The Conceptualization of a Configurable Multi-party Multi-message Request-Reply Conversation

Nataliya Mulyar[1], Lachlan Aldred[2], and Wil M.P. van der Aalst[1,2]

[1] Department of Technology Management, Eindhoven University of Technology
GPO Box 513, NL5600 MB Eindhoven, The Netherlands
{n.mulyar, w.m.p.v.d.aalst}@tue.nl
[2] Faculty of Information Technology, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{l.aldred}@qut.edu.au

Abstract. Organizations, to function effectively and expand their boundaries, require a deep insight into both process orchestration and choreography of cross-organization business processes. The set of requirements for service interactions is significant, and has not yet been sufficiently refined. Service Interaction Patterns studies by Barros et al. demonstrate this point. However, they overlook some important aspects of service interaction of bilateral and multilateral nature. Furthermore, the definition of these patterns are not precise due to the absence of a formal semantics. In this paper, we analyze and present a set of patterns formed around the subset of patterns documented by Barros et al. concerned with Request-Reply interactions, and extend these ideas to cover multiple parties and multiple messages. We concentrate on the interaction between multiple parties, and analyze issues of a non-guaranteed response and different aspects of message handling. We propose one configurable, formally defined, conceptual model to describe and analyze options and variants of request-reply patterns. Furthermore, we propose a graphical notation to depict every pattern variant, and formalize the semantics by means of Coloured Petri Nets. In addition, we apply this pattern family to evaluate WS-BPEL v2.0 and check how selected pattern variants can be operationalized in Oracle BPEL PM.

1 Introduction

It has been several years since Service-Oriented Architectures (SOAs) started gaining enormous popularity within organizations aiming to extend their boundaries by integrating software applications and external services into their business processes. To coordinate the interaction between service providers and consumers a set of standards and technologies were proposed which contributed in evolution of the Web-services paradigm. Standards like SOAP [1], WSDL [2], UDDI [3], etc. were proposed to interconnect independently developed web-services. A number of standardization proposals (XLang, BPML, and WSCI) [4,5,6,7]

R. Meersman and Z. Tari et al. (Eds.): OTM 2007, Part I, LNCS 4803, pp. 735–753, 2007.

were discontinued, however they have served as a basis for an ongoing standardization initiative: the Business Process Execution Language for Web-Services (BPEL4WS, BPEL, WSBPEL) [8]. The developed technologies successfully handle simple interaction scenarios, however when it comes to interactions involving large numbers of participants many issues remain open.

A business process can be defined as a set of activities executed according to a defined set of rules in order to achieve a specific goal. When two or more organizations wish to embed long-running interactions within their business processes, the focus shifts from the inside of a process to interactions of this process with an external environment. What aspects of service interaction have to be explicitly modeled? How to classify a given interaction scenario? What standard supports a desirable interaction scenario, and which system to select for the realization of the cross-organizational interaction? Answering these questions is significant for understanding of the requirements for service interaction.

To specify requirements in service interaction more extensively than it is done in BPEL4WS and to assess emerging web standards, thirteen Service Interaction Patterns [9] covering bilateral, multilateral, competing, atomic and causally related interactions were identified. A systematic review of the thirteen Service Interaction Patterns presented in [9] has revealed that the scope of the patterns is limited to simple interaction scenarios and that they suffer from an ambiguous interpretation due to their imprecise definition. *In this paper*, we address these gaps by exploring additional possibilities for request-response interactions and by providing a precise formal semantics in the form of Coloured Petri Nets (CPNs) [10,11].

Instead of listing all patterns identified, we propose a framework that allows for a multitude of pattern variants to be generated by configuring a conceptual model of a generic service interaction scenario. The framework is built upon two concepts: a *pattern variant* and a *pattern family*. Every pattern family combines a set of pattern variants that are generated by assigning different values to every aspect of a generic service interaction scenario. Note that the original Service Interaction Patterns correspond to pattern variants belonging to different pattern families we identified. We also propose a notation which can be configured to represent different pattern variants graphically. The CPN models designed to formalize the semantics of the pattern family are also configurable and can be used to simulate the behavior of every pattern variant from the given pattern family.

We have identified five pattern families related to different aspects of message handling in the multi-party conversation (Multi-party Multi-message Request-Reply Conversation), publish-subscribe scenarios (Renewable subscriptions) and correlation on the low- and high-level of abstraction (Message Correlation, Bipartite Message Correlation and Tripartite Message Correlation) [12]. Due to the space limit, in this paper we describe only one pattern family Multi-party Multi-message Request-Reply Conversation. The pattern family presented can be used as a tool for evaluation of web services standards and tools. We implement some of the pattern variants from this family in Oracle BPEL PM and analyze the support of different pattern variants by WS-BPEL v2.0.

The remainder of the paper is organized as follows. Section 2 gives an overview of the related work. Section 3 presents the conceptual background and introduces the format for describing of pattern variants. The Multi-party Multi-message Request-Reply Conversation pattern family is described in Sec. 4. Section 5 shows the implementation of a selected pattern variant in Oracle BPEL PM. Evaluation of WS-BPEL v2.0 is performed in Sec. 6. This paper concludes with Conclusions described in Sec. 7.

2 Related Work

The Service Interaction Patterns documented by Barros et al. in [13,9] describe a collection of scenarios, where a number of parties, each with its own internal processes, need to interact with one another according to pre-agreed rules. These scenarios were consolidated into 13 patterns and classified based on the maximal number of parties involved in an exchange, the maximum number of exchanges between two parties involved in an interaction and whether the receiver of a response is necessarily the same as the sender of a request. Based on this classification four groups were identified: (1) single transmission bilateral interactions (i.e. one-way and round-trip bilateral interactions where a party sends and/or receives a message to another party); (2) single transmission multilateral non-routed interactions (i.e. a party sends/receives multiple messages to different parties); (3) multi transmission bilateral interaction (i.e. a party sends/ receives more than one message to/from the same party); (4) routed interactions.

Since the Service Interaction Patterns of Barros et al. [9] lacked a formal semantics, their formalization by means of the π-calculus has been proposed in [14]. Decker and Puhlmann formalized the patterns based on their descriptions, and did not take into account issues which were related to the patterns but which were incorporated into the pattern descriptions. Thus, they showed the possibility to formalize certain aspects of service interaction, but in fact did not make the definition of patterns less ambiguous. For example, the pattern Racing Incoming Messages specifies: *A party 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 partners. The way a message is processed depends on its type and/or the category of partner from which it comes.* This pattern does not specify what happens if the party receives multiple messages at once, i.e. it is not clear how many of the received messages will be consumed and whether the rest of the messages will be discarded.

In [15] Zaha et al. formulate requirements for a service interaction modeling language, in addition to the ones covered by Barros et al. in [13]. The authors use these requirements for modeling behavioral dependencies between service interactions. In [16] Barros et al. introduce a set of correlation patterns that were used for evaluation of standards WS-addressing and BPEL. However, the framework presented by the authors does not cover relationships between different process instances. In [17] Barros et al. propose a compositional framework for service interaction patterns and interaction flows. They provide high-level

models for eight service interaction scenarios using ASM, illustrating the need for distinguishing between different interpretations of the patterns.

In [18] Cooney et al. proposed a programming language for service interaction, which has been used to describe implementations of One-to-Many Send-Receive and Contingent Requests service interaction patterns [9].

Aldred et al. have performed a detailed analysis of the notion of (de-)coupling in communication middleware using three dimensions of decoupling, e.g. synchronization, time and space, and documented coupling integration patterns [19].

This work is also related to contracting workflows and protocol patterns of van Dijk [20], who proposed a number of protocol patterns for the negotiation phase on a transaction. Work of Hohpe and Woolf on Enterprise application integration [21] covers various messaging aspects that can be encountered during application integration.

Furthermore, this work relates to the Workflow Patterns initiative [22,23], where a set of 43 Control-flow patterns [24], a set of 40 Data patterns [25] and a set of 43 Resource patterns [26] are proposed. In addition, a Workflow Pattern Specification Language (WPSL) [27] has been defined which allows various pattern variants to be described in a language-independent way. In particular, the control-flow patterns have had a considerable influence on the development of new languages, the adaptation of the existing ones and all kinds of standardization efforts. This paper should be seen as a part of the Workflow Patterns initiative.

Our work, presented in this paper, differs from the described related work in the following aspects. We broaden up the scope of the original Service Interaction Patterns and systematically describe various pattern variants along with offering a graphical notation that is suitable for representing every pattern variant. To avoid ambiguous interpretation we formalize the patterns by means of CPNs.

3 Conceptual Background

In this section we describe concepts central to the pattern family considered and present the format for describing the pattern variants.

Instead of listing the whole set of patterns identified, we underline the differences between the pattern variants belonging to the same pattern family. We introduce the key concepts used in the pattern description by means of a UML Data Object diagram. The attributes that influence the detailed semantics of each pattern variant are described separately. To clarify the semantics of the pattern we apply the formalism of CPNs. We designed a (set of) CPN model(s) and tested them using the simulator facilities of CPN Tools. Declarations used within CPN models are based on the set of the concepts introduced in the UML diagram. We depict a generic service-interaction scenario belonging to a given pattern family with an icon graphically representing a set of attributes. By setting the attributes a specific variant of a pattern family is selected.

Pattern attributes (also referred to as parameters) represent the orthogonal dimensions for classifying different aspects of the service interaction within the context of the given pattern family. All possible combinations of the attribute values result in a large set of pattern variants, each of which can be easily derived from the generic service-interaction scenario and is depicted by a corresponding icon.

For the purposes of this paper a Conversation is defined as the communication of a set of contextually related messages between two or more parties. A Party is an entity involved in communication with other parties by means of sending/receiving messages. A party may represent a process, a service, a business unit, etc. A Message is a unit of information that may be composed of one or more data fields. A message may represent a request or a reply.

We describe the pattern family using the following format:

- *Description* of a generic pattern variant belonging to a given pattern family.
- *Examples* illustrating the application of the given pattern variant in practice.
- *UML meta-model* describing concepts specific to a given pattern family.
- *Visualization*: a graphical notation representing a generic pattern variant and the description of variation points that can be used for tuning the graphical notation to represent pattern variants.
- *CPN semantics*: the semantics of a generic pattern variant illustrated in the form of CPN models and their corresponding description.
- *Issues* that can be encountered when applying a pattern variant from the given pattern family in practice.

4 Pattern Family: Multi-party Multi-message Request-Reply Conversation

In this section, we present the Multi-party Multi-message Request-Reply Conversation pattern family using the format described earlier.

Description. A Requestor posts a compound request consisting of N sub-requests to a set of M parties and expects a reply message to be received for every sub-request. There exists the possibility that some parties will not respond at all and the possibility that a Responder will not reply on some sub-requests. The Requestor queues all incoming messages in a certain order. The enabling of the Requestor for consumption of reply messages depends on the fulfillment of activation criteria. The Requestor should be able to, optionally, consume a subset of the responses and even process a subset of the consumed set - hence allowing for business use cases where only the best or fastest responses are needed. The number of times the Requestor may consume messages from the queue can be specified explicitly.

Example

- A request to submit an abstract and to submit a paper is issued by an editor to a list of people registered for participation in a workshop. Only papers and

abstracts submitted before a deadline would be reviewed. If a large amount of papers arrive, only the first 50 would be reviewed and only 10 best papers out of the reviewed ones would be published.

UML meta-model. An object diagram illustrating the pattern on the conceptual level is presented in Fig. 1. A Conversation consists of a set of messages (see a composition relation between `Conversation` and `Message`). A conversation involves an initiating process (e.g. Requestor), and at least one following process (e.g. Responder processes), depicted by associations `requestor` and `responder`. Any process may be or may not be involved in multiple conversations (see the multiplicity of the association `involves`). The Requestor generates at least one Request, while the Responder returns one or more Replies or does not react at all. The relation between request and reply messages is depicted by `corresponds to` association, and sending of request and reply messages by a party is illustrated by the dependency relations `is sent by` and `is produced by`. Requests issued by a Requester can be composite meaning that the Requestor may send several sub-requests in a single message concurrently to a single or multiple parties.

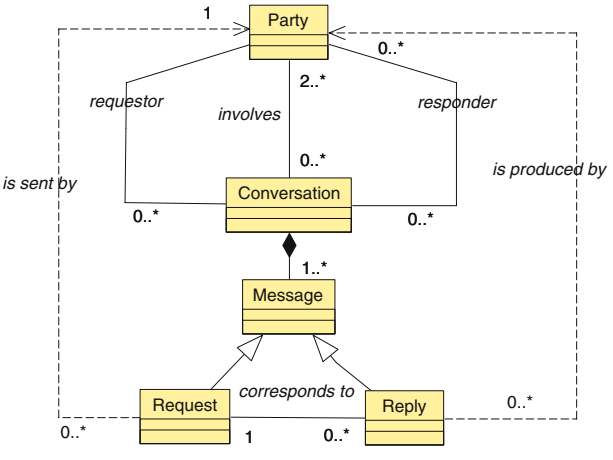


Fig. 1. UML meta-model of Multi-party Multi-message Request-Reply Conversation

Visualization. The graphical notation of the generic pattern variant is given in Fig. 2. The parties are visualized as rectangles. Directed arrows represent the direction in which a party sends a message. A message containing a single request is visualized as a black token, while a compound request is represented by multiple overlapping tokens. Parameters specific to a given party are visualized as icons residing within the boundaries of a rectangle representing a party. This graphical notation has the following set of *variation points*:

- *N* - a parameter denoting a list of sub-requests sent by a Requestor to a Responder in a single message.
  Range of values: size(N)≥1.

Default value: size(N)=1.
Visualization: This parameter is depicted by the dots on the arc from Requester to Responder. If size(N)>1 or size(N)=1 the graphical notations depicted in Fig. 3 (1a) and (1b) are used respectively.

- *M* - a parameter denoting a list of Responders involved in the conversation.
  Range of values: size(M)≥1.
  Default value: size(M)=1.
  Visualization: if size(M)>1 or size(M)=1 the graphical notations depicted in Fig. 3 (2a) and (2b) are used respectively.

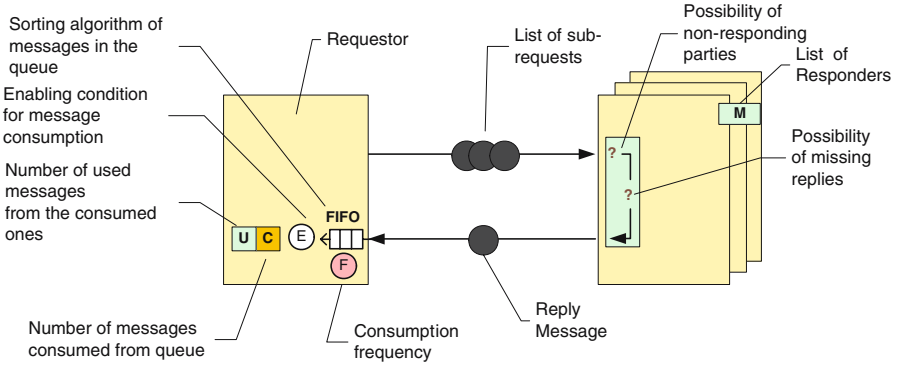


Fig. 2. Graphical notation: Multi-party Multi-message Request-Reply Conversation

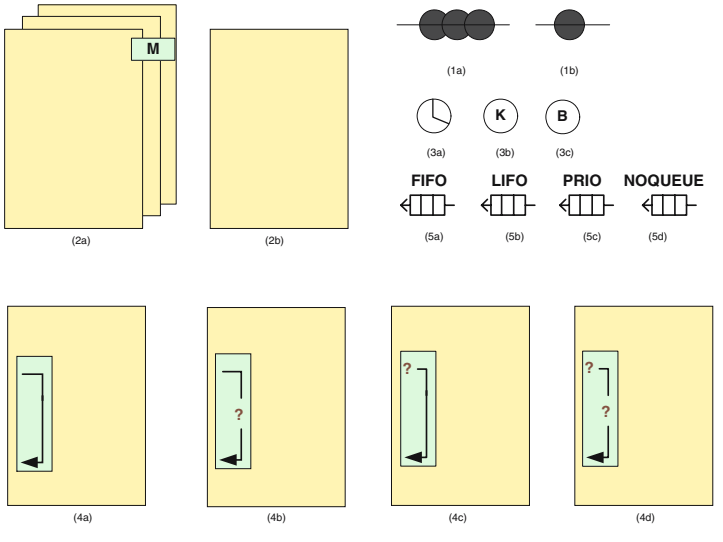


Fig. 3. Variants of graphical notation: Multi-party Multi-message Request-Reply Conversation

- *Possibility of non-responding parties* - a parameter specifying whether some of the Responders will ignore the request issued by the Requestor.
  Range of values:
  - No: all M Responders will reply at least something (for example, a request to report the level of income to the tax-office obliges all receivers to reply);
  - Yes: some Responders may not reply anything (for example, only interested parties react on the invitation to participate in a social event).

  Default value: No.
  Visualization: Fig. 3 depicts the graphical representation of four variations, where: in (4a) and(4b) all M Responders will produce at least some replies; in (4c) and(4d) some Responders may not reply on the requests received.
- *Possibility of missing replies* - a parameter specifying whether the Responder will not reply on some of the sub-requests (i.e. it is a choice of the Responder to engage in the conversation or not, and respectively to reply on all or only some of the received requests).
  Range of values:
  - No: Responders reply on all sub-requests (for example, the Responder answers on all questions in the tax declaration);
  - Yes: Responders reply only on some sub-requests (for example, a client subscribes only for two out of five journal offers received).

  Default value: No.
  Visualization: Fig. 3 depicts the graphical representation of four variations, where: in (4a) and (4c) no replies will be lost; in (4b) and (4d) some replies may not reach the Requestor.
- *Sorting of the queued messages* - a parameter specifying an ordering discipline according to which response messages queued by the sender are sorted.
  Range of values:
  - FIFO: oldest message is queued first;
  - LIFO: newest message is queued first;
  - PRIO: sorting based on some criterion (for instance, the price);
  - NOQUEUE: messages are not queued and consumed upon arrival if the sender is ready to process them, otherwise they are lost.

  Default value: FIFO.
  Visualization: Fig. 3 (5a)-(5d) depicts the graphical notation of different policies applied for sorting messages in the queue.
- *Enabling condition* - a parameter specifying the condition that has to be fulfilled to enable the Requestor to consume replies.
  Range of values:
  - a timeout (for example, requests for purchase on discount basis are accepted only until the expiration of the discount period);
  - a boolean condition, examining the properties of the queued messages (for example, at least three low-cost offers are required to select the best of them);

- a specified number of messages K ($0<K\leq N$).

Default value: K=1.
Visualization: The icon *E* residing at the Requestor's side in Fig. 1 substituted with one of the graphical notations presented in Fig. 3 (3a), (3b) and (3c) which denote the enabling condition based on a timeout, availability of specific number of messages and boolean expression respectively.

- *Consumption index* - a parameter specifying the number of reply messages to be consumed by the Requestor from the queue.
Range of values:
  - 0: none of the messages are removed from the queue (for example, messages must have enabled the process to receive, but it may need to leave them on the queue for another process to use);
  - S: S messages are removed from the queue such that $0\leq S<K$, where K is a number of replies sufficient for activation of the requester (for example, only messages selected by a boolean expression based on the property values are consumed);
  - All: all messages contained in the queue are removed.

  Default value: All.
  Visualization: The icon *C* residing at the Requestor's side in Fig. 2 substituted with a suitable value.

- *Utilization index* - a parameter specifying a number of messages from the consumed ones used by the Requestor for the processing.
Range of values:
  - 0: no messages are used for processing (for example, if no messages were consumed, or if none of the consumed messages are required by the receiving process);
  - 1: one message is used for processing (for instance, a best offer from the available ones is selected);
  - UN: a number of messages used for the processing such that $1<UN<C$, where C is a number of messages consumed (for example, a boolean condition chooses only messages that pass the boolean constraint);
  - All: all consumed messages are used for the processing.

  Default value: All.
  Visualization: The icon *U* residing at the Requestor's side in Fig. 2 substituted with its value.

- *Consumption Frequency* - a parameter specifying the number of times the sender performs the consumption of messages from the queue.
Range of values:
  - 1: the sender is activated only once, after this all remaining and arriving messages are destroyed;
  - FN: the sender consumes messages FN number of times, $1<FN$, after which all remaining and arriving messages are destroyed;
  - ∞: the sender consumes messages as long as they arrive.

Default value: 1.
Visualization: The icon F residing at the Requestor's side in Fig. 2 substituted with its value.

The pattern variant representing a scenario in which every parameter is set to the default value is presented in Fig. 4. A party A sends a single request to a party B, who sends a reply back. The party A queues the messages in the FIFO-order, and as soon as one message is received from the party B, it is consumed and processed. The presented notation may be used to represent Broadcast Remote Procedure Calls (RPC) [28], which expects one or more answers from each responding machine and treats all unsuccessful responses as garbage by filtering them out.

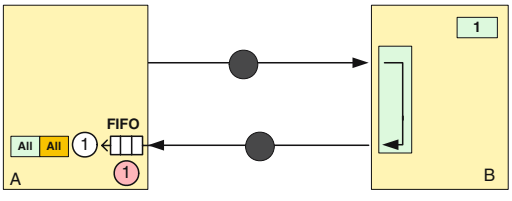


Fig. 4. Notation for the default pattern variant

CPN semantics. To avoid an ambiguous interpretation of the pattern variants related to Multi-party Multi-message Request-Reply Conversation we formalize the semantics by means of CPNs. Figure 5 depicts the top view of the CPN diagram representing the pattern.

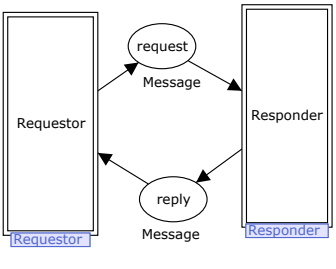


Fig. 5. CPN diagram: The main view

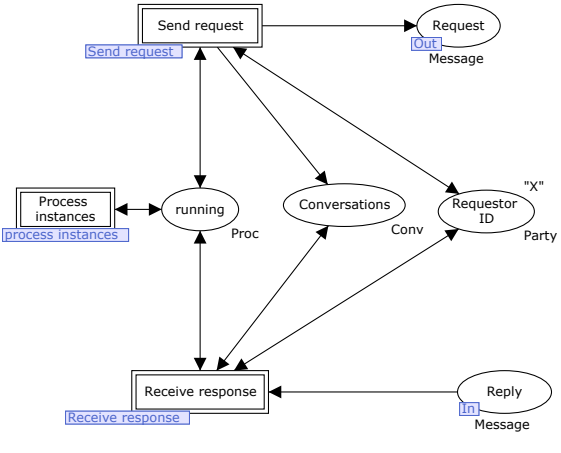


Fig. 6. CPN diagram: The `Requestor` page

Requestor and Responder are represented as substitution transitions which can be unfolded to the nets depicted in Fig. 6 and Fig. 7(c) respectively. In every given conversation the parties exchange requests and replies of type `Message`.

The Requestor (whose behavior is shown in Fig. 6) can send requests and receive response messages using substitution transitions `Send request` and `Receive response` whose decomposition is presented in Fig. 7(b) and Fig. 8. A Requestor process may have multiple process instances, whose lifecycle is shown in Fig. 7(a). Process instances available for participation in a conversation are stored in place `enabled`. When for a given process instance a conversation is started, a conversation identifier `cid` is coupled with a process instance. The uniqueness of identifiers is ensured by incrementing of a counter whose value is stored in place `Conversation counter`. A process instance chosen for conversation is stored in place `running`. Transitions `activate`, `deactivate` and

Table 1. Data types used in Figs. 5-8

```
colset Party = string;
colset Request = string;
colset Requests = list Request;
colset Reply = string;
colset Replies = list Reply;
colset ConvId = int;
colset Content = union Req:Requests + Repl:Replies; [a]
colset Message = product ConvId * Party * Party * Content; [b]
colset Count = int;
colset MTime = int;
colset Status = with active|inactive|enabled|completed; [c]
colset ConvRequest = product Parties * Requests;
colset ConvRequests = list ConvRequest;
colset ConvReply = product Parties * Replies;
colset ConvReplies = list ConvReply;
colset Pr = product ConvRequests*ConvReplies*Status;
colset Proc = product ConvId*Pr; [d]
colset ConvInfo = record start_time: MTime * last_act:MTime
    * nof_unique_messages: Count * nof_parties: Count * total_nof_messages: Count;
colset Conv = product ConvId * ConvInfo * Status;
```

[a] The content of a message is either a list of requests or a list of replies. The CPN union type is used to specify this.

[b] A message is a tuple `(cid,P1,P2,c)` where `cid` is a conversation identifier, `P1` is the requestor, `P2` is the responder, and `c` is the content. Such a message is of type `Message`.

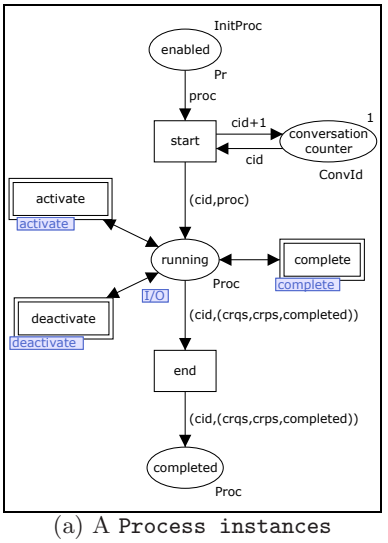
[c] The lifecycle of a process instance starts with activation of an `enabled` instance. An `active` instance can become `inactive` through deactivation, or `completed` when the instance lifecycle ends.

[d] Process instances of type `Pr` contain a list of requests sent, replies received and a status of the instance. When a conversation starts, a process instance is coupled with a conversation identifier.

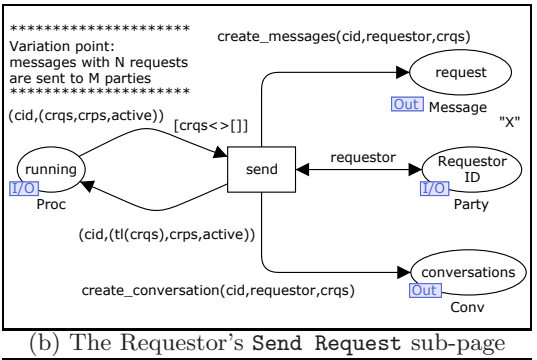
`complete` control the status of a process instance during its lifecycle. When an enabled process instance is activated, it gets the status `active` and may participate in sending and receiving of messages. Meanwhile the active process instance can become `inactive` through deactivation or can become `completed`. The lifecycle of a process instances ends upon completion and the process instance is placed to place `completed`.

The Requestor's `Send Request` sub-page in Fig. 7(b) shows that the Requestor, whose identifier is stored in place `Requestor ID`, on the moment of sending a request message creates a new conversation. Function `create_messages()` takes a list of conversation requests `crqs` of type `ConvRequests`, which contains a list of parties to whom a request should be sent, and a list of sub-requests that should be sent to each party, and creates as many messages as there are parties in the list. This function directly corresponds to the variation point specifying that messages with N sub-requests are sent to M parties.

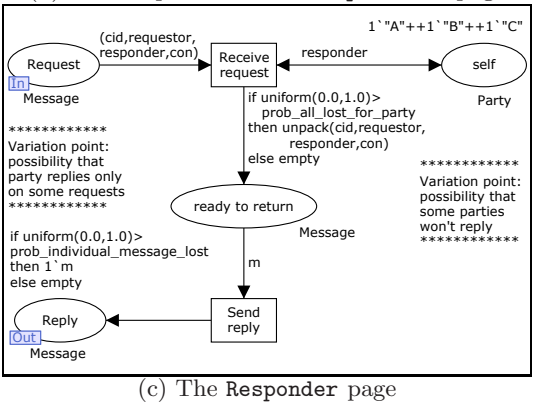
When request messages are created, a new conversation is created by means of the function `create_conversation()`. This function records the information about the conversation identifier, conversation-specific parameters (the start time of the conversation, the time of the last activation, a total number of



(a) A `Process instances` sub-page



(b) The Requestor's `Send Request` sub-page



(c) The `Responder` page

Fig. 7. CPN diagrams

messages sent, a number of parties to whom the requests have been sent, and a number of unique messages (i.e. a number of sub-requests contained in the single message)), and the status of the process instance. The recorded conversation information is used later on for the purpose of correlating response messages received with the requests sent and for identifying how many times the received messages can be consumed for processing.

The Responder page shown in Fig. 7(c) illustrates the behavior of Responders involved in the conversation. The identifiers of the Responders are stored in place `self`. They are used to relate incoming requests to a right party, based on the party identifier. When a Responder receives a request message, it unpacks the composite requests into separate messages each containing a separate sub-request. The parameter `prob_all_lost_for_party` corresponds to a variation point specifying the probability that the Responder will ignore a received composite request or will process it. If the Responder decides to reply on the request, the parameter `prob_individual_message_lost` is used as a variation point to define the probability that a reply will be sent for every unpacked sub-request.

The Requestor's `Receive response` sub-page presented in Fig. 8 illustrates the mechanism of queueing and processing of incoming responds by the Requestor. The Requestor processes only messages addressed to it (for this purpose, a `Requestor ID` is used). The response messages received are queued according to the `QueueingDiscipline()` function, which corresponds to a variation point that can be set to any of the queueing disciplines, i.e. LIFO, FIFO or PRIO.

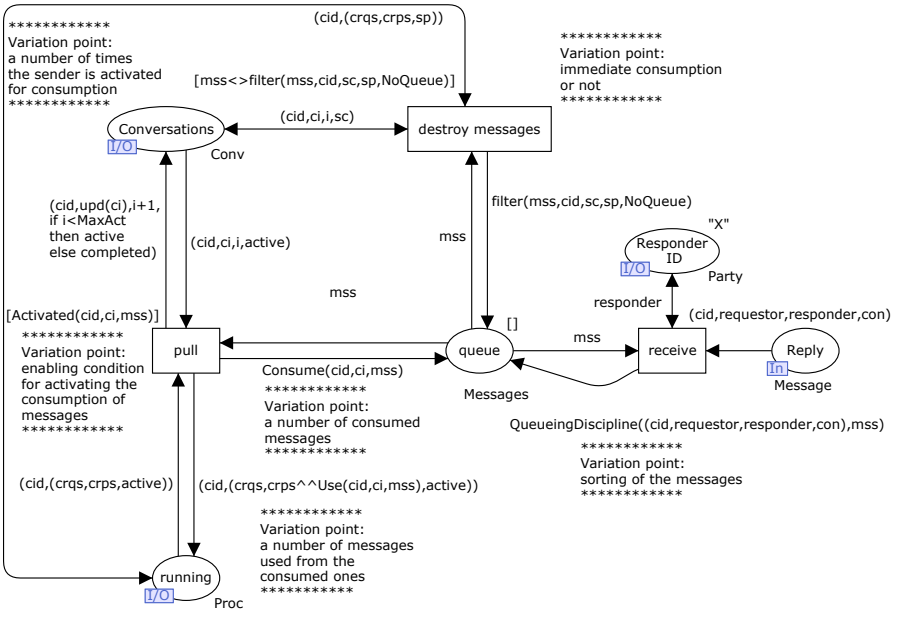


Fig. 8. CPN diagram: The Requestor's `Receive Response` sub-page

Function `Consume()` corresponds to a variation point specifying how many messages from the queued ones have to be consumed. One, several, or all available in the queue messages can be consumed. The consumption of messages happens upon the satisfaction of an enabling condition (which is a variation point) encoded as a guard of transition `Pull`. Function `Activated()` can be tuned to specify the enabling upon the availability of one or several messages in a queue, upon the satisfaction of a certain condition, or upon a timeout.

From the messages consumed only a number of messages defined by the function `Use()` are actually used by the Requestor for the processing. This variation point can be set for using either one, several or all consumed messages.

The parameter `MaxAct` corresponds to the variation point specifying how many times the Requestor may consume the messages from the queue for the given conversation. If the messages have been consumed the specified number of times, the process instance receives the status `completed` and the messages left in the queue are removed from it by means of the function `filter()`. Transition `destroy messages` is used to retrieve messages from the place `queue` if the incoming response messages do not need to be sorted and have to be consumed immediately upon arrival.

Issues. When applying pattern variants belonging to the Multi-party Multi-message Request-Reply Conversation pattern family an issue of the message correlation may arise while matching replies received with the requests sent. This issue can be solved by applying a suitable pattern variant from the Message Correlation family. If a Multi-message Multi-Party Request-Reply Conversation pattern variant has to be applied in the context of a long-running conversation, where a series of requests have to be sent one after another, the given pattern variant can be combined with a suitable pattern variant from the family of Bipartite Conversation Correlation.

5 Oracle BPEL PM: A Default Scenario in Action

In this section, we illustrate an implementation of the default pattern variant in Oracle BPEL PM v.10.1.3.1.0 (which is a tool supporting design of BPEL processes).

Figure 9(a) illustrates an asynchronous process which upon an initiation by a client performs the invocation of a synchronous service `ResponseProcess` presented in Fig. 9(b) using an invoke activity `SendRequestToResponder`.

```
<invoke name="SendRequestToResponder" partnerLink="ResponseProcess"
        portType="ns2:ResponseProcess" operation="process"
        inputVariable="RequestorInputVariable"
        outputVariable="ObtainedOutputVariable"/>
```

The content of the request sent, enclosed in the `RequestorInputVariable`, is specified by the <assign> activity `AssignInputData` in Fig. 9(a). The Responder process `ResponseProcess` is initiated by a message received from the

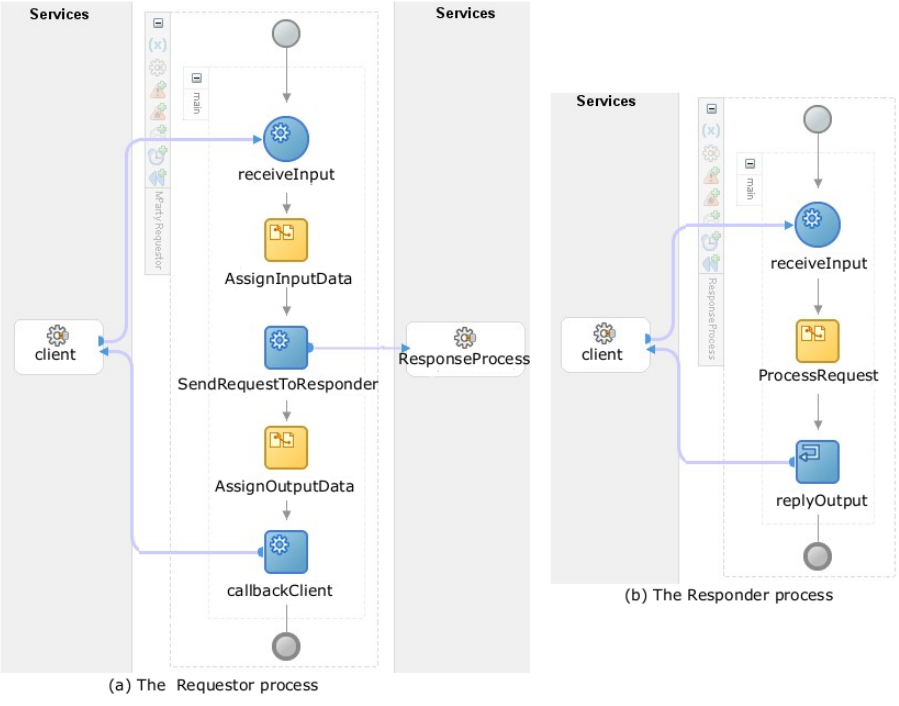


Fig. 9. Implementation of default pattern variant in Oracle BPEL PM

Requestor process. The request received is processed by an <assign> activity `ProcessRequest` in Fig. 9(b) and a response is sent back to the Requestor process using a `replyOutput` activity. The response message is assigned to an output variable `ObtainedOutputVariable` of the `SendRequestToResponder` invoke activity. Note that <invoke> activity has no attribute for message queueing, therefore response messages are not queued and are consumed and processed as soon as they arrive. We discuss the support of other pattern variants by WS-BPEL v.2.0 in the next section.

6 Evaluation of WS-BPEL v2.0

In this section we analyze what pattern variants of Multi-party Multi-Message Request-Reply Conversation are supported by WS-BPEL v2.0 by defining what values each variation point can take.

- *Number of sub-requests in a message*: an `<invoke>` activity in WS-BPEL is used to call an operation on a service. To realize a request-reply conversation a correlation pattern of the `<invoke>` activity has to be set to "request-response" and both an `inputVariable` and `outputVariable` of certain data types have to be specified. Since WS-BPEL is XML-based, a complex data

types can be defined, thus allowing to compose message from multiple sub-requests.

- *Number of Responders involved in a conversation*: many parties can be involved in a conversation with a given Requestor. A message can be sent by a Requestor to a set of Responders in parallel if every Responder is defined as a separate `PartnerLink` and a separate `<invoke>` activity is placed for every partner either in the body of the `<flow>` construct or in the `<forEach>` construct that operates in parallel mode. Therefore WS-BPEL can implement this, albeit clumsily.
- *Possibility of non-responding parties*: an `<invoke>` activity is used to call (an operation on) a service. Such an invocation can be one-way or request-response. When a request-response invocation is performed by the Requester, the `<invoke>` activity stays open until the response is received. This however does not guarantee that the service invoked will respond.
- *Possibility of missing replies*: in WS-BPEL inbound message activities (IMA) (i.e. `<receive>`, `<pick>`, `<onEvent>`) may complete only after they have received a matching message. However, in some situation an *orphaned IMA* occurs when an inbound message activity remains to be open. In this case, the standard fault `bpel:missingReply` is thrown and the orphaned IMA is not considered to be orphaned anymore.
- *Sorting of queued messages*: messages received by a process instance are not queued (NOQUEUE). WS-BPEL defines that a receiving activity needs at most one message to proceed. However, in the situation when a receiving activity is not ready for consumption and multiple messages arrive at a time, a race condition occurs. WS-BPEL does not mandate any specific mechanism for handling race conditions and leaves this decision to the BPEL engine designers.
- *Enabling condition*: an inbound message activity becomes enabled as soon as a matching message has been received by a process instance (i.e. a message of a specific type). However, it is also possible to use a `<wait>` construct in order to enable an activity for message receival after a given time period or after a certain deadline has been reached.
- *Consumption index*: only one message at a time can be consumed by an inbound message activity.
- *Utilization index*: since inbound message activities can consume only one message at at a time, therefore the message consumed is also the one used for processing.
- *Consumption Frequency*: in WS-BPEL it is possible to specify that a party may consume messages multiple times if IMA is placed in a `<while>` or `<repeatUntil>`. The consumption frequency in this case is defined by the evaluation of the boolean condition defined in these repetitive constructs.

The mapping of the pattern attributes to the WS-BPEL is not straightforward, since WS-BPEL does not have concepts able to capture the meaning of all pattern attributes or these concepts are not explicitly defined. By definition, all inbound message activities in WS-BPEL are executed as soon as a suitable

message arrives. Selection of such a behavior as a default results in quite limited capabilities of WS-BPEL to support different variants of message handling. Since WS-BPEL intentionally does not specify a mechanism for handling of the race conditions, systems supporting BPEL-processes may employ different implementations and thus support distinct pattern variants. In this case, the pattern attributes can be used to assist in selecting an appropriate system.

7 Conclusions

The approach presented in this paper shows that a multitude of pattern variants can be derived by assigning different values to variation points identified as a result of the systematic analysis of service interaction scenarios. This approach is applicable for describing other problems in the form of the configurable framework, given that all dimensions of the problem analyzed are clearly delineated and well understood. The main benefit of presenting patterns by means of a configurable pattern family is that it allows various variants of multi-dimensional complex problems to be described and referenced in a uniform way. The complexity of the Multi-party Multi-message Request-Reply Conversation pattern family is characterized by 6912 pattern variants (this number is calculated as multiplication of total number of values each of the variation points may take). The variation points identified can be used for the evaluation of tools and web-service composition standards as it has been done for Oracle BPEL PM and WS-BPEL v2.0. The analysis of WS-BPEL has shown that many pattern variants related to processing of multiple messages are not supported. Such an analysis forces us to deeply think about the requirements in service interaction and may trigger a revision of current best practices in order to capture all variation points and support more pattern variants. Furthermore, the pattern family presented can be used as a solution selection instrument, or even as a set of requirements for new languages in the area. In the future, we plan to use the pattern families as a benchmark for classification of complex service interaction scenarios.

References

1. Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., Winer, D.: Simple Object Access Protocol (SOAP) 1.1. (2000), `http://www.w3.org/TR/soap`
2. Christensen, E., Curbera, F., Meredith, G., Weerawarana, S.: Web Services Description Language (WSDL) 1.1 (2001), `http://www.w3.org/TR/wsdl`
3. Belwood, T., et al.: UDDI Version 3.0 (2000), `http://uddi.org`
4. Arkin, A., Askary, S., Fordin, S., Jekel, et al.: Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems (2002)
5. Arkin, A., et al.: Business Process Modeling Language (BPML), Version 1.0 (2002)
6. Thatte, S.: XLANG Web Services for Business Process Design (2001)
7. Peltz, C.: Web services orchestration: a review of emerging technologies, tools and standards. Hewlett Packard, Co. (2003)

8. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation (2003)
9. Barros, A., Dumas, M., Hofstede, A.: Service Interaction Patterns: Towards a Reference Framework for Service-based Business Process Interconnection. QUT Technical report, FIT-TR-2005-02, Queensland University of Technology, Brisbane (2005)
10. CPN Group University of Aarhus, Denmark: CPN Tools Home Page `http://wiki.daimi.au.dk/cpntools/`
11. Jensen, K.: Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. In: EATCS monographs on Theoretical Computer Science, Springer, Berlin (1992)
12. Mulyar, N., Aldred, L., Aalst, W., Russell, N.: Service interaction patterns: A configurable framework. BPM Center Report BPM-07-07, BPM Center, BPMcenter.org (2007)
13. Barros, A., Dumas, M., ter Hofstede, A.: Service Interaction Patterns. In: Proceedings of the 3rd International Conference on Business Process Management, Nancy, France, vol. 3716/2005, pp. 302–318 (2005)
14. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: Dustdar, S., Fiadeiro, J.L., Sheth, A. (eds.) BPM 2006. LNCS, vol. 4102, pp. 414–419. Springer, Heidelberg (2006)
15. Zaha, J., Barros, A., Dumas, M., ter Hofstede, A.: Let's Dance: A Language for Service Behavior Modeling. In: OTM Conferences (1), Vienna, Austria, pp. 145–162 (2006)
16. Barros, A., Decker, G., Dumas, M., Weber, F.: Correlation Patterns in Service-Oriented Architectures. In: FASE. Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering, Braga, Portugal (2007)
17. Barros, A., Borger, E.: A Compositional Framework for Service Interaction Patterns and Interaction Flows. In: Lau, K.K., Banach, R. (eds.) ICFEM 2005. LNCS, vol. 3785, pp. 5–35. Springer, Heidelberg (2005)
18. Cooney, D., Dumas, M., Roe, P.: GPSL: A Programming Language for Service Implementation. In: Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering, Vienna, Austria (2006)
19. Aldred, L., Aalst, W., Dumas, M., Hofstede, A.: Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware. BPM Center Report BPM-06-19, BPMcenter.org (2006)
20. van Dijk, A.: Contracting Workflows and Protocol Patterns. In: Business Process Management, pp. 152–167. Springer, Heidelberg (2003)
21. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Addison-Wesley Professional, Reading (2003)
22. Aalst, W., Hofstede, A., Kiepuszewski, B., Barros, A.: Workflow Patterns. Distributed and Parallel Databases 14(1), 5–51 (2003)
23. WPHP: Workflow Patterns Home Page, `http://www.workflowpatterns.com`
24. Russell, N., Hofstede, A., Aalst, W., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPMcenter.org (2006)
25. Russell, N., Hofstede, A., Edmond, D., Aalst, W.: Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane (2004)

26. Russell, N., Hofstede, A., Edmond, D., Aalst, W.: Workflow Resource Patterns. In: WP 127, Eindhoven University of Technology, Eindhoven. BETA Working Paper Series (2004)
27. Mulyar, N., Aalst, W., ter Hofstede, A.H.M., Russell, N.: Towards a WPSL: A Critical Analysis of the 20 Classical Workflow Control-flow Patterns. Technical report, Center Report BPM-06-18, BPMcenter.org (2006)
28. Broadcast RPC: Programming with Remote Procedure Calls. http://ou800doc.caldera.com/en/SDK_netapi/rpcpC.bcast.html