WSMX Process Mediation

Emilia Cimpian, Mick Kerrigan

DERI, National University of Ireland, Galway, Ireland
emilia.cimpian@deri.org, mick.kerrigan@deri.org

1 Introduction

The amount of resources currently available on-line provides the requestor of a service with practically an unlimited number of options, an unlimited number of services which are able to satisfy its request. Unfortunately this number is drastically reduced by the heterogeneity of the resources, since a client can not benefit from invoking a service with whom it does not know how to communicate.

This inability may have two causes: data heterogeneity or public process heterogeneity. The data heterogeneity problem, and how it can be overcame has been the focus of intense research activity during the last years. Many technologies have been developed to tackle this problem, unfortunately all of them require human user inputs at one stage or another in the mediation process.

The second problem, which still remains unsolved, is refereing to the public process heterogeneity, which express the behavior of a certain participant in a conversation during the interaction with the other partner. This paper proposes a solution for coping with this problem, illustrating the functionality of a Process Mediator (PM) developed as part of the Web Service Execution Environment (WSMX) [1] and its mediation algorithm.

2 WSMX Process Representation

A common classification of processes is public vs private processes; the private processes are carried out internally by an organization, and usually are not visible to any other entity, while the public processes define the behaviour of the organization in collaboration with other entities. As a consequence, WSMX is concerned with the representation of the public processes, and how two business partners can communicate based strictly on their public processes.

WSMX process representation is similar with the one used in WSMO choreography [3] definition. This dependency is a straight forward one considering that WSMX public processes refer to the communication pattern of an entity, while WSMO choreography describes the behavior of a service from a user point of view (that is, how the user should interact with the Web Service for consuming its functionality). In terms of WSMX, every choreography represents a public process, which means that WSMO choreography can be seen as a subclass of WSMX process.

Based on the Abstract State Machine [2] methodology, a WSMX process consists of `states` and `guarded transitions`, this being the same representation as for WSMO Choreography [3]. A state is a WSMO ontology, and each of its concepts have an attribute `mode`, which shows if the environment can receive or send (i. e., create or modify) instances of this concept: only receive, only send, both, or no access at all. The guarded transitions are used for expressing the changes of states, which occur when a set of conditions are fulfilled. For a formal description of the states and guarded transitions we refer the reader to [3].

3 WSMX Process Mediation

WSMX process mediation is concerned with determining how two public processes can be combined in order to provide a certain functionality. In other words, how two business partners can communicate, considering their public processes.

The Process Mediator is invoked when a message sent by one of the participants in a conversation can not be received as it is by the targeted partner; the Process Mediator's role is to determine, based on the incoming message and on previously stored information regarding the conversation, if it can generate a message, for either one of the two partners, that can trigger an action (not necessary to change a state, but to eliminate one condition for changing a state).

3.1 Process Mediator Functionalities

WSMX Process Mediator aims to provide the following functionalities:

Stop an unexpected message: if one of the partners sends a message that the other one does not want to receive, the mediator should just retain and store it. This message can be send later, if needed.

Inverse the order of messages: in case one of the partners sends the messages in a different order than the other partner expects them. The messages that are not yet expected are stored and sent when needed.

Split a message: if one of the partners sends in a single message multiple information that the other one expects to receive in different messages.

Combine messages: in case one of the partners expects a single message, containing information sent by the other one in multiple messages.

Send dummy acknowledgement: in case one of the partners expects an acknowledgement for a certain message, and the other partner does not intend to send it, even if it receives the message.

3.2 Process Mediator Algorithm

For providing the described functionalities, the PM performs the following steps:

1. Loads copies of the participants choreographies (choregraphies' instances); All the computations will be performed on these copies.
2. Adds the instances contained in the message to the corresponding choreography instance (the sender's choreography instance).

3. Mediates the incoming instances in terms of the targeted partner ontology, and checks if the targeted partner is expecting them, at any phase of the communication process, which is done by checking the value of the `mode` attribute. The instances that are expected are stored in an internal repository.
4. For all the instances from the repository, the PM checks if they are expected at the current phase, which is done by evaluating the transition rules. The evaluation of a rule returns the first condition that can not be fulfilled, that is, the next expected instance for that rule. The possibility that various instances from this repository can be combined in order to obtain a single instance, expected by the targeted business partner is also considered.
5. Each time the PM determines that one instance is expected, it sends it, deletes it it from the repository, updates the targeted partner choreography instance, and restarts the evaluation process (step 4). This recurrence stops when no instance from the repository is expected.
   When a transition rule can be executed, it is marked as such and not reevaluated at further iterations. The PM only checks if a transition rule can be executed, and does not execute it, since it can not update the two choreography instances without receiving inputs from one of the partners.
6. For each instance forwarded to the targeted partner, the PM checks if the sender expects an acknowledgement. If it does, but the targeted partner does not intend to send it, the PM generates a dummy acknowledgement and sends it. Simultaneously, it updates the sender's choreography instance.
7. The PM checks all the sender's rules and mark the ones that can be executed.
8. The PM checks the requestor's rule, to see if all of them are marked; when all are marked, the communication is over.

4 Conclusions

In this document we proposed an approach and a mechanism for coping with the process heterogeneity problem. The processes we are addressing in this paper are the public processes of business entities (services or a requestors of services), which express their public processes as WSMO choreographies. The proposed approach is based on the semantic description of the Web Services and of their behavior, as well as on how a requestor that wants to interact with a Web Service expresses the expected behaviour of the Web Service.

References

1. E. Cimpian, M. Moran, E. Oren, T. Vitvar, and M. Zaremba. Overview and Scope of WSMX. Technical report, WSMX Working Draft, `http://www.wsmo.org/TR/d13/d13.0/v0.2/`, February 2005.
2. Y. Gurevich. Evolving algebras 1993: Lipari Guide. In Egon Börger, editor, *Specification and Validation Methods*, pages 9–37. Oxford University Press, 1994.
3. D. Roman, J. Scicluna, C. Feier, M. Stollberg, and D. Fensel. Ontology-based Choreography and Orchestration of WSMO Services. Technical report, WSMO Working Draft, `http://www.wsmo.org/TR/d14/v0.1/`, March 2005.