Maestro for Let's Dance: An Environment for Modeling Service Interactions

Gero Decker[1], Margarit Kirov[1], Johannes Maria Zaha[2], Marlon Dumas[2]

[1] SAP Research Centre, Brisbane, Australia
`(g.decker,margarit.kirov)@sap.com`
[2] Queensland University of Technology, Brisbane, Australia
`(j.zaha,m.dumas)@qut.edu.au`

Abstract. In emerging web service development approaches, the description of interactions both from a global and from a local perspective plays an increasingly important role. In earlier work we presented a visual language (namely Let's Dance) for modeling service interactions at different levels of abstraction. In this paper we present a modeling tool for Let's Dance. The tool supports the static analysis of global models, the generation of local models from global ones, and the interactive simulation of both local and global models.

1 Introduction

As the first generation of web service technology based on XML, SOAP, and WSDL gains maturity, a second generation targeting collaborative business processes is gestating. Development methods associated to this second generation of web services generally rely on the explicit representation of service interaction behavior from two complementary perspectives: one where interactions are seen from the perspective of each participating service, and the other where they are seen from a global perspective. This leads to two types of models: In a *global model* (also called a *choreography*) interactions and their dependencies are captured from the viewpoint of an ideal observer who oversees all interactions between a set of services. Meanwhile, a *local model* focuses on the perspective of a given service, capturing only those interactions that directly involve it. Local models are suitable for implementing individual services while choreographies are instrumental during the early phases of analysis and design, when domain analysts need a global picture of the system.

In previous work [4] we have identified requirements for a service interaction modeling language and argued that existing languages, e.g. WS-CDL [2], fail to fulfill these requirements. Indeed, existing languages are targeted at application developers rather than at domain analysts who play a key role in the construction of these models. Accordingly, we have designed a language, namely Let's Dance, intended to support analysts in capturing both global and local service interaction models. This paper introduces a tool that enables analysts to capture and to analyze Let's Dance choreographies. After analysis, the local models for the parties involved in the choreography can be generated and the execution of both global and local models can be interactively simulated.

The paper is structured as follows. Section 2 gives an overview of the Let's Dance language. The architecture and the features of the tool are presented in Section 3. Section 4 discusses future extensions.

2 Let's Dance Choreographies

A choreography consists of a set of interrelated service interactions corresponding to message exchanges. At the lowest level of abstraction, an interaction is composed of a message sending action and a message receipt action (referred to as communication actions). Communication actions are represented by non-regular pentagons (symbol for send and for receive) that are juxtaposed to form a rectangle denoting an elementary interaction. As illustrated in Figure 1, a communication action is performed by an actor playing a role, specified at the top corner of a communication action. Roles are written in uppercase and the actor playing this role (the "actor reference") is written in lowercase between brackets. The name of the message type for the receive actions can be omitted (since the same type applies for both send and receive).

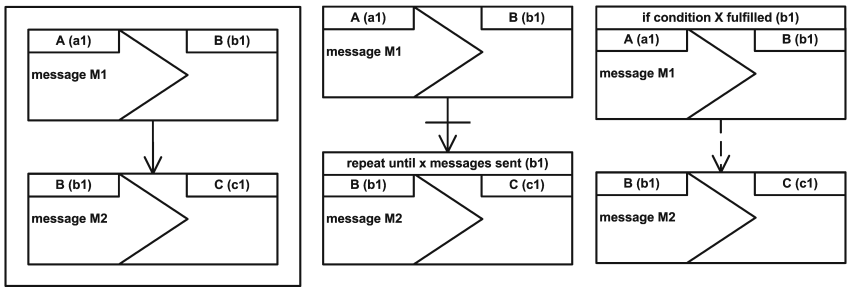
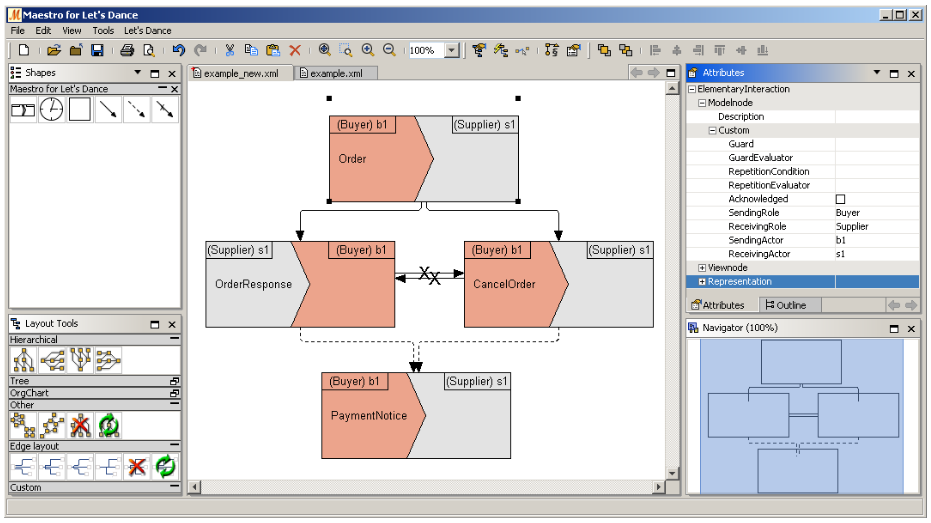


Fig. 1. Constructs of Let's Dance

Interactions can be inter-related using the constructs depicted in Figure 1. The relationship on the left-hand side is called "precedes" and is depicted by a directed edge: the source interaction can only occur after the target interaction has occurred. That is, after the receipt of a message "M1" by "B", "B" is able to send a message "M2" to "C". The rectangle surrounding these two interactions denotes a composite interaction, which can be related with other interactions with any type of relationship. The relationship at the center of the figure is called "inhibits", depicted by a crossed directed edge. It denotes that after the source interaction has occurred, the target interaction can no longer occur. That is, after "B" has received a message "M1" from "A", it may not send a message "M2" to "C". The latter interaction can be repeated until "x" messages have been sent, which is indicated by the header on top of the interaction. The actor executing the repetition instruction is noted in brackets. Finally, the relationship on the right-hand side of the figure, called "weak-precedes", denotes that "B" is not able to send a message "M2" until "A" has sent a message "M1" or until this interaction has been inhibited. That is, the target interaction can only occur after the source interaction has reached a final status, which may be "completed" or "skipped" (i.e. "inhibited"). In the example, the upper interaction has a guard assigned, which is denoted by the header on top of the interaction. This interaction is only executed if the guard evaluates to true. The actor who evaluates the guard is noted in brackets.

3 Tool Overview

Figure 2 shows a screen shot of Maestro for Let's Dance. The palette on the left-hand side contains the diagram elements. Below this palette, layout options are accessible. The main drawing area in the middle. On the right, there is a pane for editing the properties of the selected element as well as a navigator that provides an overview over the diagram. The analysis and simulation functionality can be accessed via the "Let's Dance" menu item in the menu bar at the top. Analysis results and simulation status are shown within the editing area.



ElementaryInteraction	
Modelnode	
Description	
Custom	
Guard	
GuardEvaluator	
RepetitionCondition	
RepetitionEvaluator	
Acknowledged	☐
SendingRole	Buyer
ReceivingRole	Supplier
SendingActor	b1
ReceivingActor	s1
Viewnode	
Representation	

Fig. 2. Screen shot of Maestro for Let's Dance

Maestro for Let's Dance is built on top of the Maestro visual language framework developed at SAP. It is the foundation for various modeling environments including: Maestro for BPMN, an editor for the Business Process Modeling Notation, Maestro for BPEL, a modeling environment for the Business Process Execution Language, and Maestro for SAM (Status-and-Action Management), a modeling and simulation environment for business object lifecycles.

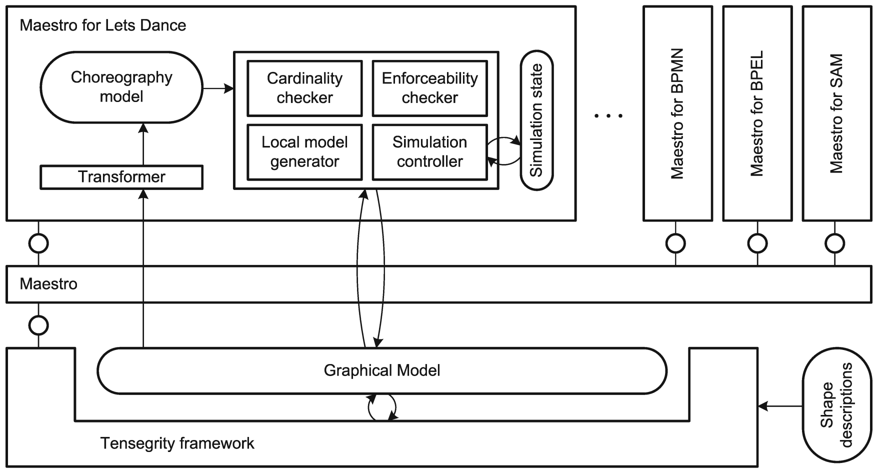


Fig. 3. Architecture overview

Figure 3 describes the main architecture of Maestro for Let's Dance using the Fundamental Modeling Concepts (FMC) block diagram notation [3]. A core component of the Let's Dance prototype is the Tensegrity framework[3]. It provides basic functionality that is needed for graphical editors such as rendering functionality, editing facilities (selection, creation, deletion, resizing, etc. of diagram elements), event propagation mechanisms, a command stack and a persistency service for diagrams. The Maestro framework extends Tensegrity and provides utility functionality for attribute management and refined application skeletons.

Tensegrity and Maestro can be compared to eclipse GMF[4] (Graphical Modeling Framework). Shape descriptions define what diagrams consist of and how they look like. Anchor points, resizing behavior, color schemes, line widths and line decorations can be defined for diagram elements. Furthermore, the palette is configured. A strong point about GMF is that it very clearly separates the data model from its graphical representation. However, GMF was not chosen because at the time of development releases were still in a pre-1.0 state and significant changes in the API from release to release caused major refactorings. It would have been an option to use eclipse GEF[5] (Graphical Editing Framework), the foundation of GMF, without using GMF itself. The fact that GEF does not include layouting functionality like it is the case for Tensegrity and the possibility of integrating the tool with Maestro for BPMN and Maestro for BPEL in the future were the final arguments for choosing Maestro.

Choreographies are described using two different data structures in the tool. The diagram data structure is optimized for the usage within the editor. E.g. entities in the data structure directly relate to edges and nodes in the diagram and layout information is attached to the entities. The choreography model data

[3] See http://www.tensegrity-software.com/
[4] See http://www.eclipse.org/gmf/
[5] See http://www.eclipse.org/gef/

structure does not contain any layout information and follows the Let's Dance meta-model introduced in [4]. The choreography model is then used as input for the analysis plugins and the simulation engine. A model transformation takes place every time a check is triggered or a simulation is started. A reverse mapping from entities in the choreography model to the entities in the diagram model is later on used for displaying output of the checkers or the simulation engine. Since EMF[6] (Eclipse Modeling Framework) includes functionality to produce XMI[7](XML Metadata Interchange)-compliant files and since there is a good integration of EMF with the UML modeling environment Rationale Rose[8], EMF was used for implementing the choreography data model.

A precondition for analyzing a choreography model is that it complies to the well-formedness criteria defined in [5]. Therefore, a well-formedness check precedes every analysis as well as the generation of local models. Examples for ill-formedness could be e.g. an unspecified actor for a send or receive action or a cycle of precedes and weak-precedes relationships.

Cardinality checker. Maestro for Let's Dance provides a *cardinality checker* that is able to identify how many times an interaction can occur, at least and at most, within one execution of a choreography. The cardinality checker takes as input a choreography and classifies its interactions into five groups: (0,0): interactions that will never be executed (i.e. unreachable); (0,1): interactions that may be executed once or skipped altogether (i.e. conditional); (1,1): interactions that will be executed exactly once; (0,n): interactions that may be executed any number of times; and (1,n): interactions that will be executed at least one.

The cardinality checker can help modelers to detect semantical errors, such as unreachable interactions or interactions that may be skipped against the modeler's intent. It can also be used to determine characteristics required from the communication channels. For example, if an interaction can be executed multiple times, the corresponding channel may be required to guarantee orderly delivery.

Cardinality analysis is largely based on reachability analysis for which we have presented two alternatives in previous work: In [1] the algorithm is based on π-calculus bi-simulation whereas in [5] an ad-hoc algorithm is introduced. It turned out that because of the computational complexity of bi-simulation analysis only very small choreographies could be processed in a reasonable time. The second algorithm has low polynomial complexity and scales up to real-world choreographies. Therefore, it was chosen for the tool.

Enforceability checker. It has been shown that there might be relations between interactions in a choreography model that cannot be enforced locally. I.e. It turns out that not all global models can be mapped into local ones in such a way that the resulting local models only contain interactions explicitly captured in the choreography model and they collectively enforce all the constraints expressed in

[6] See http://www.eclipse.org/emf/
[7] See http://www.omg.org/technology/documents/formal/xmi.htm
[8] See http://www.ibm.com/software/rational

the global model. A counter-example is given in Figure 4. In this choreography it is not possible to enforce the “precedes” constraint between the two interactions without introducing additional interactions. Indeed, how can actors ’c’ and/or ’d’ know that the interaction between ’a’ and ’b’ has taken place in the absence of any interaction between actors ’a’ and ’b’ on the one hand, and actors ’c and ’d’ on the other? Thus, either the model needs to be enhanced with an interaction between a/b and c/d, or the sequential execution constraint will not be enforced (i.e. the interactions can occur in the opposite order from the perspective of an ideal observer). Since domain analysts are supposed to sign off on a choreography model it is undesirable to automatically introduce implicit interactions to ensure the fulfilment of constraints. Instead, the modeler should be warned of such issues, so that (s)he can refine the model as needed.

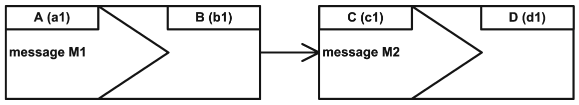


Fig. 4. Example of a non-locally enforceable choreography

Accordingly, Maestro for Let’s Dance incorporates an enforceability checker that identifies non-enforceable constraints in a choreography model and reports them to the modeler. The algorithm used for this purpose is described in [5].

Simulation In order to give choreography modelers a better idea of the semantics of their models, the tool offers the possibility to simulate choreography instances. The formalization of the execution semantics of Let’s Dance were presented in [1]. It was used as a blueprint for the implementation of the simulation engine.

An interaction can be in one of the four states *initialized* (visualized as yellow), *enabled* (green), *skipped* (red) and *completed* (gray). Each instance starts in the state *initialized*. The instance can now be skipped or it can be enabled. Therefore, the transitions to the states *skipped* and *enabled* are possible. If an instance is in state *skipped* nothing can happen to it any more. If an instance is in state *enabled* the interaction can execute. During the execution the instance can be skipped. Otherwise, it will eventually complete execution and move to state *completed*. In the case of guarded interactions the user can decide whether the interaction should be executed or if it should be skipped. In the case of repeated interactions the user can decide whether the interaction should be executed once more or if the repetition is terminated.

Local model generator. In [5] we have presented an algorithm for generating local models. This algorithm is implemented in Maestro for Let’s Dance. A new diagram containing all the interactions where a specific actor is involved is generated and the layout functionality is applied to it.

4 Outlook

The next step in the development of "Maestro for Let's Dance" is the refinement of the generation of local models. The generation of local models with implementation configurations will be an important step towards the generation of executable models, where BPEL will be the first target language. Other analysis functions will be added. Conformance checking between local models and choreography models will be one of the focus areas.

References

1. G. Decker, J. M. Zaha, M. Dumas: *Execution Semantics for Service Choreographies.* Preprint # 4329, Faculty of IT, Queensland University of Technology, May 2006. http://eprints.qut.edu.au/archive/00004329
2. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. *Web Services Choreography Description Language Version 1.0*, W3C Candidate Recommendation, November 2005. http://www.w3.org/TR/ws-cdl-10.
3. A. Knopfel, B. Grone, P. Tabeling: *Fundamental Modeling Concepts: Effective Communication of IT Systems.* Wiley, May 2006.
4. J. M. Zaha, A. Barros, M. Dumas, A. ter Hofstede: *Lets Dance: A Language for Service Behavior Modeling.* Preprint # 4468, Faculty of IT, Queensland University of Technology, February 2006. http://eprints.qut.edu.au/archive/00004468
5. J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, G. Decker: *Service Interaction Modeling: Bridging Global and Local Views.* In Proceedings of the 10th International EDOC Conference, Hong Kong, 2006.