WSMO Deliverable

# D13.2 v0.2
# WSMX Execution Semantics

WSMO Working Draft – 6th December 2004

**Authors:**
E. Oren and M. Zaremba

**Editors:**
M. Zaremba

**This version:**
http://www.wsmo.org/2004/d13/d13.2/v0.2/
**Latest version:**
http://www.wsmo.org/2004/d13/d13.2/
**Previous version:**
http://www.wsmo.org/2004/d13/d13.2/v0.1/

Copyright © 2004 DERI ® , All Rights Reserved.
DERI liability, trademark, document use, and software licensing rules apply.

# 1 Introduction

The Web Services Execution Environment (WSMX) is an execution environment for dynamic mediation, discovery and invocation of Web Services. WSMX uses WSMO to describe all aspects related to this mediation, discovery and invocation.

The goal of WSMX is to provide both a testbed for WSMO and to show the viability of using WSMO to achieve dynamic interoperable Web Services. The complete work on WSMX includes defining the conceptual model, defining an execution semantics for the environment, describing an architectural and software design and building a working implementation.

WSMX is based on event-driven architecture composed of loosely coupled components. This kind of architecture facilitates creating execution semantics for system since components activities are stimulated by appearing events and there is no fixed bindings between components. Components can create or consume events but they cannot invoke each other directly (strictly speaking, only core component can interact with other components).

In this deliverable we define execution semantics of WSMX. Accommodating for readers unfamiliar with the term execution semantics, we first describe what is meant by this concept and why we model the execution semantics. Next we explore different modelling techniques for defining execution semantics in existing literature and choose an appropriate modelling technique. Using this technique we present four mandatory WSMX execution semantics, so that its operational behavior is formally and unambiguously specified. We are aware that there will demand for defining execution semantics for WSMX by third parties. There is separate section characterizing this issue.

# 2 Methodology

## 2.1 What is execution semantics

Execution semantics, or operational semantics, is the formal definition of the operational behavior of a system. It describes in a formal language how the system behaves. Because the meaning of the system (to the outside world) consists of its behavior, this formal definition is called execution semantics.

## 2.2 Why are we modelling execution semantics

WSMX has two functions in the complete body of WSMO: it serves both as a testbed for WSMO and as an example implementation. This example implementation could for instance be used to demonstrate the viability of WSMO or as a reference for others that want to build their own WSMO execution environment. Contrary to the rest of the WSMO work it is clearly not prescriptive, it does not tell others how to build a WSMO execution environment. In that sense, the execution semantics described here are strictly part of the design process of WSMX. Its meaning and relevance should be found in improving either the design process or the result of this process, the actual software.

A perfect design process (not just software design) should result in a design that is both an adequate response to the user's requirements and a feasible directory for the implementor who will build the end result. A design therefore serves two purposes, both to guide the builder in its work, and to certify that

what will be built will satisfy the user's requirements.

### 2.3 How do we model execution semantics

Several methods exist to formally model software behaviour. These methods have different characteristics: some are more expressive than others, some are more suited for a certain problem domain than others. Some methods are graphical, some are logical; some methods have tool support for modelling, for verification, for simulation or for automatic code generation and others don't.

To choose a method, we first define our requirements: first of all, the method should be as expressive as needed to define behaviour of the software. Then, as outlined before, since the main advantage of using formal methods lies in improving the developers' understanding of the system, the resulting model should be easily understandable and unambiguous in its meaning. Thirdly, the method should allow verification of certain interesting properties of the modelled system. Lastly, since some methods are better suited for modelling a certain problem domain than others; the method should be suitable to model our specific problem domain [3].

## 3 Formal model of execution semantics

### 3.1 Modelling technique

We will describe the execution semantics using classical Petri nets. The used tool, CPNTools [4], offers the possibility to model so-called high-level Petri-nets [1], extending classical Petri nets with hierarchy, colour and time.

Hierarchy adds the possibility to decompose a transition into so-called subnets, which allows to break down a large model into smaller pieces and to model incrementally. Timed Petri nets introduce the notion of a global time, in which every token gets a certain time-stamp; this provides the possibility to describe time duration of transitions.

Coloured Petri nets are classical Petri nets extended with the notion of identity. The addition of colour means basically that tokens can be distinguished from each other, for instance by giving every token a certain type and value. Since every token now has a certain value, every transition gives its output tokens certain values; or said otherwise, a transition now becomes a relation between the value of the input tokens and the value of the output tokens. In addition, a transition can state conditions over its input tokens, that must be satisfied before the transition can become enabled.

Extending Petri nets with hierarchy and colour does not improve the expressivity of a Petri net, but does make them more concise and readable.

### 3.2 Model checking and simulation

The tool we are using makes it possible to verify certain properties of the model; it can check some simple properties such as syntactical correctness, unreachable (unused) places, or unsatisfiable conditions. The tool also allows for complex analysis of the constructed model, using state-space analysis (which is basically an exhaustive search through all possible states of the model).

The tool also allows for simulation of the constructed model. This simulation

is very useful, since it greatly enhances the modeller's understanding of the system. When a modeller is not completely well-known with Petri nets, it is quite easy to construct a Petri net that does not follow the modeller's intention. When running a simulation, these errors will easily be detected.

The simulation also helps greatly in understanding the system's functionality, and in discussing the model with others. In that sense, simulation serves as an abstract prototype of the future system, it can be used to test and analyse the modelled system in detail. To allow for simulation, our model is available for download[1].

# 4 WSMX mandatory execution semantics

Four mandatory execution semantics are required to be provided for each instance of WSMX system. Each predefined execution semantic has its specific entry point. Service requester by selecting an appropriate WSMX entry point chooses system behavior. Since WSMX is event driven system, its behavior is determined by order of events. Execution semantics are run by execution manager and selection of them is made by choosing WSMX entry point. Execution manager dispatchs events in WSMX according to selected formal specification. Please note that current version of WSMX works with WSMO conceptual model (first version of WSMX has been based on WSMX-O - a limited set of concepts derived from WSMO-Standard).

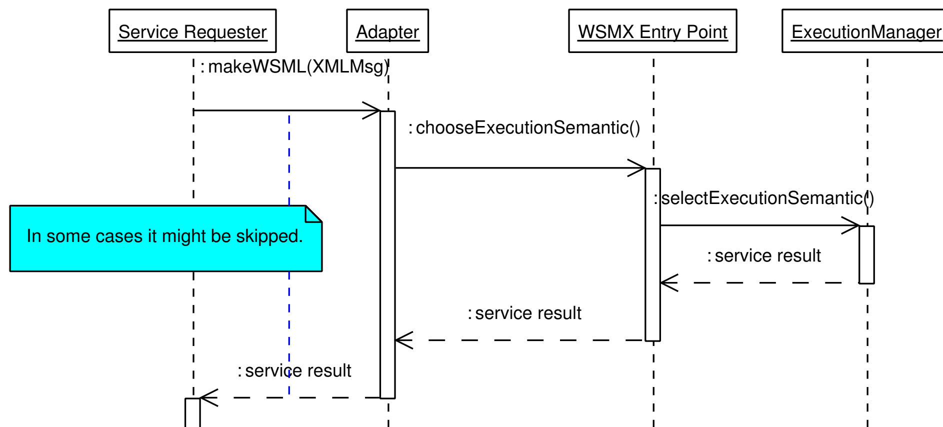


Figure 1: General entry point selection path

Each execution sematic follow the path depicted on Figure 1. However not all steps are mandatory. If service requester can understand WSML then it is not necessary to use an Adapter component.

The execution semantics of WSMX follows the component-based paradigm. This means that the execution semantics of the complete system treats components as 'black-boxes' - we do not model decisions that take place inside those components. For a complete model of the behaviour of the system the execution semantics of the components needs to be taken into account. Modelling the execution semantics of the individual components is however the responsibility of the various component owners. Components behaviour and interfaces

[1] all models created in CPN Tools are available here: wsmxcpn.zip; the tool itself can be downloaded from CPN Tools

should follow guidelines described in WSMX architecture document[new version is about to come].

## 4.1 One way goal execution

Following entry point realizes this behavior.
**realizeGoal(Goal, OntologyInstance):Confirmation**

Service requester, which expects WSMX to discover and invoke Web Service without exchanging additional messages can use this entry point by providing the formal description of Goal (in WSMO terms) and fragment instance of Ontology. This simplified scenario is based on assumption that service requester has a priori knowledge of all the data required by Web Service provider. WSMX discovers and executes service on behalf of requester. If necessary mediation can be performed. It is not mandatory to send a final confirmation by WSMX since some entities might not have permanent addressing (e.g. request could be send from dynamic IP Internet connection and user has already shutdown his connection). Usage of this execution semantic is very restricted and in real live solutions it will be rather used for testing purposes or for fixed services bindings.

In figure 2 the definition of the system is given. The behaviour of some components is specified in subsequent diagrams, which is denoted by a blue rectangle underneath the transition (e.g. discovery subnet).

First of all a list of *known web services* is created by combining internally known Web Services with external ones in a case that service requester wants to check them too. From this list, one Web Service is picked and discovery is tried. If necessary, mediation is asked for, by placing a token in the place *need mediation*. This mediation can succeed, after which the discovery can continue. The mediation can also fail, after which a new Web Service is needed (the chosen Web Service cannot be mediated, and is useless for this goal).

The discovery process continues until a useful Web Service is found (with or without mediation). If no Web Service is found, the *no more ws* transition is fired, resulting in a *discovery error*. This means that all the Web Services have been tried for the discovery, but none of them succeeded.

After discovery the *selection* component selects the web service that best fits the preferences specified in the goal[2]. Finally, the *invocation* component invokes the selected web service.

The process of discovery is specified in figure 3. This models that when deciding whether a web service matches a goal, three situations can occur: either there is a discovery (denoted by *discovery ok*), or there is no discovery which means the discovery should retry using another web service (denoted by *discovery error, new ws needed*. The third possibility is that mediation is needed, denoted by placing a token in *need mediation*, and waiting until this mediation is successfully finished *mediation ok*.

The discovery is (from the viewpoint of the WSMX system) a nondeterministic choice, either a discovery is found, or an error occurs or a mediation is needed; this choice is made not by the WSMX system, but by the *discovery* component.

This nondeterministic exclusive or is modelled by having an output place *discovery_out*, whose token is consumed by either by the *discovery_ok* transition, or by the *discovery_error* transition, or by the *need_mediation* transition.

[2]please note, that in the current version of this specification a successful discovery results in exactly one web service; however by collecting results from several invocations of discovery subnet, list of Web Services can be created

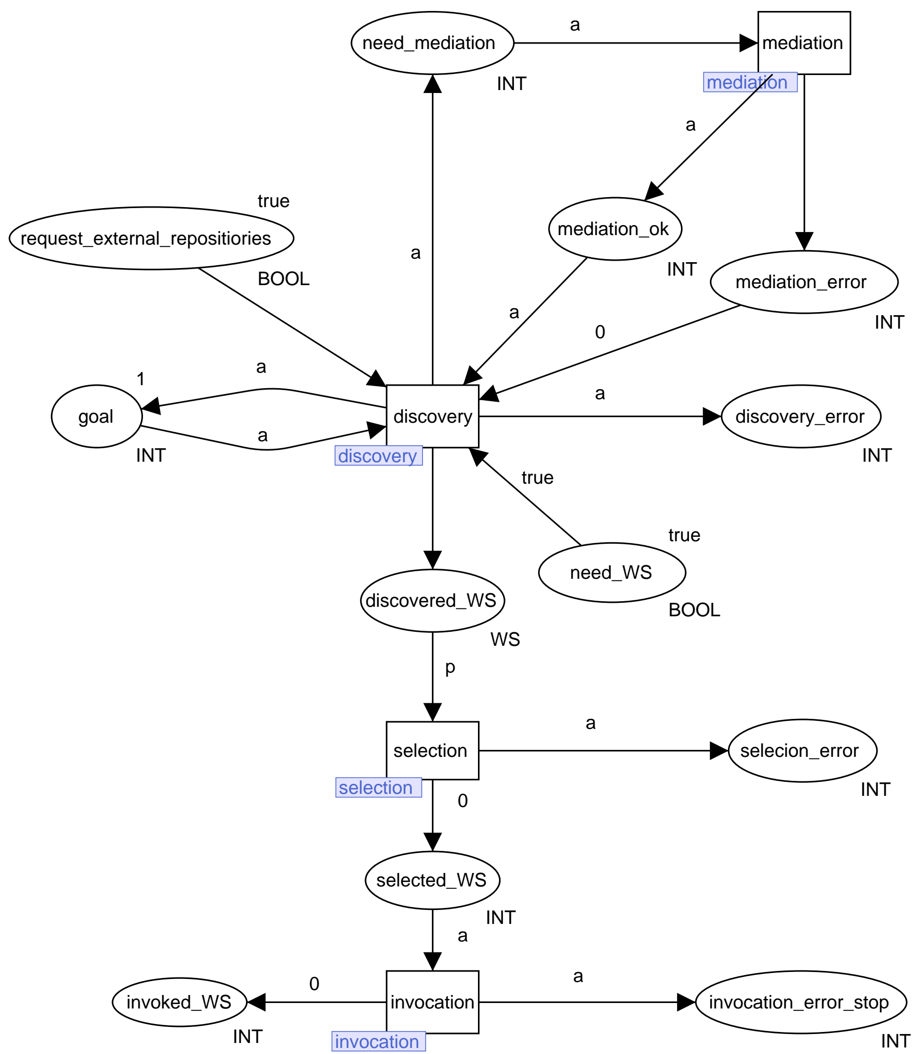


Figure 2: Overview of one way goal execution

Figure 3: Discovery inside WSMX

The same pattern repeats for modelling the other components, all of whose outcomes are nondeterministic exclusive or's (from the viewpoint of WSMX). Both the *mediation* component and the *selection* component can either fail or succeed, from the viewpoint of WSMX. This is modelled in figures 4 and 5.

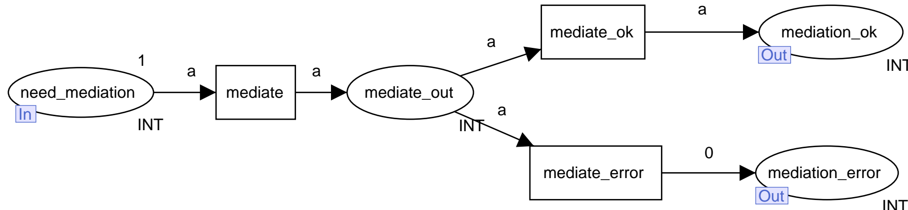


Figure 4: Mediation inside WSMX

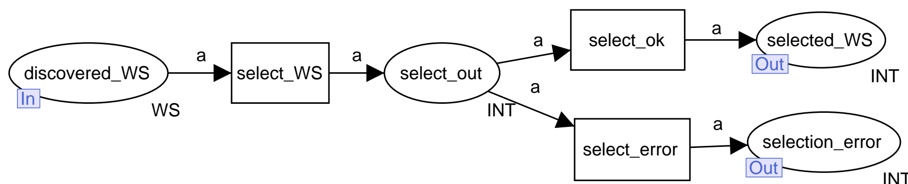


Figure 5: Selection inside WSMX

The *invocation* component is modelled slightly different, since this component to retry invocation a number of times, in case of network time-outs or other temporary errors. This component is shown in figure 6; each time an error occurs, a counter is incremented. If the counter does not exceed a certain threshold, the invocation is retried; otherwise, the invocation fails.

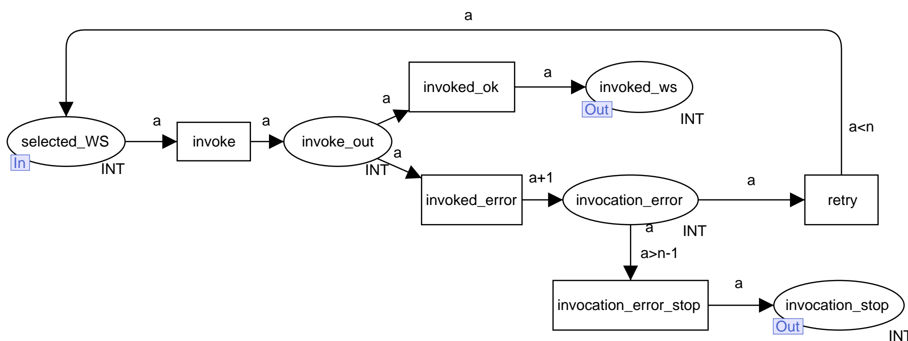


Figure 6: Invocation inside WSMX

## 4.2 List of Web Services fulfilling given Goal

Following entry point realizes this behavior.
**receiveGoal(Goal, OntologyInstance, Preferences):WebService[]**

List of Web Services is created for a given Goal and Instance of Ontology. Additionally in Preferences requester can specify number of Web Services to

be returned. Only Discovery and Mediation components carry out their tasks. Each Web Service in WebServices[] list is already mediated to requester Ontology if required and has its choreography [2]. Since returned list of Web Services are specified in the same Ontology as requester's one so requester can understand them and make a choice which one should be executed. In choreography exchange message patterns are specificated, thus both parties know what data and in what order are expected to fulfill requester Goal.

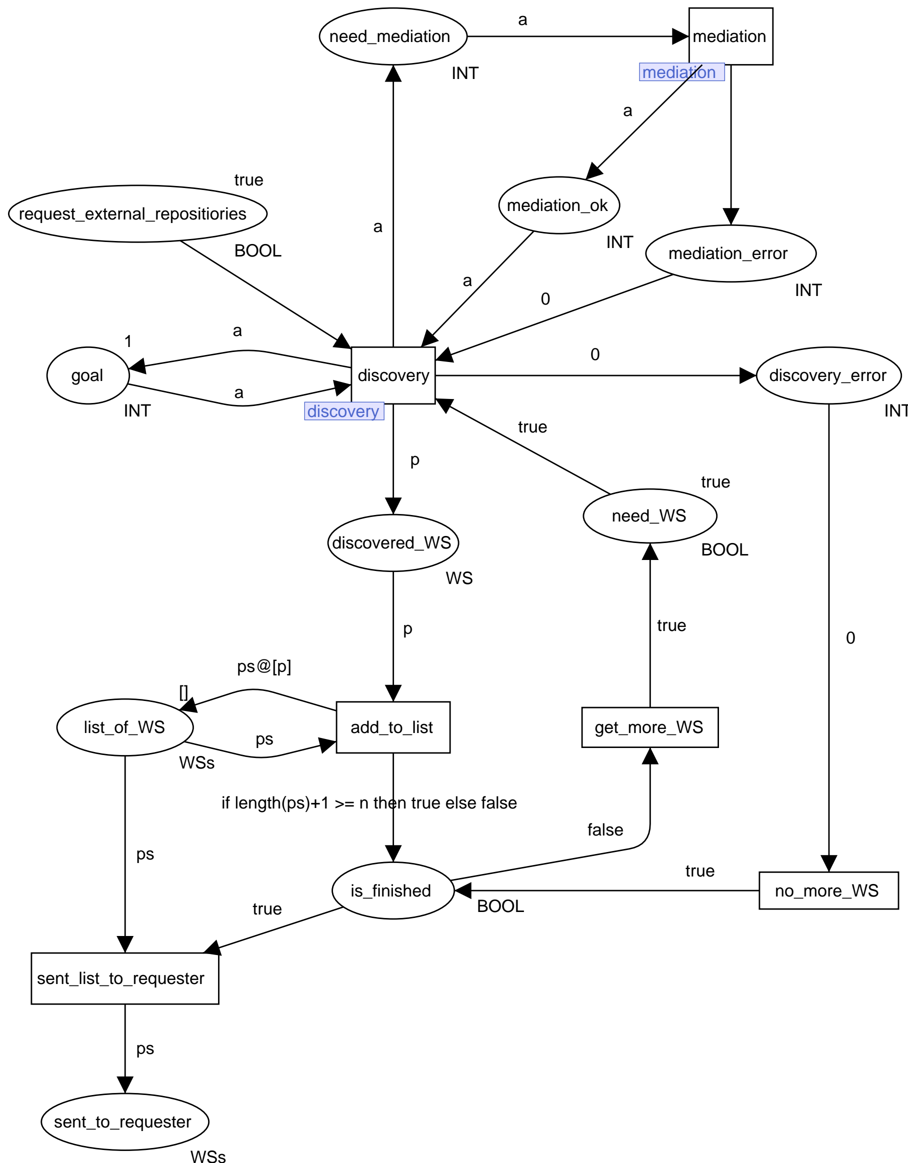


Figure 7: Overview of list of Web Services fulfiiling given Goal

Communication with service is carried out through WSMX receiveMessage(Goal, WebService, ChoreographyID):ChoreographyID entry point. This execution semantic is quite relevant when decision which Web Service to execute has to be made outside WSMX system. Decision could be taken manually or by some Web Service evaluation program.

Process of generating list of Web Services fulfilling given Goal is depicted on Fig. 7. Discovery subnet is invoked in loop as long as desired number of Web

Services is not collected or there is no more Web Services to discover. Finally, list of discovered Web Services is sent to service requester.

## 4.3 Web Service execution with choreography

Following entry point realizes this behavior.
**receiveMessage(OntologyInstance, WebServiceID, ChoreographyID):ChoreographyID**

Once service requester knows Web Service which he wants to use, back and forth conversation has to be carried out with WSMX system to provide all the necessary data to make execution of this Web Service feasible. By giving fragments of Ontology Instances (e.g. business documents such as Catalogue Items or Purchase Orders in a given ontology) it provides all data required by Web Service.

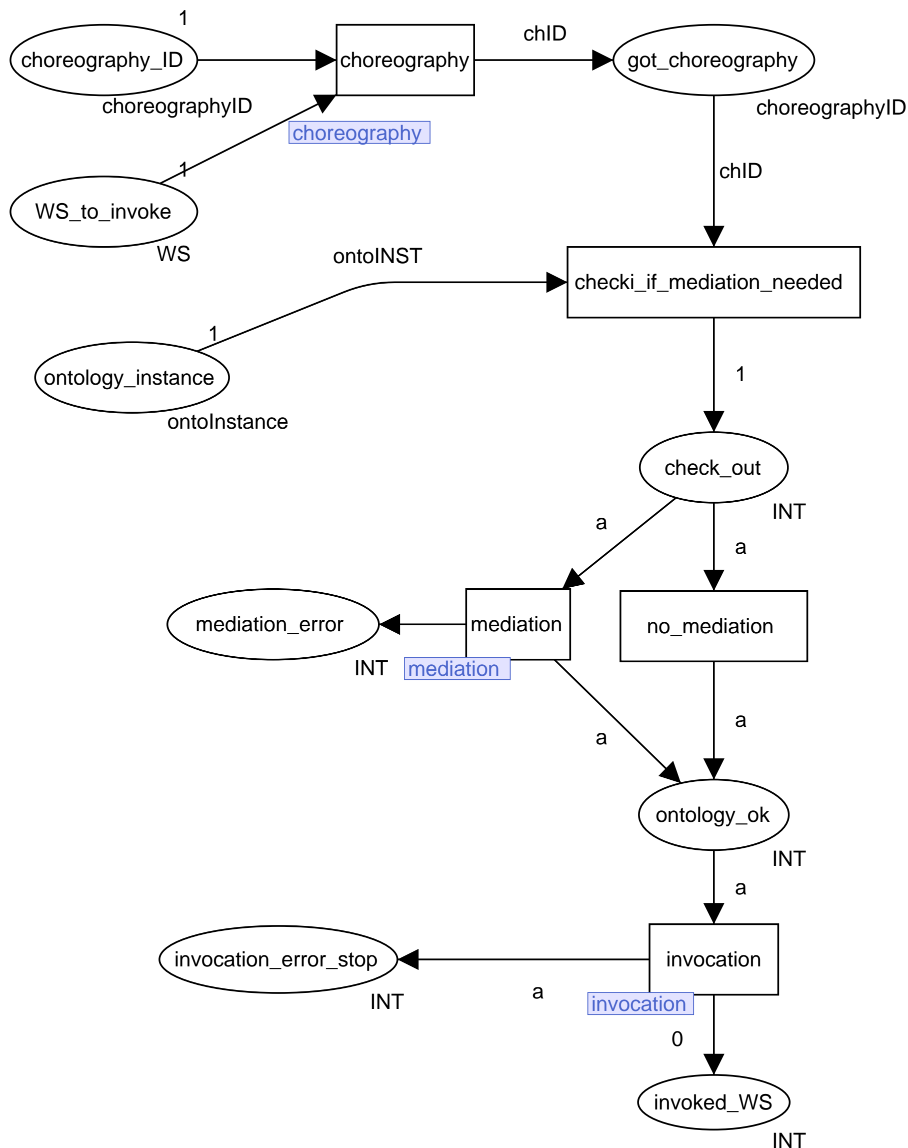


Figure 8: Overview of Web Service execution with choreography

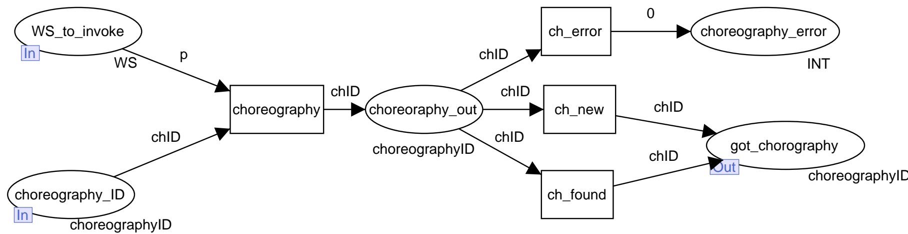


Figure 9: Choreography

This is much more realistic scenario. There is communication according to choreography previously returned together with found Web Service and Ontology Instance (Fig. 8). As depicted on Fig. 9, first time this entry point is invoked, new instance of choreography associated with given Web Service is created. Unique identifier is given to refer to newly created choreography. In next invocation this unique identifier is used to determinate currently processed step in choreography.

Next, when choreographyID is found or created communication can be carried on. It is checked whether for a given Ontology Instance mediation is required, afterwards invocation is performed.

Choreography might include miscellaneous activities. There could be for instance confirmations steps or decisions on requester side included. Main goal of Choreography is to compose Web Services and to execute them as requester expects.

### 4.4 Store WSMO entity

Following entry point realizes this behavior.
**storeEntity(WSMOEntity):Confirmation**

Store Entity entry points provides an administration interface for the system enabling to store any WSMO related entities (like Web Services, Goals, Ontologies) and making them available for other parties using WSMX system

## 5 Dynamic execution semantics

Since WSMX consists of loosely coupled components, one could easily imagine other execution semantics that will appear in future. Thanks to its architecture it can be easily enhanced with new components, components might be dynamically plug-in or plug-out. New versions of components can replace outdated ones in this manner. Since communication in WSMX is based on JMX, any component, even one deployed on some remote machine, can subscribe to WSMX Manager and receive events to process. This gives designer a flexible way to create new execution semantics.

New executions semantics specification should not be restricted to one language. They even should not be based on one paradigm (in this deliverable we consider only colour Petri Net based representation). In order to create interoperable WSMX system, WSMX should take advantage of efforts like M3PE or WFMC with its XPDL initiative. These efforts are concerned around creating interoperable workflow language that other languages would be able to map to.

Their ultimate goal is to be able to execute any workflow specification within one engine.

# References

[1] W.M.P. van der Aalst, K.M. van Hee, and G.J. Houben. Modelling workflow management systems with high-level Petri nets. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.

[2] Emilia Cimpian, Uwe Keller, Michael Stollberg, and Dieter Fensel. Choreography in WSMO. DERI Working Draft v01, 2004.

[3] B. Kiepuszewski. *Expressiveness and Suitability of Languages for Control Flow Modelling in Workflows*. PhD thesis, Queensland University of Technology, Brisbane, Australia, 2002.

[4] A.V. Rantzer, L. Wells, H.M. Lassen, M. Laursen, J.F. Qvortrup, M.S. Stissing, M. Westergaard, S. Christensen, and K. Jensen. Cpn tools for editing, simulating and analysing coloured petri nets. In W.M.P van der Aalst and E. Best, editors, *Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003*, volume 2679 of *Lecture Notes in Computer Science*, pages 450–462. Springer-Verlag, 2003.

# 6 Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, SWWS, Esperonto and h-TechSight; by Science Foundation Ireland under the DERI-Lion project; and by the Vienna city government under the CoOperate program. The editor likes to thank to all the members of the WSMO working group for their advice and input into this document.