Modeling Service Choreographies using BPMN and BPEL4Chor*

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

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

Abstract. Interconnecting information systems of independent business partners requires careful specification of the interaction behavior the different partners have to adhere to. Choreographies define such interaction constraints and obligations and can be used as starting point for process implementation at the partners' sites. This paper presents how the Business Process Modeling Notation (BPMN) and the Business Process Execution Language (BPEL) can be used during choreography design. Step-wise refinement of choreographies to the level of system configuration is supported through different language extensions as well as a mapping from BPMN to BPEL4Chor. A corresponding modeling environment incorporating the language mapping is presented.

1 Introduction

Automated electronic communication between different business partners offers big optimization potential regarding the overall business process performance. However, it also comes with certain challenges that have to be tackled. Common message formats must be agreed upon and the allowed and expected interaction sequences must be clearly defined. Legal consequences of message exchanges as well as time constraints must be captured.

Choreography languages provide a means to specify the messages exchanged between different organizations along with behavioral constraints. The Business Process Modeling Notation (BPMN [2]) offers a rich set of graphical notations for control flow constructs and includes the notion of interacting processes where sequence flow (within an organization) and message flow (between organizations) are distinguished. Therefore, BPMN is a good candidate for providing a graphical notation for choreography modeling. When it comes to refining such initial choreography models, details about timing constraints and exception handling have to be added. Finally, technical configurations are introduced for reaching unambiguity "on the wire". In order to express the different levels of details in BPMN we present several extensions to this language.

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

The Business Process Execution Language (BPEL [3]) is the de-facto standard for implementing business processes based on web services. The orchestrated web services are again exposed as services. BPEL also allows to specify ordering constraints on the messages a service accepts and produces. All in all, it only focuses specifying processes from a single organization point of view, treating the services used as opaque entities ignoring their internal structure forming separate business processes. As a consequence, choreographies cannot be described using BPEL. Therefore, we have proposed choreography extensions for BPEL in earlier work [8], adding the notion of participant topologies for gluing together different participant behavior descriptions (PBDs). PBDs are BPEL processes describing the behavior of each participant in the choreography. We propose to use BPEL4Chor as an interchange format supporting the different choreography design phases. Therefore, a transformation of BPMN choreographies to BPEL4Chor is needed.

This paper extends work from Ouyang et al. [18], where BPEL stubs are generated out of individual BPMN processes. Furthermore, this paper builds upon previous work from [10], where BPMN extensions for high-level choreography modeling were proposed, and [4], where different modeling phases and choreography viewpoints were identified. The contribution of this paper is to present the integrated usage of BPMN and BPEL4Chor during choreography design. Furthermore, we implemented a modeling environment for BPMN where BPEL4Chor choreographies are produced.

The remainder of this paper is structured as follows. The next section discusses choreography design and the use of BPMN therein. Section 3 gives an overview of BPEL4Chor. Section 4 describes the mapping of extended BPMN to BPEL4Chor and section 5 presents our modeling environment. Section 6 reports on related work in the literature, before section 7 concludes.

2 Choreography Design using BPMN

Complex choreographies cannot be created within a single step. Whenever many different business partners and many interactions are involved, choreography design must be split up into different phases each addressing different issues of the model. As reported in [22] and [4], the following phases can be distinguished.

1. At a very high level, the interaction partners are identified. It must also be captured how many partners of a particular type are involved. E.g. in a logistics scenario a number of carriers might be involved while only one breakdown surveillance service takes part. Furthermore, business documents that are exchanged between the partners are listed and agreement on the general content of the documents must be reached. E.g. it is defined that a certain contract must carry two signatures or that a request for quote must contain the quantity of the desired product. These two first steps lead to a *high-level structural view* on the choreography.
2. Choreographies reflect what interactions are needed to fulfill a certain goal. This goal can typically be divided into sub-goals or milestones that must be

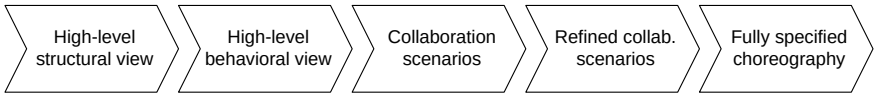

Fig. 1. Different artifacts produced in the choreography design phases

reached on the way to the overall goal. This calls for a *high-level behavioral view* on the choreography.

3. Once the milestones are defined, *collaboration scenarios* are a means to capture how to get from one milestone to another milestone. The required interaction sequences are modeled accordingly.
4. While first versions of collaboration scenario models are likely to only capture best cases, exception handling must be added subsequently. This leads to *refined collaboration scenarios* that also capture timing constraints.
5. All scenario models are aggregated into a big choreography model, including all interactions and their dependencies. Technical choices must be made, e.g. whether to use synchronous vs. asynchronous communication. This leads to the *fully specified choreography model*.

BPMN supports modeling of choreographies as collaboration diagrams. Pools model business roles or entities, while message flows represent the communication between them. High-level structural diagrams can be realized in BPMN by using empty pools and message flow between them. As it is not possible to represent that multiple participants of the same type are involved in one conversation, we added a *pool set* for this purpose.

Figure 2 is a sample structural diagram illustrating a scenario that will be used throughout this paper: A customer buys a product from a seller. The seller in turn handles payment through a payment service. Delivery is outsourced to a delivery service which in turn does not carry out the delivery by itself but rather manages the actual delivery done by a set of carriers. In some cases several carriers are involved covering a part of the overall journey by air, rail or truck.

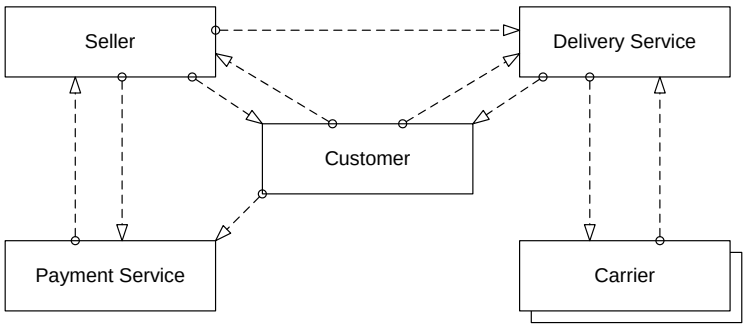

Fig. 2. High-level structural diagram in extended BPMN

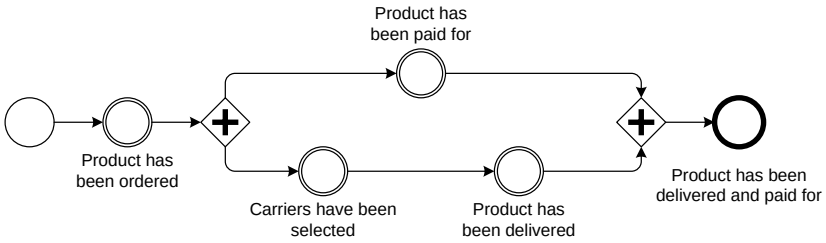

Fig. 3. High-level behavioral diagram in BPMN

The pools (rectangles) in Figure 2 represent the different participant types. Only one customer, seller, payment service and delivery service are involved in one conversation, i.e. one choreography instance. The shaded pool for type carrier represents that there might be more than one carrier involved in one conversation. The dashed arrows symbolize message flow between participants of the corresponding types, indicating who potentially sends a message to whom.

High-level behavioral diagrams can be modeled in BPMN as shown in Figure 3. Untyped events (empty circles) represent milestones which in turn are connected through control flow constructs. This example shows that the first milestone to be reached is that the customer has ordered a product. This is the precondition for the two subsequent milestones "product has been paid for" and "carriers have been selected". The first AND-gateway (diamond containing a "+") represents that the two succeeding milestones can be reached in any order. The second AND-gateway synchronizes the two branches and leads to the final milestone.

Collaboration scenarios which show how progress from one milestone to another can be achieved, are modeled as collaboration diagrams. This time, the pools are not empty but rather the ordering of the message exchanges is expressed by relating the communication activities (send and receive activities) using control flow constructs.

In Figure 4 we see how the collaboration scenario connects to other models: Two milestones from the high-level behavioral model appear again. Further connections to other models are established through the use of link events (circles containing an arrow). This is the standard BPMN way of modeling off-page connectors.

In choreographies where multiple participants of the same type might be involved, it is important to distinguish the individual participants. This is achieved by the introduction of special data object types, namely *participant references* and *participant sets*, symbolized by (shaded) artifacts with a business card icon in the upper left corner. Figure 4 illustrates how this is used in the context of different carriers that must be chosen from.

The semantics of the diagram is as follows. The seller initiates delivery by sending a delivery request to the delivery service. This service contacts all its partner carriers, asking them to check availability for the entire route or parts of the route. Upon receipt of these request, each carrier checks availability. If no

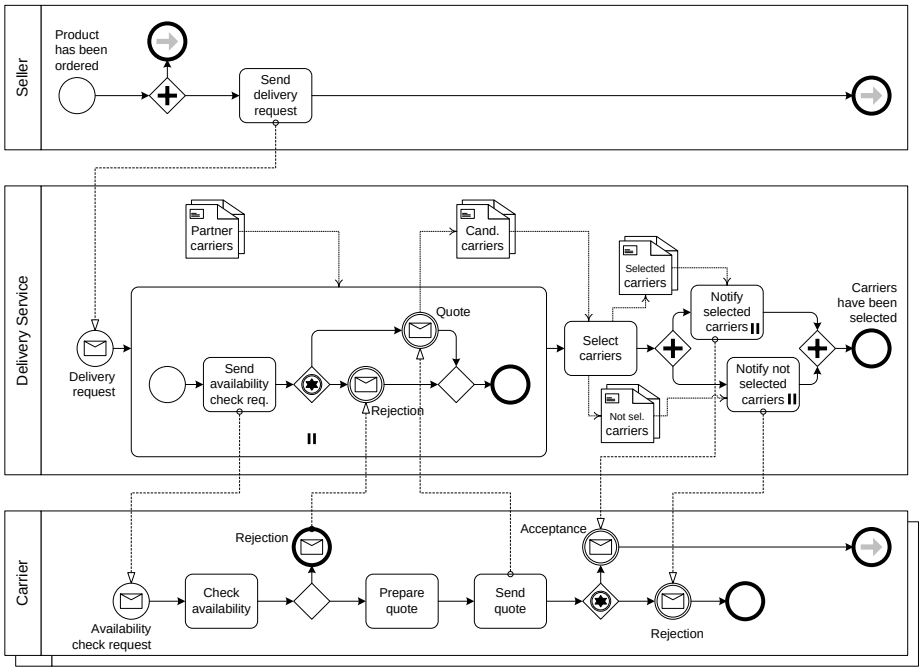

Fig. 4. Collaboration scenario in BPMN: Progressing from "product has been to ordered" to "carriers have been selected"

capacity is available the carrier answers with a rejection message. Otherwise the carrier prepares a quote and sends it back to the delivery service. The delivery service collects the quotes and remembers all carriers that have sent a quote as "candidate carriers". Once all carriers have answered, the delivery service selects one or more carriers and sends notifications to the carriers telling them whether they were selected or not. After this, the scenario ends by reaching the milestone "carriers have been selected".

The diagram illustrates how participant references and participant sets affect communication activities and multi-instance activities. The set of partner carriers serves as input for the multi-instance subprocess, indicating that one instance should be spawned for each carrier in this set. Associations from participant references to send and receive activities define that the message is sent to the referenced participant and that only a message from the referenced participant will be received, respectively.

Figure 4 only covers the best case of our collaboration scenario. It is not specified yet what happens if the carriers do not respond within a given timeframe. It is also not specified what happens if no suitable carrier can be found. This might lead to notifying the customer about a delay in delivery or even completely canceling the order.

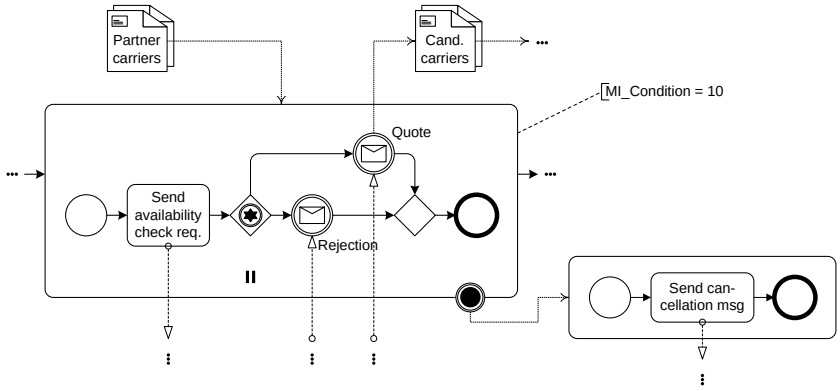

Fig. 5. Termination handlers

BPMN allows to model timeouts and exceptions by offering corresponding event types. Intermediate events attached to activities and subprocesses represent cancellation upon the occurrence of the event. Using these constructs it is possible to model a wide range of exception scenarios. However, when comparing BPMN to BPEL in terms of exception handling, we find that a number of important concepts of BPEL are missing in BPMN. As we intended to allow the modeler to refine the choreography model to fully specified models, we adopted these concepts as BPMN extensions. The following list gives an overview of these extensions. A detailed discussion on these can be found in [19].

Termination handlers. The termination handler of a subprocess defines reactions to forced termination. Especially in the case of forEach constructs with completion condition, termination handlers are needed. As soon as the completion condition is fulfilled, all remaining subprocess instances are terminated. We introduce termination handlers to BPMN. We opted for a similar graphical representation as it is used for compensation handlers. Figure 5 shows a refinement of a part of the scenario from Figure 4.

Different data object types. As already mentioned we introduce participant references and participant sets as special data object types. Additionally, we distinguish between fault variable data objects, counter variable data objects and standard variable data objects. Counter variables represent the counter in a forEach activity and fault variables hold the data of a fault that was thrown or caught.

Correlation information. Correlation is the act of relating messages received to process instances and receive activities within this instance. Typically, correlation is done based on specific message content. E.g. an order id is used to route an incoming message to the corresponding instance handling the order. While there are very complex correlation mechanisms thinkable, we opted for a correlation set semantics like it is present in BPEL. Therefore, we added corresponding attributes to the invoke and receive activities.

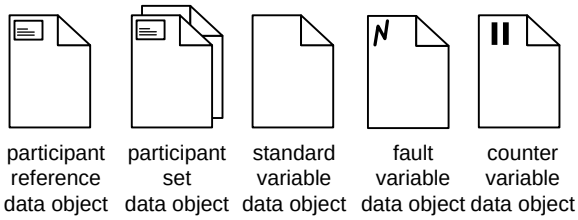

Fig. 6. Data object types

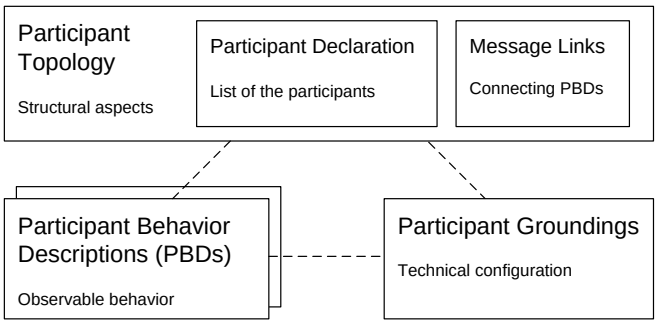

Fig. 7. BPEL4Chor artifacts

3 BPEL4Chor Overview

BPEL4Chor is a language to describe service choreographies in BPEL. It distinguishes between three aspects: (i) the participant topology, which provides a global view on the existing participants and their interconnection using message links, (ii) participant behavior descriptions, i.e. control flow dependencies in each participant and (iii) participant groundings, i.e. concrete configurations for data formats and port types.

The high-level structural view can be captured in the participant topology. The participants are listed in the participants declarations part. Here, participants and participant sets are distinguished. Each participant carries a type, which specifies the behavior of the participant. In the example participant topology shown in Listing 1, there exists one participant for participant type DeliveryService. The delivery service knows several partner carriers, therefore the topology contains the participant set PartnerCarriers. Participant sets can be used in a forEach construct, in the sense that the forEach construct iterates over this set. The current participant for the iteration is called `currentCarrier` in the listing. The messages exchanged are modeled using message links. A message link connects two participants and states which message is sent over it. Listing 1 lists an extract of the participant topology for our scenario.

Behavioral aspects are captured in the participant behavior descriptions, expressed in BPEL. Listing 2 presents the first part of the BPEL process for

Listing 1 Participant topology

```
<topology name="DeliveryTopology">
<participantTypes>
  <participantType name="Seller"
  participantBehaviorDescription="ns1:Seller" />
  <participantType name="DeliveryService" ... />
  <participantType name="Carrier" ... />
</participantTypes>
<participants>
  <participant name="Seller" type="Seller" />
  <participant name="DeliveryService" type="DeliveryService" />
  <participantSet name="PartnerCarriers" type="Carrier"
   forEach="ns2:pcarrierForEach">
    <participant forEach="ns2:pcarrierForEach" name="currentCarrier" />
  </participantSet>
  ...
</participants>
<messageLinks>
  <messageLink name="orderLink" messageName="order"
  sender="Seller" receiver="DeliveryService" />
  ...
</messageLinks>
</topology>
```

Listing 2 Participant behavior description for type delivery service

```
<process name="DeliveryService"
  <sequence>
    <receive createInstance="yes" name="ReceiveDeliveryRequest" />
    <sequence>
      <forEach name="pcarrierForEach" parallel="yes">
        <scope><sequence>
          <invoke name="SendAvailabilityCheckReq." />
          <pick>
            <onMessage wsu:id="Quote"><empty /></onMessage>
            <onMessage wsu:id="Rejection"><empty /></onMessage>
          </pick>
        </sequence></scope>
      </forEach>
      <opaqueActivity name="SelectCarriers" />
    </sequence>
    ...
  </sequence>
</process>
```

the delivery service. The communication constructs are named so that they can be interconnected. The interconnection is formed by adding the names of the activities to message links. While the message links in the sample topology in

Listing 3 Participant grounding

```
<grounding topology="DeliveryTopology">
  <messageLinks>
    <messageLink name="orderLink" portType="ds:deliveryService_pt"
    operation="getProduct" />
    ...
  </messageLinks>
</grounding>
```

Listing 1 have the attributes `sender` and `receiver` set, attributes `sendActivity` and `receiveActivity` must also be set for referring to the communication constructs in the participant behavior descriptions.

Technical choices are reflected in the participant grounding, where concrete port types and operations come in. Each message link is assigned to a port type and operation. Listing 3 presents the grounding of one message link. The grounding can then be used to generate abstract BPEL processes which are subsequently used for executable completion.

4 Mapping BPMN to BPEL4Chor

Although our extended BPMN and BPEL4Chor have a large overlap in concepts covered, not all diagrams can be transformed to BPEL4Chor. The following BPMN elements are not allowed:

- complex gateways
- ad-hoc and transactional subprocesses
- 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 or reference activities

In [18] three classes of BPMN diagrams are distinguished: (i) those that can be translated using block-structured constructs only, (ii) those that require the use of control links and finally (iii) those that require event handlers, fault handlers and message passing within one process instance for realizing control flow dependencies. For instance, the occurrence of the workflow patterns arbitrary cycles and multi merge [21] make a diagram be of category (iii), as there is no direct support for these two workflow patterns in BPEL. We argue that the BPEL code resulting from (iii) is not usable as starting point for further refining it to process implementations. Therefore, we do not transform these kind of diagrams.

General Approach. We largely base our transformation on the approach presented in [18] where a subset of BPMN is transformed to BPEL. This approach is based on the identification of *patterns* in the diagram that can be mapped onto BPEL blocks. One pattern is folded into a new activity, which

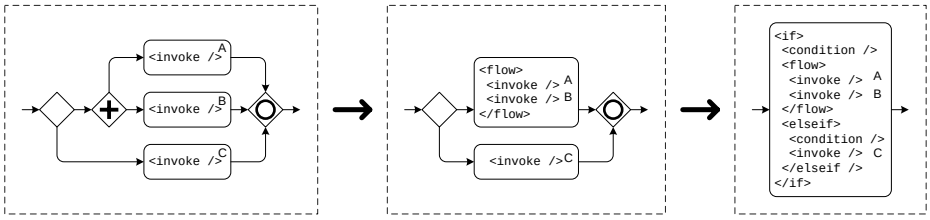

Fig. 8. Dealing with inclusive gateways

is associated with the generated BPEL code. We extend these patterns with the elements used in the extended BPMN described above. Hence, we can use that transformation for transforming processes located in a pool, pool set or subprocess to their BPEL4Chor representation. Furthermore, we loosen certain restrictions as explained in the next subsection.

Multiple start and end events. In [18] it is assumed that there is only one start event and one end event in each process. We loosen this restriction and allow certain combinations of start events as well as multiple end events. If e.g. two start events are followed by a XOR-gateway, we fold this pattern to a BPEL pick element, where the attribute createInstance is set to "yes". Also the case if they are followed by an AND-gateway can be handled and translated to BPEL4Chor. These scenarios are captured by generalized pick- and flow-patterns. While it is easy to see for these simple examples how they can be mapped, it is less obvious why some combinations are not allowed in our transformation. Imagine e.g. three start events A, B, C where A and B are merged through an AND-gateway, which in turn is merged with C through a XOR-gateway. Here, C is an alternative to the combination of A and B. Such behavior is not directly expressible in BPEL. Multiple end events are resolved by merging the different branches into an inclusive gateway.

Inclusive gateways. We allow inclusive gateways if they occur in certain combinations with other elements and can be rewritten to AND- and XOR-gateways. In order to capture these combinations, the well-structured and quasi-structured patterns from [18] are extended. This means that our transformation can handle inclusive gateways in block-structured settings only.

Figure 8 illustrates an example. It exhibits two steps to transform a BPMN diagram involving multiple invoke activities to the corresponding BPEL representation. In the first step, an AND split gateway is translated to a BPEL flow, representing concurrent invocations of A and B. In the second step, the XOR split gateway is translated to an if construct in BPEL, so that either invocations of A and B are performed concurrently or C is invoked.

Fault, compensation and termination handlers. We introduce a pattern for activities and subprocesses with attached intermediate events. This leads to the creation of a BPEL handler for each attached event. To enable direct transformation to BPEL, we only allow those fault handlers, where the outgoing

control flow from the handler is directly merged with the control flow originating from the corresponding activity or subprocess.

Other constructs. The mapping of activities and events is straightforward. 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 its corresponding 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. As the extended transformation removes elements from the model during the folding of the patterns, the topology has to be created beforehand.

1. Generate participant types in the topology from 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
   4.1. Generate the variables from the variable data objects
   4.2. Apply the extended transformation starting with the pattern for attached events

5 Choreography Modeling Environment

We have implemented a BPMN editor and the BPMN to BPEL4Chor transformation based on the Oryx framework developed at the Hasso-Plattner-Institute[3]. Oryx is a graphical editing framework written in JavaScript that uses Scalable Vector Graphics (SVG) as rendering technology. Oryx comes with a set of stencil sets for modeling pure BPMN, the extended BPMN, workflow nets and other process modeling languages. Each stencil set defines a set of elements, including their attributes, containment relationships and connection rules. The shape definitions, i.e. the graphical appearance of elements, are defined as SVG files.

Oryx strictly follows the REST (Representational State Transfer [13]) architectural style. Each process model and each element within it are considered as resources that are uniquely identified by URIs. By addressing a process model URI in a web browser, an XHTML representation of the model is retrieved, which in turn contains all model information as embedded RDF annotations. This web page also contains links to the Oryx implementation. The browser loads these scripts which turn the web page into a graphical editor application. If models are to be imported into other applications, existing XSLT stylesheets can be applied for retrieving corresponding RDF documents. Figure 9 illustrates the system architecture using the Fundamental Modeling Concepts notation [16].

[3] See `http://bpt.hpi.uni-potsdam.de/Oryx/`

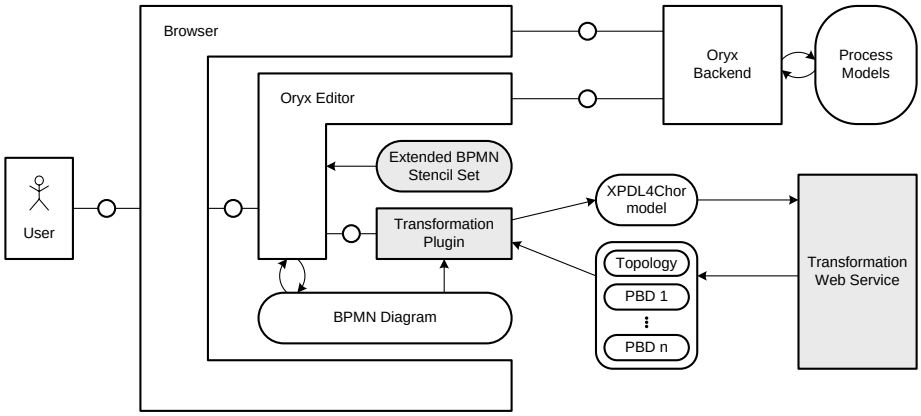

Fig. 9. Architecture of the modeling environment

The editor provides extensibility through a plugin mechanism. We used this mechanism to integrate the BPMN to BPEL4Chor transformation functionality into the editor. The transformation plugin serializes the extended BPMN diagram into an extended XPDL (XML Process Definition Language [1]) format, called XPDL4Chor. While XPDL 2.0 is a serialization format for BPMN standardized by the Workflow Management Coalition (WfMC), XPDL4Chor additionally contains the new elements and attributes we added to BPMN.

The actual transformation takes place in a separate web service. This service takes the XPDL4Chor document as input and produces the different BPEL4Chor documents. This includes the participant topology as well as the participant behavior descriptions for each participant type. The plugin offers the possibility to download these documents or to view them in the browser.

Figure 10 shows a screenshot of the Oryx editor[4]. On the left side the palette contains the different language constructs. These can be dragged onto the drawing area in the middle. Attributes of the model elements can be edited in the properties area on the right. Different editing functionality can be accessed through the buttons on the top. Output as BPEL4Chor files or output as XPDL4Chor file can be triggered through two of these buttons.

6 Related Work

There are different language proposals available for modeling choreographies. The Web Service Choreography Description Language (WS-CDL [5, 15]) was released by the World Wide Web Consortium in 2005. Differences between WS-CDL and BPEL are discussed in [17]. Let's Dance [23] 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 [6], a

[4] The editor is accessible through `http://www.bpel4chor.org/editor/`.

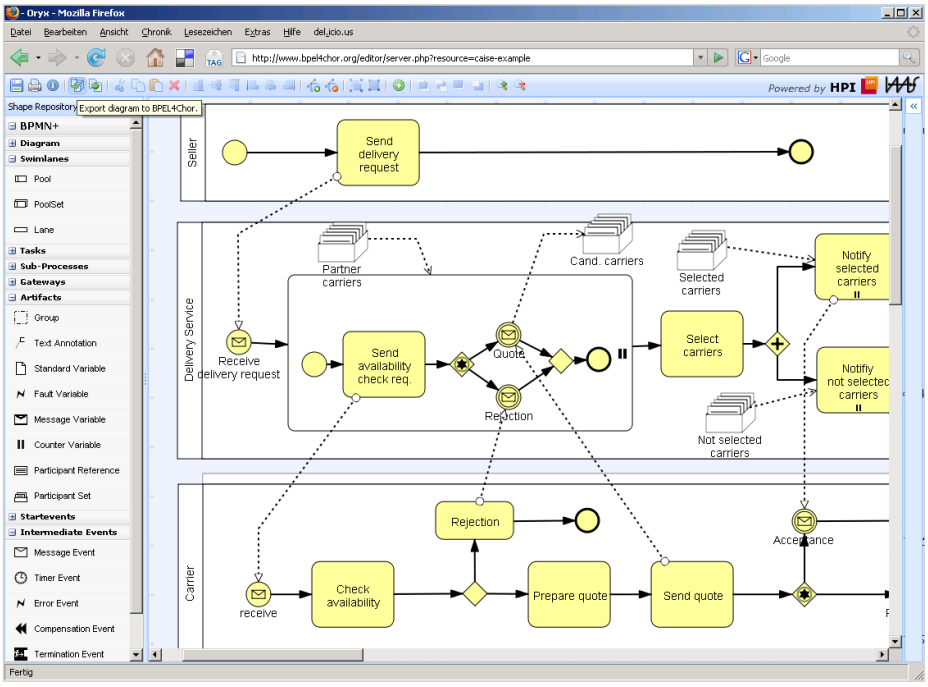

Fig. 10. Screenshot of the Oryx editor with the transformation plugin

set of recurrent choreography scenarios. An assessment of WS-CDL using these patterns can be found in [9]. An earlier and less expressive choreography language is the Business Process Schema Specification (BPSS [7]). A general introduction into the different viewpoints found in inter-organizational process modeling can be found in [12]. Already in [10] we have shown how the addition of the concepts pool set, participant references and participant sets leads to a significantly higher suitability of BPMN for choreography modeling. Such extended BPMN even surpasses WS-CDL in terms of Service Interaction Pattern support.

There are basically two different modeling styles manifested in choreography languages. In the case of *interconnected models*, send and receive activities are listed 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 [14]). 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 [11] 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 [20] 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. [18] 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.

7 Conclusion and Outlook

We presented how BPMN can be used as modeling language in the different choreography design phases. By extending BPMN we reached a higher suitability of BPMN for modeling choreographies both in early design phases as well as in late phases, where exceptions and technical configurations are added. We chose BPMN for modeling choreographies, since it is widely used in the industry and has a wide tool support.

BPMN does not specify a serialization format. We use BPEL4Chor as interchange format for the choreography models at the different levels of detail. BPEL is the standard language for describing executable workflows. Since BPEL4Chor is close to BPEL, the gap between design time and runtime is narrowed. We provided a transformation of BPMN models to BPEL4Chor, extending the transformation from [18]. BPMN elements not having a corresponding notation in BPEL4Chor are not transformed. However, these details are not required by the runtime. Finally, we presented Oryx, our graphical modeling environment that runs on the web, where a transformation plugin was added. We do not provide support for round-trip engineering: only a one-way transformation from BPMN to BPEL4Chor is provided. Modifications to the BPEL4Chor artifacts are not reflected in the BPMN diagram.

Limitations of our approach are the restrictions we impose on the BPMN models that can be transformed to BPEL4Chor. As part of that, we require that BPMN models are sound and safe, i.e. deadlock-free and without multi-token flow. We currently do not check these properties prior to the transformation. Generally, the fact that only during the transformation we can detect that we cannot transform a model, can be seen as the biggest limitation of our current implementation. It is desirable to perform a check prior to starting the transformation and to give the modeler hints how to resolve the problem. This is subject to ongoing work. As part of that we are working on integrating Petri-net-based analysis functionality into the BPMN editor.

References

1. Process Definition Interface – XML Process Definition Language, October 2005. http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf.
2. Business Process Modeling Notation (BPMN) Specification, Final Adopted Specification. Technical report, Object Management Group (OMG), February 2006. http://www.bpmn.org/.
3. Web Services Business Process Execution Language Version 2.0 – OASIS Standard, April 2007.
4. A. Barros, G. Decker, and M. Dumas. Multi-staged and Multi-viewpoint Service Choreography Modelling. In *SEMSOA 2007*.
5. A. Barros, M. Dumas, and P. Oaks. A Critical Overview of WS-CDL. *BPTrends*, 3(3), 2005.
6. A. Barros, M. Dumas, and A. ter Hofstede. Service Interaction Patterns. In *BPM 2005*, LNCS, pages 302–318, Nancy, France, 2005. Springer Verlag.
7. 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.
8. G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS 2007*.
9. G. Decker, H. Overdick, and J. M. Zaha. On the Suitability of WS-CDL for Choreography Modeling. In *EMISA 2006*.
10. G. Decker and F. Puhlmann. Extending BPMN for Modeling Complex Choreographies. In *CoopIS 2007*.
11. G. Decker and M. Weske. Local Enforceability in Interaction Petri Nets. In *BPM 2007*.
12. R. Dijkman and M. Dumas. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*, 13(4):337–368, 2004.
13. R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.
14. ITU-T. Message Sequence Chart. Recommendation Z.120, ITU-T, 2000.
15. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report, 2005.
16. A. Knopfel, B. Grone, and P. Tabeling. *Fundamental Modeling Concepts: Effective Communication of IT Systems*. Wiley, May 2006.
17. J. Mendling and M. Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *OTM 2005 Workshops*.
18. 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.
19. K. Pfitzner, G. Decker, O. Kopp, and F. Leymann. Web Service Choreography Configurations for BPMN. In *WESOA 2007*.
20. J. Recker and J. Mendling. On the Translation between BPMN and BPEL: Conceptual Mismatch between Process Modeling Languages. In *EMMSAD 2006*.
21. W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
22. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer, 2007.
23. J. M. Zaha, A. Barros, M. Dumas, and A. ter Hofstede. Let's Dance: A Language for Service Behavior Modeling. In *CoopIS 2006*.