Enhancing BPMN 2.0 Support for Service Interaction Patterns

Dario Campagna, Carlos Kavka and Luka Onesti
Research and Development Department, ESTECO SPA, Area Science Park, Padriciano 99, Trieste, Italy

Keywords: BPMN 2.0, Service Interaction Patterns.

Abstract: Choreography modeling languages have emerged in the past years as a mean for capturing and managing collaborative processes. The advancement of such languages let to the definition of the service interaction patterns, a pattern-based framework for the benchmarking of choreography languages against abstracted forms of representative scenarios. Service interaction patterns have been used to analyze the capabilities of different languages. Since its introduction, no benchmark based on this framework has been performed on the Business Process Model and Notation (BPMN) version 2.0. In this paper, we present an assessment of BPMN 2.0 support for service interaction patterns. We evidence the issues that limit the set of supported patterns, and propose enhancements to overcome them.

1 INTRODUCTION

In the past years there has been much activity in developing languages for Business Process Management systems. In particular, languages suited for describing interaction behavior between different services, i.e., for modeling service choreography, have emerged as a key instrument for achieving integration of business applications in a service-oriented architecture (SOA) setting. Examples of such languages are Lets'Dance (Zaha et al., 2006), WS-CDL (W3C, 2005), and WS-BPEL (OASIS, 2007).

With the advancement of service choreography languages came the need for consolidated insights into the capability and exploitation of the resulting standard specifications and associated implementations in terms of business requirements. In 2005, Barros et al. concluded that for service-oriented architectures to move forward, it was necessary to shift from thinking in terms of request-response and buyer-seller-shipper interaction scenarios into addressing complex, large-scale, multi-party interactions in a systematic manner. They thus presented in (Barros et al., 2005b) a set of thirteen patterns of service interactions, the *service interaction patterns*. These patterns aim to contribute to the gathering of requirements needed to shed light into the nature of service interactions in collaborative business processes, where a number of parties, each with its own internal processes, need to interact with one another according to certain pre-agreed rules (Barros et al., 2005b). The patterns capture different peculiar characteristics of such collaborative processes. The number of involved parties may be in the order of tens or even hundreds, and thus the nature of interactions is rarely only bilateral but rather multilateral. Furthermore, the assumption of strict synchronization of all responses before the next steps in a process breaks down due to the independence of the parties. More realistically, responses are accepted as they arrive. Also, while it is conventional to think of multi-cast interactions as a party sending a request to several other parties, the reverse may also apply, several parties send messages from autonomous events to a party which correlates these into a single request. Finally, not all interactions in dynamic marketplaces follow a requester-respondent-requester structure. Rather, a sender may re-direct interactions to nominated delegates. Receivers may outsource requests, choosing to "stay in the loop" and observe parts of responses. More generally, it may only be possibly to determine the order of interaction at run-time, given, for example, the content of messages passed.

The collected service interaction patterns have been derived and extrapolated from insights into real-scale B2B transaction processing, use cases gathered by standardization committees, generic scenarios identified in industry standards, and case studies reported in the literature. The patterns consolidate recurrent scenarios and abstract them in a way that provides reusable knowledge. They range from simple message exchanges to scenarios involving multi-

Campagna D., Kavka C. and Onesti L..
Enhancing BPMN 2.0 Support for Service Interaction Patterns.
DOI: 10.5220/0004989901990208
In *Proceedings of the 9th International Conference on Software Engineering and Applications* (ICSOFT-EA-2014), pages 199-208
ISBN: 978-989-758-036-9

ple participants and multiple message exchanges. On the one hand, the service interaction patterns consolidate the nature of service interactions through generalized functional classification. On the other hand, they clear the track for further and ongoing extensions. These patterns allow the assessment of web services standards, and the benchmarking of choreography and orchestration languages, making it possible for SOA technologies to progress further (Barros et al., 2005b).

Since their introduction, the service interaction patterns have been used to evaluate different choreography languages. In this paper, we focus on the latest version of the Business Process Model and Notation, i.e., BPMN version 2.0 (OMG, 2011), and present an assessment of BPMN 2.0 collaboration diagrams support for the service interaction patterns.

The remainder of this paper is structured as follows. In Section 2 we recall some of the choreography language analysis based on the service interaction patterns. In Section 3, we evaluate BPMN 2.0 for its pattern support, and point out issues that limit the set of representable patterns. To overcome such issues, we propose in Section 4 a set of enhancements for BPMN 2.0. The paper conclusions are presented in Section 5.

2 RELATED WORK

WS-BPEL (OASIS, 2007) has been the first language to be analyzed in terms of service interaction patterns. In (Barros et al., 2005b; Barros et al., 2005a) the authors show that WS-BPEL directly supports *Single Transmission Bilateral Interaction Patterns*. For *Single Transmission Multilateral Interaction Patterns*, WS-BPEL imposes some restrictions to the *Send/Receive* pattern and requires “house-keeping” code for correlation and for capturing stop and success conditions. Of *Multi Transmission Interaction Patterns*, WS-BPEL provides support for two of the three patterns. Lack of sufficient transaction support compromises a WS-BPEL solution for *Atomic Multicast Notification*. All the *Routing Patterns* are supported with the exception of *Dynamic Routing*, which is outside the scope of WS-BPEL.

In (Decker and Puhlmann, 2007) the authors show that BPMN 1.0 directly supports only five of the thirteen service interaction patterns, and present extensions for BPMN 1.0 that allow the representation of multiple participants, correlation, and reference passing. They introduce the concept of *participant set* in order to represent a set of participants of the same type involved in the same conversation, and the concept of *reference* to distinguish individual participant out of a participant set. A reference is a special data object, it can be connected to flow objects via directed associations, and can be passed to other participants connecting it to message flows with undirected associations. Thanks to these extensions, the number of patterns supported by BPMN 1.0 increases to ten. *Contingent Request* is only partially supported, while *Dynamic Routing* is excluded from the analysis.

The BPMN 2.0 specification extends the scope and capabilities of BPMN 1.0 in several areas. Among other improvements, it describes the execution semantics for all BPMN elements, defines an extensibility mechanism for process model extensions, and defines a choreography model. BPMN 2.0 choreographies are evaluated in (Cortes-Cornax et al., 2011) by using an extended quality framework, which includes the service interaction patterns. Since the patterns only cover one perspective of the requirements for choreography definition languages, the framework also includes other perspectives paying special attention to graphical notations. The evaluation identifies a number of issues in BPMN 2.0 that affects the perceptual discriminability of certain choreography modeling constructs. To address these deficiencies, the authors propose the introduction of new concepts in choreography diagrams. Examples are the concept of channel annotations, message multiplicity for message flows, and annotations for message flows to indicate which participant initiates a conversation. In (Cortes-Cornax et al., 2012) the authors considered a precise analysis of the support of the service interaction patterns in BPMN 2.0 as an important future work. However, such a study is still missing.

3 PATTERN ANALYSIS

We present in this section an assessment of BPMN 2.0 support for the service interaction patterns introduced by Barros et al. in (Barros et al., 2005b). This section is organized by following the structure of (Barros et al., 2005b). For each pattern, we present its description and issues, and propose a BPMN 2.0 implementation. The implementations and their semantics are described in natural language. For most of the patterns, we include a BPMN 2.0 graphical representation of the implementation. We provide no formal validation of the proposed solutions, since the only complete BPMN 2.0 semantics specification is presented in (OMG, 2011) by using natural language.

As we will show, BPMN 2.0 directly supports the *Single Transmission Bilateral Interaction Patterns*, two of the three *Single Transmission Multilateral In-*

teraction Patterns, the *Multi-responses* pattern, and two of the three *Routing Patterns*. With the addition of a BPMN 2.0 extension for collaborations and message queuing, it is possible to support the *One-to-many Send/Receive* pattern and the *Contingent Requests* pattern too. The *Atomic Multicast Notification* pattern can only be partially supported. We excluded from this assessment the *Dynamic Routing* pattern since its description is too imprecise, as already noted in (Decker et al., 2006a; Decker and Puhlmann, 2007).

From now on, with the term *party* we indicate a BPMN 2.0 participant instance, and with the term *parties* we indicate a set of heterogeneous BPMN 2.0 participant instances, i.e., instances of one or more BPMN 2.0 participants.

3.1 Single Transmission Bilateral Interaction Patterns

Single transmission bilateral interaction patterns correspond to elementary interactions where a party sends (receives) a message, and as a result expects a reply (sends a reply). These patterns cover one-way and round-trip non-routed bilateral interactions.

3.1.1 Send

Description. A party X sends a message to another party.

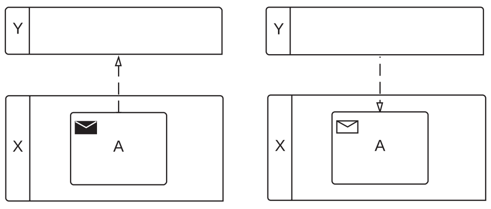
Issues. The counter-party may or may not be known at design time.

The *Send* interaction pattern can be modeled by using in a participant X a *send task* that sends a message to a participant Y, as shown in Figure 1(a). If participant Y has multiplicity greater than one (i.e., there may be more than one instance of Y in execution at the same time), then we can add to the sent message payload a reference for Y, and use context-based correlation in Y to route the message to the correct instance. It is assumed that the sender gains knowledge about the receiver reference and stores it in, e.g., a data object.

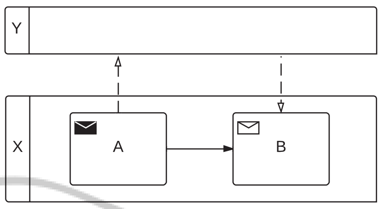
3.1.2 Receive

Description. A party X receives a message from another party.

The *Receive:* interaction pattern can be modeled by using in a participant X a *receive task* that receives a message from a participant Y, as shown in Figure 1(b). If Y has multiplicity greater than one, then we can use context-based correlation in X to accept only messages from a particular instance of Y.



(a) Send pattern. (b) Receive pattern.



(c) Send/Receive pattern.

Figure 1: Single transmission bilateral interaction patterns.

3.1.3 Send/Receive

Description. A party X engages in two casually related interactions: in the first interaction X sends a message to another party Y, while in the second one X receives a message from Y.

Issues. The counter-party may or may not be known in advance. The outgoing and incoming messages must be correlated.

The *Send/Receive* interaction pattern is depicted in Figure 1(c). It can be modeled with a *send task* followed by a *receive task* in a participant X. The former task sends a message to a participant Y, the latter receives a message from Y. If Y has multiplicity greater than one, then we can make use of context-based correlation for communicating with the desired instance of Y, and take advantage of key-based correlation to correlate outgoing and incoming messages in X.

3.2 Single Transmission Multilateral Interaction Patterns

Single transmission multilateral interaction patterns cover non-routed interactions where a party may send or receive multiple messages, but as part of different interaction threads dedicated to different parties.

3.2.1 Racing Incoming Messages

Description. A party X expects to receive one among a set of messages. Messages may be structurally different and may come from different parties. The way a message is processed depends on its type and/or the party from which it comes.

The *Racing Incoming Messages:* interaction pattern can be modeled by using in a participant X an *event based gateway* connected to *catch message events*, as depicted in Figure 2. Each catch message event receives messages of a certain type, or from a particular participant.

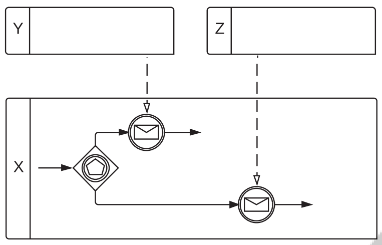


Figure 2: Racing Incoming Messages pattern.

3.2.2 One-to-many Send

Description. A party X sends a message to several other parties. All the messages have the same type (although their contents may differ).

Issues. The number of parties to whom the message is sent may or may not be known at design time.

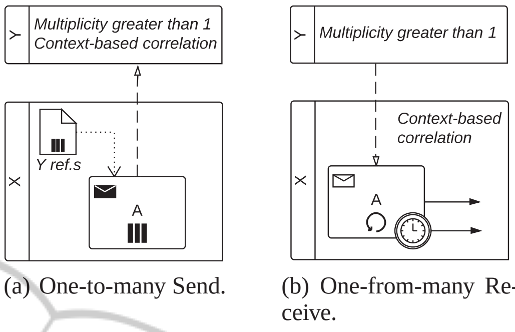
Under the assumption that receiving parties are instances of a single participant, this pattern can be thought as variant of the *Send* pattern when participant Y has multiplicity greater than one. The pattern can be modeled as shown in Figure 3(a). A *parallel multi-instance send task A* in participant X receives as input a *data object collection* containing references of participant Y instances, and sends a message to each of them. Context-based correlation can be used in Y in order to route messages to the correct instances.

3.2.3 One-from-many Receive

Description. A party X receives several logically related messages arising from autonomous events occurring at different parties. The arrival of messages needs to be timely so that they can be correlated as a single logical request.

Issues. Since messages originate from autonomous parties, a mechanism is needed to determine which incoming messages should be grouped together.

Under the assumption that sending parties are instances of a single participant, this pattern can be viewed as a variant of the *Receive* pattern when the sending participant Y has multiplicity greater than one. The pattern can be modeled as depicted in Figure 3(b). A *loop receive task A* with an *interrupting boundary timer event* is used in participant X to receive messages from participant Y instances. Context-based correlation can be used in X to accept only messages from certain instances of participant Y.


Y
Multiplicity greater than 1
Context-based correlation
X
Y ref.s
A
Y
Multiplicity greater than 1
Context-based correlation
X
A

(a) One-to-many Send. (b) One-from-many Receive.

Figure 3: One-to-many Send pattern and One-from-many Receive pattern.

3.2.4 One-to-many Send/Receive

Description. A party X sends a request to several other parties, which may be all identical or logically related. Responses are expected within a given time-frame. However, some responses may not arrive within the time-frame and some parties may even not respond at all.

Issues. The number of parties to which messages are sent may or may not be known at design time. Responses need to be correlated to their corresponding requests.

A BPMN 2.0 representation of this pattern is shown in Figure 4. We use in participant X a *multi-instance sub-process* with an *interrupting boundary timer event*, and whose *loop data input* is a *data object collection* containing references to instances of a participant Y. The sub-process contains a *send task* followed by a *receive task*. Each instance of the sub-process sends a message to an instance of Y (context-based correlation is used in Y), and then waits for a response. Responses could be correlated to their corresponding request by using key-based correlation in X. However, BPMN 2.0 correlation works at process instance level, i.e., we can only correlate a message to a specific instance of a process. To support this pattern we need to correlate received messages to a particular instance of the sub-process, and this is not possible in BPMN 2.0. To overcome this limitation, we propose a BPMN 2.0 extension for collaboration/conversations, and a modification of message correlation semantics, which will be described in Section 4.

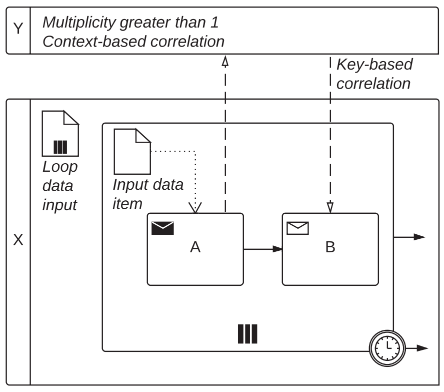


Figure 4: One-to-many Send/Receive pattern.

3.3 Multi Transmission Interaction Patterns

Multi transmission interaction patterns are dedicated to non-routed interactions in which a party sends (receives) messages to (from) the same party.

3.3.1 Multi-responses

Description. Party X sends a request to party Y, then X receives any number of responses from Y until no further responses are required. The trigger of no further responses can rise from a temporal condition, or be based on message content, which in both cases can rise from either X or Y.

Figure 5 depicts a possible representation of this pattern in BPMN 2.0. Participant X sends a message to participant Y by using the *send task D*. Such message is received in Y by the *receive task A*. Then, Y sends messages to X by using the *loop send task B*. These messages are received in X by the *loop receive task E*. X stops receiving message as soon as either the *interrupting boundary timer event* of E is triggered, or E loop condition evaluates to false, or a message sent by Y (by using the *send task C*) reaches the *interrupting boundary catch message event* of E.

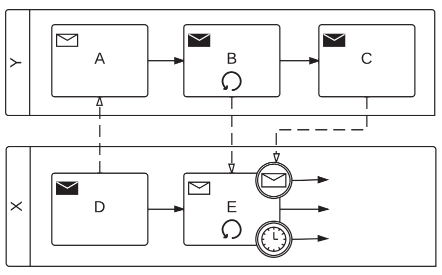


Figure 5: Multi-responses pattern.

3.3.2 Contingent Requests

Description. Party X makes a request to another party Y. If X does not receive a response within a certain time-frame, X sends a request to another party Z, and so on.

Issues. After a contingency request has been issued, it may be possible that a response arrives (late) from a previous request.

Figure 6 depicts a possible representation of the pattern (we assume that responding parties are instances of the same participant). First, a *task* in X selects a reference to an instance of Y from a *data object collection*. Then, the *send task A* sends to the selected Y instance a message (context-based correlation is used in Y). Finally, the *receive task B* waits for a response from Y. Context-based correlation is used in X to accept only messages containing the selected Y instance reference in their payloads. If no response is received before the *interrupting timer boundary event* is triggered, then another Y instance reference is selected and processed as described. Responses that arrive late from previous requests are discarded thanks to context-based correlation.

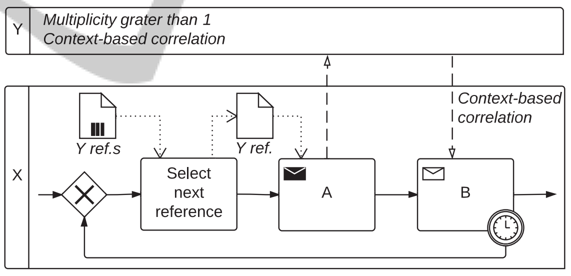

Y
Multiplicity grater than 1
Context-based correlation
Context-based correlation
X
Y ref.s
Y ref.
Select next reference
A
B

Figure 6: Contingent Requests pattern, solution (1).

The just described implementation of the pattern exploits one of the three available solutions to handle the late response issue. The first solution (1) is to disallow late arrivals altogether, and receive only the response of the current request. Another solution (2) is to accept the first response even if it is late and stop outstanding requests. The last solution (3) is to accept the first arriving response, trigger the end of outstanding request, but receive any further response that arrives (before X terminates). The pattern does not predispose which of the three solutions prevails. Solution (1) is the one adopted in Figure 6.

To support solution (2) we modify the workflow in Figure 6 adding to it a *data object* and a *task C*. The resulting workflow is shown in Figure 7. We use the new data object for context-based correlation in X. We initialize this data object with some value, and send this value in the payload of messages sent to Y

instances. Only messages from Y instances containing the chosen value in their payload are accepted in X. As soon as a response is received by B, the task C executes and changes the value of the new data object. Any other message coming from Y instances will then be discarded by context-based correlation. With the workflow in Figure 7 we accept late responses, but we may lose messages that arrive after the interrupting boundary timer event has been triggered and before the activation of B. Hence, the response we receive in X may not be the first sent from Y. To avoid losing responses, we propose a modification to the message semantics. This modification will be described in Section 4.2.

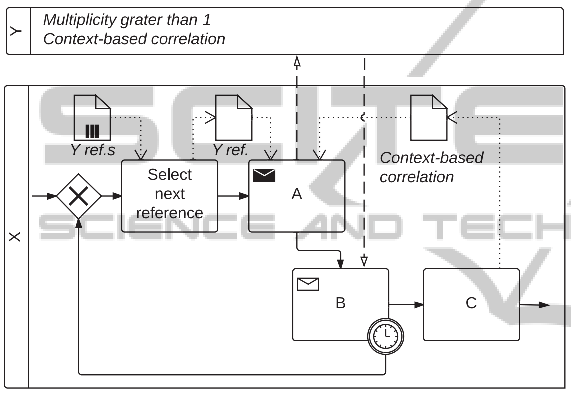

Multiplicity grater than 1
Context-based correlation
Y ref.s
Y ref.
Select next reference
A
Context-based correlation
B
C

Figure 7: Contingent Requests pattern, solution (2).

Solution (3) requires deep changes to the BPMN 2.0 message correlation mechanism in order to be implemented. A precise analysis of such changes is out of the scope of this assessment.

3.3.3 Atomic Multicast Notification

Description. A party sends notifications to several parties such that a certain number of parties are required to accept the notification within a certain time-frame. For example, all parties or just one party are required to accept the notification. In general, the constraint for successful notification applies over a range between a minimum and maximum number.

Issues. The constraint that all parties should have received the notification, means that if any one party received the notification, all the other parties also received it. Thus, some kind of transactional support is required.

The main issue of this pattern relates to atomic transactions. Atomic transactions have an all-or-nothing property: the actions taken by a transaction participant prior to commit are only tentative (typically they are neither persistent nor made visible outside the transaction); if all participants were able to execute successfully then transactions are committed; if a participant aborts or does not respond at all, all transactions are aborted. Web Service Atomic Transaction (OASIS, 2009) is an OASIS standard that defines protocols for atomic transactions, one of them is Two-Phase Commit (2PC). The 2PC protocol coordinates registered participants to reach a commit or abort decision, and ensures that all participants are informed of the final result.

BPMN 2.0 provides built-in support for business transaction through the notion of *transaction sub-process*. A sub-process marked as transactional means that its component activities must either all complete successfully or the sub-process must be restored to its original consistent state. However, business transactions are usually not ACID transactions coordinated via the 2PC protocol. The reason is they fail the isolation requirement. In order to isolate, or lock, the resource performing the component activities of the transaction, the transaction must be short-running, taking milliseconds to complete. For business transactions it is not possible to make that assumption. Business transactions are long-running, and the resources associated with their component tasks are not locked while the transaction is in progress. Instead, each activity in the transaction executes normally in its turn, but if the transaction as a whole fails to complete successfully, each of its activities that has completed already is undone by executing its defined compensating activity. Hence, BPMN 2.0 provides no support for atomic transactions, but different workarounds can be provided for the *Atomic Multicast Notification* pattern. These workarounds will be described in Section 4.3.

3.4 Routing Patterns

Routing patterns cover routed interactions, i.e., interactions involving transfers of party references.

3.4.1 Request with Referral

Description. Party X sends a request to party Y indicating that any follow-up should be sent to a number of other parties $(Z_1, Z_2, \dots, Z_n)$ depending on the evaluation of a certain condition.

Issues. Party Y may or may not have a prior knowledge of the identity of the other parties. The information transferred from X to Y must therefore allow Y to interact with the other parties.

This pattern can be represented in BPMN 2.0 as shown in Figure 8. A *data object collection* in participant X contains references to instances of participant Z that should receive the follow-ups (we assume that the referred parties are all instances of the same participant). The data object collection is transferred to participant Y through a message sent by the *send task A* in X. The *receive task B* in Y receives the message from A, and stores its payload (i.e., the collection of references) into a data object collection. Then, the *multi-instance send task C* in Y sends a message to each instance of Z referenced in the data object collection (context-based correlation is used in Z).

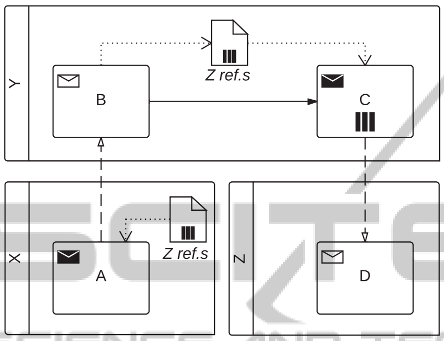


Figure 8: Request with Referral pattern.

3.4.2 Relayed Request

Description. Party X makes a request to party Y which delegates the request to other parties $(Z_1, Z_2, \ldots, Z_n)$. $Z_1, Z_2, \ldots, Z_n$ then continue interactions with X while Y observes a view of the interactions including faults.

Issues. The delegated parties $(Z_1, Z_2, \ldots, Z_n)$ may or may not have prior knowledge of the identity of the request originator, i.e., party X. The information transferred from party Y to the delegated parties must therefore allow these to fully identify and interact with X.

Figure 9 depicts the BPMN 2.0 representation of this pattern. The *send task A* in participant X sends a message containing the reference of X in its payload to participant Y. The message is received by an *intermediate catch message event* and its payload is stored into a *data object*. Subsequently, the *multi-instance send task C* in Y sends a message containing the reference of X in its payload to each instance of participant Z referenced in a *data object collection* (context-based correlation is used in Z, we assume that delegated parties are all instances of the same participant). Each message sent by task C reaches a different instance of the *receive task E*, that in its turn transfers the payload into a data object. The *send task F* and G in Z executes in parallel. The task F sends messages to Y, allowing it to monitor interactions between Z and X through the *loop receive task D*. The task G sends messages to the *loop receive task B* in order to continue the interaction with the participant X. Context-based correlation is used in X to receive messages from the delegated parties.

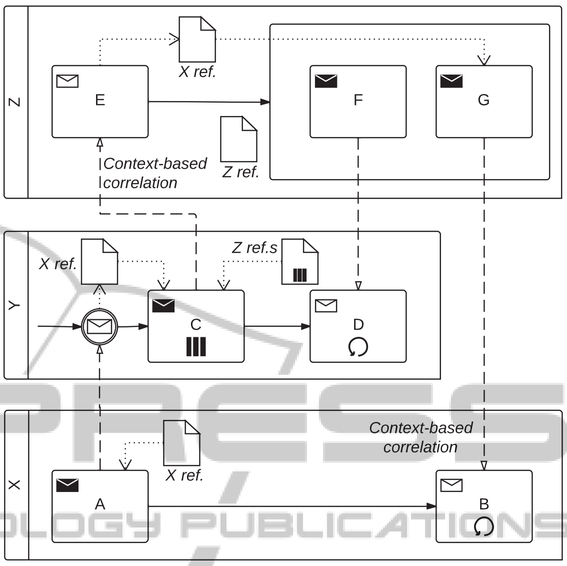


Figure 9: Relayed Request pattern.

4 BPMN 2.0 ENHANCEMENTS

We describe in this section the proposed set of enhancements for BPMN 2.0 that improve its support for service interaction patterns.

4.1 Initiator Extension

In this section, we introduce the concept of collaboration/conversation *initiator*, and modify the message correlation semantics in order to move message routing at the initiator level. Then, we show how such extensions help supporting the *One-to-many Send/Receive* pattern with the BPMN 2.0 workflow described in Section 3.2.4.

Business processes typically can run for days or even months, requiring asynchronous communication via messages. Moreover, many instances of a particular process will typically run in parallel, e.g., many instances of an order process, each representing a particular order. Correlation is used to associate a particular message to an ongoing conversation between two particular process instances. BPMN 2.0 allows using existing message data for correlation purposes, rather than requiring the introduction of technical correlation data (OMG, 2011).

The concept of correlation facilitates the association of a message to a process instance send task (throw message event) or receive task (catch message event) often in the context of a conversation, which is also known as instance routing. This association can be viewed at multiple levels, namely the collaboration (conversation), choreography, and process level. However, the actual correlation happens during runtime (e.g., at the process level). Correlations describe a set of predicates on a message (generally on the payload) that need to be satisfied in order for that message to be associated to a distinct process instance send task (throw message event) or receive task (catch message event).

In plain key-based correlation, messages that are exchanged within a conversation are logically correlated by means of one or more common correlation keys. A correlation key represents a composite key out of one or many correlation properties. A correlation property essentially specifies an extraction expressions atop a message. At run time, the first sent or received message in a conversation populates at least one of the correlation key instances. If a follow-up message derives a correlation key instance, where that correlation key had previously been initialized within the conversation, then the correlation key value in the message and conversation must match. For example, let us suppose to have participant X and Y involved in a conversation with a message flow going from a send task in X to a receive task in Y, and a message flow going from a send task in Y to a receive task in X. When the send task of the i-th instance of X sends a message, a correlation key is instantiated from the message payload. When the receive task of the i-th instance of X receives a message from Y, a correlation key instance is derived from the received message payload, and checked against the previously instantiated correlation key. If the two key instances match, then the received message is accepted. Otherwise, it is discarded by the i-th instance of X.

Key-based correlation allows one to route messages to receive tasks (or catch message events) in specific process instances, based on messages payloads. In some cases, this may be not enough. For example, in the workflow for the *One-to-many Send/Receive* pattern depicted in Figure 4, we want the task B to receive a message that correlates with the one sent by the task A. Hence, we want to route messages from Y to the task B in specific instances of the multi-instance sub-process in X.

BPMN 2.0 does not provide a way to indicate which element (e.g., participant, activity, etc.) involved in a conversation initiates the communication. Such information can be useful to better understand the sequence of interactions determined by message flows in a conversation. Moreover, the knowledge of the conversation initiator is fundamental for moving message correlation to a level different from the one of process instances. We propose a BPMN 2.0 extension, called *initiator*, which allows one to specify the `id` of the element initiating a conversation.

The following is the XSD schema for the initiator extension.

```
<xsd:schema ...>
  <xsd:element name="initiator"
               type="tInitiator"/>
  <xsd:complexType name="tInitiator">
    <xsd:attribute name="initiatorId"
                   type="xsd:string"
                   use="required"/>
  </xsd:complexType>
</xsd:schema>
```

The initiator element can be used to specify the initiator of a collaboration as shown below.

```
<bpmn:definitions ...>
  ...
  <bpmn:extension mustUnderstand="false"
          definition="esteco:initiator"/>
  ...
  <bpmn:collaboration ...>
    <bpmn:extensionElements>
      <esteco:initiator initiatorId="_11"/>
    </bpmn:extensionElements>
    ...
  </bpmn:collaboration>
  ...
</bpmn:definitions>
```

In key-based correlation, correlation key instances are associated to conversation instances. A conversation instance is associated to the process instances that it involves. We propose to associate conversation instances to their initiators. Thanks to this association, a received message can be routed to a specific initiator instance. The modified key-based correlation mechanism works as follows. When a message is sent by an initiator instance, a correlation key is instantiated and associated to the corresponding conversation instance. When a message reaches the initiator, the correlation key instance derived from the message payload is matched with correlation key instances associated to conversation instances. If a match is found, the message is routed to the initiator instance associated to the matching conversation.

Let us now consider the *One-to-many Send/Receive* workflow depicted in Figure 4. The initiator in the conversation between X and Y is the sub-process in X. At run-time, for each message sent by task A, a correlation key is instantiated and associated to an instance of the conversation. Each sub-process instance is associated to a different conversation instance. Each message sent by Y generates a correlation key instance that is matched with the correlation key instances of conversation instances. When a match is found, the message is routed to the sub-process instance associated to the matching conversation, and received by the correct instance of task *B*.

4.2 Message Queuing

In section 3.3.2 we proposed a BPMN 2.0 representation of *Contingent Requests* when the first response is accepted even if it is late (see Figure 7). This representation has a flaw, i.e., late responses may be lost.

In order to overcome this limitation, we propose the introduction of *message queuing*. That is, each message directed to a receive task or catch message event in a process, is stored in a queue when it cannot be received (e.g., when the receive task to which it is directed to is not yet active). As soon as a receive task or catch message event becomes active, it looks for messages in the message queue.

Thanks to message queuing, responses that reach participant *X* when the receive task *B* in Figure 7 is not yet active are not lost. Let us suppose a message from *Y* reaches *X* just after the interrupting timer boundary events of *B* has been triggered. The message is stored in a queue for task *B*. After the selection of the next *Y* reference and the execution of *A*, *B* becomes active and immediately receives the message that was previously stored in the queue.

4.3 Workarounds for Atomic Transactions

As we already pointed out in Section 3.3.3, BPMN 2.0 provides no support for atomic transactions. Nevertheless, different workarounds can be provided for the *Atomic Multicast Notification* pattern.

The first workaround consists in enforcing *quasi-atomicity* (Hagen and Alonso, 2000). Quasi-atomicity is related to the ability to undo certain parts of a process execution. Using this mechanism, receiving parties can perform the work associated to received requests, and compensate for it in case of failure. However, the effect of the performed work is visible to other parties, thus violating the principle of atomicity. Quasi-atomicity can be enforced in BPMN 2.0 by exploiting its built-in support for business transactions. Each receiving party can use a *transaction sub-process* to perform the work associated to the received request. Activities within the transaction that need to be undone if the transaction fails can be connected with their respective *compensating activities* by using *compensating boundary events*. Figure 10 depicts an example of usage of such BPMN 2.0 elements. Participant *Y* executes a transaction through a transaction sub-process. The transaction only consists of a task *A* connected to its respective compensating activity. After the execution of the transaction sub-process, *Y* awaits for an “Ok” or a “Fail” message. If a “Fail” message is received, the *compensation event* “Undo Transaction”, targeted to the transaction sub-process, triggers the compensating activity of *A* and rolls back the transaction to its initial state.

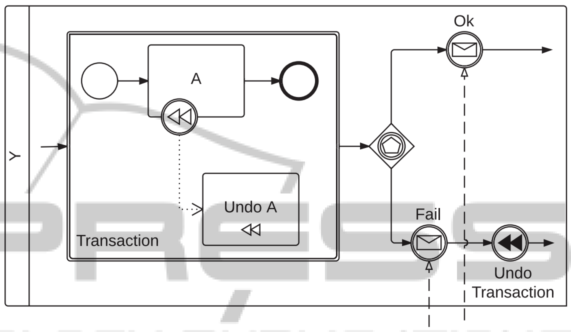


Figure 10: Example of usage of transaction sub-process, compensating activity, and compensation events.

The second workaround is a BPMN 2.0 encoding of the 2PC protocol as a sequence of sub-interactions, in a way similar to the one proposed in (Barros et al., 2005b). In the first phase, a “prepare” message is sent from the coordinating party to each receiving party. Each receiver deals with this message with a separate sub-process, which eventually will send back a “ready” message to the coordinator. After the timeout, the responses are counted to determine whether the minimum and maximum constraint are satisfied. Then, the second phase has a related set of sub-processes for each party providing a “commit” or “reject” message. Different payloads may be included in the first and second phase messages. As part of the first phase of interactions, contacted parties might only see a limited content of the message, enough to decide whether they are ready to accept the request or not. In the second phase, selected parties see all details needed to act on the request. Transaction sub-processes, compensating activities, and compensation events may be used to enforce quasi-atomicity in the second phase.

The third workaround consists in using a variant of the *One-to-many Send/Receive* pattern with a completion condition at the notifying side, as proposed in (Decker and Puhlmann, 2007).

5 CONCLUSIONS

In this paper, we investigated BPMN 2.0 collaboration diagram support for the service interaction patterns (Barros et al., 2005b), and proposed a set of enhancements to broaden it.

We assessed that BPMN 2.0 directly supports nine of the thirteen patterns, i.e., the three *Single Transmission Bilateral Interaction Patterns*, *Racing Incoming Messages*, *One-to-many Send*, *One-from-many Receive*, *Multi-responses*, *Request with Referral*, and *Relayed Request*. Standard BPMN 2.0 supports *Contingent Requests* when we choose to disallow late responses altogether. The BPMN 1.0 extensions presented in (Decker and Puhlmann, 2007) are not necessary in BPMN 2.0, since it supports multiple participants and message correlation out of the box, and since reference passing (Decker and Puhlmann, 2007) can be modeled by using data objects/data inputs/data outputs, messages, and context-based correlation.

We proposed three enhancements to broaden BPMN 2.0 support for service interaction patterns. The first is an extension called *initiator* that together with a modification of the key-based message correlation semantics allows the representation of the *One-to-many Send/Receive* pattern. The second enhancement consists in the use of *message queues* to support the *Contingent requests* pattern when we choose to accept the first response even if it is late and stop outstanding requests. The last enhancement is a set of workarounds for *Atomic Multicast Notification*. Thanks to these enhancements, BPMN 2.0 supports eleven of the thirteen patterns.

Future work will include the study of BPMN 2.0 extensions to further improve the *Contingent request* pattern support. We also plan to evaluate BPMN 2.0 as a whole, comparing and combining our results with the ones presented in (Cortes-Cornax et al., 2011). Moreover, we consider a π-calculus formalization of the BPMN 2.0 semantics as an important future work. Such a formalization would make it possible for a formal validation of the proposed pattern representations, since a π-calculus formalization of the service interaction patterns has already been presented in (Decker et al., 2006b).

ACKNOWLEDGEMENTS

The authors thank the reviewers for the very useful comments that have contributed to enhance the paper.

REFERENCES

Barros, A., Dumas, M., and Hofstede, A. (2005a). Service Interaction Patterns. In *Business Process Management*, volume 3649 of *Lecture Notes in Computer Science*, pages 302–318. Springer Berlin Heidelberg.

Barros, A., Dumas, M., and Hofstede, A. (2005b). Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. Technical Report FIT-TR-2005-02, Faculty of IT, Queensland University of Technology.

Cortes-Cornax, M., Dupuy-Chessa, S., and Rieu, D. (2012). Choreographies in BPMN 2.0: New Challenges and Open Questions. In *Proceedings of the 4th Central-European Workshop on Services and their Composition, ZEUS-2012*, volume 847 of *CEUR Workshop Proceedings*, pages 50–57.

Cortes-Cornax, M., Dupuy-Chessa, S., Rieu, D., and Dumas, M. (2011). Evaluating Choreographies in BPMN 2.0 Using an Extended Quality Framework. In *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 103–117. Springer Berlin Heidelberg.

Decker, G., Overdick, H., and Zaha, J. M. (2006a). On the Suitability of WS-CDL for Choreography Modeling. In *Proceedings of Methoden, Konzepte und Technologien für die Entwicklung von dienstebasierten Informationssystemen, EMISA 2006*.

Decker, G. and Puhlmann, F. (2007). Extending BPMN for Modeling Complex Choreographies. In *Proceedings of the 2007 OTM Confederated international conference on On the move to meaningful internet systems: CoopIS, DOA, ODBASE, GADA, and IS - Volume Part I*, OTM'07, pages 24–40. Springer-Verlag.

Decker, G., Puhlmann, F., and Weske, M. (2006b). Formalizing Service Interactions. In Dustdar, S., Fiadeiro, J., and Sheth, A. P., editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 414–419. Springer Berlin Heidelberg.

Hagen, C. and Alonso, G. (2000). Exception handling in workflow management systems. *IEEE Transaction on Software Engineering*, 26(10):943–958.

OASIS (2007). Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.

OASIS (2009). Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html.

OMG (2011). Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0.

W3C (2005). Web Services Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/.

Zaha, J., Barros, A., Dumas, M., and Hofstede, A. (2006). Let's Dance: A Language for Service Behavior Modeling. In *On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE*, volume 4275 of *Lecture Notes in Computer Science*, pages 145–162. Springer Berlin Heidelberg.