



D14v0.1. Choreography in WSMO

WSMO Working Draft 14 October 2004

This version:

<http://www.wsmo.org/2004/d14/v0.1/20041014/>

Latest version:

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

Previous version:

<http://www.wsmo.org/2004/d14/v0.1/20041004/>

Editor:

Dumitru Roman

Co-Authors:

Uwe Keller

Emilia Campian

Michael Stollberg

Dieter Fensel

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. States](#)
 - [3. Guarded transitions](#)
 - [4. Choreography description example](#)
 - [5. Conclusions and further work](#)
 - [References](#)
 - [Acknowledgement](#)
-

1. Introduction

Choreography in WSMO is part of a service interface description; it describes the behaviour of the service. The aim of this document is to provide a core conceptual model for describing the different types of choreographies in WSMO. The state-based mechanism for describing WSMO choreographies is based on the Abstract State Machines [\[Gurevich, 1995\]](#) methodology. The reason for choosing the ASMs as a basis for WSMO choreography is that ASMs provide a high flexibility

in modelling systems, being at the same time scientifically well founded. For a detailed explanation on ASMs we refer the reader to [\[Börger, 1998\]](#).

Taking the ASMs methodology as starting point, a WSMO choreography is defined as follows:

Listing 1. WSMO choreography definition

```
Class wsmoChoreography
  hasStates type state
  hasGuardedTransitions type guardedTransition
```

States

A full instantaneous description of the behaviour of the service in its communication.

Guarded transitions

Transition rules that transform the states.

The rest of the document is organized as follows: [Section 2](#) of the document describes what is the signature of a state, [Section 3](#) describes the guarded transitions, and [Section 4](#) presents an example of how a choreography is modeled and [Section 5](#) presents the conclusions and further directions.

2. States

A `state` is a specification in WSMO Ontology. The signature of the state is given by the elements of the WSMO Ontology:

- Non functional properties
- Imported Ontologies
- Used mediators
- Concepts
- Relations
- Functions
- Axioms

Besides these declarations, a signature of a state may consists also of an infinite number of WSMO identifiers.

Among the function declarations, additionally there are two function names of special importance and have a special meaning:

- `in` - a boolean, unary function. This function is an external function, meaning that its value is set by the environment and not by the service. Its argument consists of an *instance* of a concept or relation.
- `out` - a boolean, unary function. Its argument may consists of an *instance* of a concept or relation.

3. Guarded transitions

`Guarded Transitions` are used to express changes of states by means of rules, expressible in the following form:

if *Cond* then *Updates*.

Cond is an arbitrary axiom without free variables, formulated in the given signature of the state using the logical language for defining formal statements defined in WSMO.

The *Updates* consist of arbitrary WSMO Ontology instance (see [Section 3.7 of WSMO 1.0](#)) statements.

Besides this **if then** rule, we allow also rules of the form:

- **choose *x* with *Cond* do *R*,**

meaning to execute rule *R* with an arbitrary *x* among those satisfying the condition *Cond*.

- **forall *x* with *Cond* do *R*,**

meaning to execute simultaneously rule *R* for each *x* satisfying the condition *Cond*.

We freely use as abbreviations combinations of **where**, **let**, **if then else**, **case** and similar standard notations which are easily reducible to the above basic definitions.

4. Choreography description example

A service sells train tickets for a route. It receives a request for a route; based on some internal decision [1], in this case explicitly specified (if there is a route), the response of the service can either be a complete route or an error message, containing the reason for the error. If the user receives the complete route he/she is allowed to send a request for a ticket. After the user's request for the ticket, the service requests the user's credit card. Based again on some internal decision, this time not specified, the service can either send a confirmation number for the ticket, or an error message.

Listing 2 below presents the signature of the state for this example.

Listing 2. The state signature of the example

```

//The signature of the state

//concepts
concept location subConceptOf xsd:string

concept route
  source ofType location
  destination ofType location

concept completeRoute subConceptOf route
  intermediaryLocation ofType location

concept ticket subConceptOf xsd:string

concept creditCard
  name ofType xsd:string
  no ofType xsd:integer

concept confirmation
  completeRoute ofType completeRoute
  no ofType xsd:integer

concept failureReason subConceptOf xsd:string

//relations
relation routeExists
  source ofType location
  destination ofType location

relation routeFailure
  source ofType location
  destination ofType location
  reason ofType xsd:string

//functions
function getCompleteRoute
  source ofType location
  destination ofType location
  range ofType completeRoute

function getRouteFailureReason
  source ofType location
  destination ofType location
  range ofType failureReason

function getConfirmationNo
  completeRoute ofType completeRoute
  range ofType xsd:integer

function getInternalDecisionReason
  range ofType failureReason

function lastRoute

function internalDecision

```

Listing 3 below presents the guarded transitions for the example.

Listing 4. The guarded transitions for the example.

```

//Transitions rules

//If the service receives a route request, and if the route exists, then a complete route
//is sent to the user and the complete route is stored as it is needed in subsequent steps
//during the communication; if the service is not able to find a route, then the failure reason
//is sent to the user.
forall X with (in(X) and (X memberOf route))
  if routeExists(X. source, X.destination) then
    choose Y with (getCompleteRoute(X.source, X.destination) = Y
      out(Y) := true
      lastRoute := Y
    else
      choose Y with ( Y memberOf routeFailure
        source hasValue X.source
        destination hasValue X.destination
        reason hasValue getRouteFailureReason(X.source, X.destination))
      out(Y) := true

//If the service receives a ticket request from the user, the service requests the user's
//credit card.
forall X with (in(X) and X memberOf ticket))
  choose Y with (Y memberOf creditCard)
    out(Y) := true

//If the service receives the credit card, then based on some internal decision which is not
//shown to the user, it can either send a confirmation number for the ticket of the complete
//route it found at the beginning, or sends the failure reason for not being able to provide
//the user with a confirmation no for the ticket.
forall X with (in(X) and (X memberOf creditCard))
  if internalDecision then
    choose Y with (Y memberOf confirmation
      completeRoute hasValue lastRoute
      no hasValue getConfirmationNo(completeRoute))
    out(Y) := true
  else
    choose Y with (getInternalDecisionReason = Y)
    out(Y) := true

```

5. Conclusions and further work

This document presented a core conceptual model for modeling WSMO Choreographies based on the ASMs methodology. Future versions of this document will give a precise translation of the states and the guarded transitions to ASMs in order to benefit from the using of ASMs interpreters, thus having an environment for running choreographies.

References

[Börger, 1998] Egon Börger: "High Level System Design and Analysis Using Abstract State Machines", Proceedings of the International Workshop on Current Trends in Applied Formal Method: Applied Formal Methods, p.1-43, October 07-09, 1998

[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](#), and [Esperanto](#); 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] The internal decision of the service can be explicitly specified by the service in the form of a condition. This situation could help the user of the service in the sense that he/she can assume the validity of the condition, based on the definition of the condition in some ontology that the service uses.



[webmaster](#)

\$Date: 2004/10/15 15:19:31 \$
