOUGUETTAYA, A. 2005. Customized Delivery of E-Government Web Services. *IEEE Intelligent Systems,* 20(6), December 2005.

METEOR-S. 2006. METEOR-S: Semantic Web Services and Processes. http://lsdis.cs.uga.edu/projects/meteor-s/.

MIAKT. 2002. Medical Imaging and Advanced Knowledge Technologies, http://www.aktors.org/miakt/.

MOTTA, E. 1998. An Overview of the OCML Modelling Language, *In Proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML '98)*.

MOTTA, E., DOMINGUE, J., CABRAL, L., AND GASPARI, M. 2003. IRS-II: A Framework and Infrastructure for Semantic Web Services. *In Proceedings of the 2nd International Semantic Web Conference (ISWC2003,)* 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA.

NORTON, B. 2007. Reasoning About Behaviour on the Semantic Web. In Proceedings on Programming Paradigms for the Web: Web Programming and Web Services, Dagstuhl, Germany (to appear).

NORTON, B. PEDRINACI, C., HENOCQUE, L. AND KLEINER M. 2007. 3-Level Behavioural Models for Semantic Web Services. Special issue of Multi-Agent and Grid Systems. (to appear).
OASIS, SEE TC. 2006. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=semantic-ex/.
OMG. 2002. The Object Management Group: Meta-Object Facility, version 1.4, 2002. Available at http://www.omg.org/technology/documents/formal/mof.htm.
OWL. 2004. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/.
OWL-S. 2006. OWL-S 1.2 Pre-Release, (http://www.ai.sri.com/daml/services/owl-s/1.2/).
OWL-S TOOLS. 2006. http://www.daml.org/services/owl-s/tools.html.
PAOLUCCI, M., ANKOLEKAR, A., SRINIVASAN, N. AND SYCARA, K. 2003. The DAML-S Virtual Machine, *In Proceedings of the Second International Semantic Web Conference (ISWC), 2003,* Sandial Island, Fl, USA, October 2003, 290-305.
PATIL, A., OUNDHAKAR, S., SHETH, A. AND VERMA, K. 2004. METEOR-S Web service Annotation Framework, *Proceedings of the Thirteenth International World Wide Web Conference (WWW2004),* May, 2004, 553-562.
RDF. 2004. RDF Primer. http://www.w3.org/TR/rdf-primer/.
RDF SCHEMA. 2004. RDF Vocabulary Description Language 1.0: RDF Schema. http://www.w3.org/TR/rdf-schema/.
RIVA, A. AND RAMONI, M. 1996. LispWeb: A Specialised HTTP Server for Distributed AI Applications. *Computer Networks and ISDN Systems*, 28, 7-11, 953-961.
SAWSDL. 2006. Semantic Annotations for Web Services Description Language Working Group. http://www.w3.org/2002/ws/sawsdl/.
SHI, X. AND JAGANNATHAN, V. 2005. Rebuilding the Semantic Web Service Architecture, *Proceedings of the 2nd international workshop on semantic and dymanic Web processes (SDWP 2005),* in conjunction with *the International Conference on Web Services (ICWS 2005),* Orlando, Florida, USA, July 11-15 2005.
SIVASHANMUGAM, K., MILLER, J.A., SHETH AND VERMA, K. 2005. Framework for Semantic Web Process Composition. *International Journal of Electronic Commerce,* 9(2), 71-106.
SRINIVASAN, N., PAOLUCCI, M. AND SYCARA, K. 2006. Semantic Web Service Discovery in the OWL-S IDE. *In proceedings of Hawaii International Conference on System Sciences (HICSS 2006)*, Hyatt Regency Kauai, Hawaii, January 4-6, 2006.
SOAP. 2003. SOAP Version 1.2 Part 0: Primer, (http://www.w3.org/TR/soap12-part0/).
STOLLBERG, M. AND NORTON, B. 2007. A Refined Goal Model for Semantic Web Services. *In Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007)*, Mauritius, 2007.
SUPER. 2006. Semantics Utilised for Process management within and between EnteRprises, http://www.ip-super.org/.
SWSF. 2005. Semantic Web Services Framework, http://www.daml.org/services/swsf/.
SWSI. 2006. The Semantic Web Services Initiative (SWSI), http://www.swsi.org/.
TANASESCU, V., GUGLIOTTA, A., DOMINGUE, J., VILLARÍAS, L., DAVIES, R., ROWLATT, M., RICHARDSON, M., AND STINCIC, S. 2007. Geospatial Data Integration with Semantic Web Services: the eMerges Approach, *The Geospatial Web*, eds. Arno Scharl, Klaus Tochtermann, Springer.
UDDI. 2003. UDDI Spec Technical Committee Specification v. 3.0, http://uddi.org/pubs/uddi-v3.0.1-20031014.htm.
W3C. 2004a. Web Services Architecture, http://www.w3.org/TR/ws-arch/.
W3C. 2004b. Web Services Glossary. W3C Working Group Note. 11 February 2004 (Available at http://www.w3.org/TR/ws-gloss/).
WSDL. 2001. Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.
WSML. 2005. The Web Service Modeling Language WSML, http://www.wsmo.org/TR/d16/d16.1/v0.3/.
WSMO. 2007. Web Service Modeling Ontology (WSMO), http://www.wsmo.org/TR/d2/v1.4/.
WSMO STUDIO. 2006. http://www.wsmostudio.org/.
WSMX. 2005. Overview and Scope of WSMX, http://www.wsmo.org/TR/d13/d13.0/v0.2/.

APPENDICES

Appendix I. The Toplevel Concepts in the IRS-III Service Ontology

Appendix I contains a number of the main toplevel definitions for the service ontology including definitions related to the concepts of goal, Web service and mediator.

```
(def-class meta-invokable-entity () ?x
  :iff-def (or (= ?x invokable-entity)
               (subclass-of ?x invokable-entity)))

(def-class invokable-entity ()
  "Captures the input and output roles which are used within goals and Web
services."
  ((has-input-role :type role)
   (has-output-role :type role)))

(def-class meta-wsmo-entity () ?x
  :iff-def (or (= ?x wsmo-entity)
               (subclass-of ?x wsmo-entity)))

(def-class wsmo-entity ()
  ((has-non-functional-properties :type non-functional-properties)))

(def-class effect (unary-kappa-expression))

(def-class pre-condition (unary-kappa-expression))

(def-class post-condition (unary-kappa-expression))

(def-class assumption (unary-kappa-expression))

(def-class meta-goal () ?x
  :iff-def (or (= ?x goal)
               (subclass-of ?x goal)))

(def-class goal (wsmo-entity invokable-entity)
  ((used-mediator :type meta-mediator)
   (has-post-condition :type post-condition)
   (has-effect :type effect)))


(def-class meta-capability () ?x
  :iff-def (or (= ?x capability)
               (subclass-of ?x capability)))

(def-class capability (wsmo-entity)
  ((used-mediator :type meta-wg-oo-mediator)
   (has-pre-condition :type pre-condition)
   (has-post-condition :type post-condition)
   (has-assumption :type assumption)
   (has-effect :type effect)))


(def-class meta-web-service () ?x
  :iff-def (or (= ?x web-service)
               (subclass-of ?x web-service)))

(def-class web-service (invokable-entity wsmo-entity)
  ((has-capability :type meta-capability)
```

```
   (has-interface :type meta-interface)
   (used-mediator :type meta-oo-mediator)))

(def-class meta-interface () ?x
  :iff-def (or (= ?x interface)
               (subclass-of ?x interface)))

(def-class interface (wsmo-entity)
  ((has-choreography :type meta-choreography)
   (has-orchestration :type meta-orchestration)
   (used-mediator :type meta-oo-mediator)))

(def-class meta-mediator () ?x
  :iff-def (or (= ?x mediator)
               (subclass-of ?x mediator)))

(def-class mediator (wsmo-entity)
  ((has-source-component :type meta-wsmo-entity)
   (has-target-component :type meta-wsmo-entity)
   (has-mapping-rules :type mapping-rules)
   (has-mediation-service :type meta-mediation-service)))


(def-class meta-wg-mediator (meta-mediator) ?x
  :iff-def (or (= ?x wg-mediator)
               (subclass-of ?x wg-mediator)))

(def-class wg-mediator (mediator)
  ((has-source-component :type meta-web-service-or-wg-mediator)
   (has-target-component :type meta-goal-or-wg-mediator)
   (used-mediator :type meta-oo-mediator)))

(def-class meta-gw-mediator (meta-mediator) ?x
  :iff-def (or (= ?x gw-mediator)
               (subclass-of ?x gw-mediator)))

(def-class gw-mediator (mediator)
  ((has-source-component :type meta-goal-or-gw-mediator)
   (has-target-component :type meta-web-service-or-gw-mediator)
   (used-mediator :type meta-oo-mediator)))

(def-class meta-ww-mediator (meta-mediator) ?x
  :iff-def (or (= ?x ww-mediator)
               (subclass-of ?x ww-mediator)))

(def-class ww-mediator (mediator)
  ((has-source-component :type meta-web-service-or-ww-mediator)
   (has-target-component :type meta-web-service-or-ww-mediator)
   (used-mediator :type meta-oo-mediator)))

(def-class meta-gg-mediator (meta-mediator) ?x
  :iff-def (or (= ?x gg-mediator)
               (subclass-of ?x gg-mediator)))

(def-class gg-mediator (mediator)
  ((used-mediator :type meta-oo-mediator)
   (has-source-component :type meta-goal-or-gg-mediator)
   (has-target-component :type meta-goal-or-gg-mediator)))

(def-class meta-oo-mediator (meta-mediator) ?x
  :iff-def (or (= ?x oo-mediator)
               (subclass-of ?x oo-mediator)))

(def-class oo-mediator (mediator)
  ((has-source-component :type meta-oo-mediator)))
```

```
(def-class meta-wg-or-oo-mediator (meta-mediator) ?x
  :iff-def (or (= ?x wg-or-oo-mediator)
               (subclass-of ?x wg-or-oo-mediator)))

(def-class wg-or-oo-mediator (mediator) ?x
  :iff-def (or (oo-mediator ?x)
               (wg-mediator ?x)))

(def-class meta-goal-or-gg-mediator (meta-mediator) ?x
  :iff-def (or (= ?x goal-or-gg-mediator)
               (subclass-of ?x goal-or-gg-mediator)))

(def-class goal-or-gg-mediator (meta-mediator) ?x
  :iff-def (or (goal-mediator ?x)
               (gg-mediator ?x)))

(def-class meta-web-service-or-ww-mediator (meta-mediator) ?x
  :iff-def (or (= ?x web-service-or-ww-mediator)
               (subclass-of ?x web-service-or-ww-mediator)))

(def-class web-service-or-ww-mediator (mediator) ?x
  :iff-def (or (web-service ?x)
               (ww-mediator ?x)))

(def-class meta-mediation-service () ?x
  :iff-def (or (meta-goal ?x) (meta-web-service ?x)))

(def-class mediation-service () ?x
  :iff-def (or (goal ?x) (web-service ?x)))
```

Appendix II. A Subset of the Main Relations in the IRS-III Service Ontology

Appendix II contains a small portion of the main relations within the IRS-III service ontology. These relations are used by IRS-III components to manipulate user definitions of goals, mediators and Web services which are classes. Hence the relations below make use of the meta-class definitions within Appendix I.

```
(def-relation has-wsmo-input-role (?thing ?role)
  :sufficient
  (or (and (instance ?thing)
           (has-wsmo-input-role (the-parent ?thing) ?role))
      (and (class ?thing)
           (or
            (and
             (meta-invokable-entity ?thing)
             (member ?role (all-class-slot-values
                            ?thing has-input-role)))
            (and (meta-web-service ?thing)
                 (associated-goal ?thing ?goal-type)
                 (has-wsmo-input-role ?goal-type ?role))))))


 (def-relation associated-goal (?web-service ?goal)
  :sufficient
  (or (and (instance ?web-service)
           (associated-goal (the-parent ?web-service) ?goal))
      (and (meta-web-service ?web-service)
           (= ?capability (the-class-slot-value ?web-service has-
capability))
           (meta-capability ?capability)
           (= ?mediator (the-class-slot-value ?capability used-mediator))
           (meta-mediator ?mediator)
           (= ?goal (the-class-slot-value ?mediator
                                          has-source-component))
           (meta-goal ?goal))
      (and (meta-mediator ?mediator)
           (= ?web-service (the-class-slot-value ?mediator
                                                 has-target-component))
           (meta-web-service ?web-service)
           (= ?goal (the-class-slot-value ?mediator
                                          has-source-component))
           (meta-goal ?goal))))

(def-relation applicable-to-goal (?web-service-class ?goal-inst)
  :iff-def (or (not (and (= ?capability
                            (the-class-slot-value ?web-service-class has-
capability))
                         (meta-capability ?capability)
                         (= ?exp (the-class-slot-value ?capability has-
assumption))
                         (not (= ?exp :nothing))))
               (and (= ?capability
                       (the-class-slot-value ?web-service-class has-
capability))
                    (meta-capability ?capability)
                    (= ?exp (the-class-slot-value ?capability has-
assumption))
                    (not (= ?exp :nothing))
                    (holds ?exp ?goal-inst))))
```

```
(def-procedure instantiate-web-service
  (?goal-inst ?web-service-type)
  :body (in-environment
         ((?name . (new-symbol ?web-service-type)))
         (tell (append (list-of ?web-service-type ?name) nil))
         (tell (suitable-web-service ?goal-inst ?name))
         ?name))

(def-relation can-solve-goal (?goal ?thing)
  :sufficient."
  (or (and (instance ?goal)
           (can-solve-goal (the-parent ?goal) ?thing))
      (and (meta-web-service ?thing)
           (= ?capability (the-class-slot-value ?thing has-capability))
           (meta-capability ?capability)
           (= ?mediator (the-class-slot-value ?capability used-mediator))
           (meta-mediator ?mediator)
           (= ?goal (the-class-slot-value ?mediator
                                          has-source-component)))))
```

Appendix III. A Portion of the IRS-III Service Ontology for Internal Components

```
(def-class internal-goal (goal))

(def-class internal-mediator (mediator))

(def-class internal-capability (capability))

(def-class internal-web-service (web-service)
  ((has-internal-method :type symbol)))

(def-class suitable-web-service-goal (internal-goal)
  ((has-input-role
    :value has-goal :value has-actual-role-pairs
    :value has-web-service :value has-combined-oo-mediator-ontology)
   (has-input-soap-binding
    :value (has-goal "sexpr")
    :value (has-actual-role-pairs "sexpr")
    :value (has-web-service "sexpr")
    :value (has-combined-oo-mediator-ontology "sexpr"))
   (has-output-role
    :value has-goal-and-web-service-instances)
   (has-goal-and-web-service-instances :type list)
   (has-output-soap-binding
    :value (has-goal-and-web-service-instances "sexpr"))
   (has-goal :type goal-type)
   (has-actual-role-pairs :type list)
   (has-web-service :type meta-web-service)
   (has-combined-oo-mediator-ontology :type ontology)
   (has-post-condition
    :value
    (kappa (?goal)
           (is-suitable-for-goal
            (instantiate (has-role-value ?goal has-goal)
                         (has-role-value ?goal has-actual-role-pairs))
            (has-role-value ?goal has-web-service))))))

(def-class suitable-web-service-mediator (wg-mediator)
  ((has-source-component :value suitable-web-service-goal)))

(def-class suitable-web-service-web-service (internal-web-service)
  ((has-internal-method :value suitable-web-service-internal-method)
   (has-capability :value suitable-web-service-capability)))

(def-class suitable-web-service-capability (capability)
  ((used-mediator :value suitable-web-service-mediator)))

(def-class find-web-services-for-goal (internal-goal)
  ((has-input-role :value has-goal :value has-ontology)
   (has-input-soap-binding
    :value (has-goal "sexpr")
    :value (has-ontology "sexpr"))
   (has-output-role :value associated-web-services)
   (associated-web-services :type list)
   (has-output-soap-binding
    :value (associated-web-services "sexpr"))
   (has-goal :type meta-goal)
   (has-ontology :type ontology)
   (has-post-condition
    :value
    (kappa (?goal)
           (and (member ?web-service
                        (has-role-value ?goal associated-web-services))
```

```
                (can-solve-goal
                 (has-role-value ?goal has-goal) ?web-service))))))

(def-class find-web-services-for-goal-mediator (wg-mediator)
  ((has-source-component :value find-web-services-for-goal)))

(def-class find-web-services-for-goal-web-service (internal-web-service)
  ((has-internal-method :value web-services-which-solve-goal-internal-
method)
   (has-capability :value find-web-services-for-goal-web-service-
capability)))

(def-class find-web-services-for-goal-web-service-capability (capability)
  ((used-mediator :value find-web-services-for-goal-mediator)))
```