WSMX Deliverable

D13.2 v0.3
WSMX Execution Semantics

WSMX Working Draft – 24th October 2005

Authors:
Maciej Zaremba

Editors:
Maciej Zaremba

This version:
http://www.wsmo.org/TR/d13/d13.2/v0.3/20051024/
Latest version:
http://www.wsmo.org/TR/d13/d13.2/
Previous version:
http://www.wsmo.org/TR/d13/d13.2/v0.2/20051010/

Copyright © 2005 DERI ®, All Rights Reserved.
DERI liability, trademark, document use, and software licensing rules apply.

Contents

1 Introduction **3**
1.1 Purpose of this Document . . . 3
1.2 Document Overview . . . 3

2 Methodology **4**
2.1 Rationale behind using Execution Semantics . . . 4
2.2 UML Activity Diagrams . . . 4

3 WSMX mandatory execution semantics **5**
3.1 One-way goal execution . . . 7
3.2 List of Semantic Web Services fulfilling given Goal . . . 10
3.3 Semantic Web Service execution with choreography . . . 11
3.4 Register communication with WSMX . . . 12

4 Dynamic execution semantics **12**

5 Glossary **13**

1 Introduction

The Web Services Modelling Execution Environment (WSMX) is an execution environment for dynamic discovery, mediation and invocation of Semantic Web Services. WSMX is based on the Web Services Modelling Ontology (WSMO)[10], an ontology for describing various aspects related to Semantic Web Services. Web Service can be registered with WSMX by describing it in terms of WSMO, using the Web Service Modelling Language (WSML)[5], and then by invoking a registration interface provided by WSMX. A service requestor with a goal to achieve, submits their goal to WSMX. The WSMX environment takes responsibility for matching the requestor's goal to capabilities of Semantic Web Services registered to WSMX, selecting the most appropriate Semantic Web Service, mediating between the ontologies of service requestor and provider, and carrying out conversation between service requestor and picked up Semantic Web Service.

In a nutshell, execution semantics specifies a system behavior in a formal way what makes it unambiguous and enables its simulation and analysis prior to its execution. WSMX is based on event-driven architecture composed of loosely coupled components what facilitates the creation of various execution semantics for a system since the activities of the components are stimulated by events as they occur and there are no fixed bindings between components. Components can create or consume events but they cannot invoke each other directly. Components can cooperate with each other on interface level but they do not refer to each other directly. Strictly speaking, only core component provides facilities to the component enabling interactions with other components.

1.1 Purpose of this Document

In this deliverable we define the execution semantics of WSMX. 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 give the rationale behind using execution semantics and choose an appropriate modelling technique for our purpose. Using this technique we present execution semantics of four mandatory WSMX behaviours (each WSMX instance has to provide them), so that its operational behavior is formally and unambiguously specified. We are aware that there will be a demand for defining new execution semantics for WSMX by third parties tailored to their needs. These new dynamically created execution semantics can be created and feed to WSMX taking advantage of its loosely coupled components. There is a separate section characterizing this issue.

1.2 Document Overview

Section 2 answers what is execution semantics, why are we modelling it, and what methods can be used for this purpose. Section 2.1 provides a rationale behind using formal methods for software modelling, whereas Section 2.2 gives an description of a chosen methodology. Section 3 provides a detailed description of four mandatory execution semantics that has to be provided with each instance of WSMX, and Section 4 gives a description of dynamic execution semantics that can be feed by a third parties to the WSMX.

2 Methodology

2.1 Rationale behind using Execution Semantics

A design process should result in a design that is both an adequate response to the user's requirements and an unambiguous model for the developers who will build actual software. A design therefore serves two purposes, both to guide the builder in the work of building the system, and to certify that what will be built will satisfy the user's requirements. Actual software should reflect

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

Major rationale behind using formal methods for system modelling is two-fold:

- **Foundations for model testing.** It is highly desirable to perform simulation of the model before the actual system is created and enacted . It allows one to detect anomalies like: deadlock, livelock or tasks that are never reached. However, as pointed out by Dijkstra in [6] model simulation allows to point out presence of errors, but not lack of them. Nevertheless, it is a paramount to detect at least some of system malfunctions during a design time instead of the run-time. Therefore, semantics of utilized notations has to be perfectly sound in order to create tools enabling simulation of created models. Only formal, mathematical foundations can meet this requirements.

- **Improved model understanding among humans.** Giving sound foundations for utilized notation rules out ambiguities in model comprehension by involved parties.

Several methods exist to model software behaviour. Some of them model system behaviour in a general way like UML diagrams, other impose more formal requirements on model like Petri nets based methods. These methods have different characteristics: some are more expressive than others, some are more suited for a certain problem domain than others. Some methods provide graphical notation like UML or Petri nets, some are based on logical terms like Fuzzy logic; some methods have tool support for modelling, verification, simulation or automatic code generation and others do not.

We impose two major requirements on the methodology utilized for modelling WSMX behaviour. Firstly it has to use understandable and straightforward graphical notation, secondly it has to be unambiguous. These two requirements are met by UML Activity Diagrams with execution semantic specified by Eshuis in [7].

2.2 UML Activity Diagrams

UML 2.0 (Unified Modeling Language) is a widely accepted and applied graphical notations in software modelling. It compromises of a set of diagrams for system design capturing two major aspects, namely static and dynamic system properties. Static aspects specify system structure, its entities (objects or components) and dependencies between them. These structural and relational aspects are modelled by diagrams like: Classes Diagram, Component Diagram and Deployment Diagram. Dynamic aspects of the system are constituted by

control and data flow within the entities. Diagrams like: Sequence Diagram, State Machine Diagram, Activity Diagram capture these aspects. Originally UML was created for modelling aspects in object-oriented programming (OOP). However, it has to be emphasized, that UML is not only restricted to the usage in OOP area, but is also applied in other fields like for instance Business Process Modelling.

Main goal of UML diagrams is to enable common comprehension of structure and behaviour of modelled software among the involved parties (e.g. designers, developers, stakeholders, etc.). To the detriment of UML notation, this information is conveyed in informal way that may lead to ambiguities in certain cases as pointed out in [9]. Since we want to model behaviour of WSMX in formal, unambiguous yet easily comprehensive manner thus appropriate modelling method has to be chosen.

Due to the wide proliferation of UML notation several efforts were carried out to firm up its execution semantics, especially regarding the dynamic aspects of UML notation. Execution semantic of UML Activity Diagrams applied in area of workflow modelling was specified in [8, 7]. UML Activity Diagrams fulfill our requirements, therefore they are going to be used throughout this deliverable according to their semantics given by Eshuis in [7] to specify operational behaviour of WSMX

Activity Diagram depict a coordinated set of activities that captures behaviour of modelled system. Activity Diagrams specify a control and data flow between the entities providing language constructs that enable to model elaborate cases like parallel execution of entities or flow synchronization.

3 WSMX mandatory execution semantics

We define four mandatory execution semantics for each instance of WSMX system that are described in further subsections of this document. The WSML message content determines which of the predefined execution semantics will be selected. Since WSMX is an event driven system, its behaviour is specified by the order of events. Event exchange is conducted via Tuple Space[1] which provides persistent shared space enabling seamless interaction between components without direct events exchange between them. In the future WSMX is going to use a Triple Space[2] where exchanged entities are RDF triples what provides far more elaborate querying capabilities.

Interaction is carried out by exploiting a publish-subscribe mechanism. Execution semantics are started according to selected entry-point method (i.e. it initiates exchange of events via Tuple Space). WSMX works with the WSMO conceptual model and with the wsmo4j [1] data model that is compliant with the WSMO specification v1.0.

Each WSMX execution semantics follows the path depicted on Figure 1. Although using Adapter component to communicate with WSMX is not mandatory if the service requestor can understand WSML. In such a case, service requestor itself has to implement *receive(WSMLMessage, Context)* method in order to receive asynchronous WSML messages from WSMX.

The execution semantics of WSMX follows the component-based paradigm, taking advantage of various loose coupled WSMX components as depicted on Figure 2. 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

[1]http://wsmo4j.sourceforge.net

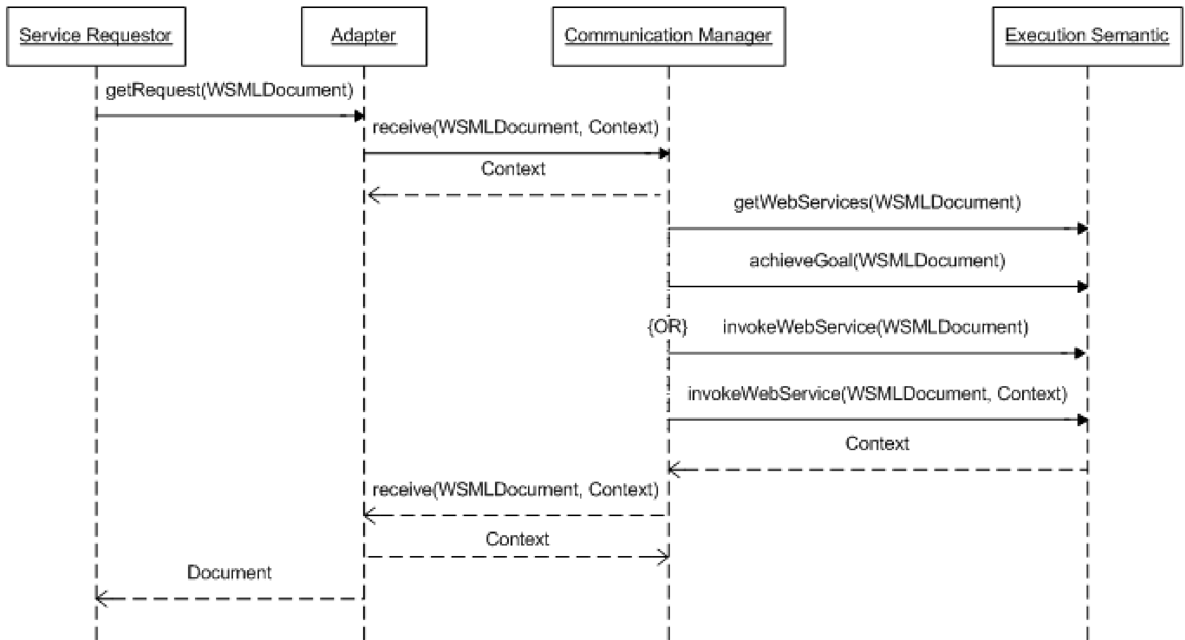

Figure 1: General entry point selection path

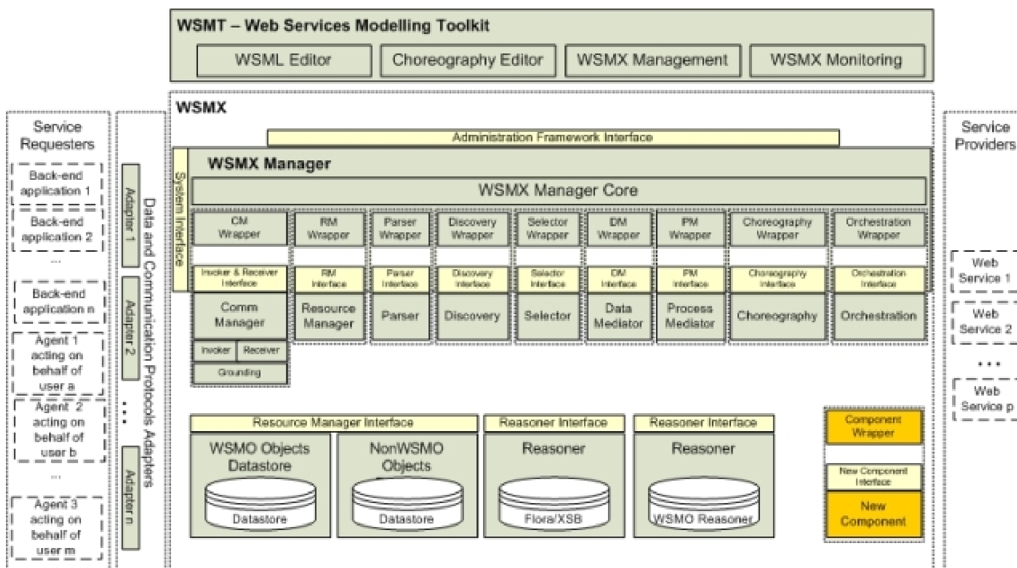

Figure 2: WSMX architecture

the internal behaviour of the components needs to be taken into the account. Modelling the execution semantics of the individual components is, however, the responsibility of the component owners. Component behaviour and interfaces should follow the guidelines described in WSMX architecture document [11]. Following subsections will present execution semantics of WSMX in more detail.

3.1 One-way goal execution

The following entry-point initiates this system behavior.
achieveGoal(WSMLDocument): Context

The service requestor, which expects WSMX to discover and invoke a Semantic Web Service without exchanging additional messages can use this entry-point by providing the formal description of Goal (in WSMO terms), an instance of Ontology (i.e. message to be sent), and a preferences that specifies whether external repositories of Web services should be searched and contains requirements for non-Functional properties (e.g. QoS, security, financial aspects, etc.). All these information are contained in WSML Document, whilst return Context is a unique identifier that allows to associate ongoing conversation and is also used for confirmation of message receival.

This simplified scenario is based on the assumption that service requestor has prior knowledge of all the data required by the Semantic Web Service provider. WSMX discovers and executes the service on behalf of requestor. If necessary, Data Mediation component can be asked for the assistance during Discovery process. Usage of this execution semantic is rather restricted and is preferable to use it for testing purposes or for fixed services bindings only.

In Figure 3 the definition of the system is given. The behaviour of the Discovery component is specified in more detail in Figure 4.

First, as depicted in Figure 4, a list of *list of SWS* is created by combining internally known Semantic Web Services with the ones from external repositories in a case the service requestor wants to check them too. From this list, one SWS is picked at the time and an attempt is made at matching. If necessary, *Data Mediation* is requested (arc *need DM*) when ontology of Goal and SWS being matched differ. This Data Mediation may succeed, after which the matching can continue. The Data Mediation may also fail, after which a new SWS is needed (the SWS being checked cannot be mediated, and is useless for the given goal).

The *Discovery* process continues (with or without *Data Mediation*) until a list of Semantic Web Services is completed. The list of Semantic Web Services is completed when the predefined number of them is collected or if there are no more SWS on the *list of SWS*. The list of found SWS is returned as a result of the *Discovery* component. The list is empty if no Semantic Web Services were found. This means that all available SWS have been tried for matching, but none of them succeeded.

After *Discovery* the *Selection* component selects the Semantic Web Service out of the list returned by *Discovery* component that fit best the user's preferences. Finally, the *Invoker* component sends a message (invokes) the selected Semantic Web Service.

The process of Discovery depicted on Figure 4 includes *Matching*. The matching is (from the viewpoint of the WSMX system) a nondeterministic choice, either a matching is found, or an error occurs or a Data Mediation is needed; this choice is made not by the WSMX system, but by the *matching* process that infers on user's Goal and semantic description of a Web Service. The same pattern is repeated for modelling the other components, all of whose outcomes are

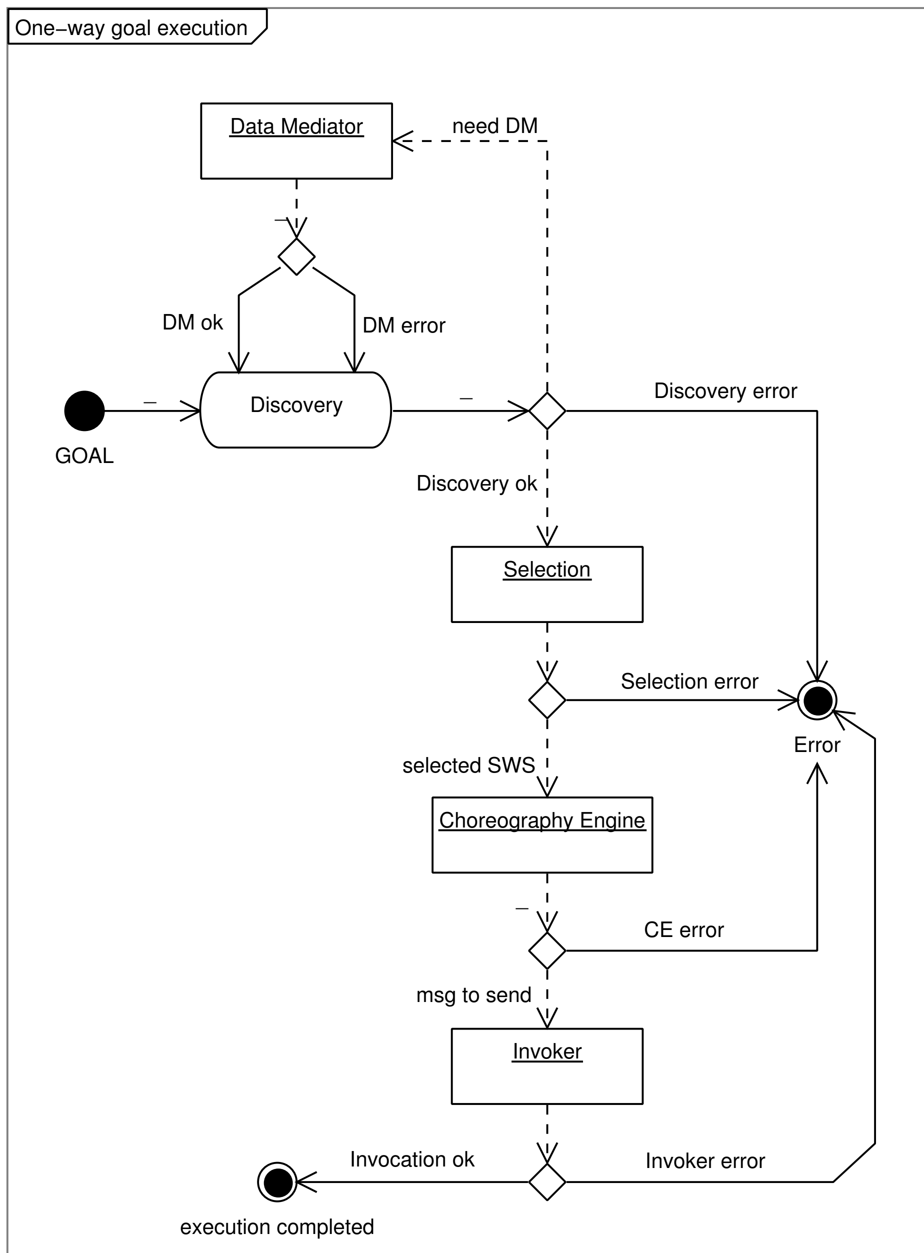

Figure 3: Overview of one-way goal execution

nondeterministic exclusive ORs (from the viewpoint of WSMX). Both the *Data Mediation* component and the *Selection* component can either fail or succeed, from the viewpoint of WSMX.

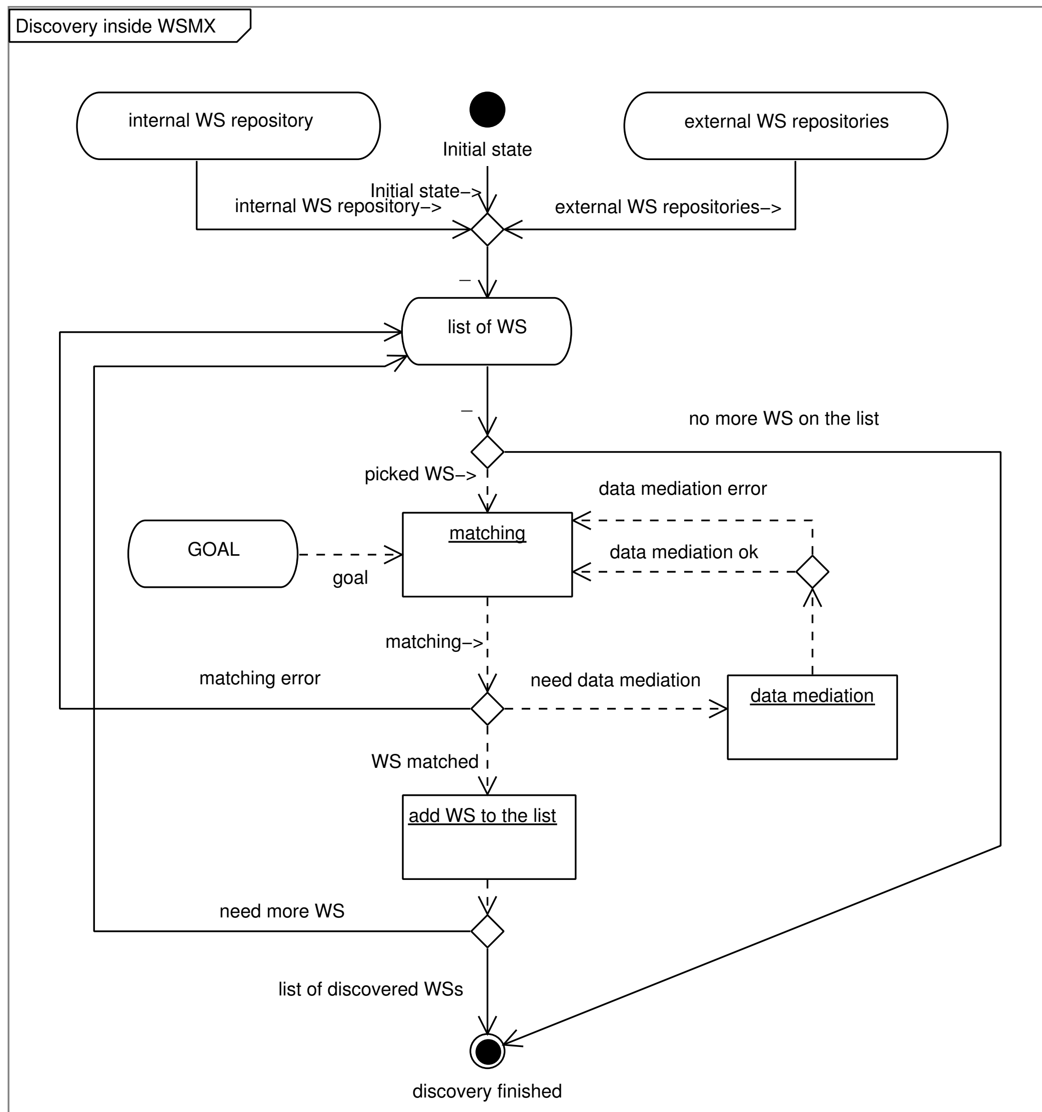

Figure 4: Discovery inside WSMX

The *Invoker* component is responsible for sending outgoing messages. Seamless approach to sending messages is provided. It is assumed that either service requestor or Semantic Web Service found by WSMX expose their interfaces as a choreography. If service requestor does not expose SWS interface it has to use Adapter as an intermediary layer located outside of WSMX. Adapter enables communication between WSMX and non-WSML service requestor. Communication with both of them is carried out in asynchronous manner, i.e. Invoker component does not block in awaiting for response. Context is included in outgoing message header, and preserved in recipient's response sent to Receiver component, thus response can be explicitly associated with context of conversation. If any of interacting parties communicates synchronously, the Adapter is utilized as a intermediary layer that carries out any blocking operations.

3.2 List of Semantic Web Services fulfilling given Goal

The following entry-point initiates the creation of the list.
getWebServices (WSMLDocument): Context

A list of Semantic Web Services is created by the *Discovery* component for a given Goal and Instance of Ontology and Preferences. Additionally in Preferences the requestor can specify the number of Semantic Web Services to be returned. Only *Discovery* and *Data Mediation* components carry out their tasks. Each Semantic Web Service on the *SWS list* is already mediated to the requestor Ontology if necessary and has its choreography [4]. Since the returned list of Semantic Web Services is specified in the same Ontology as the requestor's one, the requestor can understand them and make a choice which one should be executed. Exchange message patterns are specificated by choreography, thus both parties expose their expectation with regard to the exchanged messages and their order.

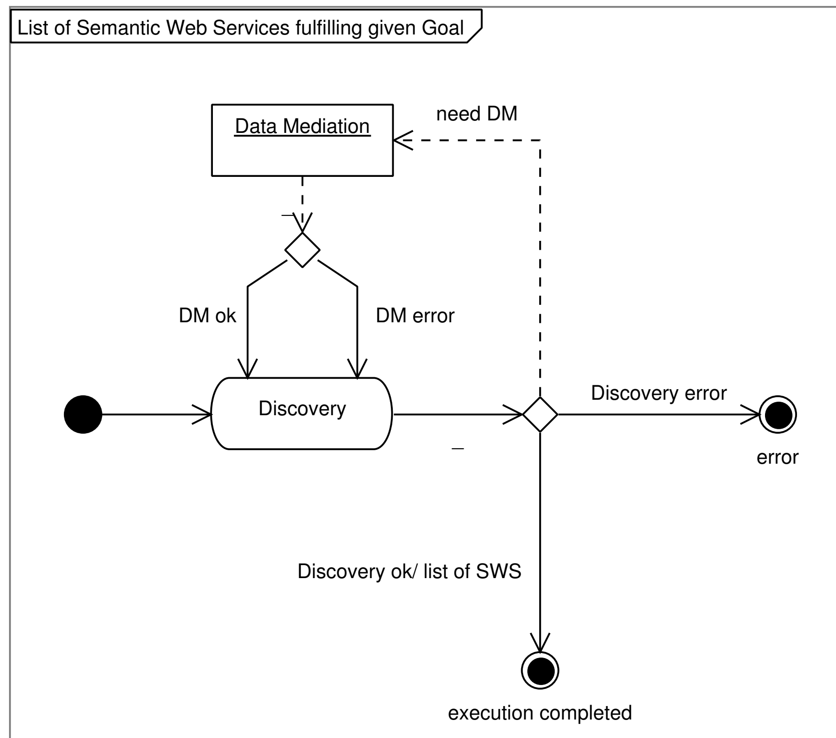

Figure 5: Overview of list of Semantic Web Services fulfilling given Goal

This behaviour is quite relevant when a decision about which Semantic Web Service to execute has to be made outside WSMX system. The decision could be taken manually or by some Semantic Web Service evaluation program.

The process of generating a list of Semantic Web Services that meet a given Goal is depicted in Figure 5. The *Discovery* component runs in loop until the desired number of Semantic Web Services is collected or until there are no more of them to discover. Finally, list of discovered Semantic Web Services is sent to the service requestor.

3.3 Semantic Web Service execution with choreography

The following entry-point initiates this execution semantic:
invokeWebService(WSMLDocument, Context):Context

Once the service requestor knows which Semantic Web Service he wants to use, back-and-forth conversation has to be carried out with the WSMX system to provide all the necessary data to make the execution of this Semantic Web Service feasible. This execution semantic involves Process Mediation component [3] that mitigates differences in the choreographies of the interacting parties. By giving fragments of Ontology Instances (e.g. business documents such as Catalogue Items or Purchase Orders in a given ontology) it provides all the data required by the Semantic Web Service.

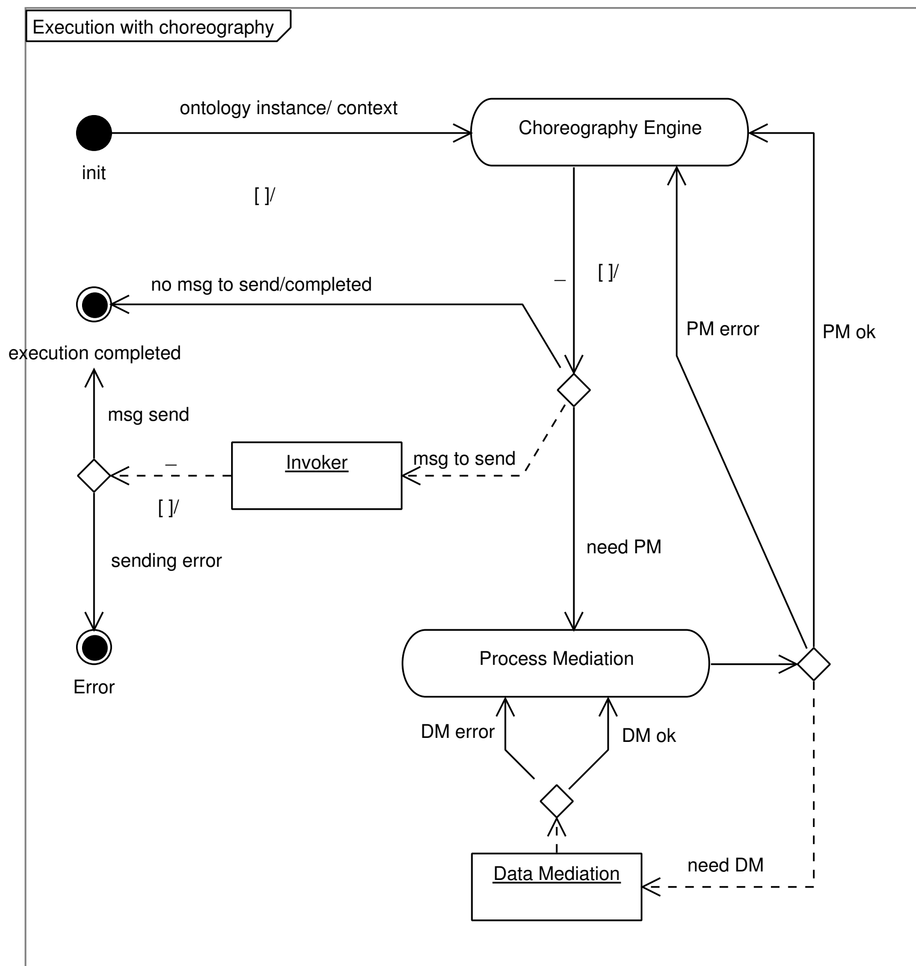

Figure 6: Overview of Web service execution with choreography

As presented on Figure 6 conversation between the given parties takes place according to service requestor's and SWS's choreography. Context previously obtained from *Register communication with WSMX* execution semantic is sent along with WSMLDocument as the parameters of this execution semantics and have to be preserved by both interacting parties. This approach facilitates keeping track of ongoing conversation status and refers to additional data required

to make communication between the two parties feasible (e.g. data related to Process Mediation like previously received messages or status of internally initialized choreographies).

Next, Process Mediation is invoked. Process Mediation mitigates a differences between choreographies of interacting parties (in this case requestor's and service's). For instance, it can change sequence of messages, generate some dummy messages, preserve some messages to send them later on or drop some of received messages. In general, its goal is to enable communication between parties despite of their different expectations of communication patterns (i.e. choreographies). In each invocation of Process Mediation, a Context is used to determine currently processed steps and to update internally initialized choreographies of the interacting parties. Process Mediation can also require some additional data preserved in the Context, such as previously received messages.

Process Mediation as an input requires instances of choreographies of interacting parties and Ontology Instance. If necessary the Data Mediation component is requested when differences between used ontologies occur to mediate Ontology Instance into the target ontology (i.e. message recipients ontology). The ready message and recipient address is forwarded to the Invoker component. Next, a message is sent asynchronously. Response message is received by the Receiver component and then it is again passed on to the Process Mediation.

It should be stressed that the approach presented in this execution semantic allows seamless communication with the outside world. One type of Invoker and Receiver component can be exploited for communication with both interacting parties. From the point of view of these components, there is no differentiation in interacting with requestor or Semantic Web service.

3.4 Register communication with WSMX

The following entry-point initiates this execution semantic:
invokeWebService(WSMLDocument):Context

This entry-point in invoked in order to register the conversation between a given Semantic Web Service and service requestor. A unique Context identifier is assigned and their choreographies are internally initialized by WSMX regardless of their own choreographies. Since they belong to the same conversation context and can be treated as a pair. This Context allows one to keep track of ongoing conversation between registered parties.

4 Dynamic execution semantics

Since WSMX consists of loosely-coupled components, one could easily imagine other execution semantics that will appear in the future. Thanks to its architecture WSMX can be easily enhanced with new components. Components can be dynamically plugged in to or plugged out of the system. New versions of components can replace outdated ones in this manner. Components can be deployed on remote machines and still be able to subscribe to WSMX events through Tuple Space and to process them. This gives the designer a flexible way to create new execution semantics.

The new executions semantic specifications need not be restricted to one language. They even need not be based on one paradigm (in this deliverable we consider only colour Petri net based representation). In order to create inte-

roperable WSMX systems, WSMX should take advantage of efforts like M3PE or WFMC with its XPDL initiative. These efforts are concerned with 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.

5 Glossary

Semantic Web Services (SWS) - semantically enriched Web Services. Their description are rooted in ontologies and uses logic expressions. It allows to perform their dynamic discovery by employing logical reasoning about their capabilities.

Execution Semantic - formal, unambiguous representation of the modelled system. Sound, mathematical foundation has to be provided for the given methodology to make it unambiguous.

Unified Modelling Language (UML) - is a graphical notations for software modelling. It compromises of a set of diagrams for system design capturing two major aspects, namely static and dynamic system properties. UML has been originally created for modelling Object-Oriented Programming software, however it is also applied in other areas like for instance Business Process Modelling.

Activity Diagram - one of the dynamic UML diagrams. It specifies a control and data flow between the entities providing language constructs that enable to model elaborate cases like parallel execution of entities or flow synchronization.

Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, Ontoweb, SEKT, and SWWS; by Science Foundation Ireland under the DERI-Lion project; and by the Austrian government under the CoOperate program.

The editors would like to thank to all the members of the WSMO, WSML, and WSMX working groups for their advice and input into this document.

References

[1] G. Alonso, D. Agrawal, A. El Abbadi, C. Mohan, R. Gnthr, and M. Kamath. Exotica/FMQM: A Persistent Message-Based Architecture. In *In the proceedings of IFIP Working Conference on Info Sys for Decentralized Organizations*, Trondheim, August 1995.

[2] Ch. Bussler, E. Kilgarriff, and R. Krummenacher others. WSMX Triple-Space Computing. WSMX Working Draft v01, 2005.

[3] E. Cimpian and A. Mocan. Process Mediation in WSMX. WSMX Working Draft v01, 2005.

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

[5] J. de Bruijn, H. Lausen, et al. D16.1v0.2 The Web Service Modeling Language WSML. WSML Final Draft, March 2005.

[6] Edsgar W. Dijkstra. Notes on structured programming. *Structured Programming*, 1972.

[7] H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling.* PhD thesis, University of Twente, Twente, Netherlands, 2002.

[8] R. Eshuis and R. Wieringa. Comparing Petri Nets and Activity Diagram Variants for Workflow Modelling- A Quest for Reactive Petri Nets. In H. Ehrig, W. Reisig, and G. Rozenberg, editors, *Petri Net Technologies for Communication Based Systems*, Lecture Notes in Computer Science. Springer-Verlag, Berlin, 2002.

[9] D. Harel and B. Rumpe. Meaningful Modeling: Whats the Semantics of "Semantics"?

[10] D. Roman, H. Lausen, and U. Keller. Web Service Modeling Ontology (WSMO). WSMO Working Draft v1.1, 2005.

[11] M. Zaremba, M. Moran, and T. Haselwanter. WSMX Architecture. WSMO Working Draft v02, 2005.