Extending BPMN for Modeling Complex Choreographies

Gero Decker and Frank Puhlmann

Business Process Technology Group
Hasso Plattner Institut for IT Systems Engineering
University of Potsdam
D-14482 Potsdam, Germany
{gero.decker,frank.puhlmann}@hpi.uni-potsdam.de

Abstract. Capturing the interaction behavior between two or more business parties has major importance in the context of business-to-business (B2B) process integration. The Business Process Modeling Notation (BPMN), being the de-facto standard for modeling intra-organizational processes, also includes capabilities for describing cross-organizational collaboration. However, as this paper will show, BPMN fails to capture advanced choreography scenarios. Therefore, this paper proposes extensions to broaden the applicability of BPMN. The proposal is validated using the Service Interaction Patterns.

1 Introduction

With the rise of service-oriented architectures (SOA [1]), business process definitions are more and more used as configuration artifacts for information systems. Services, being loosely coupled components described in a uniform way, ideally have such a granularity that they have business meaning. These services can be orchestrated in executable business processes, e.g. described in BPEL [2]. This enables an organization to quickly adapt to changing requirements and business environments. Especially in inter-organizational settings, interconnected business processes realized as services are at the center of attention. This calls for languages suited for describing the interaction behavior between the different services (a.k.a. the service choreography). Examples for such languages are Let's Dance [3] and WS-CDL [4]. Our aim is to use the popular Business Process Modeling Notation (BPMN [5]) as choreography language.

The Service Interaction Patterns [6] describe a set of recurrent choreography scenarios. They range from simple message exchanges to scenarios involving multiple participants and multiple message exchanges. These patterns can be used to evaluate choreography languages. Although the BPMN allows to define choreographies through a swimlane concept and a distinction between control flow and message flow, it only provides direct support for a limited set of the patterns. This papers discusses these deficiencies and increases the suitability of BPMN for choreography modeling by introducing language extensions.

The remainder of this paper is structured as follows. The next section will introduce a choreography example and assess BPMN for its pattern support. Section 3 gives an overview of the proposed extensions, before section 4 validates the extensions by investigating on their pattern support and section 5 further discusses our results. Section 6 reports on related work and finally section 7 concludes and gives an outlook to future work.

2 Assessment of BPMN

Figure 1 shows an auctioning scenario represented in BPMN. It involves three roles, namely seller, bidder and auctioning service. Every time a seller decides to initiate an auction, it sends an auction creation request to the auctioning service. This triggers the instantiation of the auctioning service's process. The auction is scheduled to start at a defined point in time. Once this moment is reached, different bids are received by the auctioning service. Bids are placed by different bidders and an individual bidder is allowed to place several bids. The latter allows a bidder to react on higher bids from other bidders. Once the auction is over, it is checked if at least one bid has been received. If this is not the case, the auctioning service informs the seller and the choreography instance (a.k.a. conversation) ends. Otherwise, the winning bidder is selected and the seller is informed about who won. Those bidders who were not successful are notified. The winning bidder is informed, which finally leads to payment and the delivery of the goods.

The figure illustrates the different participant behavior descriptions which are interconnected through message flow. We omitted the (required) BPMN end events due to space reasons. We represented the receipt of multiple bids via a loop marker in the "receive bid" task. The end of the auction is denoted with an intermediate timer event attached to the receive and send bid activities of the auctioning service and the bidder. We used the event-based gateway to route the sequence flows of the seller and the bidder according to the outcome of the auction. Furthermore, we used the multiple instances marker to represent the parallel emission of all sorry messages.

The resulting BPMN diagram captures the processes of each participant, the bidder, the auctioning service, as well as the seller. However, several aspects could not be captured.

- **Multiplicity of participants.** In our scenario, several bidders take part in one conversation. All bidders must conform to the same interaction behavior as depicted in the BPMN diagram. However, in addition to the mere fact that we have many bidders involved, we need to distinguish them: Only one bidder can win the auction. It is only her to receive the completion message, whereas the others receive sorry messages. It is only her to perform payment and to receive the product.
- **Correlation.** The auctioning service receives messages from different bidders. As we are dealing with an asynchronous setting, it is essential for a

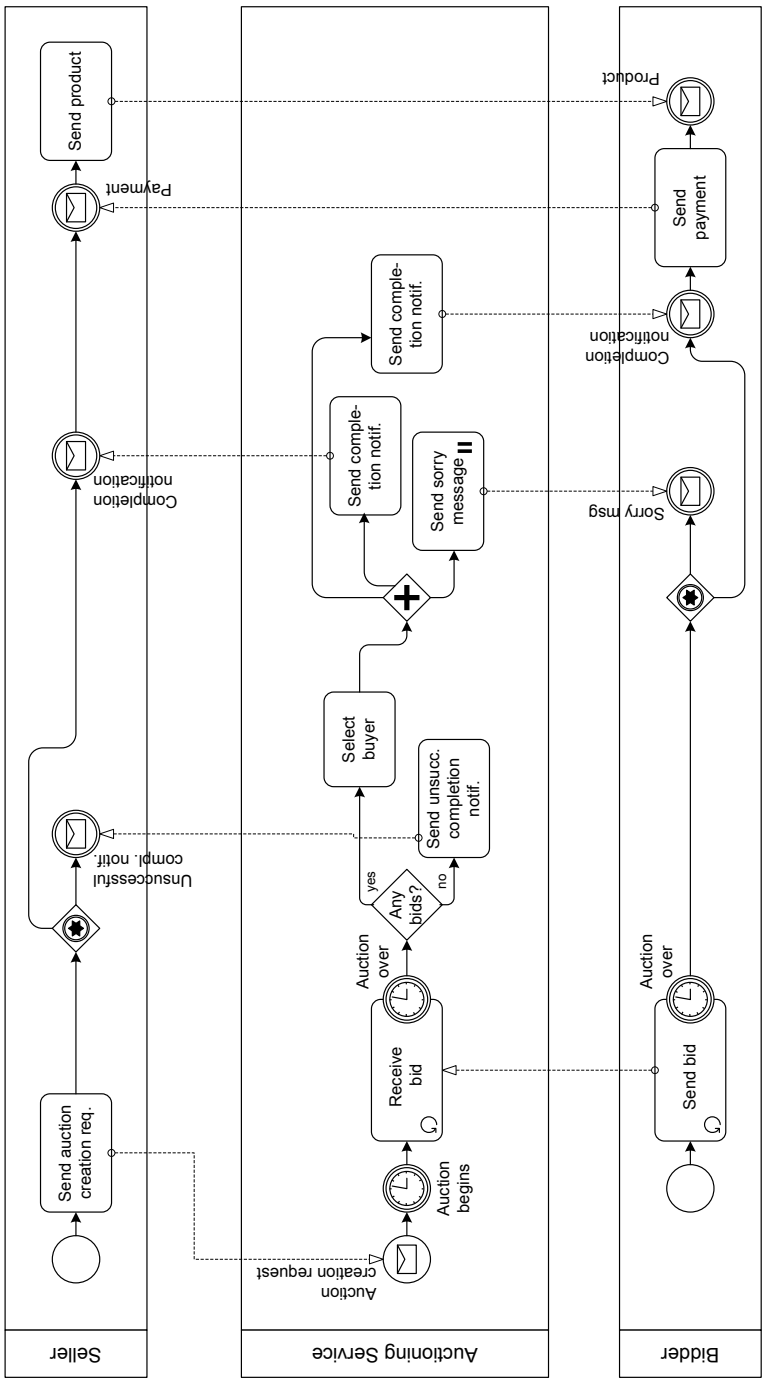


Fig. 1. BPMN choreography describing an auctioning scenario

participant to correlate messages exchanged with the same interaction partner. Imagine more complex sub-conversations between the auctioning service and the bidders. Here, the different sub-conversations with the different bidders need to be distinguished from each other.
- **Participant reference passing.** The winning bidder—the buyer—needs to contact the seller that is former unknown to her. To make this happen, she somehow needs to gain knowledge about the seller. Hence, the auctioning service passes the reference to the seller to her. In turn, the seller needs to make sure that she only accepts payment from the winning bidder. This can only be ensured if the auctioning service actually tells her who has won.

None of the these requirements can be properly represented in BPMN. This has an effect on BPMN's support for the Service Interaction Patterns as these requirements also appear in the set of patterns.

Three of the four "single-transmission multilateral interaction patterns" involve a set of participants, where the exact number might only be known at runtime. Therefore, BPMN does not support this group of patterns (except for *Racing Incoming Messages*, which is directly supported through the event-based gateway). The multiplicity problem also applies to *Contingent Request*, where a participant sends a request to another participant. If this participant does not respond within a given timeframe, a third participant is contacted. As the length of the list of potential recipients of request might not be known at design-time, BPMN does not support this pattern.

Request with Referral involves participant reference passing. Also *Relayed Request* might involve reference passing. Here, a participant A makes a request to a participant B who delegates it to yet another participant C. C subsequently interacts with A, while B observes a view of the interactions. As C might not know A in advance, B might need to send the reference of A when delegating the request. Therefore, both *Request with Referral* and *Relayed Request* are not supported in BPMN. In analogy to [7], we do not consider *Dynamic Routing* in this assessment as the pattern description is too imprecise.

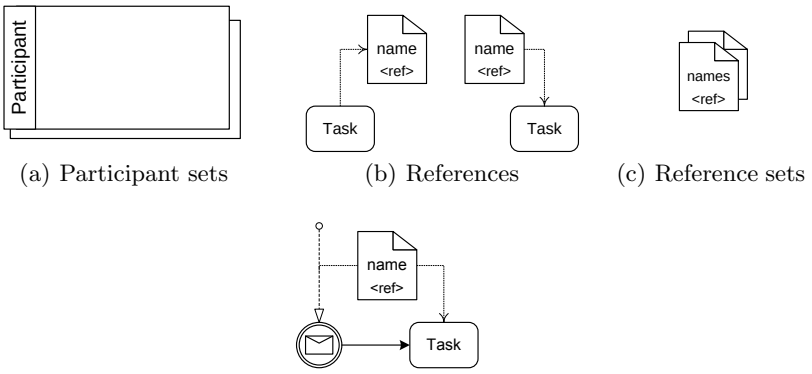
Only patterns *Send*, *Receive*, *Send/receive*, *Racing Incoming Messages* and *Multi-responses* are directly supported in BPMN. Therefore, we present BPMN extensions that overcome the illustrated issues in the next section.

3 BPMN Extensions

We introduce extension for the BPMN that allow the representation of multiple participants, correlations, and reference passing.

3.1 Participant Sets

Pools can represent individual participants in BPMN. As we have seen in the previous section we need to distinguish those cases where at most one participant of a particular participant type is involved in one conversation or if there



(a) Participant sets (b) References (c) Reference sets

(d) Reference passing

Fig. 2. BPMN extensions

can be potentially many participants involved. In our auctioning example, there is exactly one seller and one auctioning service involved in one conversation. However, we have potentially many bidders involved.

For representing multiple participants we introduce shadowed pools as new notational element, shown in figure 2(a). A set of participants of the same type involved in the same conversation is called a *participant set*.

3.2 References

The main challenge with participant sets is that we need to distinguish individual participants out of this set. We do this via references as shown in figure 2(b). A reference is a special data object enhanced with ⟨ref⟩. A reference can be connected to a flow object via associations. We give the following semantics to the different connection directions:

- A reference can be written by a flow object (represented by an association from the flow object to the reference). (i) If the flow object is a receive activity, e.g. an intermediate message event or an activity with incoming message flow, the reference will point to the sender upon message receipt. If the reference already pointed to a participant, the reference will simply be overwritten. (ii) If the flow object is not a receive activity, it is not specified what participant the reference will point to. Consider the selection of the buyer in our example.
- A reference can be read by a flow object (represented by an association from the reference to the flow object). (i) If the flow object is a send activity, the message will be sent to the participant the reference points to. In our example the auctioning service sends a completion notification to exactly that bidder out of the bidder set, who was selected to have won the auction. (ii) If the flow object is a receive activity, then a message is only awaited

from the defined participant. E.g. the seller only waits for payment from the buyer. (iii) If the flow object is neither a send nor a receive activity, it is not specified what happens with that reference inside the activity.

References cover those cases where an individual participant needs to be identified. However, we might need to select subsets of the participants involved in one conversation. In our example, this is the case for those bidders who did not win the auction. We need to send a sorry message to all of them—but we must not send this message to the winning bidder. We introduce reference sets as shown in figure 2(c) with the following semantics:

- A reference set can be modified by a flow object (represented through an association from the flow object to the reference set). (i) If the flow object is a receive activity, a reference to the sender of the message will be added to the reference set if such a reference is not already contained in the set. In our example we find a "receive bids" activity where bids from different bidders are received. In case a bidder who has already placed a bid in the same auction places another bid, no reference will be added to the set. However, if a new bidder takes part, a reference will be added. (ii) If the flow object is not a receive activity, it is not specified, what exactly happens with the set. It might be overwritten completely or references might be added, removed or changed.
- A reference set can be read by a flow object. (i) If the activity is a looped activity, i.e. a sequential loop or a multiple-instances activity, the reference set determines the number of repetitions or instances. This requires that at most one reference set serves as input for a looped activity. A special case is a looped send activity. Here, a message is sent to every of the referenced participants. In those cases, where the looped activity is a complex activity, a reference can be placed inside this activity which will represents the selected reference out of the set for a particular instance or repetition. (ii) If the activity is not a looped activity, it is not specified how the reference set is used within the activity. In our example, the "select buyer" activity takes the reference set as input and selects the winning bidder.

3.3 Reference Passing

References can be passed to other participants as shown in figure 2(d). The reference is connected to a message flow with an undirected association. The passed reference can be connected to other flow objects with directed associations. In figure 2(d), the passed reference is used in the task.

3.4 Example

The example from figure 1 is extended with the proposed extensions, shown in figure 3. First of all, a shadow was added to the pool of the bidder to represent a participant set. The "receive bid" task of the auctioning service collects

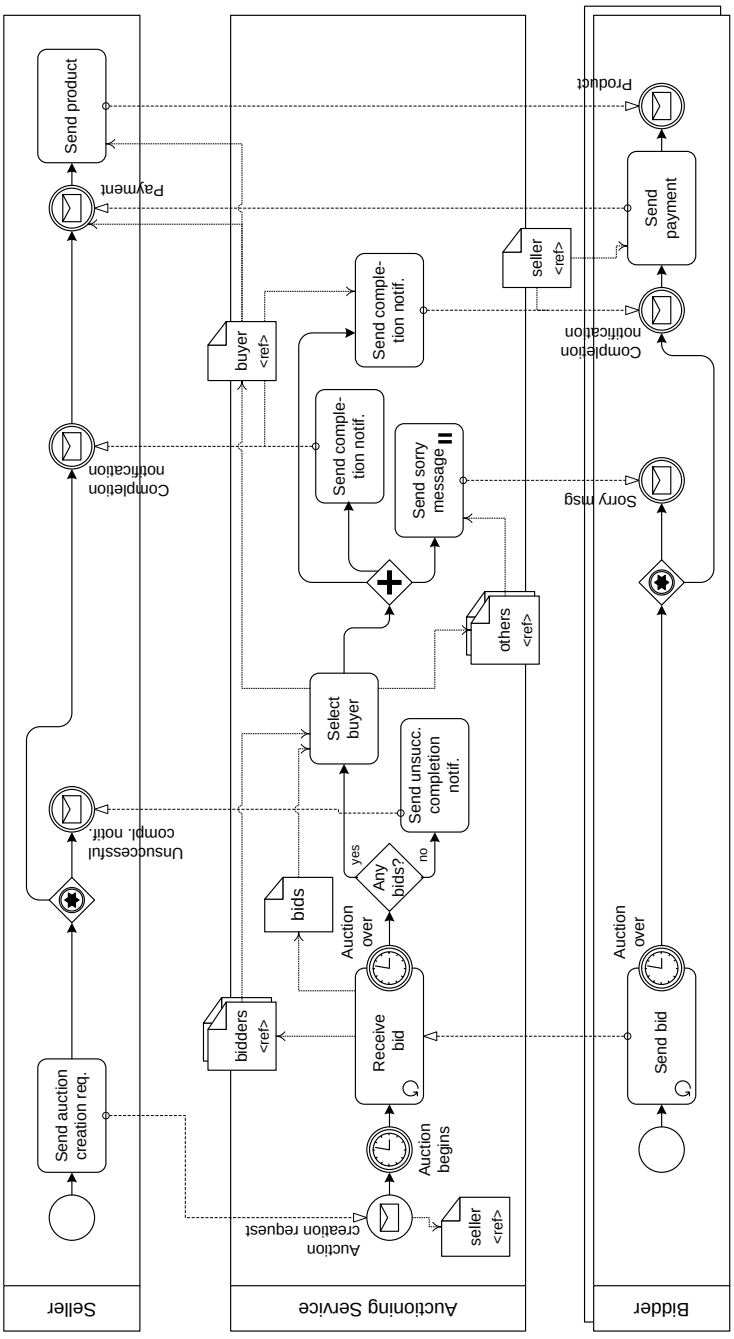


Fig. 3. The auctioning scenario represented using the extended BPMN

the references of the different bidders into a reference set. The reference set is forwarded to the "select buyer" task. Inside this task, the successful bidder is selected and placed into a new reference, denoted as buyer. The remaining references of the bidders reference set are placed into an others reference set. The others reference set is used as an input to the "send sorry message" task. Here, an instance is created for each element of the set. Hence, all unsuccessful bidders are notified. The buyer reference is forwarded to the "send completion notification" task, where it determines the instance of the bidder that should be contacted. Furthermore, it is passed to the seller, where it is used as an input for the reception of the payment as well as determining the reference of the bidder's instance to which the product should be sent. Finally, a reference of the seller is passed to the successful bidder. This reference is acquired implicitly via the initial interaction between the seller and the auctioning service.

4 Validation

This section validates the proposed BPMN extensions by investigating how the Service Interaction Patterns can be represented. It is notable that many of the patterns require multiple participants and/or dynamic binding of interaction partners via reference passing.

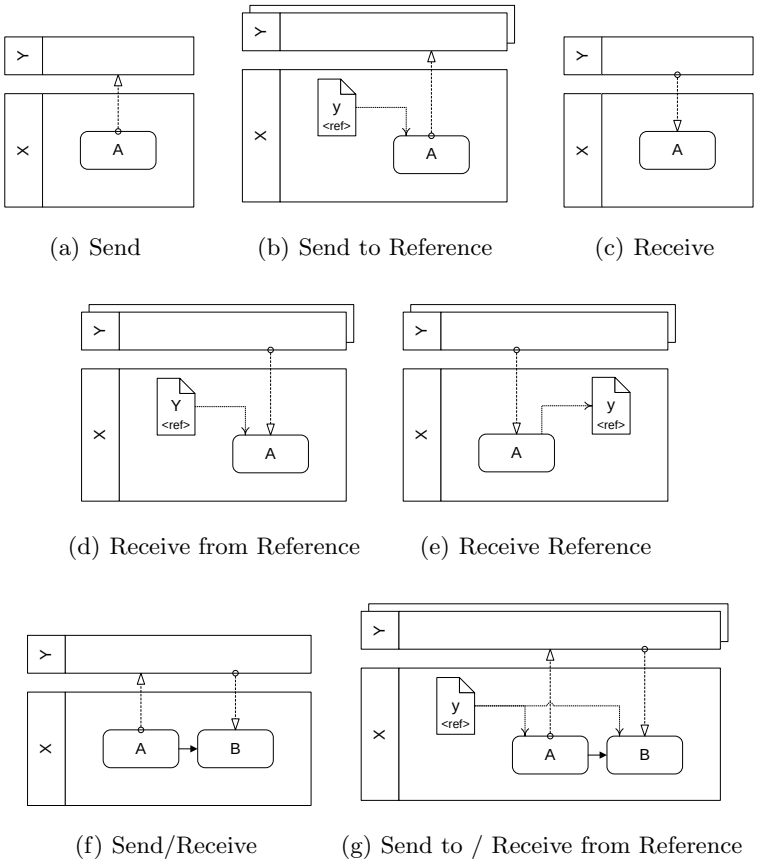
4.1 Single Transmission Bilateral Interaction Patterns

The single transmission bilateral interaction patterns represent basic interaction behavior. Graphical representations are shown in figure 4.

Send: A process sends a message to another process. The *Send* interaction pattern is depicted in figure 4(a). It is an assumption that the receiver gains knowledge about the reference of the requester. If the message flow is targeted at a participant set, the matching instance has to be determined via a reference, shown in figure 4(b).

Receive: A process receives a message from another process. The *Receive* interaction pattern is depicted in figure 4(c). According to the previous pattern, the receiver automatically gains knowledge about the reference of the requester. If the message should be received from a particular instance of a participant set, a reference according to figure 4(d) has to be used. If a message is received from an unspecified instance of the participant set, the corresponding reference can be collected, shown in figure 4(e).

Send/Receive: A process X engages in two causally related interactions. In the first interaction X sends a message to another process Y (the request), while in the second one X receives a message from Y (the response). A combined send/receive interaction is shown in figure 4(f). Once again, due to a one to one multiplicity, the correlation between requester and provider is evident. If the interaction partner is a certain instance of a participant set, a reference according to figure 4(g) has to be used.



(a) Send (b) Send to Reference (c) Receive

(d) Receive from Reference (e) Receive Reference

(f) Send/Receive (g) Send to / Receive from Reference

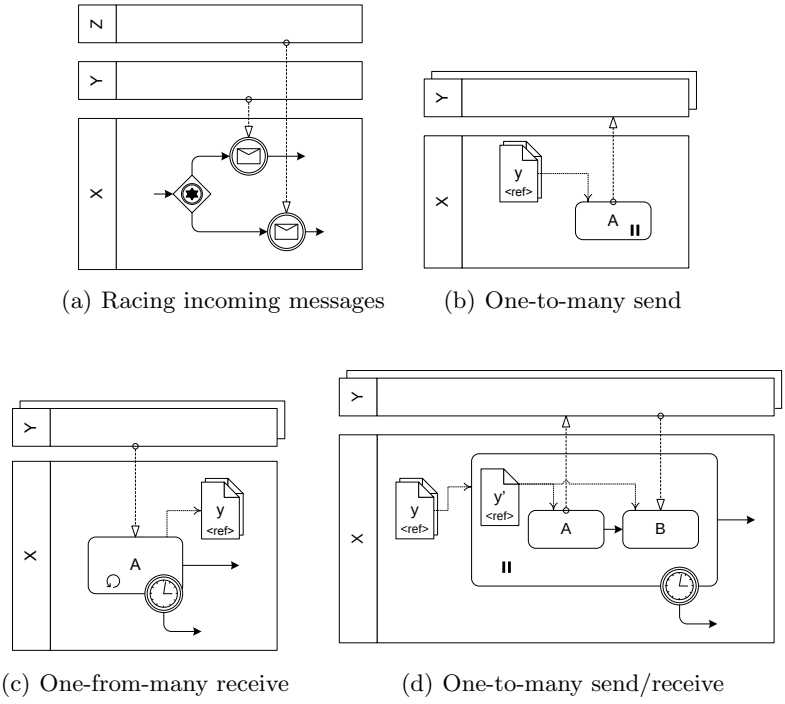
Fig. 4. Single transmission bilateral interaction patterns

4.2 Single Transmission Multilateral Interaction Patterns

The single transmission multilateral interaction patterns represent one to many or many to one interactions. Graphical representations are shown in figure 5.

Racing Incoming Messages: A process expects to receive one among a set of messages. These messages may be structurally different (i.e. different types) and may come from different categories of processes. The way a message is processed depends on its type and/or the category of processes from which it comes. Figure 5(a) shows the solution to this pattern. If several instances of a participant set should be used instead of Y and Z, a single receive task is sufficient.

One-to-many Send: A process sends messages to several other processes. The messages all have the same type (although their contents may differ). This pattern is depicted in figure 5(b). The multiple instance task A sends a message to



(a) Racing incoming messages (b) One-to-many send

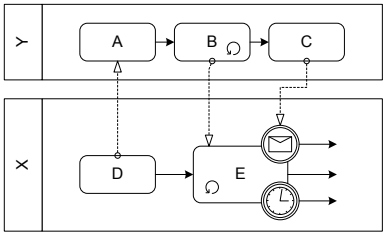
(c) One-from-many receive (d) One-to-many send/receive

Fig. 5. Single transmission multilateral interaction patterns

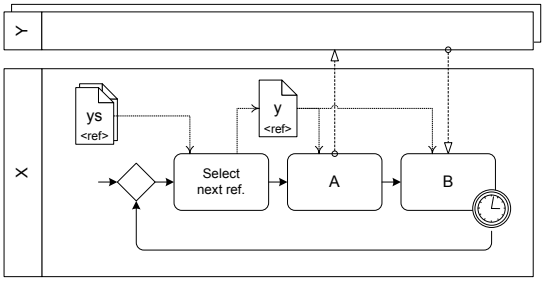
each reference contained in the reference set. We assume that all participants referenced are of the same type.

One-from-many Receive: A process receives a number of logically related messages that arise from autonomous events occurring at different processes. The arrival of messages needs to be timely so that they can be correlated as a single logical request. The one-from-many receive pattern is shown in figure 5(c). The references of the senders are collected in a reference set created in the loop-type task A. If enough messages have been gathered (decided internally inside A), the standard outgoing sequence flow is activated. If instead a timeout occurred, the interaction failed.

On-to-many send/receive: A process sends a request to several other processes, which may all be identical or logical related. Responses are expected within a given timeframe. However, some responses may not arrive within the timeframe and some processes may even not respond at all. The *One-to-many Send/receive* pattern is shown in figure 5(d). The associated reference set points to the participants that should be included. Like in the preceding pattern, also in this pattern the task B decides if enough responses have been gathered in the given timeframe. The figure includes a reference *y'* used within the sub-process. This



(a) Multi-responses



(b) Contingent requests

Fig. 6. Multi transmission interaction patterns

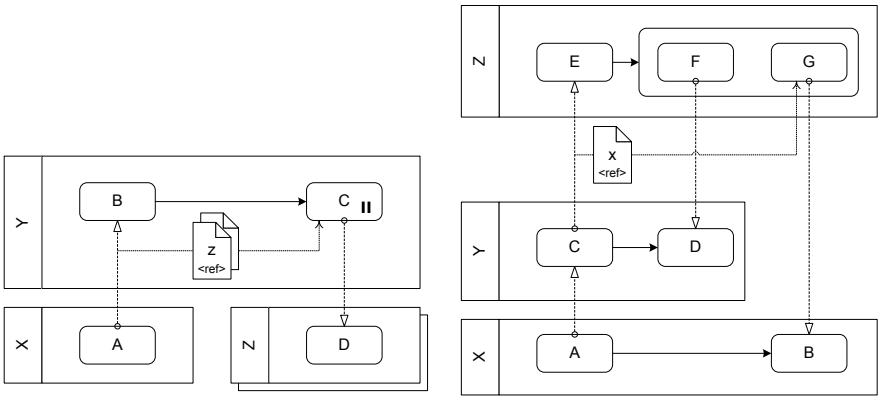
reference is to be filled for every instance that is spawned, as already mentioned in section 3.2.

4.3 Multi Transmission Interaction Patterns

The multi transmission interaction patterns represent many to many interactions. Graphical representations are shown in figure 6.

Multi-responses: A process X sends a request to another process Y. Subsequently, X receives any number of responses from Y until no further responses are required. The trigger of no further responses can arise from a temporal condition or message content, and can arise from either X or Y's side. This pattern is depicted in figure 6(a). The task D of X sends an initial request to task A of Y. Task B of Y responds until they are no more responses. Task E in X receives the responses until (1) a timeout occurs, (2) E decides to have gathered enough responses, or (3) a stop messages arrives from Y.

Contingent Requests: A process X makes a request to another party Y. If X does not receive a response within a certain timeframe, X alternatively sends a request to another process Z, and so on. This pattern is shown in figure 6(b). Initially, a reference set is passed to a task that selects a certain reference out of the set. The



(a) Request with Referral (b) Relayed Request

Fig. 7. Routing patterns

downstream task A receives this reference and initiates a request. Task B tries to receive the response. If no response is received in the given timeframe, another reference out of the reference set is selected and processed as described. What cannot be captured with our extensions, however, is the reception of messages from previous requests that failed due to a timeout.

Atomic Multicast Notification: A process sends notifications to several parties such that a certain number of parties are required to accept the notification within a certain timeframe. This pattern requires transactional behavior spanning multiple processes. Transactions are included in BPMN, however, they must only be applied within one process. Distributed transactions are not supported. Therefore, we can only provide a workaround for this pattern in our extended BPMN. It looks similar to *One-to-many Send/receive* with a completion condition at the notifying side.

4.4 Routing Patterns

The routing patterns describe flexible interaction behavior between a set of processes. Graphical representations are shown in figure 7.

Request with Referral: Process X sends a request to process Y indicating that any follow-up response should be sent to a number of other processes (Z1, Z2, . . . , Zn) depending on the evaluation of certain conditions. The solution to this pattern is shown in figure 7(a). It uses reference passing to denote the instances of Z that should receive the follow-up responses.

Pattern	BPMN	ext. BPMN
Send	+	+
Receive	+	+
Send/Receive	+	+
Racing Incoming Messages	+	+
One-to-many Send	-	+
One-from-many Receive	-	+
One-to-many Send/Receive	-	+
Multi-reponses	+	+
Contingent Request	-	+/-
Atomic Multicast Notification	-	-
Request with a Referral	-	+
Relayed Request	-	+

Table 1. BPMN vs. extended BPMN

Relayed Request: Process X makes a request to process Y which delegates the request to other processes (Z1, ..., Zn). Processes Z1, ..., Zn then continue interacting with process X while process Y observes a "view" of the interactions including faults. The interacting parties are aware of this "view". The *Relayed Request* pattern is shown in figure 7(b). While participant Z has immediate knowledge of Y, it needs a reference to participant X. This is received via reference passing from Y.

4.5 Validation Summary

A comparison on the supported Service Interaction Patterns for the standard BPMN as well as our proposed extension is shown in table 4.5. As already argued previously, we do not support *Atomic Multicast Notification* and did not consider *Dynamic Routing* in this assessment. *Contingent Requests* is also only partly supported, since (late) responses from earlier requests are ignored.

5 Discussion

Our proposals make heavy use of refined data objects. A major problem with BPMN data objects is that their semantics is not clearly defined in the BPMN specification. E.g. it is unclear what it means if different activities write on the same data object. Here, we simply assume that if an activity has write-access to a data object, it (might) overwrite the entire content of the data object upon completion. BPMN does not have the notion of collections or buffers, as they are present in UML Activity Diagrams [8]. Therefore, we introduced a distinction between simple data objects and data object sets, where we assume that write-access to a data object set typically means that the activity (might) add an object to the set. We do not require that data objects are only placed within pools or only accessed from within one pool. However, we have to leave a detailed

discussion on BPMN data objects and their semantics to future work.

The BPMN extensions presented in this paper are aligned with the work done on BPEL4Chor [9], an extension to abstract BPEL for modeling choreographies. This becomes evident in the semantics of references and reference sets. Awaiting messages from any sender vs. awaiting messages from a particular sender expressed through the absence or presence of a read-relationship between references and receive activities is analogous to the semantics of BPEL4Chor participant references that are either uninitialized or already set. Furthermore, the notion of adding references to a reference set in case a message is received from a sender that is not yet referenced in the set, is analogous to the notion of containment of a BPEL4Chor participant reference in a participant reference set. However, a detailed transformation of our extended BPMN to BPEL4Chor goes beyond the scope of this paper and must be left to future work.

References express correlation in those cases where receive activities read references. This defines who messages are to be received from. However, this notion of correlation only covers a limited set of scenarios. Imagine settings, where the same pair of participants engage in different parallel conversations. Here, our notion of references is not sufficient to distinguish the different conversations. Furthermore, it might be important to specify what message parts correlation is actually based on. E.g. a customer id or a shipment invoice number might be used as concrete correlation identifiers. There might be even more sophisticated correlation mechanisms needed, such as ranges of values or time-based correlation of messages. [10] provides a set of correlation patterns that might be a starting point for further refinements for correlation support.

In this paper we have left BPMN unchanged as much as possible while providing increased support for the Service Interaction Patterns. However, there is a general discussion whether the *interconnection modeling* approach, as it is the case for BPMN, is suited for choreography modeling at all. We have seen that we basically define control flow on a per-participant basis. Corresponding send and receive activities are connected through a message flow, jointly representing interactions.

An alternative to this approach is *interaction modeling*, where atomic interactions are the basic building blocks and control and data flow is defined between them. The main advantage of this approach is that incompatibility between different participants cannot occur in choreography models. It also reduces the number of modeling elements for representing a certain choreography. This increases modeling speed and helps to keep the models readable. Control and data flow dependencies are defined from a global perspective in the sense that (for most constructs) it does not need to be expressed explicitly, to what particular participant it actually belongs. Techniques for generating participant behavior descriptions out of the interaction model then care about which participant actually has to enforce a certain dependency later on.

Sometimes it is not possible to generate participant behavior descriptions such that all dependencies in the choreography are collectively enforced by them without adding synchronization interactions. Such choreographies are called *locally unenforceable*. For details please refer to [11] and [12]. A detailed comparison between interconnection models and interaction models goes beyond the scope of this paper and needs to be discussed in future work.

6 Related Work

BPMN enjoys widespread use in both industry and academia. [13] delivers an assessment of BPMN regarding its support for the Workflow Patterns [14] as well as its capabilities for the data and resource allocation perspective. However, this assessment does not include the Service Interaction Patterns.

A range of languages where introduced for modeling choreographies. BPEL4-Chor [9] adds a thin layer on top of abstract BPEL, interconnecting the different participant behavior descriptions. Let's Dance [3] and WS-CDL [4] follow the interaction modeling approach as described in the previous section. Like BPMN, Let's Dance is mainly targeted at business analysts and comes with a graphical notation. WS-CDL is tightly linked to other web services standards such as WSDL. Both languages have been assessed for their Service Interaction Pattern support (cf. [7]). It turns out that Let's Dance directly supports most patterns. WS-CDL is a little less suited for choreography modeling as it only comes with limited support for expressing those scenarios where multiple participants of the same type are involved in a conversation and the actual number of participants is only known at runtime. Event-driven Process Chains (EPC) is another widely-used process modeling language. In [15] extensions for inter-organizational process modeling are proposed. However, there has not been an assessment using the Service Interaction Patterns regarding their suitability.

There has also been work on mapping (subsets of) BPMN to formalisms. Dijkman et al. present a mapping to Petri nets in [16], enabling the verification of soundness and liveless. However, the formalization does not include message flows. Therefore, reasoning on choreographies is out of scope of their work. Wong et al. present another formalization of BPMN based on Communicating Sequential Processes (CSP) in [17]. In [18] they then show how compatibility checking can be carried out for BPMN choreographies. Other approaches for compatibility checking in choreographies are introduced by Martens [19], Puhlmann et al. [20] and Massuthe et al. [21]. A general introduction into the different viewpoints of choreographies can be found in [22].

7 Conclusion

In this paper we have identified weaknesses of BPMN regarding its suitability for choreography modeling. We based our assessment on the Service Interaction Patterns and concluded that there is direct support for only five out of the twelve

patterns considered. We then proposed extensions to overcome these limitations and validated the extended BPMN with the patterns.

In future work we are going to introduce a formal mapping for the new concepts. This enables the verification of complex choreographies, including compatibility and conformance checking. In [23] we have already shown that name creation and restriction in π-calculus are useful concepts for formalizing choreographies. Therefore, we consider using π-calculus or a Petri net version enhanced with a name concept, e.g. similar to ν-nets as presented in [24], as formal basis. The latter would enable us to reuse and extend the Petri-nets-mapping in [16].

References

1. Burbeck, S.: The Tao of E-Business Services (2000)
2. Fallside, D.C., Walmsley, P.: Web Services Business Process Execution Language Version 2.0. Technical report (2005) `http://www.oasis-open.org/apps/org/workgroup/wsbpel/`.
3. Zaha, J.M., Barros, A., Dumas, M., ter Hofstede, A.: A Language for Service Behavior Modeling. In: Proceedings 14th International Conference on Cooperative Information Systems (CoopIS 2006), Montpellier, France, Springer Verlag (2006)
4. Kavantzas, N., Burdett, D., Ritzinger, G., Lafon, Y.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation. Technical report (2005) `http://www.w3.org/TR/ws-cdl-10`.
5. OMG.org: Business Process Modeling Notation. 1.0 edn. (2006)
6. Barros, A., Dumas, M., Hofstede, A.: Service Interaction Patterns. In Aalst, W., Benatallah, B., Casati, F., eds.: Business Process Management, volume 3649 of LNCS, Berlin, Springer Verlag (2005) 302–318
7. Decker, G., Overdick, H., Zaha, J.M.: On the Suitability of WS-CDL for Choreography Modeling. In: Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen (EMISA 2006), Hamburg, Germany (2006)
8. : UML 2.0 Superstructure Specification. Technical report, Object Management Group (OMG) (2005)
9. Decker, G., Kopp, O., Leymann, F., Weske, M.: BPEL4chor: Extending BPEL for Modeling Choreographies. In: Proceedings International Conference on Web Services (ICWS). (2007)
10. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE), Braga, Portugal (2007)
11. Zaha, J.M., Dumas, M., ter Hofstede, A., Barros, A., Decker, G.: Service Interaction Modeling: Bridging Global and Local Views. In: Proceedings 10th IEEE International EDOC Conference (EDOC 2006), Hong Kong (2006)
12. Decker, G., Weske, M.: Local Enforceability in Interaction Petri Nets. In: Proceedings 5th International Conference on Business Process Management (BPM 2007). LNCS, Brisbane, Australia (2007)
13. Wohed, P., van der Aalst, W.M., Dumas, M., ter Hofstede, A., Russell, N.: On the Suitability of BPMN for Business Process Modelling. In: Proceedings 4th International Conference on Business Process Management (BPM 2006). LNCS, Vienna, Austria, Springer Verlag (2006)

14. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.P.: Workflow Patterns. Distributed and Parallel Databases **14** (2003) 5–51
15. Seel, C., Vanderhaeghen, D.: Meta-model based extensions of the epc for inter-organisational process modelling. In: Proceedings 4th GI-Workshop EPK 2005 - Geschäftsprozessmanagement. (2005)
16. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. Preprint 7115, Queensland University of Technology, Brisbane, Australia (2007)
17. Wong, P.Y., Gibbons, J.: A process semantics for BPMN. Technical report, Oxford University Computing Laboratory (2007) `http://web.comlab.ox.ac.uk/oucl/work/peter.wong/pub/bpmnsem.pdf`.
18. Wong, P.Y., Gibbons, J.: Verifying business process compatibility. In: Proceedings 3rd International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord'07), Paphos, Cyprus (2007)
19. Martens, A.: Analyzing Web Service based Business Processes. In Cerioli, M., ed.: Proceedings of Intl. Conference on Fundamental Approaches to Software Engineering (FASE'05), Part of the 2005 European Joint Conferences on Theory and Practice of Software (ETAPS'05). Volume 3442 of Lecture Notes in Computer Science., Edinburgh, Scotland, Springer-Verlag (2005)
20. Puhlmann, F., Weske, M.: Interaction Soundness for Service Orchestrations. In Dam, A., Lamersdorf, W., eds.: Service-Oriented Computing – ICSOC 2006, volume 4294 of LNCS, Berlin, Springer Verlag (2006) 302–313
21. Massuthe, P., Reisig, W., Schmidt, K.: An Operating Guideline Approach to the SOA. Annals of Mathematics, Computing & Teleinformatics **1** (2005) 35–43
22. Dijkman, R., Dumas, M.: Service-oriented Design: A Multi-viewpoint Approach. International Journal of Cooperative Information Systems **13** (2004) 337–368
23. Decker, G., Puhlmann, F., Weske, M.: Formalizing Service Interactions. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Business Process Management, volume 4102 of LNCS, Berlin, Springer Verlag (2006) 414–419
24. Rosa-Velardo, F., de Frutos-Escrig, D.: Name creation vs. replication in petri net systems. In: Proceedings 28th International Conference on Application and Theory of Petri Nets and other Models of Concurrency (Petri Nets 2007), Siedlce, Poland (2007)