Choreography in IRS-III – Coping with Heterogeneous Interaction Patterns in Web Services

John Domingue, Stefania Galizia and Liliana Cabral

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

Abstract. In this paper we describe how we handle heterogeneity in web service interaction through a choreography mechanism that we have developed for IRS-III. IRS-III is a framework and platform for developing semantic web services which utilizes the WSMO ontology. The overall design of our choreography framework is based on: the use of ontologies and state, IRS-III playing the role of a broker, differentiating between communication direction and which actor has the initiative, having representations which can be executed, a formal semantics, and the ability to suspend communication. Our framework has a full implementation which we illustrate through an example application.

1 Introduction

Web services provide a mechanism to connect applications regardless of the underlying software/hardware platform and their location. From an Information Technology (IT) perspective the key features of web services are that, a) they are based on standard XML based protocols which can run over the internet and b) the descriptions of a web service are distinct from the actual implementation. From a business perspective one key feature is that web services 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, Amazon Web Services allows software developers to directly access their technology platform and product data [1].

Interest in web service technology is high. Many of the major IT vendors (e.g. Microsoft, IBM, SAP) now provide web service based solutions. Moreover, current predictions indicate that the market for web service based solutions will be worth $2.9 billion in 2006 growing to $6.2 billion by 2008 [13].

The web service community is now beginning to accept that the majority of the current problems associated with web services are related to the fact that all of the technologies are based on syntactic descriptions such as WSDL [25] and UDDI [21]. Because syntactic level descriptions are not amenable to computer based interpretation, all of the tasks associated with creating applications from web service based components are carried out manually. Requiring IT specialists to discover, compose and deploy web services manually is time-consuming, costly and error-prone. Moreover, as stated by Larry Ellison:

> ***"Semantic differences remain the primary roadblock to smooth application integration, one which Web Services alone won't overcome….When I pass customer data across [the Web] in a certain format using a Web services interface, the receiving program has to know what that format is. You have to agree on what the business objects look like."*** [8]

The most significant task when connecting software components together is not the plumbing (the data and control flow) but coping with the semantic differences. Two main types of communication mismatches can occur. The first is that the data can have different underlying representations. For example, one web service may represent an address as a number followed by a street name and town, whereas another may represent an address as a number followed by a postal code. The second type of mismatch is related to interaction. Each web service will have a specific interaction pattern related to how the underlying processes are implemented. For example, one web service may require credit card details (e.g. card number, card expiry date) to be sent one at a time whereas another may require that all details are sent in a single message.

In this paper we describe how we cope with heterogeneous web service interaction patterns in the context of IRS-III [7]. IRS-III is a framework and implemented infrastructure which supports the creation of semantic web service based applications. IRS-III has been used to teach semantic web services in a number of tutorials [19] and is currently being deployed in a number of application areas in the context of the DIP [6] project. Following the WSMO [17] framework we use the term choreography to denote the IRS-III component which deals with web service interaction. Our primary contributions which we describe in this paper include: a set of design principles for choreography, a formal definition of choreography based on abstract state machines, a well founded set of ontology based choreography specific primitives and a full implementation.

The rest of this paper is structured as follows: in the following section we describe related work, then we present an overview of IRS-III framework. The section 4 describes the choreography within IRS-III outlining the design principles, our formal model, the main primitives and the execution. In section 5 we describe an example application and the final section concludes the paper.

2 Related Work

The existing approaches describing the communication among web services propose different definitions of choreography, and some of them do not clearly distinguish between choreography and message exchange pattern defininitions.

A message exchange pattern (MEP) is a syntactic template that represents a model for the exchange of messages between web services; a choreography should also describe patterns semantically. However, some approaches view choreography as the composition of atomic MEPs [23], without the support of semantics. Actually, the

only standard choreography definition, available on the W3C glossary [24], states simply that choreography concerns the interaction of services with their users.

However, the requirements emerging from e-Business necessitate that web services exchange information at the semantic level. Thus, the choreography of a semantic web service should include a communication protocol specification, which represents service interactions at a semantic level.

The Web Service Choreography Description Language (WS-CDL) provides the choreography representation from a global point of view [12]. According to this vision, the choreography describes the behaviour observable from an external point of view, emphasizing the collaboration of parties, where the communication progresses only when jointly agreed ordering rules are satisfied. Furthermore, the model depicted by the WS-CDL working group describes the choreography at three levels of abstraction: abstract, portable and concrete [22]. An abstract choreography definition will contain descriptions of the data types used and the conditions under which a given message is sent. A portable choreography includes descriptions of the physical structure of the information exchanged and of the technologies used. A concrete choreography extends a portable description including destination URLs, and specific rules, such as information about digital certificates to be used for securing messages. When creating a choreography, the chosen level of abstraction would depend on the current context (e.g. the type of organization it was designed for) and the level of reusability and extendibility required.

Another global approach is presented by Dijman and Dumas [5]. They depict both static and dynamic aspects of the global communication among heterogeneous web services using Petri Nets.

The main current approaches to representing web service communication at a semantic level are proposed by the WSMO [17] and OWL-S [16] working groups.

A web service description within WSMO contains an interface definition. An interface includes a definition of orchestration – how a composite web service invokes subsidiary web services - and a choreography. WSMO adopts, furthermore, the Abstract State Machine (ASM) formalism to model the behavioral aspects of the communication.

In contrast OWL-S does not provide an explicit definition of choreography but instead focuses on a process based description of how complex web services invoke atomic web services.

Within IRS-III, our viewpoint is based on the WSMO approach, which is different from the global approaches described above, as it represents the choreography of a single web service. That is, we describe how one web service talks to one other.

We strictly keep to the WSMO vision, in fact, by separating the orchestration and choreography concepts, WSMO emphasizes the difference between communication and cooperation among web services.

There are also other choreography descriptions that follow the WSMO approach, for instance, Arroyo and Duke propose a Conceptual Model for a Semantic Choreography Engine (SOPHIE) [2], where they aim to separate in a clear-cut way the syntactic and the semantic level and adopt the ASM formalism to model the communication.

In the rest of this paper we give a detailed description of choreography in IRS-III.

3 IRS-III 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 [4] supported the creation of knowledge intensive systems structured according to the UPML framework [9] and IRS-II [15] integrated the UPML framework with web service technologies. Within IRS-III we have now incorporated and extended the WSMO ontology [17].

IRS-III has three main classes of features which distinguish it from other work on semantic web services.

Firstly, it supports *one-click publishing* of 'standard' program code. In other words, it automatically transforms programming code (currently we support Java and Lisp environments) into a web service, by automatically creating an appropriate wrapper. Hence, it is very easy to make existing standalone software available on the net, as web services.

Secondly, by extending the WSMO goal and web service concepts, clients of IRS-III can directly invoke web services via goals - that is IRS-III supports *capability-driven* service invocation.

Finally, IRS-III services are web service compatible – standard web services can be trivially published through the IRS-III.

The main components of the IRS-III architecture are the IRS-III Server, the IRS-III Publisher and the IRS-III Client, which communicate through the SOAP protocol. The IRS-III server holds descriptions of Semantic Web Services at two different levels. A knowledge level description is stored currently represented internally in OCML [14], an Ontolingua-derived language which provides both the expressive power to express task specifications and service competencies, as well as the operational support to reason about these.

Publishing with IRS-III entails associating a specific web service with a WSMO web service description. When a web service is published in IRS-III all of the information necessary to call the service, the host, port and path are stored within the choreography associated with the web service.

The IRS publishing platform is furthermore responsible for the actual invocation of a web service; additionally, it automatically generates wrappers which turn standalone code into a web service. The platform also copes with the syntactic level differences between the various web service platforms e.g. AXIS and Apache.

IRS-III was designed for ease of use, in fact a key feature of IRS-III is that web service invocation is capability driven. The IRS-III Client 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 an appropriate web service semantic description and then invokes the underlying deployed web service.

In the rest of the paper we will use the terms "IRS" and "IRS-III" interchangeably.

4 IRS-III Choreography model

A choreography is described in IRS-III by a grounding declaration 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`, `output-role-soap-binding`. The *guarded transitions* are the set of rules, which represent the interaction between IRS-III and the Web Service on behalf of an IRS client. They are applied when executing the choreography. This model is executed at a semantic level when IRS-III receives a request to achieve a goal.

In the rest of this section we list the main design principles which motivate our choreography model.

4.1 Design principles

Ontology Based. Ontologies form a central pillar of the semantic web. Founding our choreography descriptions on ontologies means that we can refer to relevant domain dependent concepts or relations within guarded transitions.

IRS as a Broker. As we mentioned earlier the IRS acts as a broker for capability based invocation. A client sends a request to achieve a goal and the IRS finds, composes and invokes the appropriate web services. The choreography to the IRS is thereby fixed. We assume that IRS clients are able to formulate their request as a goal instance. This means that we only require choreographies between the IRS and the deployed web services. Our choreography descriptions are therefore written from the perspective of IRS as a client of the web service.

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

Given the above we decided to adopt the Abstract State Machine (ASMs) model [3] to represent IRS choreography. Additionally, ASMs are also used within WSMO [18] which is the ontology adopted within IRS-III. A further reason for using ASMs is that they combine mathematical rigor with a practical execution model to represent message exchange patterns.

By representing ASM as rules, the sequence of operations and the message pattern instantiations are generated through the evaluation of conditions. A condition is a generic statement on the current situation, for instance, that an error has occurred. The executive part of the guarded transitions (after 'then') updates the state.

The general form of a guarded transition is given below:

"**if** currentstate = s $\wedge$ Cond **then** currentstate = s1 "

Open. The major components of IRS-III are semantic web services represented within the IRS-III framework. This feature enables the main functionalities of the IRS to be redefined to suit specific requirements. Following this the IRS choreography engine is itself a semantic web service.

Communication Representation. We have chosen to classify the communication in IRS choreography according to two dimensions, following the system-client cooperation model proposed in KADS [11], namely:

- The initiative in the communication, and
- The direction of the communication.

The initiative expresses which actor, either IRS 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.

The reason for preferring this communication model is that in this way we can verify at every state which actor has initiative. Initiative is associated with the actors who in some sense have control of the conversation. For example, only actors with initiative are allowed to start a conversation or update data previously sent.

A message exchange *event* is a kind of transfer task, an elementary executed operation by an actor during a conversation.

From the IRS perspective, and according to Greef and Breuker's communication representation, we consider six kinds of events: *obtain, present, provide, receive, obtain-initiative, present-initiative*. When the IRS does not have the initiative, *receive* and *provide* messages are used. Conversely, *obtain* and *present* events occur when the IRS is in control of the conversation. *Obtain-initiative* and *present-initiative* allow the initiative to be transferred. For detailed event descriptions see [10].

When a client, that can also be a web service, invokes the IRS, in order to achieve a goal, the choreography engine runs. We depict a simple invocation goal scenario below, underlining the events involved during choreography execution.

Figure 1 depicts the event sequence for this typical goal driven web service invocation scenario.

Fig. 1 A typical sequence of choreography events occurring during goal based web service invocation in IRS-III.

The client initiates the communication with IRS by requesting that a goal be achieved. Within our model this corresponds to *receive* and *obtain-initiative* events as the client delegates initiative to the IRS to invoke the required service. During a second phase the IRS invokes a web service which returns a response. In this phase the IRS has the initiative and therefore the occurring events are *present* and *obtain*.

Ability to Suspend Communication. There will be some situations where it is necessary to suspend the current dialog and resume it later. For example, either the IRS or the web service may not have some required data or a web service may go offline.

Executable Semantic Descriptions. The semantic representations of choreography should be executable directly or should be able to be compiled to a runnable representation. Our underlying modelling language OCML [14] is operational. Additionally, extensions within the IRS allow us to attach OCML functions to deployed web ser-

vices. This means that within a guarded transition one can refer to external data, for example, to "today's exchange rate".
Formalization. A formal semantics allows us to reason about the choreography descriptions which is useful if we want to automatically compose web services. For this reason, we adopt ASMs and our formal model is described in the following section.
Easy to use. If we want our system to be used widely, it is important that the components are easy to use. For this purpose we have defined a relatively small set of choreography specific primitives.

4.2 Formal Definition

Our abstract model of choreography is represented by four main entities: *events, states, conditions,* and *guarded transitions.*
We perform the IRS-III choreography through the tuple $\langle E, S, C, T \rangle$, where

- *E* is 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 the conditional guarded transitions.

The events that can occur are: {*obtain, present, provide, receive, obtain-initiative, present-initiative*} [10]. Every event maps to an operation during the conversation viewed from the IRS perspective.
The states are the possible message exchange pattern instantiations. A state $s_i \in S$ at a given conversation step Ti, is represented by a set of instances. It contains a constant subset, the web service host, port, location, that is invariant whenever the same web service is invoked, and the event instantiation, dependent on the event that occurred at step Ti.
The web service host, port and location are defined during the IRS publishing process – see section 3.1.
A condition $c \in C$ depicts a situation occurring during the conversation.
The guarded transitions, according with WSMO definition [18], express changes of states by means of rules:
A guarded transition $t \in T$, is a function $t : \left(S, 2^C\right) \underset{E}{\rightarrow} S$, that associates a couple (s, $\{c_1, .., c_j\}$) to s', where s and s' $\in S$, *and every* c_k $(1 \leq k \leq i) \in C$.
A guarded transition updates the communication state by an event $e \in E$.

4.3 Choreography primitives

We have defined a set of choreography specific primitives which can be used in guarded transitions. Our primitives provide an easy to use interface to control a conversation between the IRS 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 step the IRS has the initiative and it is ready to start the communication.

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

The type of event which occurs with send-message is "present" since the IRS holds the initiative and the communication direction is from the IRS to the web service (see figure 1).

Send-suspend. Suspends the communication between IRS and the web service, without stopping the choreography executions. This action will occur, for example, when the IRS 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 the IRS to the web service and the IRS 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 and the communication direction is from the web service to IRS.

Figure 2 shows all events which occur when a web service suspends communication. Initially IRS has initiative, but it is handed over to the web service which suspend the communication through the event "receive". When the web service resumes the dialog the associated event is "receive" again, because the web service has the initiative.

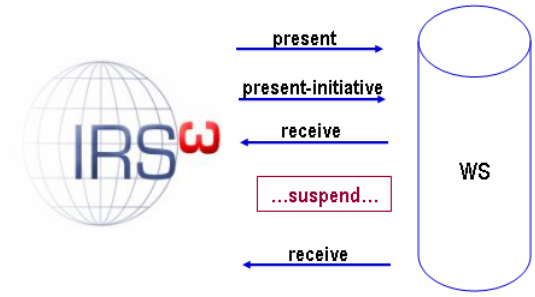

Fig. 2. The occurring choreography events if the web service suspends the communication.

Received-message. Contains the result of a successful send-message for a specific operation. In the general case the trigged event is "obtain" as shown in figure 1. If however the web service had previously suspended the communication it will be "receive" (see figure 2). In the both situations the message direction is from the web service to the IRS, but in the former one, IRS 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 to occur then the received-error primitive is used. The parameters of received-error include the error message and the type of error which occurred. In a fashion similar to received-

message, described above, the event taking place is either “obtain” (see figure 1), or “receive” (see figure 2).

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

4.4 Choreography execution

IRS uses a forward-chaining-rule engine to execute a choreography. This means the rules belonging to a choreography are fired according to the state.
Within the IRS there is an internal method which selects one guarded transition when two or more are selected.

One important feature of the execution environment of IRS 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 server carries out inferences at an ontological level. 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).

Lift. Lifts an XML string into an ontological construct, represented in OCML. A generic version of this relation is defined within the IRS 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 the classes within the underlying ontology and the particular web services selected. In order to cope with XML based input the lift primitive utilizes an inbuilt SAX based XML parser.

Lower. Lowers the ontological construct to XML. The input parameters to lower are: `instance-name` and a class `web-service`. 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.

5 Virtual Travel Agency Example

Our example application is based on the WSMO Virtual Travel Agency (VTA) application [20]. The overall scenario is to provide a portal where clients can ask for train tickets between any two cities in Europe specifying a departure time and date. The portal maintains a profile for regular users which contains personal preferences.

Our implementation of the VTA includes four web services which can book tickets for specific countries (e.g. Austria, France) and two which can book tickets for travellers with particular profiles (e.g. students and business people). In the rest of this de-

scription we will focus on one particular web service – the train ticket service for Germany - and describe its choreography.

German-buy-train-ticket-service-choreography
grounding:

```
  normal
   book-german-train-journey
      has-person "sexpr"
      has-departure-station "sexpr"
      has-destination-station "sexpr"
      has-date-and-time "sexpr"
   "string"

  first-class-upgrade
    book-first-class-upgrade-german-train-journey
    …..

  standard-class
    book-standard-class-german-train-journey
    …..

  acknowledge-error
   acknowledge-error-message
      has-acknowledgement "int"
   "string"
```

guarded-transitions:

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

  accept-first-class-upgrade
    received-message normal ?result
    upgrade-class ?result
    operation-input normal has-person ?person
    accept-upgrade ?person ?accept-upgrade
  then
    send-message 'first-class-upgrade
    end-choreography

 date-error-transition
   received-error normal ?error-message ?error-type
   date-format-error ?error-type
 then
   send-message-with-new-input-role-pairs
      'acknowledge-error (has-acknowledgement 0)
   end-choreography
```

If the traveller booking the train ticket is a gold card member the German train ticket service offers a free upgrade to first class. Travellers can state that they automatically accept these offers within their profile. The choreography definitions below enable the IRS to interact with the web service so that the correct types of bookings are made. The choreography starts with the guarded transition containing `init-choreography` and it ends with the `end-choreography` execution.

The choreography contains two components. The first is a grounding which maps between semantic operations and the implementation level. Above we show the full grounding for the `normal` and `acknowledge-error` operations and only partial definitions for the other operations. After the operation name the next part of the grounding shows 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 it would be 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 three guarded transitions. `Start` initializes the choreography session and then invokes the deployed service by sending the message associated with the `normal` operation. `Send-message` is a choreography specific relation which takes the values of the input roles from the associated goal instance, transforms the values to an XML representation (using a relation called `lower`), and then invokes the web service. `Accept-first-class-upgrade` uses 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 an offer of an upgrade and whether the traveller's profile states that s/he automatically accepts upgrades. The executive part of the guarded transition sends a message for the `first-class-upgrade` operation and ends the choreography.

The final guarded transition shown, `date-error-transition`, handles errors. If invoking a web service causes an error then an instance of the relation `received-error` is created. The signature of this relation is `<operation> <error-message> <error-type>`. `Error-type` is an instance of a subclass of the `invocation-error` class. The condition for this guarded transition checks to see if the error is a date format error. When this is the case the `acknowledge-error` operation is invoked. Note that because the input-role name and value (`has-acknowledgement` and `0`) are not present in the original goal invocation they are provided here. Hence the use of the relation `send-message-with-new-input-role-pairs`.

Every guarded transition execution updates the choreography state.

```
German-buy-train-ticket-service-publisher-
information
web-service-host: "137.108.24.227"
web-service-port: 3001
web-service-location: "/soap"
```

Once the semantic descriptions have been created we 'publish' the web service through a simple dialog where we state the URL of the appropriate publishing platform. The definition created for the `german-train-ticket-service` is shown above. Host, port and location represent also the invariant part of choreography state when a given web service is invoked.

Before running a set of guarded transitions the IRS creates a new ontology which inherits from the ontology in which the web service is defined. All new assertions are made within the new ontology which is deleted after the choreography completes (with `end-choreography`). This allows the IRS to cope with simultaneous goal driven web service requests. Additionally, the ontology is used to capture the current state of a choreography run when a suspend primitive is invoked.

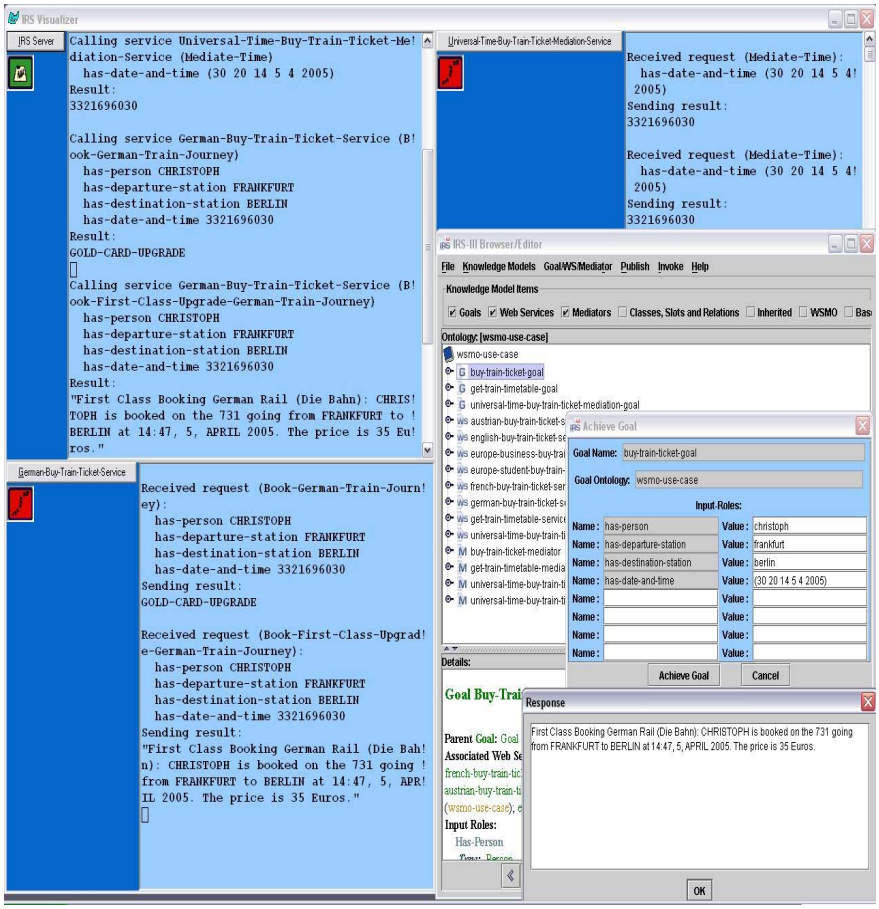

Fig. 3. A screen snapshot showing the VTA running on IRS-III

Figure 3 shows a screen snapshot of the VTA application running in IRS-III. The bottom right of the figure contains three windows. The *Invocation Client* (with title "Achieve Goal") provides a dialog where the client has specified the input role values

for the `buy-train-ticket-goal` (Christoph wants to travel from Frankfurt to Berlin at 14:20:30 on the 5th April 2005). Below the *Response* window shows the final result – Christoph has a first class booking on the German rail system at 14:47 for 31 Euros. Behind the Invocation Client and Response window we can see the *IRS-III Browser/Editor.* The top part displays a list of the goals, web services and mediators defined within the `wsmo-use-case` ontology. The bottom part shows a detailed description of `buy-train-ticket-goal`, where every item, classes, relations and instances, can be inspected by clicking on it.

The main window (titled "IRS Visualizer") is a simple visualization system which displays the interactions between the IRS server and the published web services. The top left pane within the visualization (with the label "IRS Server") shows the goal based requests received by the server and the web service invocation requests sent.

The portion of the interaction history shown contains a call to the `universal-time-buy-train-ticket-mediation-service`, a mediation service which converts the date from (30 20 14 5 4 2005) into 3321696030 format. The `german-buy-train-ticket-service` is called twice. The first call with the implementation component identifier `book-german-train-germany` results in the response `GOLD-CARD-UPGRADE`. This call corresponds to the invocation with the `start` guarded transition. The second call with the implementation component identifier `book-first-class-upgrade-german-train-journey` results in the response shown in the Response window. The second call corresponds to the invocation associated with the `accept-first-class-upgrade` guarded transition.

The pane on the top right of the visualizer (labelled "Universal-Time-Buy-Train-Ticket-Mediation-Service") shows that the mediation service was called twice. As mentioned earlier, during the web service selection process the IRS evaluates the logical expression within the assumption slot of a web service's capability. If the logical expression evaluates to true the corresponding web service is deemed to be selected. Before evaluating the expression the IRS runs the web service's associated mediators to transform the values within the invoked goal instance. The date and time mediation service is run twice because both the German and Austrian rail services within our application use a universal date and time format.

The pane of the bottom left of the visualizer (labelled "German-Buy-Train-Ticket-Service") shows that two invocations were made to the `german-buy-train-ticket-service`. The first with the component identifier `book-german-train-germany` and the second invocation with `book-first-class-upgrade-german-train-journey`. Within the Lisp publishing platform these correspond to a Lisp function name. As mentioned earlier for a standard web service the identifier would correspond to a WSDL operation.

6 Conclusions and Future Work

Enabling heterogeneous software components, available on the internet, to be integrated is a primary aim for research in the area of semantic web services. In this paper we have described how IRS-III is able to handle heterogeneity related to web service interaction patterns through a choreography.

The choreography execution occurs in IRS-III from the client perspective, that is to say, to carry out a web service invocation, the IRS executes the choreography as well as a requester client.

Our underlying design principles are based on the use of ontologies and state, the IRS acting as a broker for capability based invocation, the dimensions of initiative and communication direction, the provision of a formal description, and semantic descriptions which are realised within simple-to-use constructs that can be executed.

We have shown through a detailed example how choreographies can be defined and executed with little effort with our framework. As mentioned earlier a key element of our design is that the choreography component of IRS-III is itself a semantic web service allowing developers to easily replace our choreography execution engine with another if desired.

We have recently used our platform and the choreography execution in various tutorials: at the European Semantic Web Conference (ESWC 2005), the International Conference on Web Engineering (ICWE2005), and the Knowledge Web Summer School (SSSW 2005) and we will continue to evaluate the framework at the European Conference on Web Services (ECOWS 2005) and at this year's International Semantic Web Conference (ISWC 2005).

Additionally, we are currently deploying an IRS-III based application within an e-Government demonstrator in the context of the DIP project.

In relation to future work we plan to semi-automatically generate client choreographies from the choreography descriptions of WSMO-compliant web services.

The IRS-III browser/editor and publishing platforms are currently available at http://kmi.open.ac.uk/projects/irs/. We periodically release executable versions of the server for specific usage contexts.

Acknowledgements

This work is supported by DIP (Data, Information and Process Integration with Semantic Web Services) (EU FP6 - 507483) and AKT (Advanced Knowledge Technologies) (UK EPSRC GR/N15764/01) projects.

References

1. Amazon (2005). Web Services (Available at http://www.amazon.com/gp/browse.html/104-6906496-9857523?%5Fencoding=UTF8&node=3435361).
2. S. Arroyo, S. and A., Duke (2005). SOPHIE - A Conceptual Model for a Semantic Choreography Framework. In proceedings of the Workshop on Semantic and Dynamic Web Processes (SDWP 2005). Orlando, Florida, USA, July 2005.
3. 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, p.1-43, October 1998.
4. Crubezy, M., Motta, E., Lu, W. and Musen, M. (2002). Configuring Online Problem-Solving Resources with the Internet Reasoning Service. IEEE Intelligent Systems 2002.

5. Dijkman, R. and Dumas, M. (2004). Service-Oriented Design: A Multi-Viewpoint Approach. International Journal of Cooperative Information Systems 13(4): 337-368, 2004.
6. DIP (2005). The DIP Project. http://dip.semanticweb.org/.
7. 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. CEUR Workshop Proceedings, ISSN 1613-0073 II.
8. Ellison, L. (2002). Looking Toward the Next Phase for Web Services. (Available at http://webservicesadvisor.com/doc/09586).
9. Fensel, D. and Motta, E. (2001). Structured Development of Problem Solving Methods. IEEE Transactions on Knowledge and Data Engineering, Vol. 13(6). 913-932.
10. 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. (Available at http://CEUR-WS.org/Vol-113/paper7.pdf).
11. Greef, H. P. and Breuker, J. A. (1992). Analysing system-user cooperation in KADS. Knowledge Acquisition, 4:89–108, 1992.
12. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T. and Lafon, Y. (Eds) (2004). Web Service Choreography Description Language Version 1.0. W3C Working Draft 17 December 2004. (Available at http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/).
13. Kerner, S. M. (2004). Web Services Market to Explode (Available at http://www.internetnews.com/dev-news/article.php/3413161)
14. Motta, E. (1998). An Overview of the OCML Modelling Language. In proceedings of the 8th Workshop on Knowledge Engineering Methods and Languages (KEML '98).
15. Motta, E., Domingue, J., Cabral, L. and Gaspari, M. (2003). IRS-II: A Framework and Infrastructure for Semantic Web Services. In proceeding of the 2nd International Semantic Web Conference (ISWC2003). Sundial Resort, Sanibel Island, Florida, USA. LNCS 2870, pp. 306–318.
16. OWL-S Working Group (2004). OWL-S: Semantic Markup for Web Services (Available at http://www.daml.org/services/owl-s/1.1/overview/).
17. Roman, D., Lausen, H. and Keller, U. (Eds) (2005). The Web Service Modeling Ontology WSMO, final version 1.1. WSMO Final Draft D2, 2005.
18. Roman, D., Sciluna, D. and Feier, C. (Eds) (2005). Ontology -based Choreography and Orchestration of WSMO Services. Final Draft D14.
19. Stollberg, M. and Arroyo, S. (2005). WSMO Tutorial. WSMO Deliverable (Available at http://www.wsmo.org/TR/d17/)
20. Stollberg, M. and Lara, R. (Eds) (2004). D3.3 v0.1 WSMO Use Case: Virtual Travel Agency.
21. UDDI (2003). UDDI Spec Technical Committee Specification v. 3.0, http://uddi.org/pubs/uddi-v3.0.1-20031014.htm
22. W3C [a] (2004). Web services choreography model overview. W3C Working Draft 24 March 2004 (Available at http://www.w3.org/TR/2004/WD-ws-chor-model-20040324).
23. W3C [b] (2004). Web Services Architecture. W3C Working Draft 11 February 2004 (Available at http://www.w3.org/TR/ws-arch/).
24. W3C [c] (2004). Web Services Glossary. W3C Working Group Note. 11 February 2004 (Available at http://www.w3.org/TR/ws-gloss/).
25. WSDL (2001). Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.