



D14v0.1. Choreography in WSMO

DERI Working Draft 20 June 2004

This version:

<http://www.wsmo.org/2004/d14/v0.1/20040620/>

Latest version:

<http://www.wsmo.org/2004/d14/v0.1/>

Previous version:

<http://www.wsmo.org/2004/d14/v0.1/20040527/>

Editors:

Dumitru Roman
Laurentiu Vasiliu
Michael Stollberg
Christoph Bussler

This document is also available in non-normative [PDF](#) version.

Copyright © 2004 [DERI](#)®, All Rights Reserved. [DERI](#) liability, trademark, document use, and software licensing rules apply.

Table of contents

- [1. Introduction](#)
 - [2. Conceptual Model](#)
 - [3. Conclusions and further work](#)
 - [References](#)
 - [Acknowledgement](#)
-

1. Introduction

Choreography in WSMO is part of the interface description of a WSMO service. It defines how the service communicates in the environment it is deployed (i.e. other WSMO service) in order to consume its functionality. It is also the choreography where the service specifies how it makes use of existing technologies (e.g. web services, space-based computing, etc).

The aim of this document is to identify the concepts that are needed to model choreographies in WSMO and to propose a way for representing WSMO

choreographies in a formal way. A formal representation of choreographies would help to match/adapt choreographies of WSMO services in order to be able to converse, in case they defined their choreographies a-priori, or automatic generation of a choreography in case the user (human or not) of a WSMO service has no a-priori definition of a choreography and need to generate one in order to be able to converse with the WSMO service. This document makes no assumptions on the existing technologies (e.g web services, space-based computing, etc) that choreographies should be grounded. The aim here is to provide a conceptual model that can then be easily grounded to different existing technologies.

The document is structured as follows: Section 2 presents what a choreography should consists of (i.e. defines the conceptual model for choreographies) and some initial ideas on how to formally represent them, and Section 3 concludes the document and presents further intentions.

2. Conceptual Model

Before describing what needs to be model when defining a choreography, we present where a choreography is needed when defining a WSMO service.

First of all a choreography is needed for describing how to communicate with the WSMO service in order to consume its functionality (i.e. how the capability of the service is achieved). This is depicted in the left side of the WSMO Service 1 in Figure 1.

However, as shown in Figure 1, choreographies (or better say choreography-interfaces) may also appear when defining the orchestration of the service. In order to achieve its capability, WSMO Service 1 might use WSMO Service 2 and WSMO Service 3 (the order in which they are used is defined in the orchestration of the service and is not important here). WSMO Service 1 needs to describe in a choreography the way it behaves when interacting with WSMO Service 2. If WSMO Service 2 is known at design time, then a choreography, consisting of WSMO Service 2's complementary behavior, is associated to its corresponding proxy (P1 in the figure). If WSMO Service 2 is not known at design time, then a goal is associated to its corresponding proxy (P1) and the choreography will be, ideally, generated at runtime when the service that may solve the goal is known. Note that in this case WSMO Service 1 acts as an invoker. There might also be the case that WSMO Service 1 acts as an invokee in the orchestration when interacting with WSMO Service 3; case in which a capability and a behavior is associated to the proxy (P2 in the figure) corresponding to WSMO Service 3. Note that this time it is not the choreograph associated to the proxy P2 that needs to be generated or adapted at the run time, but the choreography of WSMO Service 3.

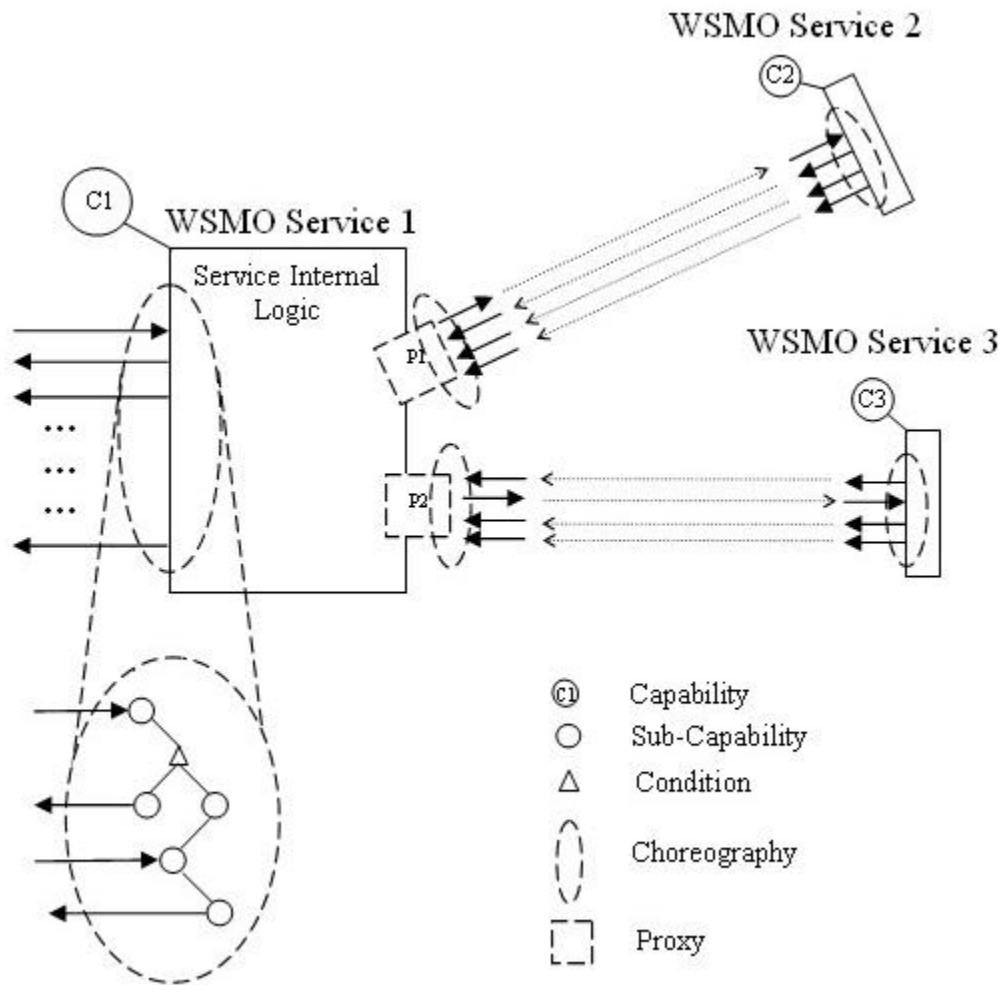


Figure 1. Choreography descriptions needed when defining a WSMO service.

Choreographies (or better say choreography interfaces) thus are needed not only when a user interacts with a service in order to consume its functionality, but also when the service tries to achieve its functionality when making use of other WSMO Services.

In order to achieve the functionality of the service, the service's choreography uses a Message Exchange Pattern (MEP). A message exchange pattern can be:

- Simple MEP - specifies a communicative act that either send or receive messages.
- Complex MEP - specifies a sequence of conditional communicative acts that either send or receive messages and non-specified actions.

A communicative act is defined as follows:

Listing 1. Communicative act definition

```
CommunicativeActDefinition[
type => type
sender => sender
receiver =>> receiver
content => content
]
```

Type

Represents the type of the communicative act.

Sender

Represents the identity of the sender (i.e. WSMO service) of the message.

Receiver

Represents the identity of the receiver (i.e. WSMO service) of the message.

Content

Represents to content of the message that is either sent or received.

The types of the communicative acts that a `simple MEP` may specify are (although the set of the communicative acts types we consider here can be extensible, it will ensure the usability of the MEP by a variety of services):

- `Request` - The sender requests the receiver to perform some action.
- `Inform` - The sender informs the receiver that a given fact is true (e.g. an action has been successfully performed).
- `Failure` - The sender informs the receiver that an action was attempted but the attempt failed.

`Complex MEP` specify a sequence of, possibly conditional, communicative acts that either send or receive messages (i.e. a sequence of, possibly conditional, simple MEPs) and non-specified actions. Thus, the `complex MEP` is where the overall capability of the service is decomposed in `sub-capabilities` (i.e. simple MEPs and non-specified actions).

The `state` of a complex MEP is given by the values of the inputs/outputs of the sub-capabilities.

`State transitions` are given by rules, having the form: **if *Cond* then *Updates***. *Conds* an arbitrary condition and *Updates* consists of sub-capabilities' inputs/outputs updates, which are executed simultaneously.

3. Conclusions and further work

This document presented a preliminary draft related to choreography in WSMO. Given the fact that the way the choreography modeled in WSMO is similar to Abstract State Machine [Gurevich, 1995] model, we intend to investigate how Abstract State Machine Language [AsmL] can be used for modelling WSMO choreography.

References

[AsmL] Abstract State Machine Language. Available at <http://research.microsoft.com/fse/asml/>

[Gurevich, 1995] Yuri Gurevich: *Evolving Algebras 1993: Lipari Guide*", Specification and Validation Methods, ed. E. Börger, Oxford University Press, 1995, 9--36.

Acknowledgement

The work is funded by the European Commission under the projects DIP, Knowledge Web, SEKT, SWWS, Esperanto, and h-TechSight; by Science

[Foundation Ireland](#) under the [DERI-Lion](#) project; and by the Vienna city government under the [CoOperate](#) program.

The editors would like to thank to all the [members of the WSMO working group](#) for their advice and input into this document.

[1] <http://www.engr.uconn.edu/%7Eibrahim/publications/acl.htm>

[2] <http://www.fipa.org/specs/fipa00037/SC00037J.html>

[webmaster](#)