Web Service Choreography Configurations for BPMN*

Kerstin Pfitzner[1], Gero Decker[2], Oliver Kopp[1], Frank Leymann[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany
{pfitzner,kopp,leymann}@iaas.uni-stuttgart.de
[2] Hasso-Plattner-Institute, University of Potsdam, Germany
gero.decker@hpi.uni-potsdam.de

Abstract. The Business Process Modeling Notation (BPMN) and the Business Process Execution Language (BPEL) are the de-facto standards for process modeling and implementation. While BPMN allows to define choreographies, i.e. the interaction behavior of interconnected services, BPEL only allows an endpoint-centric view on services. To achieve applicability of BPEL in the choreography space, we defined BPEL4Chor as choreography extensions for BPEL in previous work. This paper extends on this work and makes a contribution for the model-driven development of service-based systems in inter-organizational settings: It extends BPMN for enabling the generation of fully defined BPEL4Chor choreographies and presents how to carry out this transformation.

1 Introduction

The Service-oriented Architecture (SOA) is an architectural style for building software systems based on services. Services are loosely coupled components described in a uniform way that can be discovered and composed. Services can be orchestrated for implementing long-running business processes. In the web services platform architecture [6], web service orchestrations are described using the Business Process Execution Language (BPEL [2]), which are in turn exposed as web services. A typical visual notation used to describe these processes is the Business Process Modeling Notation (BPMN [1]).

BPMN is implementation-independent and especially allows the definition of choreographies by interconnecting different processes using message flows. Such choreographies are of high value especially in inter-organizational settings, where different business partners agree on their interaction behavior before they actually interconnect their information systems. Participant behavior descriptions are first used as views on the choreography from the perspective of an individual partner. Afterwards, they serve as specifications for the implementation of new services or the adaptation of existing services.

While BPMN choreographies are suitable for listing all interactions and defining the control flow between them, detailed technical configurations cannot

* Partially funded by the German Federal Ministry of Education and Research (project Tools4BPEL, project number 01ISE08)

be expressed in this language. In contrast to this, BPEL allows such configurations. However, it only considers individual processes which communicate with a black box environment. In other words, BPEL does not support the specification of choreographies. In order to overcome this limitation, we introduced a choreography extension for BPEL called BPEL4Chor [7]. In this paper, we tackle the integration of BPMN and BPEL4Chor. Existing work provides the basis for generating BPEL stubs out of BPMN, e.g. [14]. We extend this work in two dimensions: (i) We integrate the additions of BPEL4Chor to BPEL into the transformation. (ii) We target the generation of fully defined BPEL4Chor choreographies: Ideally no further refinements are necessary in the BPEL4Chor model. This calls for BPMN extensions focusing on technical configuration.

The remainder of this paper is structured as follows. Section 2 provides a choreography example and section 3 introduces BPEL4Chor. Section 4 identifies the missing concepts in BPMN for generating fully defined choreographies and section 5 gives an overview of the BPMN extensions realizing these concepts. Section 6 presents the mapping from extended BPMN to BPEL4Chor. Section 7 will report on related work. Section 8 concludes and points to future work.

2 BPMN Example

BPMN supports modeling of choreographies as collaboration diagrams. message flows are used. Pools model business roles or entities, while message flows represent the communication between them. An example for a collaboration diagram is presented in Fig. 1 which illustrates the following scenario in BPMN: A client requests the price of a book from a book shop. The book shop is a reseller and has business relationships with a set of suppliers. Hence, the book shop requests the price from the suppliers. This request causes each supplier to calculate the price for the book in question. If the book is available at the supplier, its price is sent back to the book shop. Otherwise, a fault message is sent. Upon receipt of a fault message, fault handling is triggered, where the supplier data is updated. Thus, the book shop can collect statistical data or use the data in an improved process, in which the book shop does not request the supplier for the price of this book again. After having received an answer from all suppliers, the book shop checks whether any supplier has responded a price. If a price has been received, the book shop determines the cheapest supplier and sends the respective price to the client. Otherwise the supplier sends an error message to the client. Meanwhile, the client waits for the response of the book shop. After the client has received a response, the collaboration ends.

3 BPEL4Chor Overview

BPEL4Chor is a language to describe choreographies using BPEL. The behavior of each participant is specified in a *participant behavior description* (PBD). The PBD itself is expressed in abstract BPEL, where internal details of the process are omitted. Thus, BPEL and BPEL4Chor contain the same activity types.

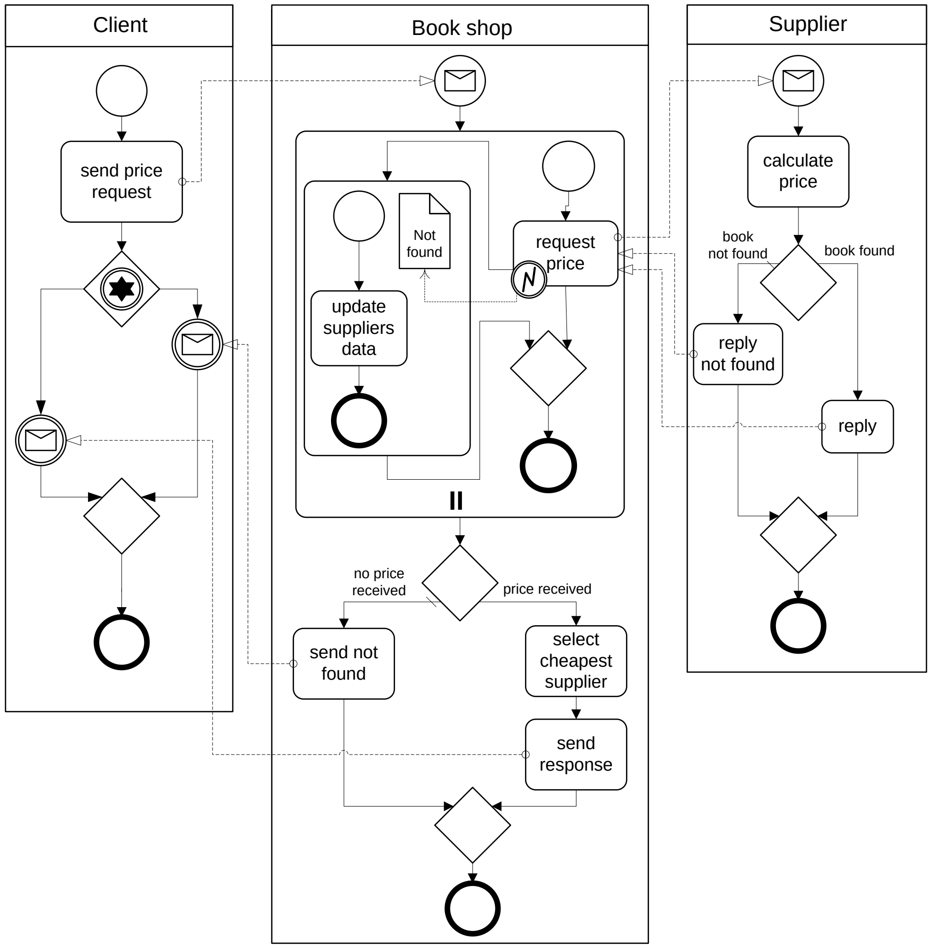

Fig. 1. Book shop choreography in BPMN

The communication activities of the PBDs are interconnected using message links. A message link determines that a message is sent from one activity to another one. Thus, message links replace port types and operations. Message links themselves are specified in a *participant topology*. The participant topology also specifies which participants take part in the choreography. Multiple instances of a participant type are specified by the notation of a participant set. Participant types form the connection between the participant behavior descriptions and the concrete participants. The technical configuration, i.e., the concrete port types and operations to be used at the execution phase of the choreography are given in the *participant grounding*. There, each message link is grounded to a WSDL operation. An abstract BPEL process containing partner links, port types, and operations can be generated for each PBD. These abstract BPEL processes can then be used as starting point for the generation of executable BPEL files, which can be deployed on a BPEL workflow engine.

4 Web Service Choreography Configurations Missing in BPMN

BPEL4Chor uses abstract processes, which are also the basis for BPMN collaboration processes. This section lists the missing elements for each aspect.

Basic activities. BPMN knows different task and event types. Following task and event types can be mapped to basic BPEL activities: a task of the type service to the invoke activity, a send task to the reply activity, a receive task or message event to the receive activity, an error event to the throw activity, a compensation event to the compensate activity and a timer event to the wait activity. The other BPEL basic activity types assign, validate, empty and opaque do not have an equivalent in BPMN.

Structured activities. Structured BPEL activities can be modeled using BPMN sub-processes, gateways and sequence flow. [15] shows how this works for the sequence, if, pick, flow, while and repeatUntil activity. A scope activity can be modeled using a sub-process not marked as looping activity. The forEach activity can be modeled as a sub-process or task marked as multiple instance loop. Note that while and repeatUntil activities can be modeled using a standard loop. BPEL scopes can be isolated, whereas BPMN is not aware of the notion of isolated sub-processes or tasks.

Event handlers. BPEL event handlers can be triggered by messages or alarms (timeouts) and run concurrently to the activities within a process or scope. BPMN attached intermediate events interrupt an activity. Thus, modeling of BPEL event handlers using BPMN attached intermediate events is not applicable. Furthermore, event handlers can be instantiated multiple times, while their parent process or scope is active. Such behavior cannot be modeled in BPMN.

Termination handlers. The BPMN specification does not clearly state what will happen during the interruption of an activity when an intermediate attached event is triggered. There is no event type in BPMN that triggers a termination handler.

Multiplicity of participants. It is not possible to indicate in the diagram that there are multiple participants of the same type involved in a choreography. For example, in the book shop choreography presented in Fig. 1 we cannot determine whether there are several suppliers involved or whether one supplier will receive the request several times.

Variables and messages. BPEL4Chor transfers data along message links using a message. This corresponds to BPMN message flows and the message attribute. BPEL4Chor allows to specify participant references and participant sets to be sent over a message link. These can be annotated using data objects. In addition, BPEL uses variables to store data. BPMN does not distinguish between different types of data objects. For example, a data object modeling a variable cannot be distinguished from a data object modeling a participant reference.

Attributes. Some additional information about the process can be defined using the element attributes in BPMN. For instance, the correlation sets of a BPEL process can be defined using the properties attribute of a BPMN process. However, element attributes are not sufficient for representing all BPEL4Chor information. For example, the import elements of a process or the from and to elements in a variable declaration cannot be expressed.

Participant grounding. In BPMN the implementation details are part of the graphical elements and are defined in element attributes such as Implementation or Message. This does not satisfy the requirement of BPEL4Chor, which demands that the implementation details should be separated from the model.

5 BPMN Extensions for Service Choreographies

BPMN is intended to be extensible and allows to add new markers and indicators associated with the BPMN elements. The indicator *pool set* is introduced as a shaded pool to denote multiple instances of participants of the same type.

Scopes are sub-processes without any marker. We introduce the *isolated attribute* for sub-processes to identify isolated scopes.

We use markers to distinguish the *task types assign, empty, validate and opaque* as presented in Fig. 2. These tasks are not allowed to be connected with message flows, since they do not take part in the communication between the participants. A *new trigger for the termination event* is added as shown in Fig. 3. The purpose of the termination event is to trigger a termination handler. Thus, it can only be attached to the boundary of tasks and to sub-processes.

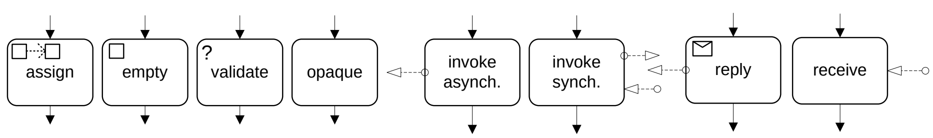

Fig. 2. Configured Tasks

To distinguish event, fault, termination and compensation handlers, marked sub-processes are introduced. Event handler sub-processes cannot be connected with sequence flows. The parent process or sub-process of an event handler determines the context in which it can be triggered. Fault, termination and compensation sub-processes can only be connected with the appropriate attached event.

Different types of data objects need to be distinguished. Therefore, the indicator participant set data object is introduced. Furthermore, markers for the data object are introduced to denote participant reference data objects and variable data objects.

The *participant set* and *participant reference data objects* are used to denote which participant is actually taking part in a conversation. A pool set represents

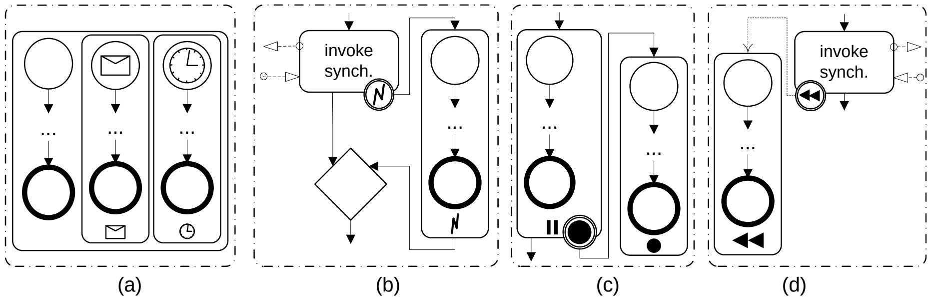

Fig. 3. (a) Event, (b) fault, (c) termination and (d) compensation handlers

multiple participants. Thus, participant data objects are needed for conversations with activities located in a pool set to denote the concrete participant instance being communicated with. Participant reference and participant set data objects can be associated with communicating activities (invoke task, reply task, message event) to define the participants taking part in the communication. The participant data objects have different semantics depending on the direction of the association they are connected with. Fig. 4 illustrates that an association emanating from a participant reference data object leading to a sending task denotes that a message is sent to this participant. If an association leads to a receiving flow object (message event, invoke task), a message from this participant is expected. A participant set data object association with a multiple instance task or sub-process denotes that the loop will iterate over that participant set.

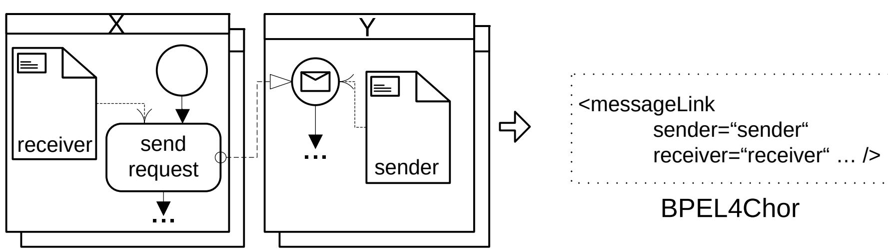

Fig. 4. Sender and receiver

A directed association from a participant reference data object to a multiple instance sub-process denotes that the participant reference acts as loop counter.

Fig. 5 shows an association from a receiving flow object to a participant set data object. This association denotes that a message is expected from an arbitrary participant and a reference to the sender of the message will be stored in the associated set. The actual participant reference in the set is represented by the participant reference data object associated with the flow object.

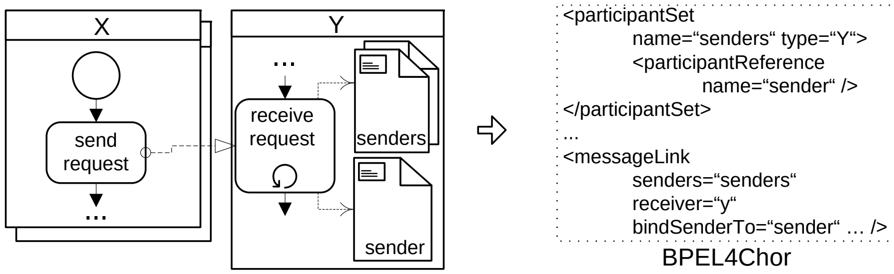

Fig. 5. Storing sender in a participant set

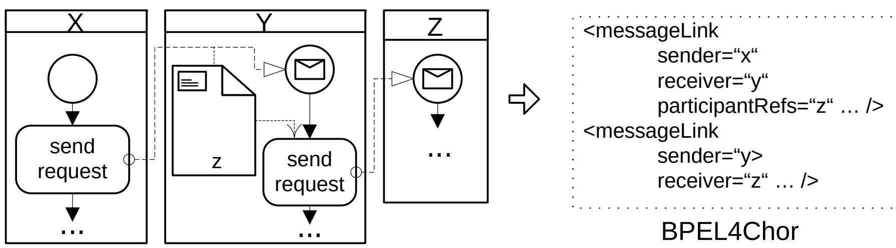

Fig. 6. Passing a participant reference over the message flow

Besides communicating activities, participant reference data objects and participant set data objects can be associated with message flows as presented in Fig. 6. This realizes link passing mobility: The associated participant data objects are references passed over the message flow.

Variable data objects are used to denote variables. They must be located within the boundaries of a pool, pool set or sub-process and they can be associated with flow objects only. The association must not cross the boundary of the pool, pool set or sub-process where the variable is located. Variable data objects can be associated with message events, fault events, invoke tasks and reply tasks without any further requirements. Complex associations between variables and tasks (e.g., for assign tasks) are defined in additional element attributes introduced for each task type. We introduce four different types of variable data objects: standard variables express the defined variables within a process or sub-process, counter variables represent the counter in a forEach activity, fault variables hold the data of a fault that was thrown or caught and message variables contain the message that triggers a message event of a message event handler. Each variable type offers different attributes in which the relevant data is stored.

To keep the BPEL4Chor grounding replaceable, we introduce the attribute *grounding* for the diagram. This attribute denotes the URI of the participant grounding file.

An extension of the example in Section 2 with the introduced elements is shown in Fig. 8. The pool set expresses the multiple suppliers. The participant

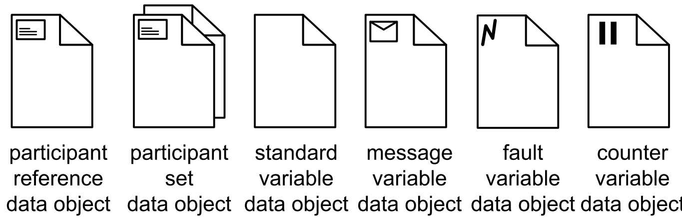

Fig. 7. Configured data objects

set data object associated with the multiple instance sub-process represents the set of suppliers the loop will use for the iteration. The current supplier of each looping instance is stored in the participant reference data object. This data object is located in the multiple instance sub-process and associated with it. Furthermore, it is associated with the "request price" task to denote that the request is sent to the current supplier. The fault handler sub-process has a special marker. A fault variable data object is associated with the attached fault event to indicate the storage of the fault data in this variable. The reply tasks in the supplier pool set can be distinguished from the invoke tasks in the book shop pool set with the aid of a special marker.

6 Mapping Configured BPMN to BPEL4Chor

Regarding the transformation to BPEL4Chor we restrict the configured BPMN. The following elements are not allowed:

- complex gateways
- ad-hoc and transactional sub-processes
- link, rule and multiple start events
- all end events except the non-triggered ones
- cancel, rule, link, multiple or non-triggered intermediate events
- user, script, abstract, manual, reference or non-typed tasks

In addition, we do not allow arbitrary cycles in BPMN diagrams, as their translation leads to convoluted BPEL code: As there is no direct support for arbitrary cycles in BPEL, message exchanges within one process in combination with event handlers and fault handlers are resorted to as workaround in [14]. Furthermore, gateways are only allowed to have either more than one incoming or more than one outgoing sequence flow.

A transformation from BPMN to BPEL has been developed in [15] for a subset of BPMN elements. This transformation is based on the identification of *patterns* in the diagram that can be mapped onto BPEL blocks. One pattern is folded into a new task, which is associated with the generated BPEL code. We extend these patterns with the elements used in the configured BPMN described above. Hence, we can use that transformation for transforming processes located in a pool,

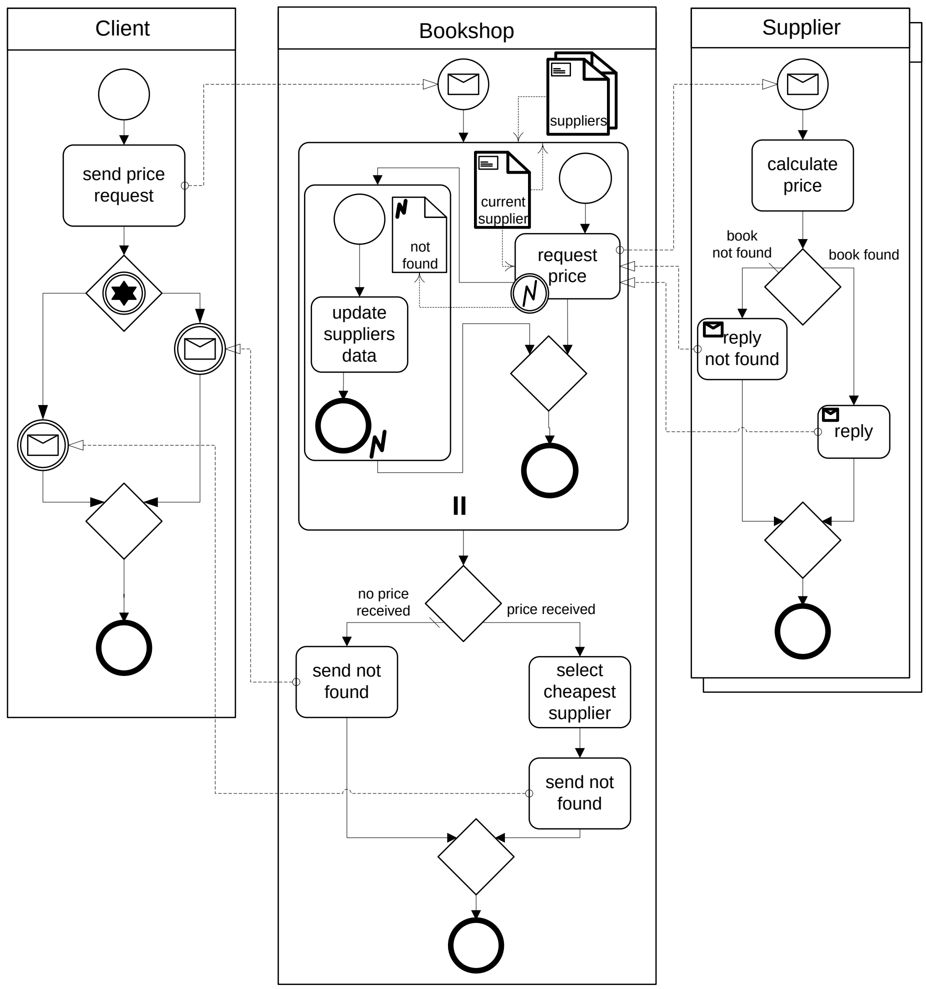

Fig. 8. Choreography modeled in configured BPMN

pool set or sub-process to their BPEL4Chor representation. The transformation assumes that there is only one start and one end event in each process.

We have to introduce a pattern for tasks and sub-processes with attached events leading to the appropriate handlers. This *attached event pattern* creates a handler for each attached event. The nesting is handled by the newly introduced *nesting pattern*, where the nested content is handled as a process.

Event handlers are not connected with any sequence flow. Thus, they cannot be treated as pattern and must be handled before the mapping of the process flow as shown in Fig. 9, step 4.

The tasks and events are mapped during the mapping of the patterns. The variable data objects are not folded because they may be associated with flow objects in other patterns. Each pool and pool set is mapped to a participant type. For a simple pool a participant reference with the type can be generated directly. Additional references are generated from participant reference data objects. The mapping of message flows to message links depends on the connected activities, the participant reference and participant data objects associated with these activities and the message data objects associated with the message flows. The extended transformation removes elements from the model during the folding of the patterns. Thus, the topology has to be created beforehand. An overview of the whole transformation is given in Fig. 9.

1. Generate participant types in the topology depending on the pools and pool sets
2. Generate participant references and participant sets from the participant reference and participant set data objects
3. Generate message links from the message flow, the associated participant reference and message data objects
4. Transform the processes within the pools and pool sets
   (a) Generate the variables from the variable data objects
   (b) Generate event handlers. Treat the content as process and convert it according to step 4
   (c) Apply the extended transformation starting with the pattern for attached events

Fig. 9. General transformation approach from configured BPMN to BPEL4Chor

7 Related Work

There are different language proposals available for modeling choreographies. The Web Service Choreography Description Language (WS-CDL [12,3]) was released by the World Wide Web Consortium in 2005. Differences between WS-CDL and BPEL are discussed in [13]. Let's Dance [17] is another choreography language. Like BPMN, it is implementation-independent and comes with a visual notation. This language was designed to support all Service Interaction Patterns [4], a set of recurrent choreography scenarios. An assessment of WS-CDL using these patterns can be found in [8]. An earlier and less expressive choreography language is the Business Process Schema Specification (BPSS [5]). A general introduction into the different viewpoints found in inter-organizational process modeling can be found in [10].

There are basically two different choreography modeling styles manifested in the languages. In the case of *interconnection models*, send and receive activities are enlisted for each role and control and data flow dependencies are defined on a per-role-basis. In contrast to this, *interaction models* are made up of atomic interactions and control and data flow is defined globally, i.e. it is not directly assigned to any of the roles. Examples for the first group are BPMN and BPEL4Chor, but also simpler languages such as Message Sequence Charts (MSC [11]). Examples for the second group are WS-CDL, BPSS and Let's Dance. Bridging these two modeling styles is not trivial and requires for sophisticated transformation algorithms as presented in [9] for the case of interaction Petri nets and their corresponding participant behavior descriptions. This is not needed in our case, as BPMN and BPEL4Chor follow the same modeling style.

There has been some work on comparing BPMN and BPEL and carrying out transformations. Comparison was done e.g. in [16] on the general concepts covered in both languages and on the respective Workflow Pattern support: the authors' conclusion is that the expressiveness of BPMN has to be restricted if a full mapping to BPEL is desired.

A major challenge in transforming BPMN to BPEL are the differences in control flow constructs available in the languages. Ouyang et al. [14] restricted BPMN and mapped that subset completely to BPEL.

Several commercial tools allow to define BPEL-specific configurations for BPMN-models and implement transformation algorithms. However, typically only a small subset of BPMN is allowed and then translated. None of the tools provides a transformation to BPEL4Chor.

8 Conclusion and Outlook

We presented an approach to extend BPMN to support modeling of BPEL4Chor choreographies. Our approach is independent of the structural transformation of BPMN to BPEL4Chor. If new results including OR gateways are available, these can be applied to our approach. We took the transformation of Ouyang et al. as example and extended it to support our BPMN choreography extensions. Thus, we showed how BPEL4Chor can be generated out of configured BPMN. Validation is provided through the ongoing implementation of an editor supporting configured BPMN and the transformation of configured BPMN to BPEL4Chor. The editor is web based and already comes with a support for BPMN. It stores the diagrams in an extended XPDL format reflecting the extensions of the configured BPMN. The transformation is implemented as Java web service that will be integrated in the editor. The extended XPDL format is used as input for the transformation web service.

We introduced participant reference data objects to reflect participant references in BPMN. On the other hand, participant references are first class citizens in BPEL4Chor. We suggest adding the notation of participant reference objects to BPMN, as we argue that participant references have major importance.

References

1. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. `http://www.bpmn.org/`.

2. Web Services Business Process Execution Language Version 2.0 – Committe Specification. Technical report, OASIS, Jan 2007.
3. A. Barros, M. Dumas, and P. Oaks. A Critical Overview of WS-CDL. *BPTrends*, 3(3), 2005.
4. A. Barros, M. Dumas, and A. ter Hofstede. Service Interaction Patterns. In *BPM 2005*, LNCS, pages 302–318, Nancy, France, 2005. Springer Verlag.
5. J. Clark, C. Casanave, K. Kanaskie, B. Harvey, N. Smith, J. Yunker, and K. Riemer. ebXML Business Process Specification Schema Version 1.01. Technical report, UN/CEFACT and OASIS, May 2001. `http://www.ebxml.org/specs/ebBPSS.pdf`.
6. F. Curbera, F. Leymann, T. Storey, D. Ferguson, and S. Weerawarana. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More*. Prentice Hall PTR, 2005.
7. G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4chor: Extending BPEL for Modeling Choreographies. In *Proceedings International Conference on Web Services (ICWS)*, 2007.
8. G. Decker, H. Overdick, and J. M. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *EMISA 2006*, Hamburg, Germany, Oct 2006.
9. G. Decker and M. Weske. Local Enforceability in Interaction Petri Nets. In *Proceedings 5th International Conference on Business Process Management (BPM 2007)*, LNCS, Brisbane, Australia, 2007.
10. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
11. ITU-T. Message sequence chart. Recommendation Z.120, ITU-T, 2000.
12. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report, November 2005. `http://www.w3.org/TR/ws-cdl-10`.
13. J. Mendling and M. Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In R. Meersman, Z. Tari, and P. H. et al., editors, *Proceedings of OTM 2005 Workshops. Lecture Notes in Computer Science 3762*, pages 506–515. Springer Verlag, October 2005.
14. C. Ouyang, M. Dumas, S. Breutel, and A. H. ter Hofstede. Translating Standard Process Models to BPEL. In *CAiSE 2006*, Luxembourg, June 2006.
15. C. Ouyang, M. Dumas, A. H. ter Hofstede, and W. M. van der Aalst. Pattern-based translation of BPMN process models to BPEL web services. *International Journal of Web Services Research (JWSR)*, 2007.
16. J. Recker and J. Mendling. On the translation between BPMN and BPEL: Conceptual mismatch between process modeling languages. In T. Latour and M. Petit, editors, *CAiSE 2006 Workshop Proceedings - Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD 2006)*, pages 521–532, June 2006.
17. J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. A Language for Service Behavior Modeling. In *Proceedings 14th International Conference on Cooperative Information Systems (CoopIS 2006)*, Montpellier, France, Nov 2006. Springer Verlag.