The Choreography Model for IRS-III

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

This paper describes how we manage the interaction between different heterogeneous web services through choreographies within IRS-III.

IRS-III is a framework and platform for developing WSMO based semantic web services. Our choreography framework is based on the KADS system-client co-operation model that represents communication through two dimensions: the message direction and which actor has the initiative. Our formalism is state based and is thus compliant with Abstract State Machine (ASM) model used within WSMO.

In addition to describing our approach in this paper we provide a formal semantics for IRS Choreography and a full implementation which we illustrate through an example application.

1. Introduction

Web Services promise to turn the web of static documents into a vast library of interoperable running computer programs and as such have attracted considerable interest, both from industry and academia.

Recent efforts to enhance the Web Services technologies have tried to raise the web to a new level of service. Software application available on the Web can be accessed, executed and composed thanks to the idea of Web Services.

Web Services provide a mechanism to connect applications regardless of the underlying software/hardware platform and their location. From an 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 Service (AWS) 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].

Many efforts are still necessary to obtain a complete growth of Web Services. Currently the Web Service technologies fall on the restricted capability to support automated service composition, recognition and comparison. Their limit comes out from the total absence of semantic representation of the services available on the internet. The descriptions, represented in XML-based description languages such as WSDL [23] and UDDI [21], mostly focus on the specification of the input and output data types and the access details. These specifications are obviously not powerful enough to support automatic discovery, mediation and composition of web services. A software agent cannot find out what a web service actually does, by reasoning about a WSDL specification. Analogously the same agent cannot locate the appropriate service in UDDI registry, given a specification of a target functionality. As a result, existing web service infrastructures by and large support a manual approach to web service management: only manual discovery can be supported and only 'static', manually configured web applications are possible.

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. Requiring IT specialists to discover, compose and deploy web services manually is time-consuming, costly and error-prone. Semantic differences remain the primary roadblock to smooth application integration, one which Web Services alone won't overcome [7].

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 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 problem 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 [6]. 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 [5] 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 follow section we provide an overview on WSMO and IRS-III, then we present choreography within IRS-III outlining our formal model, the informal description and the main primitives. In section 4 we describe an example application and the final section concludes the paper.

2. Background

In this section we present an overview on the Choreography definitions, an introduction of the Web Service Modeling Ontology (WSMO) [17] and of the IRS-III framework.

2.1. Choreography definitions

The Choreography of a Web Service defines the communication protocol between the server and a deployed Web Server in terms of message exchanges [22]. Several approaches are currently available to describe the interactions between web services. A global vision of 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 [12]. Dijman and Dumas [4] depict both static and dynamic aspects of the global communication among heterogeneous web services using Petri Nets. Many levels of abstractions are also proposed. W3C depicts three abstraction choreography levels: abstract, portable and concrete [22]. However the requirements emerging from eBusiness necessitate that web services exchange information at a semantic level. Thus the choreography of a semantic web service should include a communication protocol specification which represents functionality at a semantic level.

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.

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.

2.2. WSMO

The *Web Service Modeling Ontology* (WSMO) [17] is a formal ontology and language for describing the various aspects related to Semantic Web Services.

The main goal of WSMO is to enable e-commerce by applying Semantic Web technologies to Web Services. The semi-automated discovery, composition and execution of Web Services is based on logical inference-mechanisms. WSMO has the Web Service Modeling Framework (WSMF) [8] as a starting point; it refines this framework and develops a formal ontology and a formal language. The WSMF is composed by four different main elements: Goals, Web Services, Ontologies and Mediators.

The Goals represent the targets, the final problems solved by Web Services, requested by the user. The WSMO definition of goal concerns the state of the desired information space, describes the desired state of the world after the execution of a given Web Service. A goal can import existing concepts and relations defined elsewhere either extending or simply reusing them as appropriate.

The Web Services description concerns all aspects of the Web Services. They represent the functional behaviour which must be semantically described in order to allow operations of discovering, invocation, composition, execution, and everything concerns its semi-automated use. The interface is part of Web Services, and is composed by Choreography and Orchestration.

Ontologies are the key elements in WSMO, they provide the terminology used by other elements, allow to link machine and human terminologies and define the information formal semantics.

Mediators provide the means to link the three components described above. The incorporation of mediators in WSMO facilitates the clean separation of different interoperability mechanisms.

2.3. IRS-III

The IRS project has the overall aim of supporting the automated or semi-automated construction of semantically enhanced systems over the internet. IRS-I [3] supported the creation of knowledge intensive systems structured according to the UPML framework [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 four 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.

Thirdly, IRS-III is programmable. We allow IRS-III users to substitute their own semantic web services for some of the main IRS-III components.

Finally, IRS-III services are web service compatible – standard web services can be trivially published through the IRS-III and any IRS-III service automatically appears as a standard web service to other web service infrastructures. In the rest of the paper we will use the terms "IRS" and "IRS-III" interchangeably.

3. IRS-III choreography

A choreography is described in IRS-III by a grounding declaration and a set of guarded transitions. The *grounding* specifies the operations involved in the invocation of a Web Service and their mapping to the implementation level. More specifically, the grounding definitions have the following structure: operation-name, input-roles-soap-binding, output-role-soap-binding. The *guarded transitions* are the set of rules applied when executing the choreography.

In the rest of this section we present our choreography model at several levels of abstraction. We provide, at first, the formal model and in following the design principles in a more informal way.

3.1. Formal model

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 2.3.

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 $(1 \le k \le i) \in C$.

A guarded transition updates the communication state by an event $e \in E$.

3.2. Informal description

In this section we list the main design principles which motivate our choreography model.
Our choreography description is state-based, it means that any message sent by the 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 [2] 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.

Once the grounding is defined, the sequence of operations and the message instance 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. We can easily represent the guard transition execution and state updating by a decision tree shown in figure 1. Every node symbolizes a state and the labeled edges are the triggering events by guarded transitions, given a set of condition and a state.

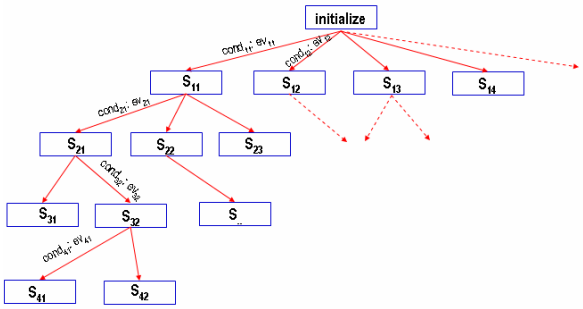

Fig. 1 Decision tree representing the guarded transitions execution.

The general form of a guarded transition is given below:

"**if** currentstate = s ∧ Cond **then** currentstate = s1 "

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 [10] [11], 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.

Fig. 2 A typical sequence of choreography events.

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". Figure 2 depicts the event sequence for this typical goal driven web service invocation scenario.

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. In these case both, IRS and the web service are the ability to suspend the communication and resume it later.

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.

3.3. Choreography primitives

If we want our system to be used widely, it is important that the components are easy to use. For this reason our set of choreography specific primitives is relatively small.

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. At this step 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 (see figure 4).

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 2).

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 3 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.

Received-message. Contains the result of a successful send-message for a specific operation (see figure 4). In the general case the trigged event is "obtain" as shown in figure 2. If however the web service had previously suspended the communication it will be "receive" (see figure 3). 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.

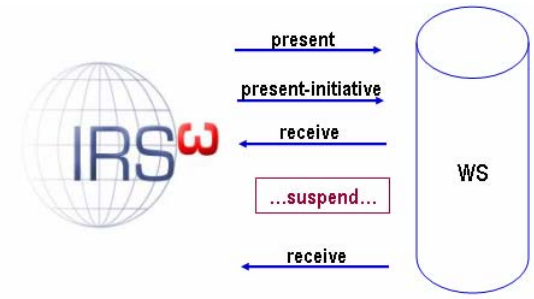

Fig. 3. The occurring events when the web service suspends the communication.

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 2), or "receive" (see figure 3).

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

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 primitives which map a) from the ontological level to XML (lower) and b) from XML to the ontological level (lift) how shown in figure 4.

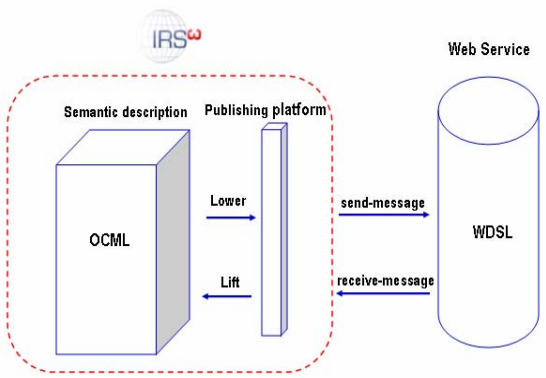

Fig. 4. The main choreography running primitives during the goal invocation.

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.

The IRS publishing platform is 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.

4. An example: Virtual travel agency (VTA)

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.

As mentioned earlier one of the overall design features for IRS-III is that it is open. The main components of IRS-III are defined as semantic web services which can be overridden by developers if desired. Below we show the goal definition for the choreograhpy component.

```
Run-Choreography-goal
Input Roles:
  Has-web-service web-service "sexpr"
  Has-input-role-value-pairs
  list "sexpr"
Output Role:
  Has-output ocml-thing "sexpr"
```

The choreography starts with the guarded transition containing init-choreography and it ends with the end-choreography execution.

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.

The goal has four input roles: Has-person, Has-departure-station, Has-destination-station, Has-date-and-time. Each input role has a type which is a class from the underlying ontology and a soap binding. For example, Has-departure-station has the type city and sexpr soap type (we created our own specific soap type for Lisp s-expressions). The goal has a single output role Has-booking-order which has the type string and soap type string. The date and time are given in a list format containing seconds, minutes, hours, date, month and year e.g. (30 20 14 5 4 2005) for 14:20:30 on 5th of April 2005.

```
Buy-train-ticket-goal
Input Roles:
 Has-person person "sexpr"
 Has-departure-station city "sexpr"
 Has-destination-station city "sexpr"
 Has-date-and-time list-date-and-time
 "sexpr"
Output Role:
 Has-booking-order string "string"
```

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

```
German-buy-train-ticket-service
Input Roles:
 Has-date-and-time
 universal-date-and-time "int"
Capability:
 german-buy-train-ticket-capability
Interface:
  german-buy-train-ticket-service-
  interface
```

The train ticket service `German-buy-train-ticket-service` simply points to the functional definition contained in the capability and the deployed web service details contained in the interface. The web service is linked to the buy-train-ticket-goal through the `universal-time-buy-train-ticket-mediator` (the definition is below). Web service's linked in this way automatically 'inherit' the input and output roles from the associated goal. For the definition above we have overridden the type and soap bindings for one of the goal's input roles has-date-and-time. The deployed web service which implements `German-buy-train-ticket-service` takes time in universal time format which is a single integer.

```
German-buy-train-ticket-capability
Used-mediator: universal-time-buy-train-
ticket-mediator
Assumption:
    (kappa (?goal)
      (and (not (student (wsmo-role-value
      ?goal 'has-person)))
         (not (business-person
         (wsmo-role-value
        ?goal 'has-person)))
         (is-in-country
         (wsmo-role-value
         ?goal 'has-departure-station)
                           germany)
         (is-in-country
         (wsmo-role-value
        ?goal 'has-destination-station)
                          germany))))
```

The `German-buy-train-ticket-capability` definition contains the `universal-time-buy-train-ticket-mediator` which is shown below. The assumption contains an OCML expression which is used by IRS-III to select the web service. The assumption expression above states that the `German-buy-train-ticket-service` should be selected if the person is not a student or business person and both the departure and destination stations are in Germany.

```
Universal-time-buy-train-ticket-mediator
source-component:
 buy-train-ticket-goal
mediation-service:
 universal-time-buy-train-ticket-
 mediation-goal
```

The mediator above transforms a list representing the date and time given to `Buy-train-ticket-goal` into a single integer in universal time format. The mediation service `universal-time-buy-train-ticket-mediation-goal` is a standard WSMO goal with an associated web service which is able to carry out the transformation.

```
German-buy-train-ticket-service-
interface
choreography:
 german-buy-train-ticket-service-
 choreography
```

The interface above simply points to the choreography for the web service.

```
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
```

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 is 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`.

```
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.

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 ends (with `end-choreography`). This allows the IRS to cope with simultaneous goal driven web service requests.

5. Conclusions and future work

In this paper we have described how IRS-III is able to handle heterogeneity related to web service interaction patterns through a choreography.

Enabling heterogeneous software components, available on the internet, to be integrated is a primary aim for research in the area of semantic web services. Our underlying design principles are based on the use of state, the dimensions of initiative and communication direction, the provision of a formal

description, and semantic descriptions which are based on simple-to-use constructs and 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 are currently deploying an IRS-III based application within an e-Government demonstrator in the context of the DIP project. The IRS-III browser/editor and publishing platforms are currently available at http://kmi.open.ac.uk/projects/irs/.

Ackowledgements

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] 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.

[3] Crubezy, M., Motta, E., Lu, W. and Musen, M. (2002). Configuring Online Problem-Solving Resources with the Internet Reasoning Service. IEEE Intelligent Systems 2002.

[4] Dijkman, R. and Dumas, M. (2004). Service-Oriented Design: A Multi-Viewpoint Approach. International Journal of Cooperative Information Systems 13(4): 337-368, 2004.

[5] DIP (2005). The DIP Project. http://dip.semanticweb.org/.

[6] Domingue, L. Cabral, F. Hakimpour,D. Sell and E. Motta (2004). IRS III: A Platform and Infrastructure for Creating WSMO-based Semantic Web Services. Proceedings of the Workshop on WSMO Implementations (WIW 2004) Frankfurt, Germany. CEUR Workshop Proceedings, ISSN 1613-0073 II.

[7] Ellison, L., (2002). Looking Toward the Next Phase for Web Services. (Available at http://webservicesadvisor.com/doc/09586).

[8] D. Fensel, D. and Bussler, C. (2002). The web service modeling framework wsmf. Electronic Commerce Research and Applications, 1(2):113–137, 2002.

[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. 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., 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, 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., Keller, U. (Eds) (2005) The Web Service Modeling Ontology WSMO, final version 1.1. WSMO Final Draft D2, 2005.

[18] Roman, D., Sciluna, D., 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., 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 (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] WSDL (2001) Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315.