Service Interaction Patterns: A Configurable Framework

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

[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,n.russell}@tue.nl
[2] Faculty of Information Technology, Queensland University of Technology
GPO Box 2434, Brisbane QLD 4001, Australia
{l.aldred}@qut.edu.au

Abstract. In this paper we present a framework for describing a series of pattern variants encountered in the context of Service Interaction. The original Service Interaction Patterns [9] only covered 13 interactions scenarios. Moreover, some important aspects of service interaction of a bilateral and multilateral nature are not addressed in [9]. Furthermore, these patterns allow for ambiguous interpretation due to the absence of a formal semantics. The scope of the patterns generated by means of the framework described in this paper is much broader. To avoid ambiguities we formalize the semantics of the patterns by means of Colored Petri Nets (CPN). In addition, we propose an intuitive graphical notation that can be used to denote the various pattern variants. This paper also provides an evaluation of WS-BPEL v2.0 standard using these patterns.

Keywords: Service Interaction Patterns, Colored Petri Nets, BPEL, correlation.

1 Introduction

In recent years the Service-Oriented Architecture (SOA) has started gaining popularity within organizations aiming to integrate third party 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 underpin the notion of web-service. Standards like SOAP [14], WSDL [16], UDDI [13], etc. were developed to interconnect independently developed web-services. A number of standardization proposals (XLANG, BPML and WSCI) [6, 7, 32, 27] were discontinued, however they have served as the basis for two ongoing standardization initiatives: the Business Process Execution Language for Web-Services (BPEL4WS, BPEL, WSBPEL) [5] and the Web Services Choreography Description Language (WSCDL) [24]. The resultant technologies successfully handle simple interaction scenarios, however when it comes to interactions involving large numbers of participants many issues remain unresolved.

The concepts of orchestration and choreography come into focus when two or more organizations wish to embed complex long-running multi-party interactions

within their business processes. A business process can be defined as a group of activities executed according to a defined set of rules in order to achieve a specific goal. A choreography defines the sequence of dependencies between interactions between multiple parties in order to implement a business process comprising multiple web services [26]. In other words, choreography describes the externally visible behavior of the interacting parties. Orchestration defines the control and data flow between web services that are necessary to achieve a business process, from the viewpoint of one participant [26].

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 service interaction? Answering these questions is significant for understanding of the requirements for service interaction.

To specify requirements in service interaction more extensively than what has been done in the content of 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 these 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 broadening the scope of the service interaction patterns and by providing a precise formal semantics in the form of Colored Petri Nets (CPNs) [18, 23]. We also propose a notation to represent different pattern variants.

The concept of patterns was introduced by Christopher Alexander [3], who developed a set of patterns in the architectural domain and combined them in a pattern language related to architecture. This triggered the discovery and definition of patterns in many other fields and resulted in a number of pattern collections, in areas such as software design [21], architecture [12, 28] and workflow management [34, 29].

In this paper, we continue the work on patterns in the context of service interaction. The framework proposed in this paper consists of five *pattern families*. Each pattern family contains a large set of *pattern variants* that address the same issue in different ways. A *pattern configuration* is a set of configuration parameters defined for a particular pattern family to differentiate between different pattern variants. By setting configuration parameters to different values from the defined range various different pattern variants can be generated. Figure 1 illustrates main building blocks the given framework consists of, and indicates how many meaningful pattern variants can be generated from a pattern configuration of the pattern families identified. Note that the original thirteen Service Interaction

Patterns [9] correspond to pattern variants belonging to different pattern families presented in this paper.

In order to visualize a pattern variant, for each pattern family we propose an intuitive notation that allows different values of configuration parameters to be graphically depicted. Each configuration parameter is associated with a label depicted differently for each of the values in the predefined range. Thus, the graphical notation serves as a means of visualizing and distinguishing pattern variants.

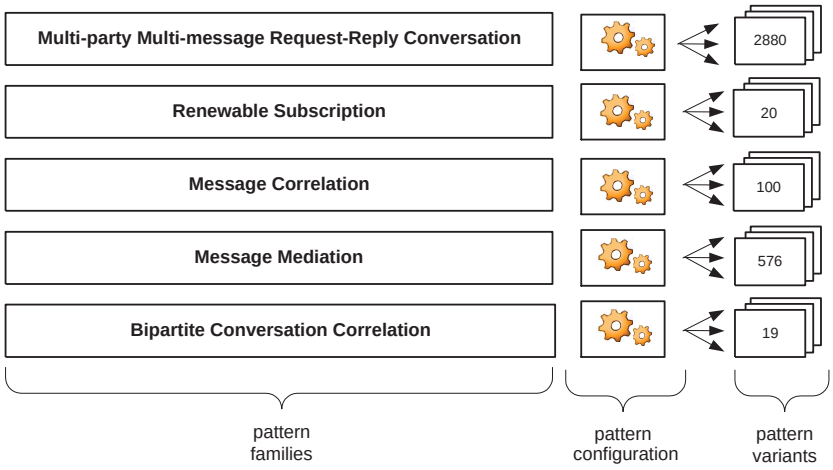


Fig. 1. The configurable framework for service interaction

On the conceptual level, all pattern families are built up around the concepts defined in a meta-model, which serves as the foundation of the framework. Each pattern family augments a set of concepts defined in the meta-model with concepts specific to the pattern context. Besides defining the pattern families on the conceptual level, we formalize the semantics of pattern variants by means of CPNs [23]. For every pattern family, we have developed a set of CPN models and tested them by means of CPN Tools [18].

The five pattern families described in this paper address the following problems.

- The *Multi-party Multi-message Request-Reply Conversation* pattern family considers conversations in which a single party interacts with multiple parties using multiple messages. It addresses the problems of non-guaranteed response and different aspects of message handling. For this family, there are potentially 6912 pattern configurations, althought only 2880 of these are actually meaningful pattern variants.
- The *Renewable Subscription* pattern family addresses problems related to long-running conversations (subscriptions) where either of the two parties

involved in the conversation can take the initiative when initializing and renewing a subscription. In total, 20 meaningful pattern variants can be derived from the pattern configuration of this family.

- The *Message Correlation*, *Message Mediation* and *Bipartite Conversation Correlation* pattern families address the problem of correlation, where the first pattern family addresses the problem at a low-level of abstraction, while the latter two do so at a higher-level of abstraction.
  - In the context of interactions between two tightly-coupled parties, the *Message Correlation* pattern family addresses the issue of correlating incoming messages with previously exchanged messages related to the same conversation. This pattern family combines 100 meaningful pattern variants.
  - The *Message Mediation* pattern family concentrates on the particularities of message exchange between two-loosely coupled parties that interact via an intermediary. This pattern family combines 64 pattern variants for Mediated Introduction and 512 pattern variants for Mediated Interaction where the role of the intermediary is to introduce one party to another or to forward messages back-and-forward between these parties respectively.
  - The *Bipartite Conversation Correlation* pattern family concentrates on message correlation in the context of a long-running conversation between two parties, where knowledge accumulated during message correlation may change in the course of the conversation. This pattern family combines 19 meaningful pattern variants.

Patterns that can be derived by means of the framework proposed in this paper also can be used as a benchmark for evaluating tools and standards. Furthermore, they can serve as a language for communicating problems and solutions within the domain.

The remainder of the paper is organized as follows. Section 2 gives an overview of related work. Section 3 presents the concepts common to each of the pattern families and introduces the format for describing pattern variants. The Multi-party Multi-message Request-Reply Conversation pattern family is described in Section 4. The Renewable Subscription pattern family is described in Section 5. The Message Correlation pattern family is described in Section 6. Scenarios related to the Message Mediation are described in Section 7, and the Bipartite Conversation Correlation is described in Section 8. We evaluate the Web Services Business Process Execution Language (WS-BPEL v2.0) in Section 9. This paper concludes with future work and conclusions in Section 10.

2 Related work

The Service Interaction Patterns documented by Barros et al. [10, 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); and (4) routed interactions.

The Service Interaction Patterns [9] lacked formal semantics, hence their formalization by means of the π-calculus has been proposed by [20]. The majority of these patterns can be interpreted in various ways. Decker and Puhlmann formalize the patterns based on the pattern descriptions and some additional assumptions about the value of variable pattern attributes, however they do not specify the whole range of values the selected pattern attributes may take. Thus, they show the possibility of formalizing certain aspects of service interaction, but in fact do 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 [36] Zaha et al. formulate requirements for a service interaction modeling language, in addition to the ones covered by Barros et al. in [10]. The authors used these requirements for modeling behavioral dependencies between service interactions.

Barros et al. [8] introduced five correlation mechanisms, five conversation patterns, and eight relationships between process instances and conversations that were used for evaluation of standards WS-Addressing and BPEL. The framework presented by the authors does not cover relationships between different process instances. In this paper, we address the latter issue by analyzing correlation on the low- and high-level of abstraction.

Aldred et al. [2] 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 .

In [17] Cooney et al. proposed a programming language for service interaction, which has been used to describe the implementations of two Service Interaction Patterns, i.e. One-to-Many Send-Receive and Contingent Requests [9].

This work is also related to contracting workflows and protocol patterns of van Dijk [33], who proposed a number of protocol patterns for the negotiation phase of a transaction.

Barros et al. [11] proposed a compositional framework for service interaction patterns and interaction flows. They provided high-level models for eight service

interaction scenarios using ASM, illustrating the need to distinguish between different interpretations of the patterns.

The work of Hohpe and Woolf on Enterprise application integration [22] covers various messaging aspects that may be encountered during application integration.

Furthermore, this work relates to the Workflow Patterns Initiative [1, 35], where a set of 43 Control-flow patterns [29], a set of 40 Data patterns [30] and a set of 43 Resource patterns [31] are proposed. In particular, the control-flow patterns have had a considerable influence on the development of new languages, the adaptation of the existing ones and various standardization efforts. This paper should be seen as a part of the Workflow Patterns Initiative.

The work presented in this paper, differs from the work described above in a number of ways. We broaden the scope of the original Service Interaction Patterns and systematically describe various pattern variants. Moreover, we offer a graphical notation that is suitable for representing every pattern variant. To avoid multiple interpretations, we formalize the patterns by means of CPNs.

3 New Service Interaction Patterns

In this section we present a framework for describing new Service Interaction Patterns. Instead of listing all of the patterns identified, we identify the differences between the pattern variants belonging to the same pattern family, i.e. individual pattern variants are obtained by setting the configuration parameters for a particular pattern family.

The patterns we introduce in this paper adopt a consistent presentation format. We depict a pattern variant belonging to a given pattern family with a label which graphically represents a set of attributes. By configuring the attributes, a specific variant of a pattern family is enabled. We introduce the key concepts used in the pattern description by means of a UML class diagram (see Figure 2). The main purpose of this diagram is to represent the pattern at a conceptual level. Later, various parts of this core model will be extended by concepts specific to each of the pattern families.

The meta-model depicted in Figure 2 represents concepts that are common to all of the pattern families identified. For the purpose 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. The association `involves` shows that two or more parties `Party` may be involved in a `Conversation`. The composition relation between `Conversation` and `Message` indicates that at least one message is exchanged in a conversation. 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* as visualized by specializations `Request` and `Reply` of the `Message`. There is an association between `Request` and `Reply` indicating that there may be multiple replies corresponding to a given request. To represent the concepts specific

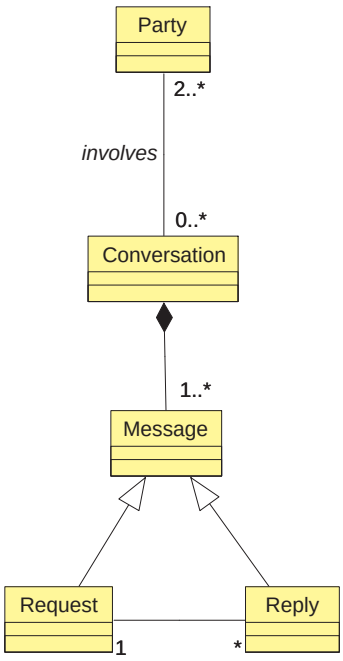


Fig. 2. UML Meta-model of service interaction concepts shared by all pattern families

to a given pattern family we will augment this meta-model by adding relevant concepts for this family.

The attributes that influence the detailed semantics of each pattern variant are described separately. To clarify the semantics of the pattern we use a formalism based on CPNs. For each pattern family we have 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 diagrams.

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

The five pattern families presented in this paper are described using the following format:

- *Description*: of service interaction scenarios combined in the given pattern family.
- *Examples*: illustrating the application of the given pattern variant in practice.
- *UML meta-model*: a description of concepts specific to a given pattern family.

- *Visualization*: a graphical notation representing a pattern configuration and the description of configuration parameters that can be used for configuring the graphical notation to represent specific pattern variants.
- *CPN semantics*: the semantics of a generic pattern illustrated in the form of CPNs presented by means of the screen-shots of the models and their corresponding description.
- *Issues*: that can be encountered when applying a pattern variant from the given pattern family in practice and corresponding solutions.

These five pattern families generate 3595 pattern variants in total. We describe each of the pattern families in detail below.

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

The first pattern family we describe using the format discussed in the previous section is the Multi-party Multi-message Request-Reply Conversation pattern family.
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 also the possibility that a responder will not reply to 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 use in 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

- Requests to submit an abstract or to submit a paper are issued by an editor to a list of 117 people registered for participation in a workshop. Only papers and abstracts submitted before the deadline will be reviewed. If a large number of papers arrive, only the first 50 will be reviewed and only 10 best papers out of the reviewed ones will be published.

UML meta-model An object diagram illustrating the pattern at the conceptual level is presented in Figure 3. A conversation consists of a set of messages (see the composition relation between `Conversation` and `Message`). A conversation involves an initiating process (e.g. requestor), and at least one following process (e.g. responder), depicted by associations `requestor` and `responder`. A 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` message, while the responder returns one or more `Reply` messages or does not react at all. The relation between `Request` and `Reply` messages is depicted by the `corresponds to` association, and the sending of request and reply messages by a party is illustrated by the `is sent by` and `is produced by` dependency relations). Requests issued by a requester can be composite. This means that the

requestor may send several sub-requests in a single message concurrently to a single or to multiple parties.

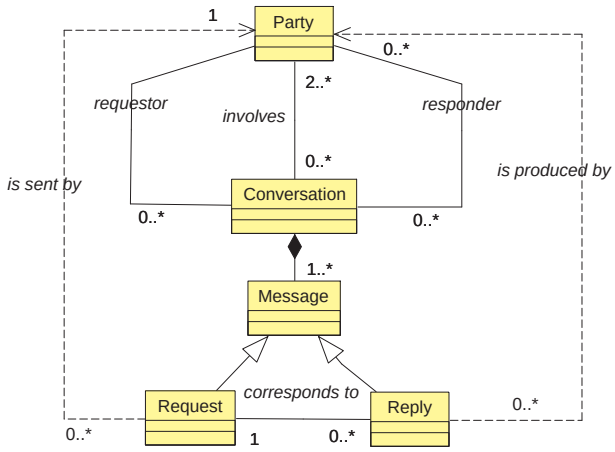


Fig. 3. UML meta-model of
Multi-party Multi-message Request-Reply Conversation

Visualization The graphical notation of the configurable *Multi-party Multi-message Request-Reply Conversation* is given in Figure 4. 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 labels residing within the boundaries of a rectangle representing a party. This graphical notation is used to set the following set of *configuration parameters*:

- 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. For size(N)>1 and size(N)=1 the graphical notations depicted in Fig. 5 (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: For size(M)>1 and size(M)=1 the graphical notations depicted in Figure 5 (2a) and (2b) are used respectively.

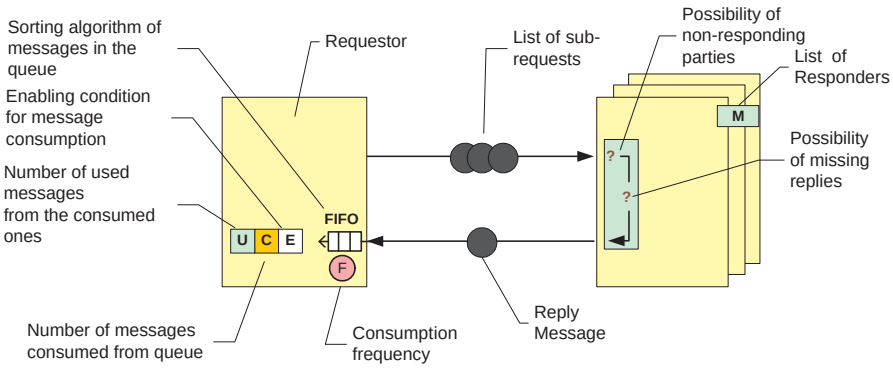


Fig. 4. Graphical notation:
Multi-party Multi-message Request-Reply Conversation

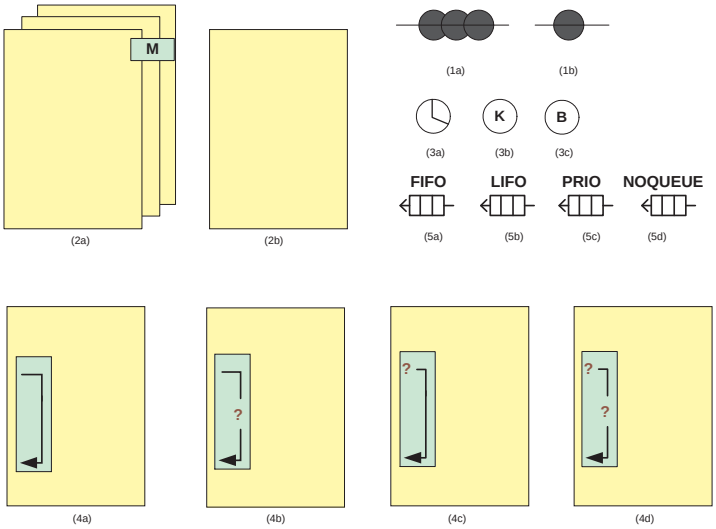


Fig. 5. Variants of graphical notation:
Multi-party Multi-message Request-Reply Conversation

- *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 at all (for example, only interested parties react on the invitation to participate in a social event).

  Default value: No.
  Visualization: Figure 5 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 to all 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 to all sub-requests (for example, the responder answers on all questions in the tax declaration);
  - Yes: responders reply only to some sub-requests (for example, a client subscribes only to two out of five journal offers received).

  Default value: No.
  Visualization: Figure 5 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 the first one in the queue;
  - LIFO: newest message is the last one in the queue;
  - PRIO: sorting based on some criterion (for instance, on price);
  - NoQueue: messages are not queued and are consumed upon arrival if the sender is ready to process them, otherwise they are lost.

  Default value: NoQueue.
  Visualization: Figure 5 (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 a discounted basis are accepted only until the expiration of the discount period);
  - a specified minimal number of messages K ($0<K\leq N$);
  - 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).

  Default value: K=1.
  Visualization: the *E* label residing at the requestor's side in Figure 4 substituted with one of the graphical notations presented in Figure 5 (3a), (3b) and (3c) which denote activation conditions based on a timeout, availability of specific number of messages and a 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 be necessary to leave them on the queue for another process to use);
- S: S messages are removed from the queue such that 0<S<K, where K is the number of replies sufficient for activation of the requester (as specified in the enabling condition);
- All: all messages contained in the queue are removed.

Default value: All.
Visualization: the *C* label residing at the requestor's side in Figure 4 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 the number of messages consumed;
  - All: all consumed messages are used for the processing.

  Default value: All.
  Visualization: the *U* label residing at the requestor's side in Figure 4 substituted with its value.
- *Consumption Frequency* - a parameter specifying the number of times the requestor performs the consumption of messages from the queue.
  Range of values:
  - 1: the requestor is activated only once, after this all remaining and arriving messages are destroyed;
  - FN: the requestor consumes messages FN number of times, 1<FN, after which all remaining and arriving messages are destroyed;
  - ∞: the requestor consumes messages as long as they continue to arrive.

  Default value: 1.
  Visualization: the *F* label residing at the requestor's side in Figure 4 substituted with its value.

The pattern variant representing a scenario in which every parameter is set to the default value is presented in Figure 6. A party A sends a single request to a party B, who sends a reply back. The party A does not queue messages, and consumes them as soon as they arrive. Only one message is necessary for the party A to become enabled and start consumption and processing of the messages. All other messages that may arrive later on will be discarded.
Illustrative example To illustrate how the pattern configuration of the *Multi-party Multi-message Request-Reply Conversation* can be applied in practice, we revisit the example presented earlier for which we describe the corresponding pattern variant by defining values for configuration parameters.

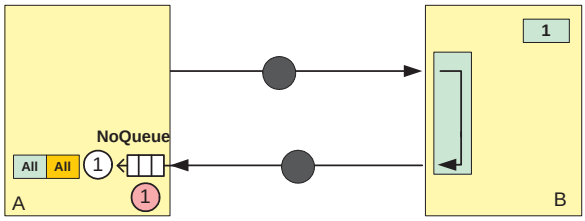


Fig. 6. Notation for the default pattern variant of
the Multi-party Multi-message Request-Reply Conversation pattern family

In this example, the editor plays the role of the requestor and people registered for participation in a workshop represent multiple responders (size(M)=117). The editor sends a request to submit an abstract or to submit a paper, thus the request is of the composite nature and size(N)≥1. Since there is no guarantee that the responders will reply, there is the possibility of non-responding parties. Since not all responders may respond by sending an abstract and a paper, there is also the possibility of missing replies. Reply messages are sorted according to a FIFO policy. The enabling condition for consumption of replies for processing is set to the timeout corresponding to the indicated deadline. The consumption index is set to 50 papers, this means that all other messages will be discarded. The utilization index is set to 10 (only 10 best papers will be reviewed). The consumption frequency is set to 1, no messages are stored in the queue for subsequent processing. The graphical notation representing the pattern configuration for this example is shown in Figure 7.

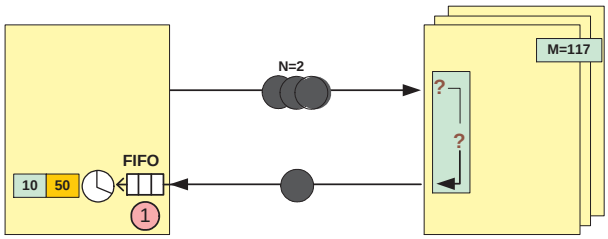


Fig. 7. Notation for the paper submission example

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 8 depicts the top view of the CPN diagram representing the pattern. Requestor and responder are represented as substitution transitions which can be unfolded to the nets depicted in Fig. 9 and Fig. 10(c) respectively. In every conversation the parties exchange requests and replies of type `Message`.

The requestor (whose behavior is shown in Figure 9) can send requests and receive response messages using substitution transitions `Send request` and `Receive`

`response` whose decompositions are shown in Figure 10(b) and Figure 11. A requestor process may have multiple process instances, whose lifecycle is shown in Figure 10(a). Process instances available for participation in a conversation are stored in the `enabled` place. When for a given process instance a conversation is started, a conversation identifier `cid` is coupled with the process instance. The uniqueness of identifiers is ensured by incrementing a counter whose value is stored in the `Conversation counter` place. A process instance chosen for conversation is stored in the `running` place. Transitions `activate`, `deactivate` and `complete` control the status of a process instance during its lifecycle. When an enabled process instance is activated, it has 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 when it is complete and the process instance is added to the `completed` place.

The requestor's `Send Request` sub-page in Fig. 10(b) shows that the requestor, whose identifier is stored in place `Requestor ID`, at the moment of sending a request message creates a new conversation. Function `create_messages()` takes a list of conversation requests `crqs` of the `ConvRequests` type, 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 configuration parameter specifying that messages with N sub-requests are sent to M parties.

Table 1. Data types used in Figures 8 -11

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]

[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`.

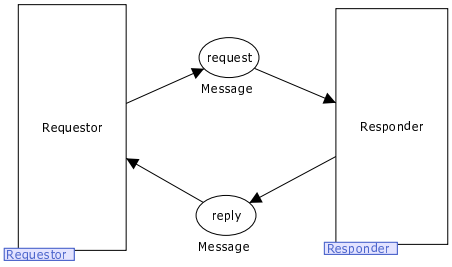


Fig. 8. CPN diagram: The top view of Multi-party Multi-response Request-Reply Conversation

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 messages sent, the number of parties to whom the requests have been sent, and a number of unique messages (i.e. a number of sub-requests can be 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 Figure 10(c) illustrates the behavior of responders involved in the conversation. The identities of the responders are stored in place `self`. They are used to relate incoming requests to the 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.

```
colset Count = int;
colset MTime = int;
colset Status = with active|inactive|enabled|completed; [a]
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; [b]
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;
```

Table 2. Data types (cont.)

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

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

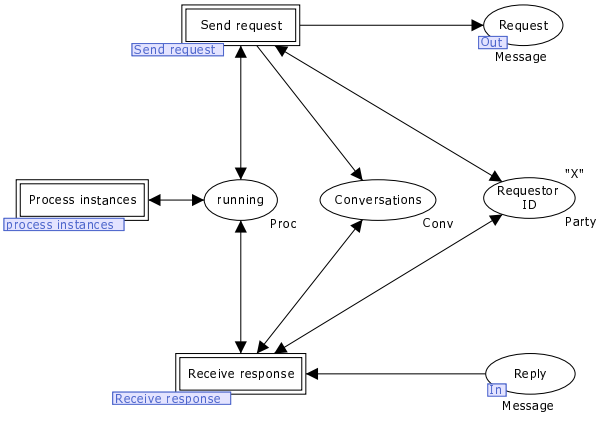
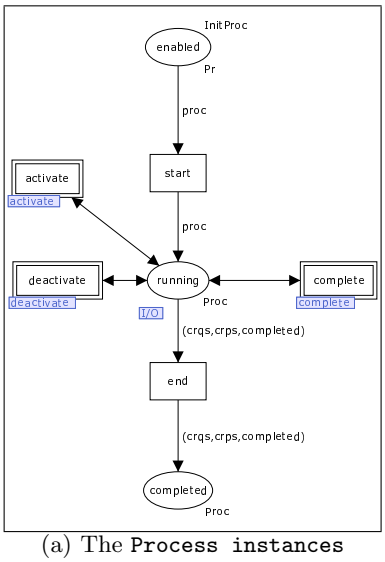
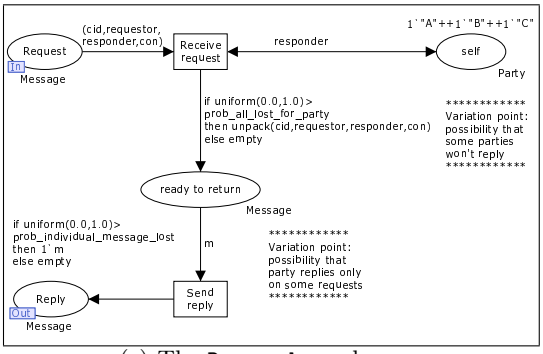


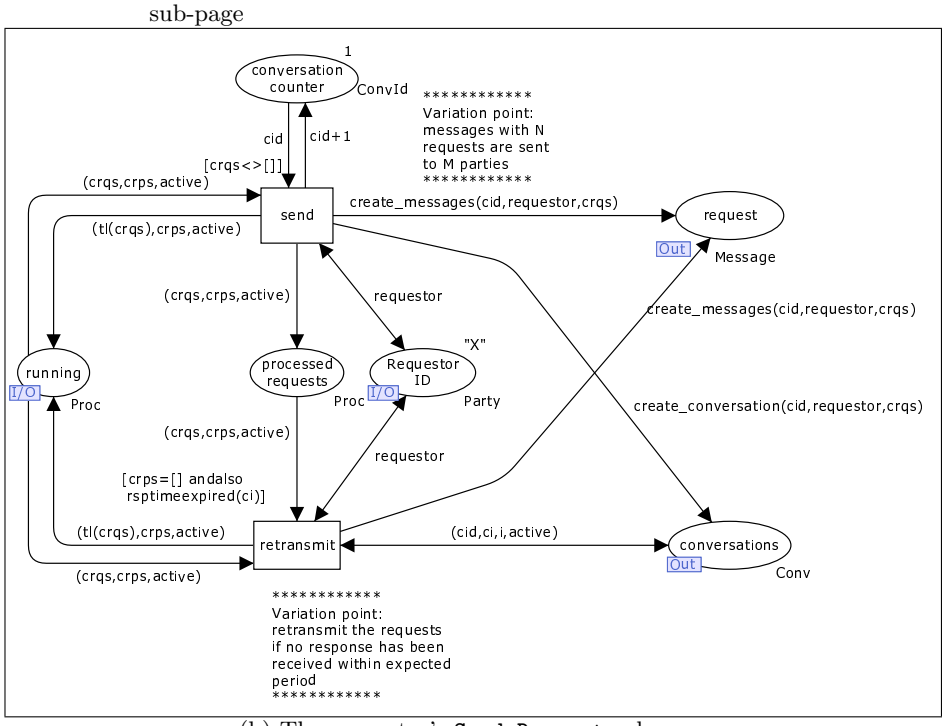
Fig. 9. CPN diagram: The `Requestor` page of Multi-party Multi-message Request-Reply Conversation



(a) The `Process instances` sub-page



(c) The `Responder` sub-page



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

Fig. 10. CPN models of Multi-party Multi-message Request-Reply Conversation

The parameter `prob_all_lost_for_party` corresponds to a configuration parameter 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 configuration parameter to define the probability that a reply will be sent for every unpacked sub-request.

The requestor's `Receive response` sub-page presented in Figure 11 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 the configuration parameter that can be set to any of the queueing disciplines, i.e. LIFO, FIFO or PRIO. If messages should not be sorted (NoQueue), they are consumed immediately.

Function `Consume()` corresponds to the configuration parameter specifying how many messages from the queued ones have to be consumed. One, several, or all available messages in the queue can be consumed. The consumption of messages occurs when the enabling condition (encoded as a guard of transition `Pull`) is satisfied. The `Activated` function can be configured 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.

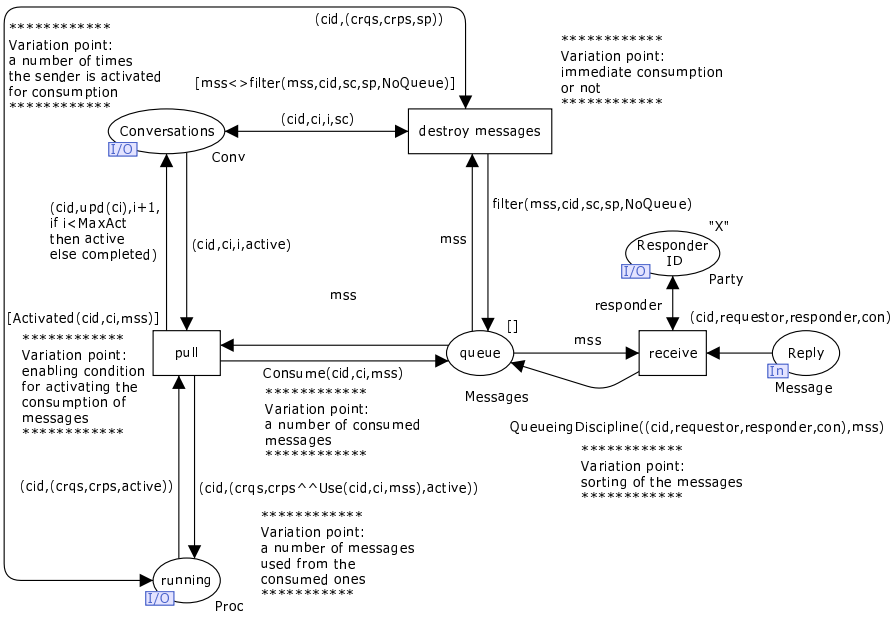


Fig. 11. CPN diagram: The requestor's `Receive Response` sub-page of Multi-party Multi-message Request-Reply Conversation

From the messages consumed only the number of messages defined by the `Use()` function are actually used by the requestor for the processing. This con-

figuration parameter can be set for using either one, several or all consumed messages.

The `MaxAct` parameter corresponds to the configuration parameter 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 `filter()` function. Transition `destroy messages` is used to retrieve messages from the `queue` place 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 the issue of the message correlation may arise while matching replies received with the requests sent. This issue can be addressed by applying a suitable pattern variant from the *Message Correlation* family (see Section 6). If the *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 pattern family *Bipartite Conversation Correlation*.

5 Pattern Family: Renewable Subscription

This section describes the second pattern family named Renewable Subscription.

Description Two parties, a provider and a customer, are involved in a conversation with each other. A provider offers a product under specific subscription terms and a customer consumes the product. Both the provider and the customer may initiate the subscription process by sending a request message. There may be a confirmation/rejection response on the subscription request or no response at all. Depending on terms of subscription, the subscription can be renewed automatically, at the initiative of the customer or the provider.

Examples

- To apply for travel insurance, a client contacts an insurance company. The insurance company informs the client about available types of insurance and duration terms. Once the client has taken out insurance, it is automatically renewed every year. The insurance can be canceled at the client's request at any time.
- A short-term trial newspaper subscription can be extended at the request of a reader.

UML meta-model Concepts specific to the Renewable Subscriptions pattern family are illustrated by means of the UML model in Figure 12. By *subscription* we understand a conversation between two parties, one party offering a product (see the `provider` association between `Party` and `Subscription`) and another party consuming it (see the `customer` association between `Party` and `Subscription`), to establish an agreement for delivery of a certain *product* according to pre-agreed

subscription terms. A party may offer zero or more products and define a set of subscription terms (see aggregation relations between `Party`, and `Product` and `SubscrTerms`). Subscription terms define a subscription period, a number of products to be delivered during this subscription period, a response period within which the customer has to acknowledge the acceptance or rejection of the offered subscription, and how the acknowledgment should be notified. The established subscription relates to one product and a particular set of subscription terms, although these could be the same for multiple subscriptions.

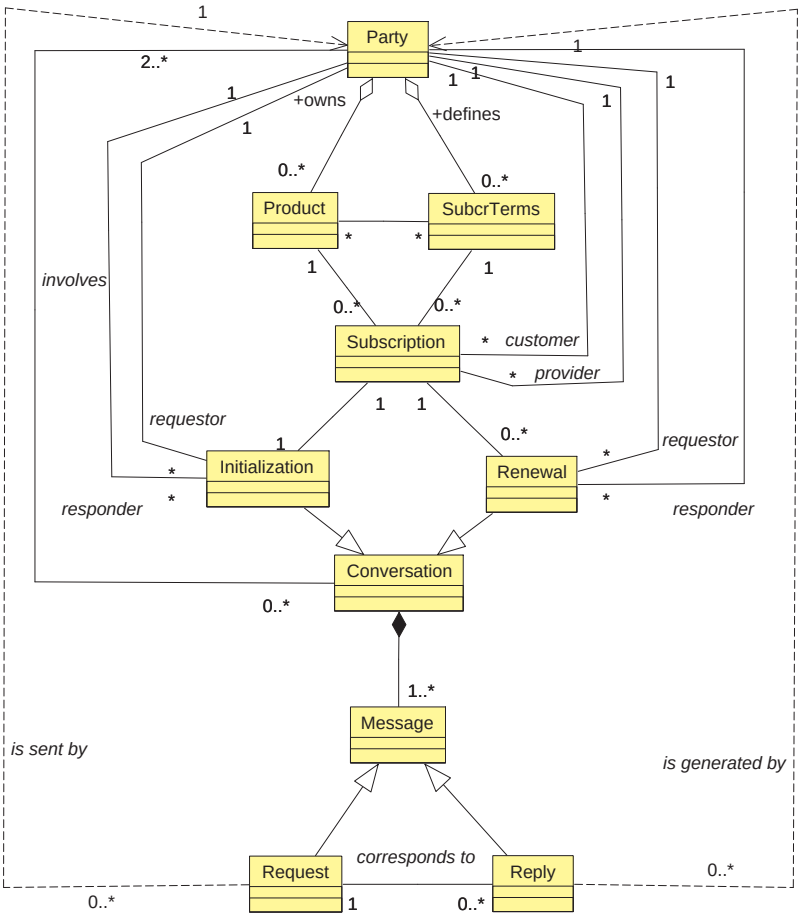


Fig. 12. UML meta-model of Renewable Subscriptions

Each subscription has one initiation phase and may have multiple renewal phases or may not have the Renewal phase at all (see the association relations between `Subscription`, `Initiation` and `Renewal`). Initiation and renewal are conversations (see specialization relation between `Conversation`, `Initiation` and `Renewal`) held for the purpose of establishing and renewing of a subscription respectively. Parties involved in the conversation, play the role of `requestor` or `responder`, where requestor initiates a phase by sending a `Request` message and responder replies with zero or more `Reply` messages.

Visualization Figure 13 illustrates the graphical notation for Subscription Renewal pattern configuration. Depending on who initiates the subscription and who takes the initiative for its renewal, six subscription renewal types can be distinguished. For each subscription renewal type listed in Table 3 there is a separate graphical notation in Figure 13.

The parties are visualized as rectangles with a vertical line in the center of a rectangle representing internal message flow. Directed arrows between rectangles represent the direction in which a party sends a message. A dashed arrow indicates that no reply may be sent back, i.e. the reply is optional. Every message is represented as a black token. The message properties are embedded into the rectangles attached to the message. The time sequence of message exchange corresponds to the time axis. Request and reply messages are denoted as `REQ` and `RPL` respectively. Message indexes `c` and `p` denote that message is sent by the customer or the provider respectively. Message indexes `init` and `renew` denote that the message is related to the initialization or renewal of a subscription, while index `cnlrenew` identifies that the message relates to the cancellation of an automatically renewed subscription.

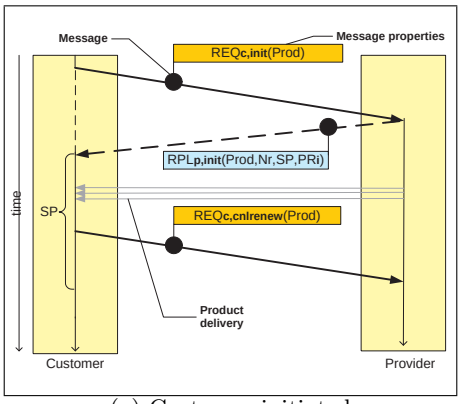
	Subscription type	Initiator	Renewer
(a)	Customer-initiated Automatically-renewed Subscription	Customer	none
(b)	Provider-initiated Automatically-renewed Subscription	Provider	none
(c)	Customer-initiated Customer-renewed Subscription	Customer	Customer
(d)	Provider-initiated Customer-renewed Subscription	Provider	Customer
(e)	Customer-initiated Provider-renewed Subscription	Customer	Provider
(f)	Provider-initiated Provider-renewed Subscription	Provider	Provider

Table 3. Renewable Subscriptions types

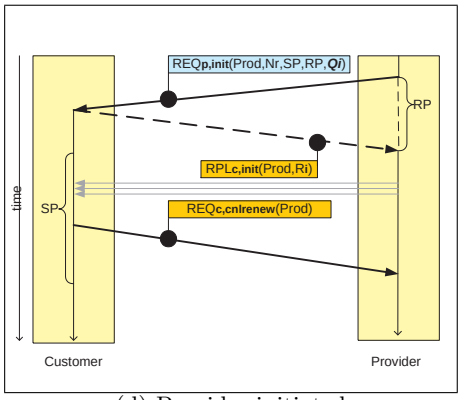
Besides selecting the subscription renewal type, there is a number of configuration parameters[1] that have to be set to a specific value in order to differentiate pattern variants:

- *Expected initiation confirmation*: confirmation expected by the provider on the subscription initiation offer sent to the customer.
  Range of values:

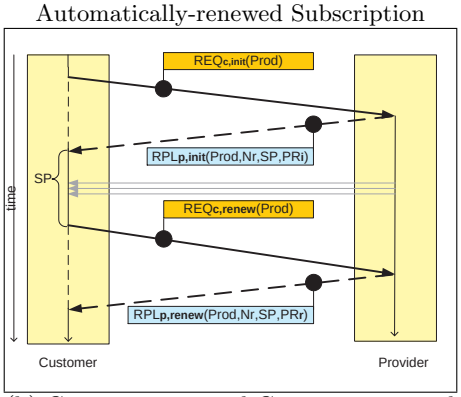
[1] We omit the default values of the configuration parameters, since these do not apply to all subscription renewal types



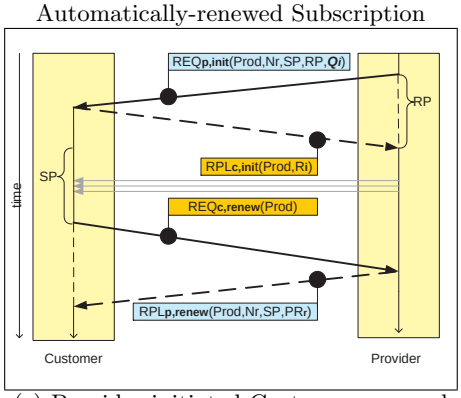
(a) Customer-initiated Automatically-renewed Subscription



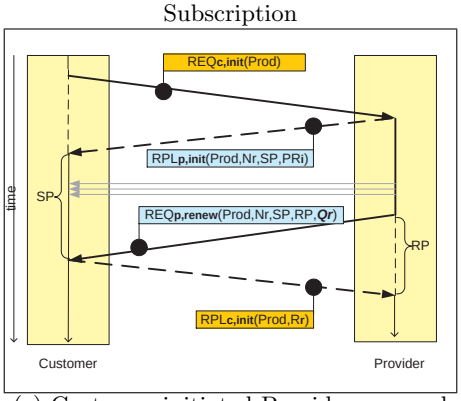
(d) Provider-initiated Automatically-renewed Subscription



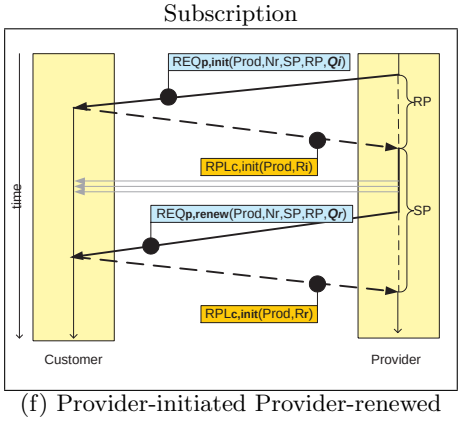
(b) Customer-initiated Customer-renewed Subscription



(e) Provider-initiated Customer-renewed Subscription



(c) Customer-initiated Provider-renewed Subscription



(f) Provider-initiated Provider-renewed Subscription

Fig. 13. Graphical notation: Renewable Subscriptions

- Yes/No: the provider requests that the customer reply with “Yes” or “No” to accept or reject an offer for initiation of a subscription. If no confirmation is received, no subscription is established.

- Yes: the provider requests that the customer reply with "Yes" in order to initiate a subscription. If the expected response is not received, no subscription is established.
- No: the provider requests that the customer reply "No" in order to terminate the initiation of a subscription which is implicitly considered to be established. If expected response is not received, the subscription is considered to be accepted.

Visualization: the *Qi* label in the message properties of an initiation request sent by the provider to the customer, substituted with a suitable value (Figure 13). An example illustrating the provider's request with an expected confirmation "Yes" is presented in Figure 14.

- *Expected renewal confirmation*: confirmation expected by the provider on the subscription renewal offer sent to the customer.
  Range of values:
  - Yes/No: the provider requests the customer to reply with "Yes" or "No" to accept or reject an offer for the renewal of a subscription. If no confirmation is received, no subscription is established.
  - Yes: the provider requests the customer to reply with "Yes" in order to renew a subscription. If the expected response is not received, no subscription is established.
  - No: the provider requests the customer to reply "No" in order to terminate the renewal of a subscription which is implicitly considered to be established. If expected response is not received, the subscription is considered to be accepted.

  Visualization: the *Qr* label in the message properties of a renewal request sent by the provider to the customer, substituted with a suitable value (Figure 13).

Besides the configuration parameters, the graphical notation in Figure 13 also contains a set of dynamic attributes. Values of dynamic attributes may vary for different examples that are based on by the same pattern variant. The dynamic attributes describe characteristics of a subscription such as the period of subscription (SP), the specific product (Prod), the number of products to be delivered (Nr) within the subscription period, and the response period (RP) during which a subscription offer has to be accepted. Furthermore, the customer's response to the subscription initiation offer and on the subscription renewal offer (denoted `Ri` and `Rr` respectively), and the provider's response to the subscription initiation request or the subscription renewal request received from the customer (denoted `PRi` and `PRr` respectively) correspond to behavioral variables that may have different values in different conversations. In particular, the customer may reply on the subscription offer received from the provider with `Yes` or `No`, or not reply at all. When the customer sends a request to the provider to initiate or renew a subscription, the provider may accept or reject the request, or may not reply at all. The behavioral variables are visualized by substituting labels `Prod, SP, RP, NR, Ri, Rr, PRi,` and `PRr`, residing in the properties of messages in the corresponding subscription renewal type in Figure 13, with a suitable value.

The example shown in Figure 14 presents a subscription offer for 4 issues of a journal "Cosmo" that will be delivered within 30 days. The customer is expected to reply on this offer with "Yes" to accept the offer within the response period of 14 days. The values of behavioral variables `Ri` and `PRr` are not specified, because these are set dynamically and thus may take different values for every conversation. In one particular instance of the subscription conversation, the customer could accept the offer from the provider and acknowledge the acceptance by sending "Yes". When the subscription period is about to finish, the customer can request the renewal of the subscription by sending a request specifying the name of the magazine. The provider may acknowledge the acceptance of the renewal request by issuing the "Accept" response, confirming the subscription period and the number of issues to be delivered.

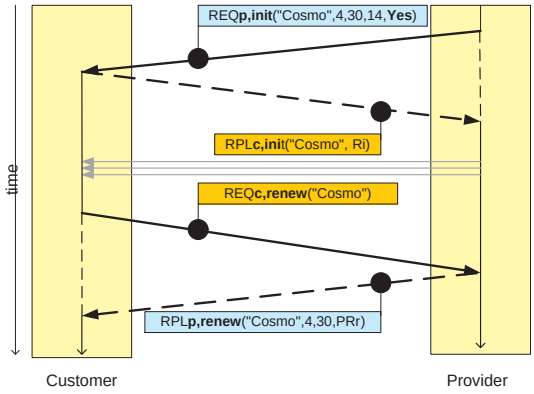


Fig. 14. An example of the Provider-initiated Customer-renewed pattern variant

Illustrative example To describe the pattern variants used in one of the examples listed earlier, let's define the values of the configuration parameters. In the travel insurance example, where a client contacts an insurance company to apply for travel insurance, the insurance is renewed automatically every year. The client and the insurance company map onto the roles of the customer and the provider respectively. This corresponds to the Customer-initiated Automatically-renewed subscription type. The graphical notation for the pattern configuration for the given example is depicted in Figure 15. In its request to the insurance provider, the client specifies the product requested "trvl insurance". The provider indicates to the client in the reply message the number of products to be delivered, i.e. 1, the subscription period of 1 year, and its acceptance response `PRi`, which can be either "Accept" or "Reject". Since the acceptance response is determined dynamically, it's value is not specified. Within the subscription period, the client may send a cancellation request tot the insurance provider to cancel the insurance.

CPN semantics In this section, we describe in detail the semantics of one Provider-initiated Customer-renewed subscription scenario. As they are similar

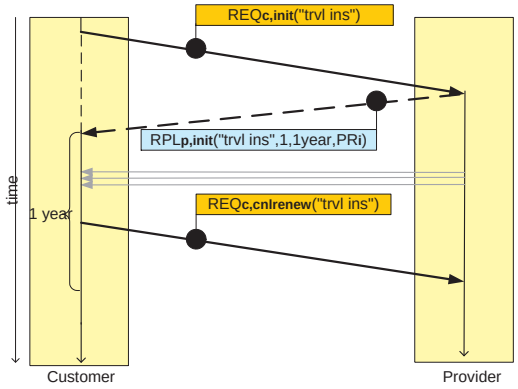


Fig. 15. Notation for the travel insurance example

in form, the CPN models for the remaining subscription renewal scenarios are listed in Appendix B.

Figure 5 illustrates the top view of the Provider-initiated Customer-renewed subscription scenario. Customer and provider are represented by substitution transitions with corresponding names that unfold to the sub-pages presented in Figure 16 and Figure 17 respectively. The definition of data types is is based on the concepts and notation introduced earlier. The subscription scenario is based on the assumption, that there is a one-to-one relation between customer and provider. This means that from the customer perspective, conversations are performed with a single provider, and the same holds for the provider. This implies that customer and provider do not need to specify their identities in the messages exchanged (this could easily be added if desired). However, since multiple subscriptions can be established between customer and provider for the same product, the subscriptions have to be differentiated. For this purpose, a subscription identifier of type `SID` is introduced in the messages exchanged.

The customer, whose behavior is presented in Figure 16, receives an initialization request $\texttt{req}_{pinit}$ and puts it in the place `Subscription offered` by means of transition `Receive init request`. If the customer decides to reply, function `createcinitreply()` generates an initialization reply message of type `RPLcinit`. A variable `r` of type `R` indicates whether the request is accepted, rejected or ignored. If the customer accepts the subscription, the subscription details are recorded in place `Subscription established` by means of the function `recordsubscr()`. Products sent by the provider are received by the customer via transition `Receive product`, which examines whether the product delivered is the product expected by the customer by means of the function `productforme()`. When the last product has been received by the customer, the customer generates a renewal request via function `createcrenewreq()` and sends it to the provider by transition `Send renew request`. In order to match future replies from the provider with the request sent, the customer stores the subscription request in place `Subscription requested`. When a renewal reply $\texttt{rpl}_{prenew}$ is received from

the provider, the function `replyforcustomer()` locates a corresponding request. If the provider accepted the renewal request, the function `updatesubscr()` records the details of the subscription renewed in the place `Subscription established`. From this moment on, the customer may continue receiving products and may perform subscription renewal requests again.

The provider, whose behavior is presented in Figure 17, initiates the conversation with the potential customer by sending an initialization request for subscription $\texttt{req}_{pinit}$ of type $\texttt{REQ}_{pinit}$. Decomposition of a substitution transition `Send`

Table 4. Data types used in Fig. 5-18

colset Prod = string; [a]
colset RP = int; [b]
colset SP = int; [c]
colset Nr = int; [d]
colset Q = with YesNo|Yes|No; [e]
colset R = with Yes|No|none; [f]
colset PR = with Accept|Reject|Neglect; [g]
colset SID = INT; [h]
colset REQpinit = product SID * Prod * Nr *RP * SP * Q; [i]
colset REQprenew = product SID * Prod * Nr * RP * SP *Q; [j]
colset REQcinit = Prod; [k]
colset REQcrenew = product SID * Prod; [l]
colset RPLpinit = product SID * Prod * Nr * SP * PR; [m]
colset RPLprenew = product SID * Prod * Nr * SP * PR; [n]
colset RPLcinit = product SID * Prod * R; [o]
colset RPLcrenew = product SID * Prod * R; [p]

[a] Product name
[b] Response period
[c] Subscription period
[d] Number of products to be delivered
[e] Confirmation expected by provider
[f] Confirmation sent by customer
[g] Confirmation sent by provider
[h] An identifier for distinguishing identical subscriptions
[i] Init. request of provider
[j] Renewal request of provider
[k] Init. request of customer
[l] Renewal request of customer
[m] Init. reply of provider
[n] Renewal reply of provider
[o] Init. reply of customer
[p] Renewal reply of customer

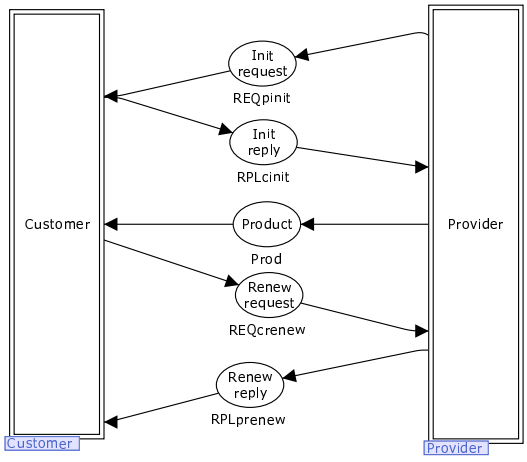


Table 5. CPN diagram: The top view of Provider-initiated Customer-Renewed Subscription

`init request` is presented in Figure 18. The `createpinitrequest()` function generates an initialization request based on the product offers available at the provider in `Product offers` place and a variable `q` of type `Q` that represents the confirmation expected by the provider on the given request. This is a configuration parameter that is set dynamically to one of the values of the small color-set `Q`. Subscription offers sent are stored in place `Offered subscription`.

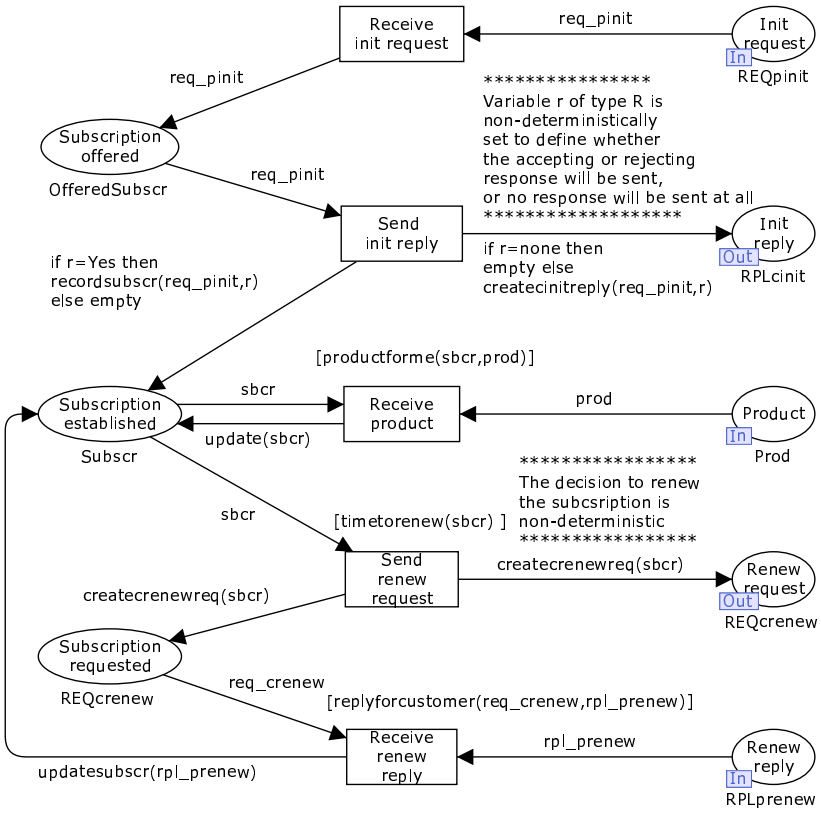


Fig. 16. The `Customer` page of Provider-initiated Customer-Renewed Subscription

colset Offer = product Prod * Nr * RP * SP * Ren; [a]
colset Subscr = product SID * Prod * Nr * SP *Ren; [b]
colset OfferedSubscr = product SID * Prod * Nr * RP * SP * Q * Ren; [c]
colset RequestedSubscr = product SID * Prod; [d]

[a] Offers of provider
[b] Established subscription
[c] Subscription offered
[d] Subscription requested

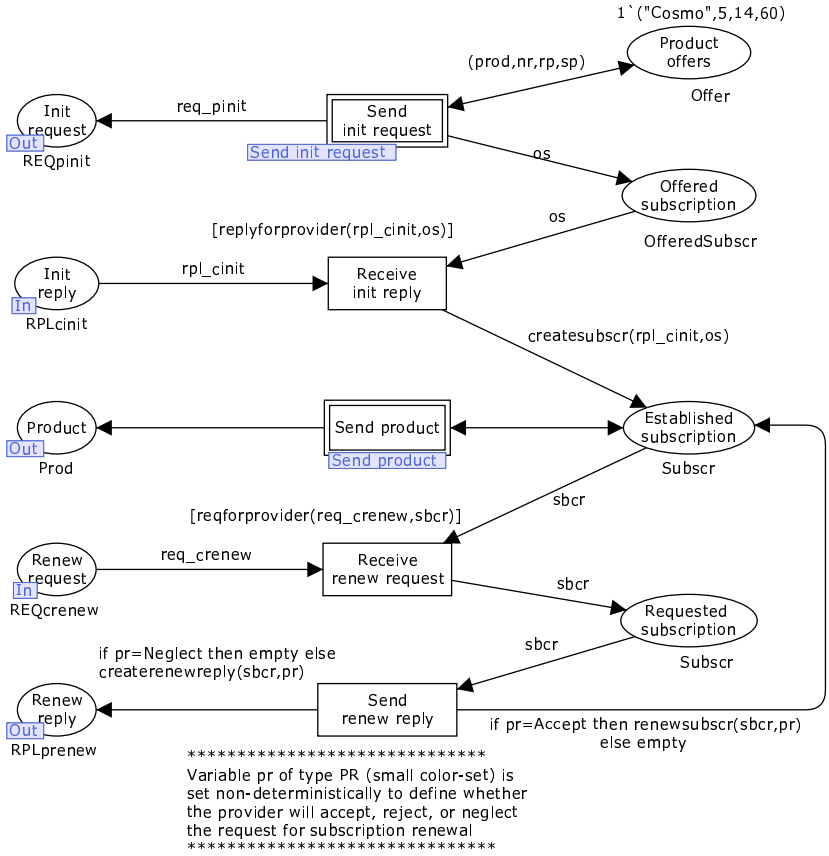


Fig. 17. CPN diagram: The `Provider` page of Provider-initiated Customer-Renewed Subscription

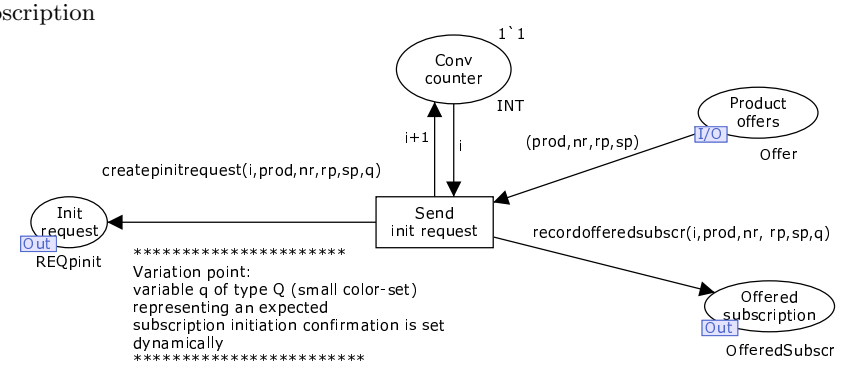


Fig. 18. CPN diagram: The `Send init request` sub-page of Provider-initiated Customer-Renewed Subscription

When a customer replies on the initialization request, the provider at first examines the reply message ($\texttt{rpl}_{cinit}$) received. The `replyforprovider()` function checks whether the reply received corresponds to any of the requests sent. Then, for correlated requests the subscription details are recorded by the `createsubscr()` function in the `Established subscription` place if the customer confirms the acceptance of the subscription. For established subscriptions, the provider delivers a specified number of products `nr` during the agreed subscription period `sp`.

Products are sent to the customer via transition `Send product`. When the provider receives a renewal request $\texttt{req}_{crenew}$ from the customer, it examines by the `reqforprovider()` function whether there is a corresponding subscription that could be renewed. A subscription which can be renewed is stored in place `Requested subscription`. If the provider decides not to neglect the reply, a reply message is created by the `createrenewreply()` function, and if the provider accepts the request received, the subscription details are recorded by the `renewsubscr()` function to the `Established subscription` place.

Issues The pattern variants belonging to a family of *Renewable Subscription* consider one-to-one relation between a customer and a provider. However, in many real life scenarios a customer may have multiple subscriptions with the same or different providers and a provider may have multiple subscriptions with the same or different customers. Such context conditions obviously require a deep insight into the message correlation issue. The issues of correlation can be addressed by combining a *Renewable Subscription* pattern variant with a suitable pattern variant from the families of *Message Correlation* or *Bipartite Conversation Correlation*. The involvement of multiple providers and customers as well as requests for multiple subscriptions can be expressed by combining a renewable subscription pattern variant with the pattern variants of the *Multi-party Multi-message Request-Reply Conversation* family.

6 Pattern Family: Message Correlation

The third pattern family, named Message Correlation, addresses issues of correlation at a lower-level of abstraction. This family is described in this section.

Description A Party communicating with other parties has to handle incoming messages in accordance with a history of established conversations. Message Correlation is the act of identifying the relevant conversation for a message received by the Party.

Example

- An insurance company handles claims for refund of lost baggage, medical costs, etc. When a claim is received, an insurance advisor determines whether the client has a valid insurance policy and whether there are any records related to the claim received. If the client has no valid insurance, the advisor may provide the client with a new policy or may refuse to handle the claim.

UML meta-model Concepts specific to the Message Correlation pattern family are illustrated by means of the UML diagram in Figure 19. A Party may partic-

ipate in multiple conversations with other parties (illustrated by the association relation between `Party` and `Conversation`). The Party may send messages to and receive messages from other parties (see the relations `send` and `receive` between `Party` and `Message`). Messages exchanged can be either of type `Request` or `Reply`, where every request may have multiple replies, but each reply corresponds to exactly one request (see dependency relation between `Request` and `Reply`). It is an assumption of this pattern that messages exchanged between parties contain information about the sender, the receiver and as well as additional content in the format (From, To, Content). The party sending out a message determines what information relating to the sender's and receiver's identity it wants to reveal in the message.

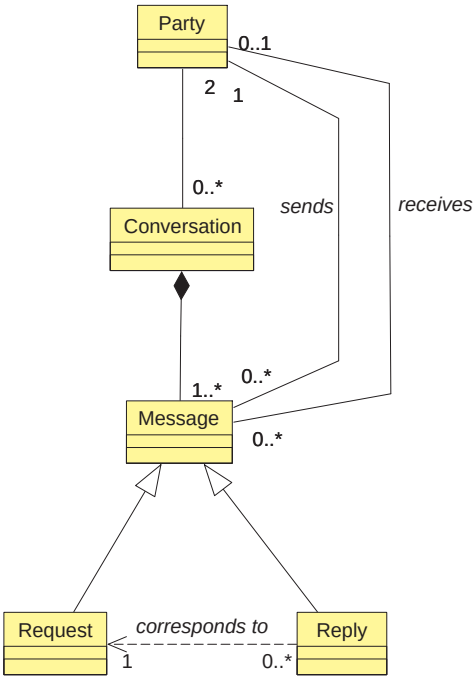


Fig. 19. UML Meta-model of Message Correlation

Visualization Figure 20 illustrates the graphical notation for Message Correlation. A party is visualized as a rectangle. The direction of arrows linked to the party node indicates the direction of message flow. Information about the message sender contained in the message is enclosed in the `From`-field. Information about the message receiver contained in the message is enclosed in the `To`-field. Information the receiving party has about its own credentials before correlating

a message received is enclosed in the `Me` field. Information the party has about the credentials of the other party involved in a conversation is enclosed in the `You` field. Information the party has about its own identity and the identity of the other party from whom a message has been received after message correlation is enclosed in fields `Me` and `You`' respectively.

This graphical notation contains a set of static attributes and configuration parameters. Both static attributes and configuration parameters have to be configured for each of the pattern variants. For all pattern variants, the value of static attributes is fixed and does not change, while configuration parameters can be configured in accordance with values in the specified range.

Values of the static attribute representing the knowledge of the receiving party about own identity before and after message correlation are denoted by the `Me` label. `Me` is a pair (P_r, C_r) where P_r denotes the id of the party-receiver and C_r denotes the id of the conversation used by the receiving party to correlate the message received. To illustrate a pattern variant, the `Me` label is substituted with the value (P_r, C_r).

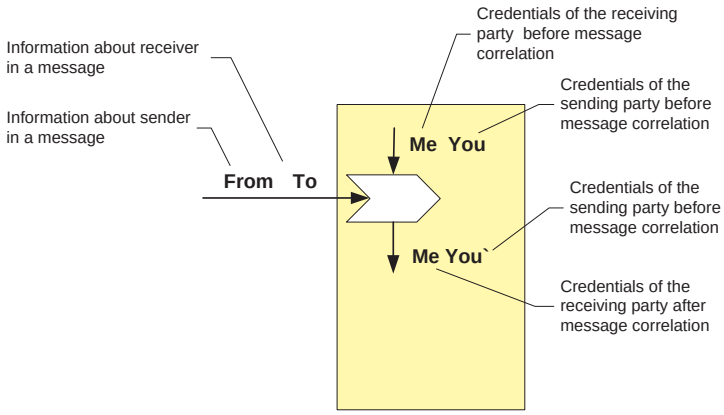


Fig. 20. Graphical notation:
Message Correlation

The graphical notation shown in Figure 20 illustrates that this pattern family has the following set of configuration parameters:

- *Message Sender field*: the extent of the information revealed by the sender of a message regarding its identity.
  Range of values: `From` is a tuple of potential sender identifier and conversation identifier. The sender identifier is denoted P_s and the conversation identifier used by the sender for correlation purposes is denoted C_s. Either the id of the sender, the id of the conversation or both can be missing in the `From`-field. Missing information can be either intentionally or accidentally underspecified by the message sender (for instance, when a party wants to hide its id or when it forgets to include some information). Missing information is denoted as $\perp$.

So, possible values of the `From`-field are $(P_s, C_s), (P_s, \perp), (\perp, C_s)$ and $(\perp, \perp)$.
Default value: (P_s, C_s) (i.e. both the party identifier and the conversation are supplied by the sender).
Visualization: the `From` label shown in Figure 20 substituted with a suitable value. An example specifying the default value of the message sender field is shown in Figure 21.

- *Message Receiver field*: the extent of information specified by the sender of a message regarding the receiver's identity.
  Range of values: `To` is a tuple comprised of the intended receiver identifier and its conversation identifier. The receiver identifier is denoted P_r and the conversation identifier used by the receiver for correlation purposes is denoted C_r. Either the id of the receiver, the id of the conversation or both can be omitted in the `To`-field. So, possible values of the `To`-field are $(P_r, C_r), (P_r, \perp), (\perp, C_r)$ and $(\perp, \perp)$.
  Information specified by the message sender about the identity of the receiving party is required by the message receiver to uniquely identify a conversation related to the message received.
  Default value: $(P_r, \perp)$ (i.e. only the id of the intended party is specified).
  Visualization: the *To* label shown in Figure 20 substituted with a suitable value. An example specifying the default value for the message receiver field is shown in Figure 21.
- *Credentials of the message sender before message correlation*: receiver's knowledge in regard to the credentials of the sending party involved in the conversation with the receiving party.
  Range of values: `You` is a tuple including the potential message sender identifier and its conversation identifier. The same notation as introduced earlier is used to denote the id of the party-sender and its conversation identifier. Since the receiver's information about the message sender may be incomplete, possible values of the `You`-field are $(P_s, C_s), (P_s, \perp), (\perp, C_s)$ and $(\perp, \perp)$.
  Default value: $(P_s, \perp)$ (i.e. the receiving party has knowledge only about the identity of the sending party).
  Visualization: the `You` label shown in Figure 20 substituted with a suitable value. An example specifying the default value of the message receiver field is shown in Figure 21.
- *Credentials of the message sender after message correlation*: information about the credentials of the sending party involved in the conversation with the given receiving party, updated after message correlation.
  Range of values: `You‘` is a pair comprised of the possible sender identifier and its conversation identifier. Since information the party has about the sender id and its conversation id available before the message correlation might be incomplete, some of the missing knowledge can be gained by the receiving party from the information provided in the message sender field. For instance, if `You`$=(\perp, \perp)$ and $From = (P_s, C_s)$, then `You'`$= (P_s, C_s)$ if all missing information is recorded. Note that some of the missing information may be forgotten, and the resulting value of $You‘$ may be $(P_s, \perp), (\perp, C_s)$ or may

even remain unchanged $(\perp, \perp)$. Table 6 illustrates possible values of the `You`‘ field calculated based on the information available in the `You` field, the information provided in the `From` field and the possibility of not recording the provided information.

Table 6. Enumeration of all scenarios regarding possible information gained

From	You	You'
(Ps, Cs)	(Ps, Cs)	(Ps, Cs)
	$(Ps, \perp)$	(Ps, Cs)
		$(Ps, \perp)$
	$(\perp, Cs)$	(Ps, Cs)
		$(\perp, Cs)$
	$(\perp, \perp)$	(Ps, Cs)
		$(Ps, \perp)$
		$(\perp, Cs)$
		$(\perp, \perp)$
$(Ps, \perp)$	(Ps, Cs)	(Ps, Cs)
	$(Ps, \perp)$	$(Ps, \perp)$
	$(\perp, Cs)$	(Ps, Cs)
		$(\perp, Cs)$
	$(\perp, \perp)$	$(Ps, \perp)$
		$(\perp, \perp)$
$(\perp, Cs)$	(Ps, Cs)	(Ps, Cs)
	$(Ps, \perp)$	(Ps, Cs)
		$(Ps, \perp)$
	$(\perp, Cs)$	$(\perp, Cs)$
	$(\perp, \perp)$	$(\perp, Cs)$
		$(\perp, \perp)$
$(\perp, \perp)$	(Ps, Cs)	(Ps, Cs)
	$(Ps, \perp)$	$(Ps, \perp)$
	$(\perp, Cs)$	$(\perp, Cs)$
	$(\perp, \perp)$	$(\perp, \perp)$

Default value: (P_r, C_r).
Visualization: the `You'` label shown in Figure 20 substituted with a suitable value. An example specifying the default value for the message receiver field is shown in Figure 21.

Figure 21 illustrates the graphical notation used for the Message Correlation pattern variant where all configuration parameters are set to the default values.

Illustrative example To illustrate the example presented earlier, we identify values for the configuration parameters of the *Message Correlation* pattern configuration, and show the graphical notation for the given pattern configuration (see Figure 22). In the insurance claims handling example, when a claim is received, an insurance advisor determines whether the client has a valid insurance policy and whether there are any records related to the claim received. Let `Ps`

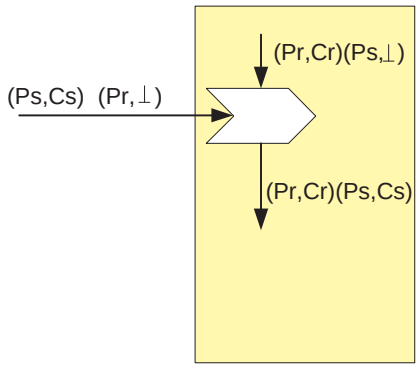


Fig. 21. Default notation: Message Correlation

be an identity of the client specified by the client, `Cs` be the insurance number specified by the client for handling the claim, `Pr` be the identity of the insurance advisor, and (Cr) be a client number associated with the client used within the insurance company. It is the assumption in this example that the client has a valid insurance. The client specifies both information about their identity, the insurance number in its claim, and the identity of the insurance company. Since no correlation-related information is gained by the insurance advisor from the incoming message, the knowledge about the client after handling the claim does not change.

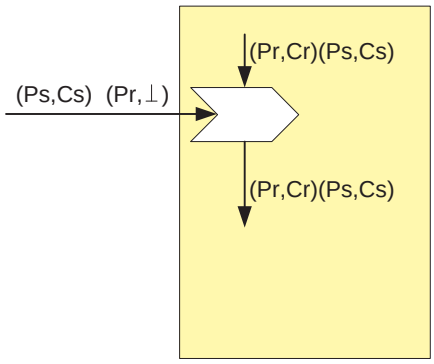


Fig. 22. Notation for the claim handling example

CPN semantics In this section, we formalize the semantics of Message Correlation by means of CPN. The CPN diagram presented in Figure 23 illustrates the act of correlating an incoming message of type `Message` with the history of existing conversations `Conv` via transition `Correlate`. Messages that have to be processed by the party are supplied in the form `(from,to,cont)`, where `from` (of type `From`) represents the identity of the message sender; `to` (of type `To`) represents the identity of the party to which the message is dedicated, and `cont` (of

type `Cont`) represents the content of the message. The knowledge based on the history of existing conversations is available to the party before a received message has been correlated has the form `(me,you,cont_old)`. The element `me` of type `Me` represents the knowledge of the receiving party itself. The element `you` (of type `You`) represents the knowledge about the message sender from previous interactions. The element `cont_old` (of type `Content`) represents the content of all messages previously exchanged in the given conversation.

Values `NoPartyId` and `NoConvId` (which correspond to the symbol $\perp$ introduced in the pattern visualization section above) denote the absence of the information about the party identifer and conversation identifier respectively. Note that a union type is used to incorporate such "missing values".

The `abletocorrelate()` function matches information supplied in the incoming message with the history of previous and current conversations. In particular, matches are performed between variables `to` and `me`, and `from` and `you`. In this example, the `abletocorrelate()` function performs correlation by matching the identities of the message sender and message receiver supplied in the message with the corresponding identities available to the party-receiver in the history of conversations. Note that instead of correlating based on matching the infor-

Table 7. Data types used in Fig. 23

```
colset PartyId = union smallstr + NoPartyId; [a]
colset ConvId = union smallint + NoConvId; [b]
colset PxC = product PartyId*ConvId;
colset Content = string;
colset From = PxC;
colset To = PxC;
colset Me = PxC;
colset You = PxC;
colset Conv = product Me*You*Content;
colset Message = product From*To*Content;
```

[a] The id of a party is a small color-set of a certain type (STRING in this case) or is denoted by NoPartyId if not specified.

[b] The id of a conversation is a small color-set of a certain type (INT in this case) or is denoted by NoPartyId if not specified.

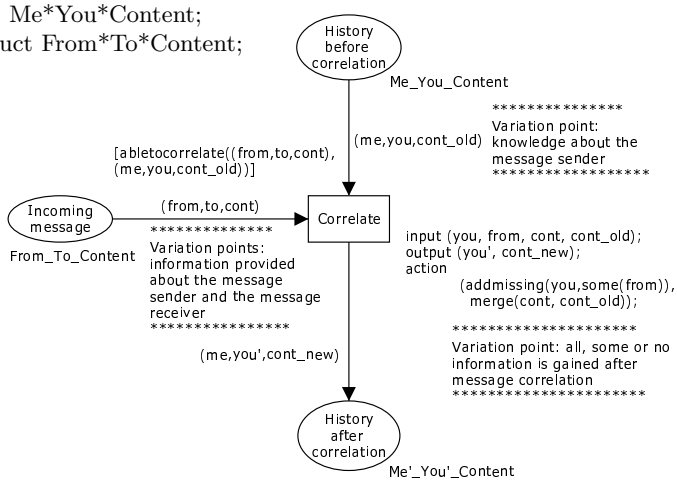


Fig. 23. CPN diagram: Message Correlation

mation contained in the incoming message with local knowledge of the party, also some analysis of the message content can be performed (this issue is discussed in more detail in the next paragraph). If correlation is successful, i.e. the `abletocorrelate()` function has identified a conversation to which the message received can be uniquely correlated, then the history of existing conversations is updated. In particular, the `merge()` function adds the content `cont` of the message received to the old content `cont_old` of the identified conversation. The `addmissing()` function identifies if there is any missing information regarding the credentials of the message sender, and where such information is identified, adds it to the existing information. Note that since not all information provided necessarily has to be recorded, the `some()` function defines how much of the missing information will be actually recorded.

In the net presented in Figure 23, messages which can not be correlated are blocked in the `Incoming messages` place. This net could be extended with mechanisms for handling correlation failure. In particular, constructs could be added for creating a new conversation or for discarding an incoming message for which no corresponding conversation has been found in the history of existing conversations.

Issues The *Message Correlation* pattern family identifies the issues experienced by a party receiving a message that must be correlate. The given pattern family considers correlation between two tightly-coupled parties. When fewer dependencies are desired between the parties, the tight binding between them may be relaxed. The issue of message mediation among two loosely-coupled parties are addressed in the *Message Mediation* pattern family. Furthermore, the *Message Correlation* pattern family concentrates on a single interaction between two parties in the context of already existing conversation. Thus, it does not provide any insight into how information used for correlation changes during the course of the conversation. The issue of varying correlation information in the context of a bipartite conversation is addressed in the *Bipartite Conversation Correlation* pattern family.

From a functional point of view, message correlation is performed by a party using either a *key-matching* or a *property-analysis* approach. A party may use key-matching to uniquely identify a conversation related to the message received. For this, message fields *From* and *To* are used as keys for correlation. The mapping of these keys to the values of `You` and `Me` should be unique, i.e. no two conversations can be identified as a result of applying key-matching. In situations, where key-matching results in a non unique mapping, some additional analysis of the message content might be needed. The property-analysis approach is used to identify a conversation related to the message received based on the examination of the message content. For analysis purposes, other information available to the party may also be used. Based on the content of the message (for instance, message id, conversation id, time-stamp, etc.) together with other information available to the party it may be possible to determine the conversation to which the message received relates. Note that in this section we only considered key matching.

The perfect scenario leading to successful correlation would contain the maximal possible information for all configuration parameters. However, in real-life situations not all information is guaranteed available. For instance, information specified by the message sender in the message may be incomplete or information available to the receiver about the sender may be missing. If no identifier for the receiving party is specified in the message, no guarantee can be given that the message will be delivered to the right party. If the message sender does not disclose its id in the message, there is no guarantee that the follow-up response will be correctly delivered. If no conversation id, used by the receiver for correlation purposes, is contained in the message, then there is the possibility of it being correlated with the wrong conversation.

An exceptional situation may occur if, as a result of correlation, no conversation related to the message received can be identified (i.e. in case of the correlation failure). To deal with such an exception, a party may *create* a new conversation or *discard* the message received.

The majority of existing products rely on the key-matching approach, i.e. they perform correlation based on matching of information supplied in the message with information available to the party. These products discard messages which do not provide complete information about the credentials of the message receiver, while the identity of the party or the conversation identifier could be inferred based on analysis of available message content.

7 Pattern Family: Message Mediation

In this section, we describe the fourth pattern family called Message Mediation. This pattern family concentrates on mediation of messages related to a conversation between two loosely-coupled parties.

Description A party requesting a service may not know the credentials of the party providing this service or may know the party credentials but not be willing to engage in a message exchange with them directly. To establish a conversation between two loosely-coupled parties, the customer and the provider, a third intermediary party named "the mediator" is required. The customer posts a request to the mediator and expects the response to be received back. The mediator forwards both requests from the service customer to the service provider and replies from the provider back to the customer. Alternatively, the mediator may provide a party with details of the credentials of the other party involved in the conversation to allow for future interactions to occur directly between them.

Examples

- To organize a business trip, a manager asks their secretary to arrange the trip and provide itinerary details when they become available.
- To order office equipment an employee contacts a secretary, who forwards the employee's request to a supplier. After processing the request, the supplier delivers the equipment ordered directly to the employee.
- Air-miles card owners may exchange accrued air-miles for reservation of flights provided by one of the airline partners. The miles-to-flight exchange is per-

formed via the Air-miles web-site, which mediates data exchange between the client and the airline operator. The itinerary details provided to the client contain contact details of the airline operator for any future enquiries regarding the booking.

UML meta-model Concepts specific to Message Mediation are illustrated by means of the UML diagram in Figure 24. A conversation between a customer and a provider involves a third party, i.e. a mediator (the three parties involved in a conversation are illustrated by the association relations between `Party` and `Conversation`). Each party involved in the conversation may send and receive messages (see the association relations between `Party` and `Message`). Messages exchanged can be either of type `Request` or `Reply`, where each reply corresponds to one request, and one request may have multiple replies.

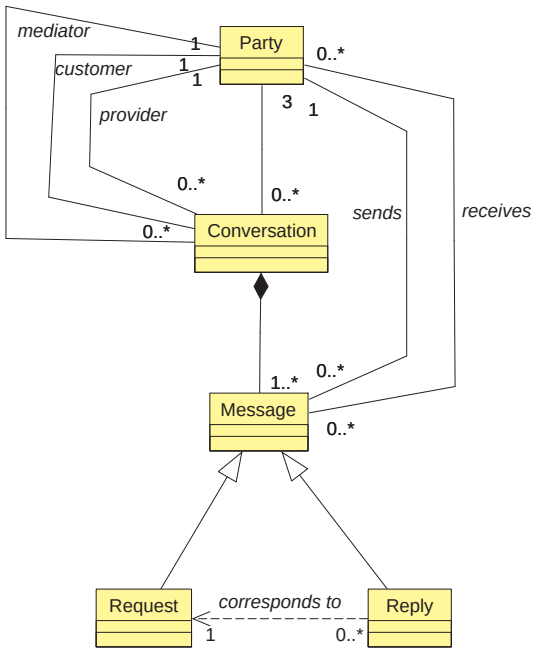


Fig. 24. UML meta-model of Message Mediation

Visualization The graphical notation for Message Mediation is presented in Figure 27. The mediator may play two different roles in the tripartite conversation: (1) forward both a request from the customer to the provider and the provider's reply back to the customer; (2) route a request from a customer to a provider and supply the provider with a reference to the customer's credentials (so that the provider can reply to the customer directly). The role of the mediator is one of

the configuration parameters that have to be set in order to distinguish between two types of message mediation: Mediated Interaction and Mediated Introduction denoted using the notation presented in figures 29 and 27 respectively. This notation is based on the graphical notation used for the Message Correlation pattern family. It is assumed that messages exchanged between parties involved in the tripartite conversation are of the format `(From, To, Them, Expose, Content)`, where `From` represents the credentials of the message sender, `To` represents the credentials of the message receiver, `Them` identifies the third party involved in the given conversation, `Expose` indicates permission to reveal the credentials of the message sender to the third party in the conversation, and `Content` represents additional content of the message. Note that compared to the basic correlation scenario described in the previous section, there are two additional fields: `Them` and `Expose`.

Both graphical notations contain a set of configuration parameters (i.e. parameters that have to be set to a specific value in order to configure a pattern variant), a set of static attributes (i.e. pattern attributes whose value is fixed for all pattern variants derived from the pattern configuration), and a set of dynamic attributes (i.e. pattern attributes whose value is derived from other pattern attributes once all configuration parameters for a specific pattern variant have been set to a specific value).

For the sake of clarity, we describe the pattern configurations for Mediated Introduction and Mediated Interaction separately.

Let $P1$, $P2$, and $P3$ denote the party identifiers of the customer, the mediator, and the provider respectively. Let $C1$, $C2$, and $C3$ denote the conversation identifiers used by the customer, the mediator and the provider for correlation purposes respectively. Let $\perp$ denote the absence of either party or conversation identifier in the message.

The graphical notation of the Mediated Interaction pattern configuration is shown in Figure 29. This notation contains the following configuration parameters:

- *Customer request for specific provider*: denotes information revealed by the customer about the identity of provider, which the mediator has to forward at the customer's request.
  Range of values: *Them1* is a pair comprising a party identifier and a conversation identifier, potentially representing the credentials of the provider supplied by the customer to the mediator. The customer may know the identity of provider or may not know who the potential provider will be. If the customer knows the identity of the provider, it may indicate it in the *Them1* field in the form of $(P3, \perp)$. The mediator will forward the request to the specified party. If the customer does not know the provider's identity or does not want to specify it, the value $(\perp, \perp)$ is assigned to the `Them1` field.
  Default value: $(\perp, \perp)$.
  Visualization: the *Them1* label substituted with a suitable value in Figures 29. An example specifying a default value is given in Figures 26.
- *Customer's permission to expose its credentials*: denotes permission granted to the mediator by the customer to disclose its credentials to the provider.

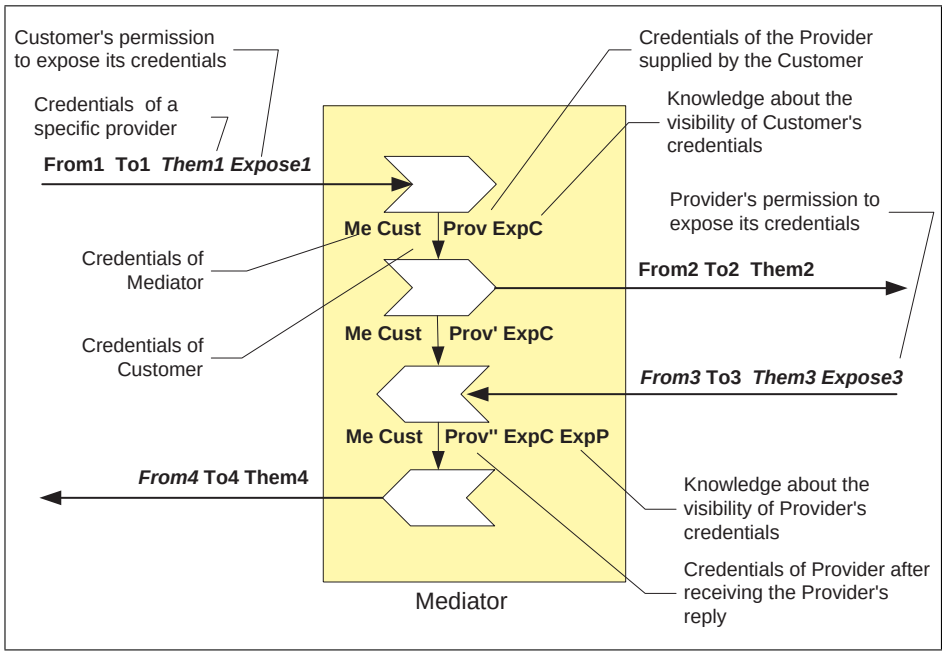


Fig. 25. Graphical notation: Mediated Interaction

Range of values: *Expose1* is a Boolean variable, whose value can be `true` or `false` indicating that the customer allows the mediator to expose its identity to the provider or prohibits it to do so in order to remain anonymous respectively.
Default value: `false` (i.e. the customer does not give permission for their credentials to be revealed).
Visualization: the *Expose1* label substituted with a suitable value in Figure 29. An example specifying a default value is given in Figure 26.

- *Visibility of the provider's credentials*: denotes whether permission is granted to the mediator by the provider to disclose its credentials to the customer.
Range of values: `Expose3` is a Boolean variable, indicating whether the provider allows the mediator to expose its identity to the customer or prohibits it to do so in order to remain anonymous. The provider may expose its identity in order to allow the customer to interact directly with them in the future independently of the mediator.
Default value: `false` (i.e. the permission to expose the credentials is not granted).
Visualization: the *Expose3* label substituted with a suitable value in Figure 29. An example specifying a default value is given in Figure 26.

- *Provider's knowledge about credentials of customer*: denotes information revealed by provider in the response message to the mediator about the credentials of the customer. This information may be used by the mediator to

correlate response messages received from the provider.
Range of values: *Them3* is a pair comprising the customer identifier and the conversation identifier. The provider may specify none, some or all information received from the mediator about the identity of the customer retrieved from the `Them2` field. So, possible values for *Them3* are $(P1, C1), (P1, \perp), (\perp, C1)$ or $(\perp, \perp)$.
Default value: $(\perp, \perp)$.
Visualization: the *Them3* label substituted with a suitable value in Figure 29. An example specifying a default value is given in Figure 26.

- *Information about the message sender in the response from provider to mediator*: denotes credentials of the provider revealed by the provider in the response message sent to the mediator.
  Range of values: `From3` is a pair comprising the provider identity and the conversation identifier. The provider may underspecify some information about its identity in the message sent to the mediator. If an underspecified identity is passed by the mediator to the customer, the customer may fail to start a direct interaction with provider in the future. Thus, possible values of the `From3` field are $(P3, C3), (P3, \perp), (\perp, P3)$ or $(\perp, \perp)$.
  Default value: $(P3, C3)$.
  Visualization: the *From3* label substituted with a suitable value in Figure 29. An example of specifying a default value is given in Figure 26.
- *Information about the message sender in the response from mediator to customer*: denotes the identity of mediator revealed in the response message sent by the mediator to the customer.
  Range of values: `From4` is a pair comprising a party identifier and a conversation identifier. The mediator may underspecify some information about its credentials, therefore possible values of the `From4` field are $(P2, C2), (P2, \perp), (\perp, C2)$ and $(\perp, \perp)$.
  Default value: $(P2, C2)$.
  Visualization: the *From4* label substituted with a suitable value in Figure 29. An example of specifying a default value is given in Figure 26.

The static attributes of the Mediated Interaction pattern configuration are listed below:

- `From1`: information specified by the customer about its credentials in the request message to the mediator. It is assumed that the customer reveals all information about its credentials, therefore `From1` $= (P1, C1)$.
- `To1`: information about the mediator's credentials specified by the customer in the message receiver field of the request message sent to the mediator. Initially, the customer has no knowledge about the conversation identifier used by the mediator for correlation, therefore `To1` $= (P2, \perp)$.
- `Me`: knowledge of the mediator about its party identifier and the associated conversation identifier: `Me` $= (P2, C2)$.
- `From2`: information specified by the mediator about its credentials in the request message to the provider. It is assumed that the mediator reveals all

information about its credentials in order for the response message to be delivered at the correct address, therefore From2 = $(P2, C2)$.

The dynamic attributes for the Mediated Interaction pattern configuration are listed below.

- Cust: information supplied by the customer about its credentials in the request to the mediator. This information corresponds to retained knowledge held by the mediator for correlation purposes. The knowledge about the identity of the customer is gained by the mediator from the From1 field: Cust=From1.
- Prov: information supplied by the customer in the request sent to the mediator about the identity of a provider. This information is retained knowledge held by the mediator in order to to forward the request received from the customer to the party with the indicated identity (if it has been provided). This knowledge is gained from the Them1 field: Prov=Them1.
- To2: information about the provider's credentials specified by the mediator in the message receiver field of the request message forwarded to the provider. The identity of the provider is set to Prov if it has been provided by the customer, or is defined by the mediator based on its implicit knowledge and set to $(P3, \perp)$.
- Prov': credentials of the provider to whom the mediator has sent the request, recorded for the purpose of the future correlations. This information is derived from the To2 field: Prov'=To2.
- Prov'': knowledge about the credentials of the provider available to the mediator after receiving the response message from the provider. The mediator extends information about the provider's identity stored in the Prov' field with missing knowledge gained from the From3 field of the response message received from the provider. Since the provider could underspecify some knowledge about its identity, the possible values of Prov'' are $(P3, C3)$ or $(P3, \perp)$.
- ExpC: knowledge of the mediator about permission granted by the customer to disclose its credentials to the provider. It is assumed that the mediator hides the identities of parties involved in the conversation and discloses them only if the corresponding party has given the permission to do so, i.e. ExpC=Expose1.
- Them2: information specified by the mediator in the request message to the provider about the credentials of the customer. The knowledge retained in the Cust field is used to set the value of the Them2 field only if the customer has granted the permission to disclose its credentials, i.e. Them2=Cust=$(P1, C1)$ if and only if ExpC=true, otherwise Them2=$(\perp, \perp)$.
- ExpP: knowledge of the mediator as to whether permission granted by the provider to disclose its credentials to the customer. This knowledge is gained from the Expose3 field: ExpP=Expose3.
- Them4: information specified by the mediator in the response message to the customer about the credentials of the provider. The knowledge retained in the Prov field is used to set the value of the Them4 field only if the provider has

granted the permission to disclose its credentials, i.e. `Them4`=`Prov` if and only if `ExpP`=true, otherwise `Them4`=$(\perp, \perp)$.

– `To4`: information about the customer's credentials specified by the mediator in the message receiver field of the reply message sent to the customer. The mediator retrieves the credentials of the customer from the `Cust` field: `To4`=`Cust`.

Figure 26 illustrates the graphical notation of the Message Interaction pattern variant where all pattern attributes are set to the default value.

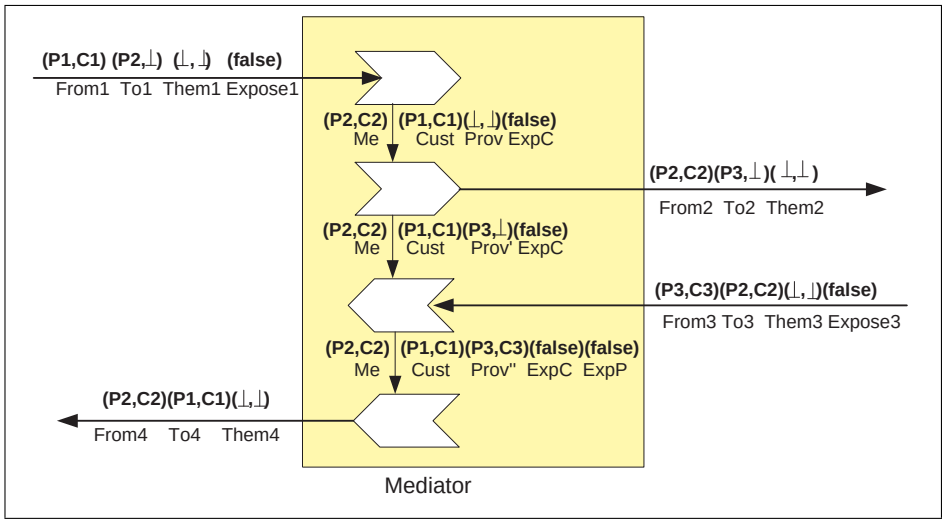


Fig. 26. Default notation: Message Interaction

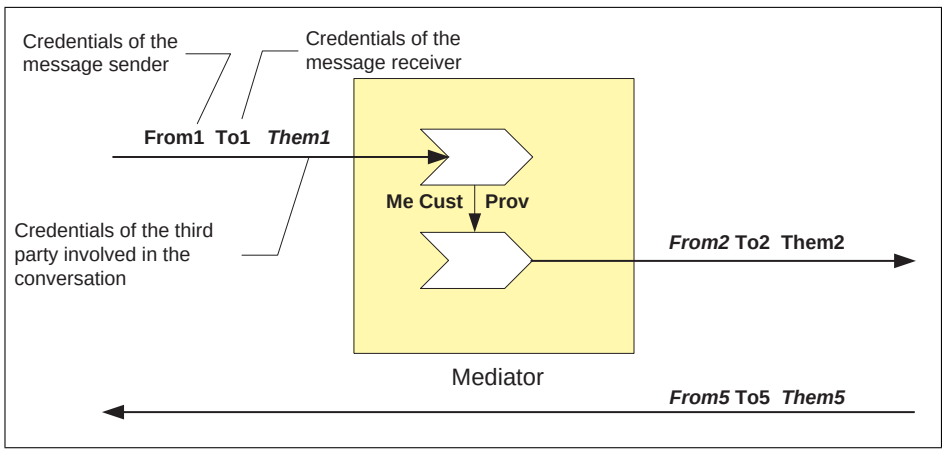


Fig. 27. Graphical notation: Mediated Introduction

The configuration parameters of the Mediated Introduction pattern configuration are described below:

- *Customer request for specific provider*: denotes information revealed by the customer about the identity of provider, which the mediator has to forward at the customer's request.
  Range of values: *Them1* is a pair comprising a party identifier and a conversation identifier, representing the credentials of the provider supplied by the customer to the mediator. The customer may know the identity of provider or may not know who the potential provider will be. If the customer knows the identity of the provider, it may indicate this in the *Them1* field in the form of $(P3, \perp)$. The mediator will forward the request to the specified party. If the customer does not know the provider's identity or does not want to specify it, the value $(\perp, \perp)$ is assigned to the `Them1` field.

  It is assumed that the customer has no knowledge about the conversation identifier used by the provider, since there was no direct interaction between them in the context of the given conversation before. Therefore, the value $(P3, C3)$ is excluded. The value $(\perp, C3)$ is also excluded since it does not give complete information about the provider's identity and corresponds to an arbitrary provider.
  Default value: $(\perp, \perp)$.
  Visualization: the *Them1* label substituted with a suitable value in Figures 27. An example specifying a default value is given in Figures 28.

- *Information about the message sender in the request from mediator to provider*: denotes information revealed by the mediator about its credentials in the request message forwarded to the provider.
  Range of values: information provided by the mediator about its credentials in the request to the provider may be underspecified. The mediator may reveal all, some or none of its identity to the provider, providing that the mediator passes a complete reference to the credentials of the customer (the latter are required for sending a response message directly back to the customer).
  So, possible values of the *From2* field are $(P2, C2)$, $(P2, \perp)$, $(\perp, C2)$ and $(\perp, \perp)$.
  Fixed value: $(P2, C2)$.
  Visualization: the *From2* label substituted with its value in Figure 27. An example specifying a default value is given in Figure 28.

- *Information about the message sender in the response from provider to customer*: denotes credentials of the provider revealed by the provider in the response message sent to the customer.
  Range of values: *From5* is a pair comprising the provider identity and the conversation identifier. The provider may underspecify some information about its identity in the message sent to the mediator. If an underspecified identity is passed to the customer, the customer may be unable to start a direct interaction with the provider in the future. Thus, possible values of the `From5` field are $(P3, C3)$, $(P3, \perp)$, $(\perp, P3)$ or $(\perp, \perp)$.
  Default value: $(P3, C3)$.

Visualization: the *From5* label substituted with a suitable value in Figure 27. An example of specifying a default value is given in Figure 28.

- *Information about mediator exposed by provider to customer*: denotes the credentials of the mediator revealed by the provider in the response message sent to the customer.
  Range of values: *Them5* is a pair comprising the party identifier and the conversation identifier, whose value is based on the `From2` field. Table 8 illustrates possible values of the `Them5` field, containing all or part of the information from the `From2` field.
  Default value: $(\perp, \perp)$.
  Visualization: the *Them5* label substituted with a suitable value in Figure 27. An example specifying a default value is given in Figure 28.

Table 8. Enumeration of all scenarios regarding revealed credentials of the mediator

From2	Them5
(Ps, Cs)	(Ps, Cs)
	$(Ps, \perp)$
	$(\perp, Cs)$
	$(\perp, \perp)$
$(Ps, \perp)$	$(Ps, \perp)$
	$(\perp, \perp)$
$(\perp, Cs)$	$(\perp, Cs)$
	$(\perp, \perp)$
$(\perp, \perp)$	$(\perp, \perp)$

The static attributes of the Mediated Introduction pattern configuration are listed below:

- `From1`: information specified by the customer about its credentials in the request message to the mediator. It is assumed that the customer reveals all information about its credentials, therefore `From1` $= (P1, C1)$.
- `To1`: information about the mediator's credentials specified by the customer in the message receiver field of the request message sent to the mediator. Initially, the customer has no knowledge about the conversation identifier used by the mediator for correlation, therefore `To1` $= (P2, \perp)$.
- `Me`: knowledge of the mediator about its party identifier and the associated conversation identifier: `Me` $= (P2, C2)$.

The dynamic attributes of the Mediated Introduction pattern configuration are listed below. Their values are derived from other pattern attributes.

- `Cust`: information supplied by the customer about its credentials in the request to the mediator. This information is retained knowledge held by the mediator for correlation purposes. The knowledge about the identity of the customer is gained by the mediator from the `From1` field: `Cust`=`From1`.

– `Prov`: information supplied by the customer in the request sent to the mediator about the identity of a provider. This information is retained knowledge held by the mediator in order to to forward the request received from the customer to the party with the indicated identity (if it has been provided). This knowledge is gained from the `Them1` field: `Prov`=`Them1`.
– `To2`: information about the provider's credentials specified by the mediator in the message receiver field of the request message forwarded to the provider. The identity of the provider is set to `Prov` if it has been provided by the customer, or is defined by the mediator based on its implicit knowledge and set to $(P3, \perp)$.
– `Them2`: information specified by the mediator in the request message to the provider about the credentials of the customer. This information will be used by the provider when sending the response to the customer. The knowledge retained in the `Cust` field is used to set the value of the `Them2` field, i.e. `Them2`=`Cust`.
– `To5`: information about the customer's credentials specified by the provider in the message receiver field of the reply message sent to the customer. The provider retrieves the credentials of the customer from the `Them2` field: `To5`=`Them2`.

Figure 28 illustrates the graphical notation of the Message Introduction pattern variant where all pattern attributes are set to the default value.

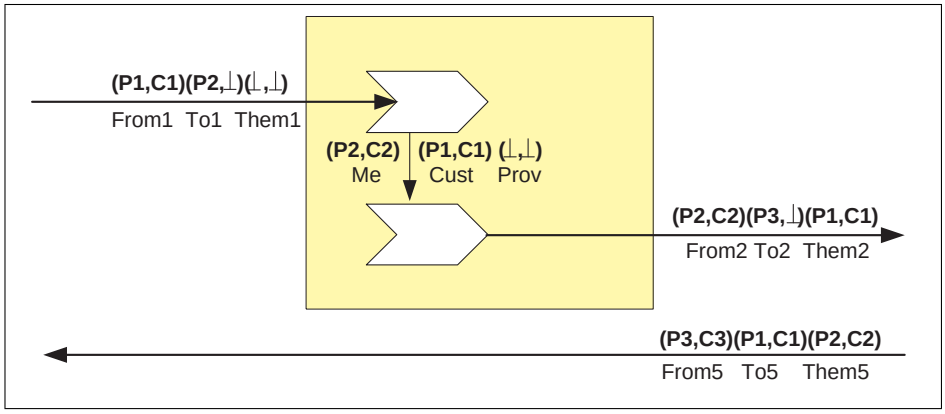


Fig. 28. Default notation: Message Introduction

Illustrative example To show how the pattern configuration can be used in practice, we analyze one of the examples presented earlier and define the corresponding pattern variant. In the air-miles exchange example, the air-miles card owner sends a request to the Air-miles web-site to exchange accrued air-miles for reservation of the specific flight. The Air-miles web-site serves as a mediator between the client and the airline operator. When forwarding the request from

the client to the selected by the client airline operator, the mediator reveals the client's credentials (these are needed to book a flight). The airline operator sends the details of the reservation back to the mediator, who in its turn forwards them to the client. Together with the details of the reservation made, the mediator specifies the contact address of the airline operator in order to allow the customer to contact the airline company in the future for any outstanding issues. In this example, the Mediated Interaction is performed by the Air-miles web-site. The label `Expose1` is set to `true`, since the clients details have to be communicated to the airline operator. The `Them1` label contains the identity of the airline operator selected by the client. The `Expose3` label is set to `true`, since the airline operator discloses its credentials to the client in order to allow for the future interactions. The `From3` label contains complete information about the credentials of the airline operator and the `From4` label also contains complete information about the Air-miles organization.

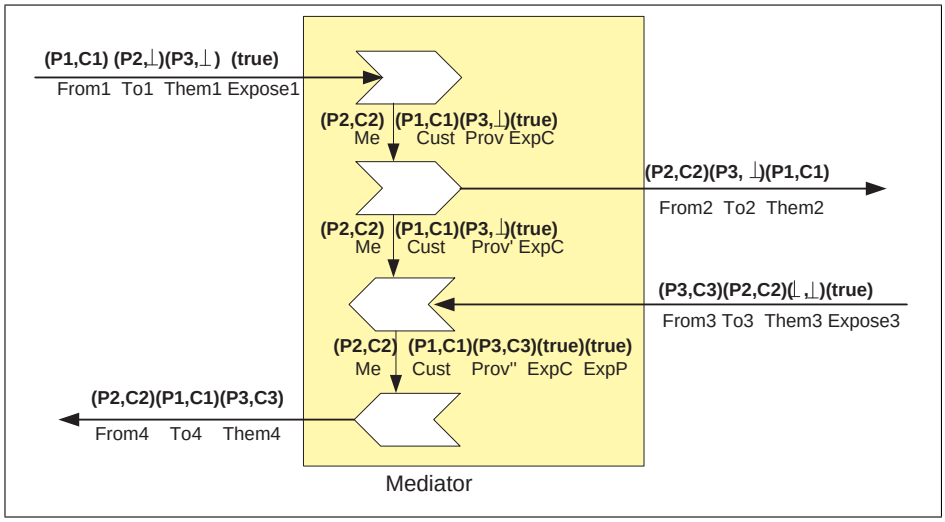


Fig. 29. Notation for the air-miles exchange example

CPN semantics In this section, we describe the semantics of both Message Mediation scenarios in the form of CPN models. The Mediated Interaction scenario is represented in Figure 30. The three parties involved in a conversation, i.e. customer, mediator, and provider, are represented as substitution transitions that decomposed in Figures 31, 32 and 33 respectively.

The behavior of the customer is shown in Figure 31. The customer communicates directly with the mediator by sending a request message of type `CustRequest` to and receiving a response message of type `MedReply` from the mediator via transitions `Send request` and `Receive response` respectively. Place `Mediator identity` stores information about the identity of mediators to one of which a request message will be sent by the customer. Initially, the customer has no knowl-

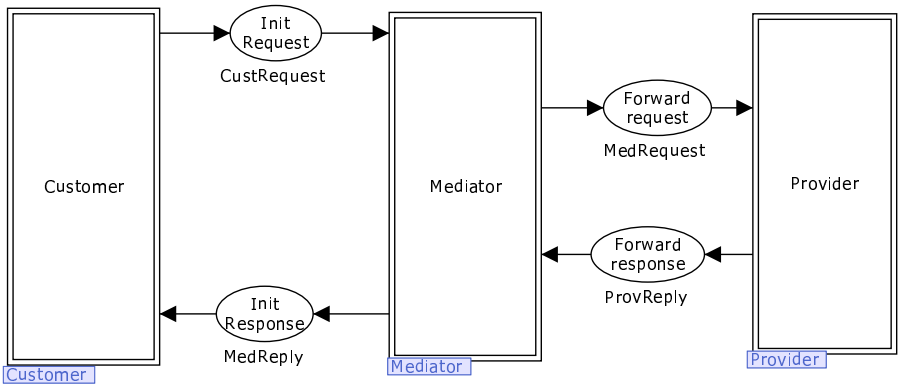


Fig. 30. CPN diagram: The top view of Mediated Interaction

Table 9. Data types used in the Mediated Interaction diagrams

```
colset PartyId = union smallstr + NoPartyId;
colset ConvId = union smallint + NoConvId;
colset PxC = product PartyId * ConvId;
colset Content = STRING;
colset From = PxC;
colset To = PxC;
colset Them = PxC;
colset Expose = BOOL;
colset ExpC = Expose;
colset ExpP = Expose;
colset Me = PxC;
colset You = PxC;
colset Cust = PxC;
colset Prov = PxC;
colset Conv = product Me * PxC * PxC * Content;
colset ConvM = product Me * Cust * Prov * ExpC * ExpP * Content; [a]
colset CustRequest = product From * To * Them * Expose * Content;
colset MedRequest = product From * To * Them * Content;
colset MedReply = product From * To * Them * Content;
colset ProvReply = product From * To* Them * Expose * Content;
```

[a] `ConvM` denotes information recorded by mediator about conversations with customer and provider. While customer and provider communicate directly only with one party and store conversations of type `Conv`, the mediator has to keep track of interaction with both parties at once.

edge about the conversation identifier used by the mediator for correlation purposes. The customer may know the identity of the provider with whom it wishes to communicate directly. The customer may either provide the mediator with the identity of the provider or may leave it out. The knowledge about the identity of the provider is enclosed in the `them1` field of the request message. The value of this configuration parameter is set by the non-deterministic `defthem1()` function that either specifies the credentials of the provider or leaves this field empty. In the request message the customer explicitly specifies whether it allows the mediator to disclose its credentials to the provider or not by setting the `expose1` variable of Boolean type to `true` or `false` respectively.

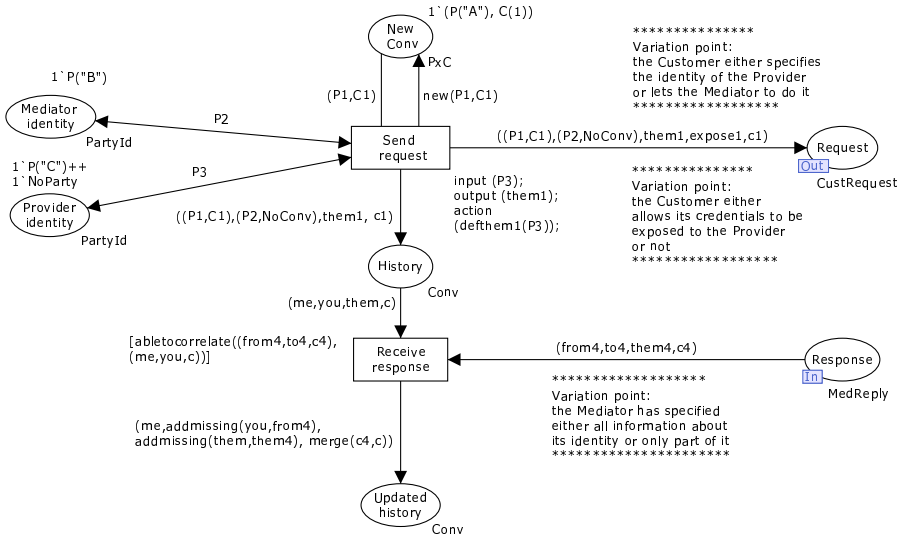


Fig. 31. CPN diagram: The `Customer` page of Mediated Interaction

The `History` place keeps track of all messages sent by the customer to the mediator, including the credentials of all parties involved in the conversation. Response messages sent by the mediator to place `Response` are received by the customer via the `Receive response` transition. A function `abletocorrelate()` in the transition guard checks whether the response message received can be correlated with any of the requests sent. In this example, correlation is performed based on matching the message receiver field with the identity of the customer. If the result of the correlation is successful, the missing information gained from the response message is recorded in the `Updated history` place by means of the `addmissing()` function. Information contained in the `them4` field of the response message can be used by the customer to send a follow-up request directly to provider.

The behavior of the mediator is shown in Figure 32. Requests sent by the customer to the `Init request` place are received by mediator via the `Receive`

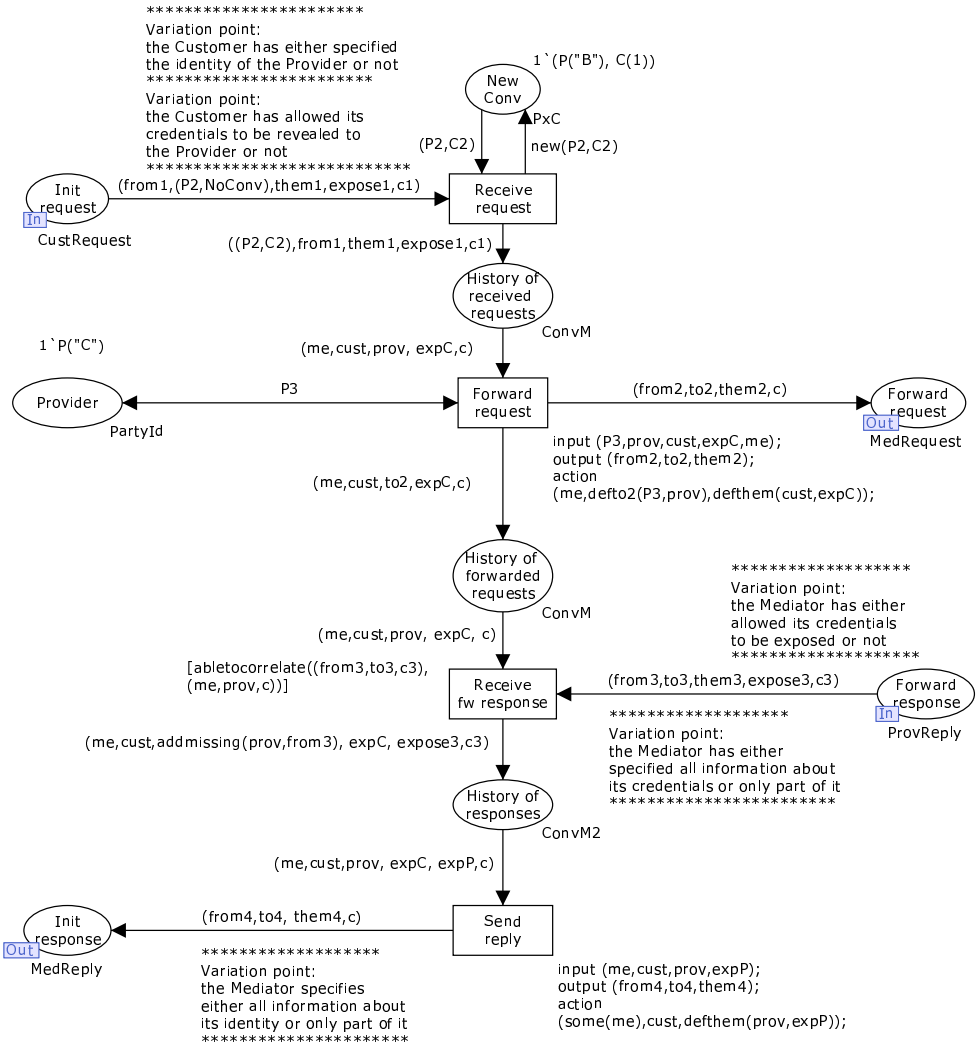


Fig. 32. CPN diagram: The `Mediator` page of Mediated Interaction

`request` transition. For each of the requests received, the mediator creates a new conversation, whose identifier is provided by the counter place `New Conv`. After receiving the request, the mediator adds a record about the request received to place `History of received requests`. This record contains information about the credentials of the customer `from1`, information provided by the customer about the identity of a provider `them1`, and information about the visibility of the customer's credentials to the provider `expose1`. When forwarding a request received from the customer, the mediator specifies the address of the message receiver, its own credentials and shows the credentials of the customer if the customer has granted the permission to disclose its credentials. The latter is done by

means of the function `defthem()`. The `defto2()` function determines the identity of the provider. If the customer has specified the identity of the provider to which the mediator has to forward the request, then this identity is used in the `to2` field, otherwise the mediator determines to which of the parties, whose identities are stored in place `Provider`, the message can be forwarded.

When response messages of type `ProvReply`, stored in place `Forward response`, are received from the provider, the mediator tries to correlate them with previous requests using the `abletocorrelate()` function. If the related conversation has been identified, the mediator records missing information gained from the response message in the `History of responses` place using the `addmissing()` function. The mediator forwards the response message to the customer via the `Send Reply` transition. In this message, the mediator may underspecify its credentials in the `from4` field using function `some()`. Function `defthem()` is used to define the value of the `them4` field. In particular, if the provider has granted permission for its credentials to be revealed to customer (i.e. a variable `expP` has been set to `true`), the credentials of provider stored in variable `prov` are assigned to the `them4` field.

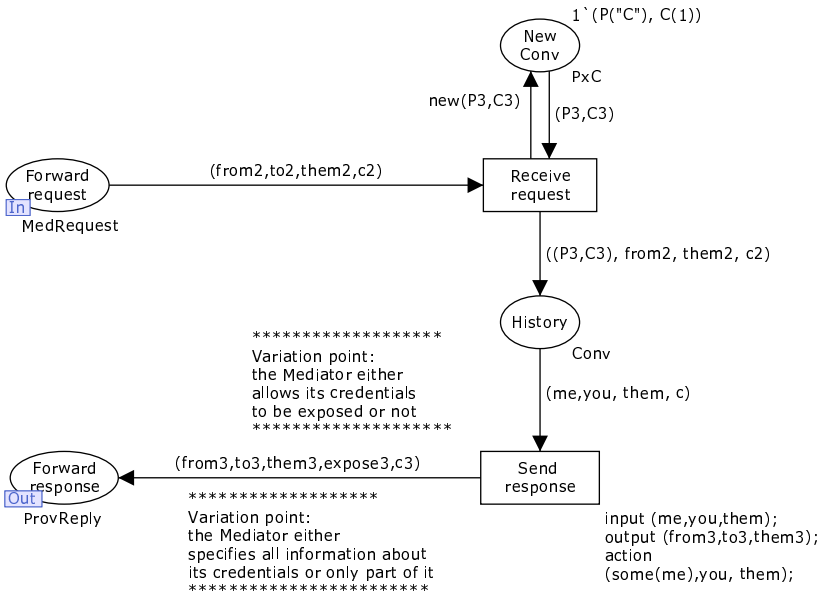


Fig. 33. CPN diagram: The `Provider` page of Mediated Interaction

The behavior of the provider is shown in Figure 33. For each of the requests forwarded by the mediator, the provider creates a new conversation and records information related to the conversation in the `History` place. The provider sends a response message to the address specified by the mediator in the `From2` field. When sending a response message, the provider may decide to reveal all, some or

none of its identity to the mediator. This information is defined by function `some` and enclosed in the `from3` field of the response message. Moreover, the provider may decide to expose its credentials to the customer, and for this, the Boolean variable `expose3` has to be set to `true`.

The main page of the Mediated Introduction pattern configuration is shown in Figure 34. The behavior of customer has been presented in Figure 31. The behavior of both the mediator and provider differs slightly from the nets described in the Mediated Interaction scenario and is presented in figures 35 and 36 respectively.

In the Mediated Introduction scenario, messages of different types are used (see Table 10). None of the messages exchanges between parties in this scenario contain the `Expose` field, because it is assumed that the credentials of the parties can be freely revealed to other parties.

Table 10. Data types used in Mediated Introduction diagrams

```
colset CustRequest = product From * To * Them * Content;
colset MedRequest = product From * To * Them * Content;
colset MedReply = product From * To * Them * Content;
colset ProvReply = product From * To* Them * Content;
```

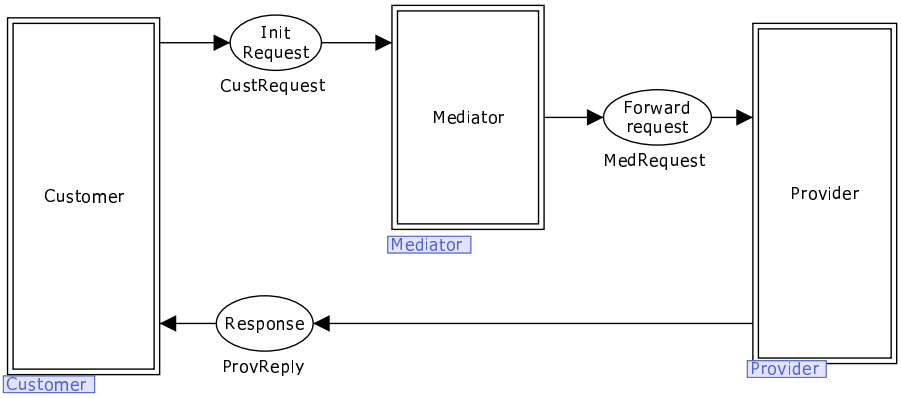


Fig. 34. CPN diagram: The top view of Mediated Introduction

The substantial difference between the behavior of the mediator presented in Figure 35 and that for the mediator in the Mediated Interaction, is that credentials of the customer recorded by the mediator in the `cust` variable are assigned to the `them2` field of the response message sent to the provider without examining the permission of the customer to expose its credentials. As has been mentioned already, this is done based on the assumption that no permission is required for the mediator to expose the credentials of the customer to the provider.

The behavior of the provider is shown in Figure 36. After receiving a response message from the customer, the provider sends the response message directly to

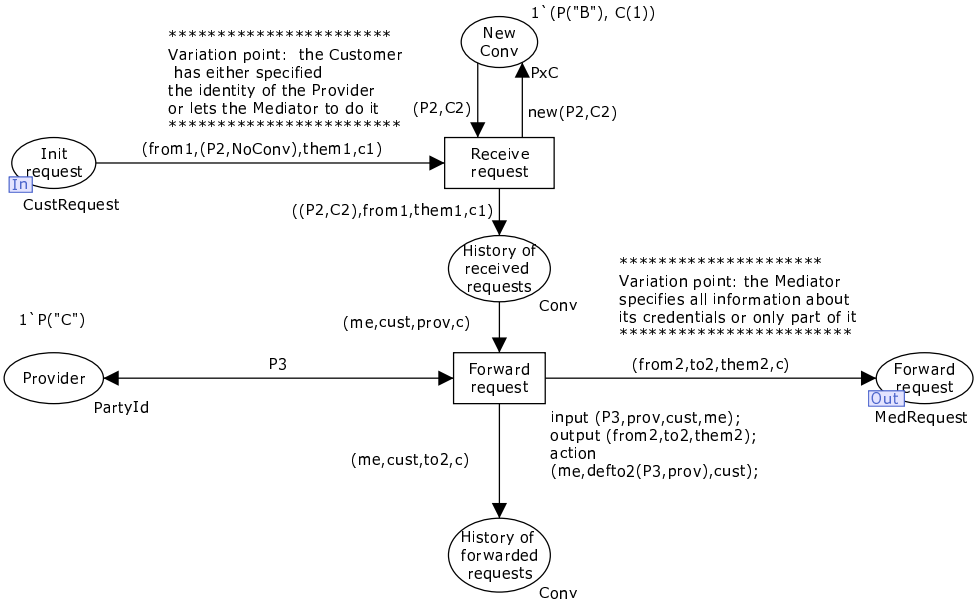


Fig. 35. CPN diagram: The `Mediator` page of Mediated Introduction

the customer. Although there was no interaction between the customer and the provider in the past, and the customer does not know the provider, therefore the provider uses the reference to the customer's credentials provided by the mediator in the `them2` field to specify the address of the message receiver `to5` in its response message.

Issues Pattern variants belonging to the *Message Mediation* family address scenarios where a single message is exchanged between the involved parties. If multiple sub-requests have to be sent to a set of mediators at once, or if a customer has to send requests to a set of mediators, who in turn might want to forward the request to a set of providers, then the given pattern variant can be combined with another suitable pattern variant belonging the *Multi-Party Multi-message Request-Reply Conversation* family. One of the roles of the mediator in the tripartite conversation may be to pass the reference from one party to another in order to enable their direct interaction in the future. In the course of long-running bipartite conversations, information used for correlation may change. This issue is addressed by the *Bipartite Conversation Correlation* pattern family.

8 Pattern Family: Bipartite Conversation Correlation

In this section we describe the fifth and final pattern family, Bipartite Conversation Correlation, which addresses the issue of correlation in a conversation between two parties.

Description Each of the two parties involved in a long-running conversation may indicate during the initiation of the conversation what correlation token it expects

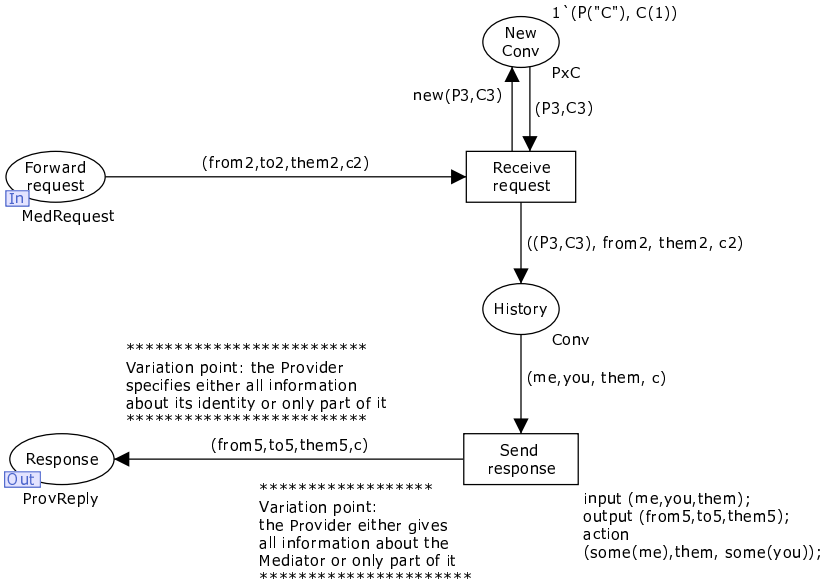


Fig. 36. CPN diagram: The `Provider` page of Mediated Introduction

in the follow-up messages in order to unambiguously relate them to the existing conversation. Independent of the correlation information provided by one party during the conversation initiation, the other party may forget or choose to ignore it in the follow-up interactions.

Example

- A client requests a telephone subscription from the telephone company. The company registers the customer and confirms the registration by sending a letter with a telephone number and the client's name to the address specified by the client. The company expects a client to indicate the telephone number in any future enquiries. However, it is likely that when making future enquiries, the client will forget to do so or will send incomplete information.

UML meta-model Concepts specific to the *Bilateral Conversation Correlation* pattern configuration are illustrated by means of the UML diagram in Figure 37. Two parties, a requestor and a responder, are involved in a bipartite conversation (see the `responder` and `requestor` association relations between `Party` and `Conversation`). Each party involved in a conversation may send and receive messages (see associations `sends` and `receives` between `Party` and `Message`). Messages can be either of type `Request` or `Reply`, where every reply corresponds to precisely one request and multiple replies can be sent for the same request. A conversation consists of two phases: initiation and follow-up interactions (represented as specializations `Initiation` and `Follow-up`). The requestor initiates a conversation by sending a message to the other party. The requestor specifies its correlation credentials (i.e. a party identifier and a conversation identifier) in the initial message sent to the other party and expects the responding party to use

the same credentials in the response messages, however the responder may choose to ignore this information is the follow-up interactions.

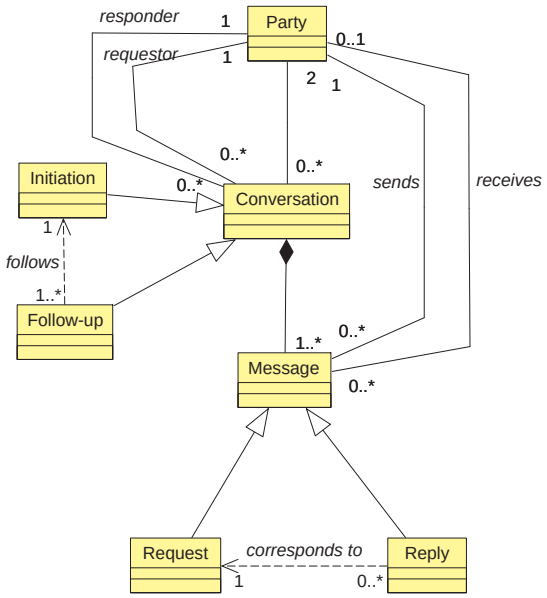


Fig. 37. UML meta-model of
Bipartite Conversation Correlation

Visualization Figure 38 illustrates the graphical notation used for illustrating Bipartite Conversation Correlation. Parties are visualized as rectangles. Arrows between rectangles indicate interactions between parties, where direction of arrows corresponds to the message flow. Interactions marked as `initial request` and `initial response` correspond to the initiation of conversation, where parties notify each other about the credentials they want to use as tokens for correlation in the follow-up interactions. It is an assumption of this pattern, that messages exchanged have the form `(From, To, Content)`, specifying the information about the message sender, message receiver, and the content of the message. Labels, marked as `From` and `To`, denote information about the message sender and the message receiver respectively.

Let $P1$ and $P2$ denote party identifiers of the requestor and responder respectively. Let $C1$ and $C2$ denote conversation identifiers used by the requestor and the responder for correlation purposes respectively. Let $\perp$ denote the absence of either party or conversation identifier in a message.

This graphical notation in Figure 38 contains a set of static attributes, dynamic attributes and configuration parameters. The static attributes refer to information that is fixed in every pattern variant. The dynamic attributes derive their value

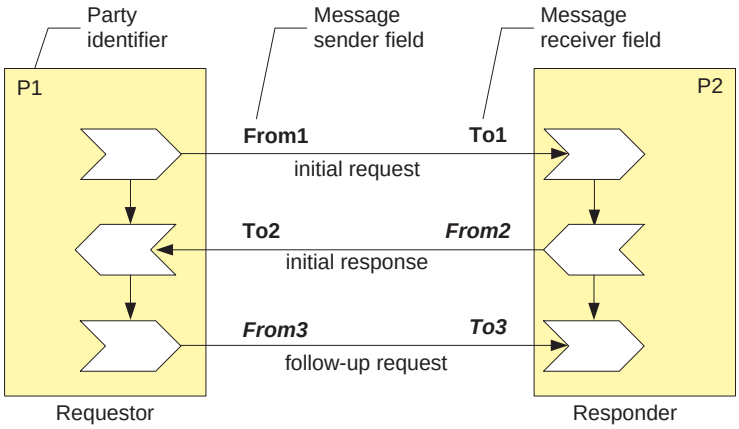


Fig. 38. Graphical notation:
Bipartite Conversation Correlation

from other pattern attributes. The configuration parameters refers to information that varies in all pattern variants, i.e. the configuration parameters have to be set to a value from the defined range in order for a specific pattern variant to be configured. The list of the static attributes is shown below:

- *Requestor credentials in the initial request*: information specified by the requestor about its identity in the conversation initial request. The requestor expects the responder to use the specified credentials when sending a response message back. `From1` is a pair comprising the requestor identifier and the conversation identifier used by the requestor for correlation purposes. It is an assumption, that the requestor always specifies the maximal possible information about its identity when initiating the conversation with another party: `From1`$=(P1, C1)$.
- *Responder credentials in the initial request*: information specified by the requestor about the credentials of the responder in the conversation initiation request. `To1` is a pair comprising the responder identifier and the conversation identifier used by the responder for correlation purposes. It is assumed, that initially the requestor does not have any knowledge about the conversation identifier the responder will use for correlation: `To1`$= (P2, \perp)$.

The graphical notation in Figure 38 contains only one dynamic pattern attribute described below:

- *Requestor credentials in the initial response*: information specified by the responder about the identity of the requestor in the response on the initiation request. `To2` is a pair comprising the requestor identifier and the conversation identifier used by the requestor for correlation purposes, such that `To2=From1`.
  Fixed value: `(P1,C1)`.
  Visualization: label `To2` in Figure 38 substituted with its value.

The list of configuration parameters is given below.

- *Responder credentials in the initial response*: information specified by the responder about its identity in the response to the initial request received from the requestor.
  Range of values: *From2* is a pair containing the responder identifier and the conversation identifier used by the responder for correlation purposes. The requestor identifier is denoted as $P2$ and its conversation identifier is denoted as $C2$. It is assumed, that the requestor may forget to specify or choose not to disclose some of the information about its credentials. The absence of this information is denoted by $\perp$. So, possible values of the `FROM2` field are $(P2, C2), (P2, \perp), (\perp, C2)$ and $(\perp, \perp)$.
  Default value: $(P2, C2)$.
  Visualization: the *From2* label in Figure 38 substituted with a suitable value. An example specifying a default value is shown in Figure 39.
- *Responder credentials in the follow-up request*: information specified by the requestor about the identity of the responder in follow-up requests.
  Range of values: *To3* is a pair containing the possible responder identifier and the conversation identifier used by the responder for correlation purposes. Although the requestor has received information about the responder credentials in the initiation response, it may decide to use all of the information provided, some of it or no information at all. Where the requestor decides to use the credentials specified by the responder in the `FROM2` field, `To3 = From2`. If the requestor decides to specify only part of the information retrieved from the `FROM2` field, either the party identifier of the responder or responder's conversation identifier may be omitted. For example, if `From2`$=(P2, C2)$, then possible values of the `To3` field are $(P2, \perp), (\perp, C2)$ and $(\perp, \perp)$.
  Default value: `From2`.
  Visualization: the *To3* label in Figure 38 substituted with a suitable value. An example specifying a default value is shown in Figure 39.
- *Requestor credentials in the follow-up request*: information specified by the requestor about its identity in the follow-up requests sent to the requestor.
  Range of values: *From3* is a pair containing the responder identifier and the conversation identifier used by the responder for correlation purposes. Depending on what information the requestor specified about the identity of the responder in the follow-up request, i.e. whether it used all credentials specified by the responder or only part of them, the requestor may specify either complete information about its identity as indicated in the `From1` field, or only part of it. So, possible values of the `From3`-field are: $(P1, C1), (P1, \perp), (\perp, C1)$ and $(\perp, \perp)$.
  Default value: $(P1, C1)$.
  Visualization: the *From3* label in Figure 38 substituted with a suitable value. An example specifying a default value is shown in Figure 39.

Figure 39 presents the graphical notation for one of the bipartite conversation pattern variants where all configuration parameters are set to their default

values. The requestor and the responder retrieve all information from the message received, and specify complete information about the identity of the message sender and the message receiver.

Illustrative example For the telephone subscription example described earlier we define the pattern configuration depicted in Figure 40. In this example, a client requests a telephone subscription from the telephone company. The company registers the customer and confirms the registration by sending a letter with a telephone number and the client's name to the address specified by the client. The company expects a client to indicate the telephone number in any future enquiries. However, when making the next enquiry, the client forgets to specify their telephone number. In the graphical notation, the client and the telephone company map directly on the roles of the responder and the provider. The requestor omits the conversation id specified by the responder (i.e. the telephone number) in the follow-up request. Note that the requestor assumes the responder to have used `C1` as an identifier.

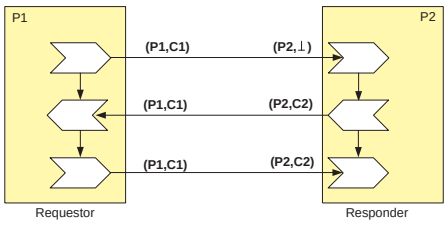


Fig. 39. Default notation: Bipartite Conversation Correlation

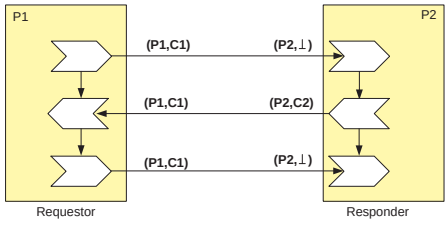


Fig. 40. Notation for the telephone subscription example

CPN semantics A CPN diagram illustrating the semantics of the Bipartite Conversation Correlation is shown in Figure 41. Two parties, a requestor and a responder, represented as substitution transitions `Requestor` and `Responder`, are involved in a conversation. The responder initiates a conversation by sending an initiation request `Init request` of type `Message`. In this initiation request the requestor specifies the credentials it wants the responder to use when responding on this request. The responder sends an initiation response `Init response` of type `Message`, where it specifies its own credentials that must be used by the requestor in the follow-up requests. Messages are exchanged between the parties take the form `(From,To,Content)`, where `From` is credentials of the message sender, `To` is credentials of the message receiver and `Content` is the content of the message.

The decomposition of transitions `Requestor` and `Responder` is shown in Figure 42 and Figure 43 respectively. The CPN models for bipartite conversation correlation are based on the correlation mechanisms and data types introduced for the Message Correlation pattern family (see Section 6).

The behavior of the requestor is depicted in Figure 42. Transitions `Send init request`, `Receive init request` and `Send follow-up request` correspond to the send conversation initiation request, receive initiation reply, and send follow-

up request respectively. To initiate a new conversation, a new unique conversation id is created by function `new()`, which increments the identifier of the last conversation. When sending out an initialization request, the requestor specifies its identity `(P1,C1)`, where `P1` is a party identifier of type `PartyId` and `C1` is a conversation identifier of type `ConvId`. In the credentials of the message receiver in the initiation request, the requestor only specifies the id of the responder `P2`. The absence of the conversation identifier is denoted by means of the value `NoConvId`. After the message has been sent, it is recorded in place `History` of type `Conv`. This place stores the history of all current conversations for use in correlating response messages from the responder.

When an initialization response arrives, transition `Receive init response` checks by means of the function `abletocorrelate()` whether the message received can be correlated with one of the conversations recorded in the place `History`. The mechanism of correlation is based on the model presented for the *Message Correlation* pattern family. If the message received can be correlated, the history of conversations is updated. Function `addmissing()` records information supplied by the responder about its identity which was not known to the requestor. Since the requestor may forget to retrieve some information, function `some()` defines how much information will be included in each case. By means of

```
colset PartyId = union smallstring +
       NoPartyId; [a]
colset ConvId = union smallint +
       NoConvId; [b]
colset PxC = product
       PartyId * ConvId;
colset Content = string;
colset From = PxC;
colset To = PxC;
colset Me = PxC;
colset You = PxC;
colset Conv = product Me*You*
       Content;
colset Message = product From*
       To*Content;
```

Table 11. Data types used in Figs. 41-43

[a] The id of a party is a small color-set of type STRING or is denoted by NoPartyId if not specified.

[b] The id of a conversation is a small color-set of type INT or is denoted by NoPartyId if not specified.

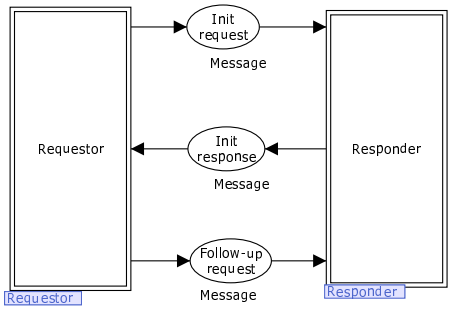


Fig. 41. CPN diagram: The top view of Bipartite Conversation Correlation

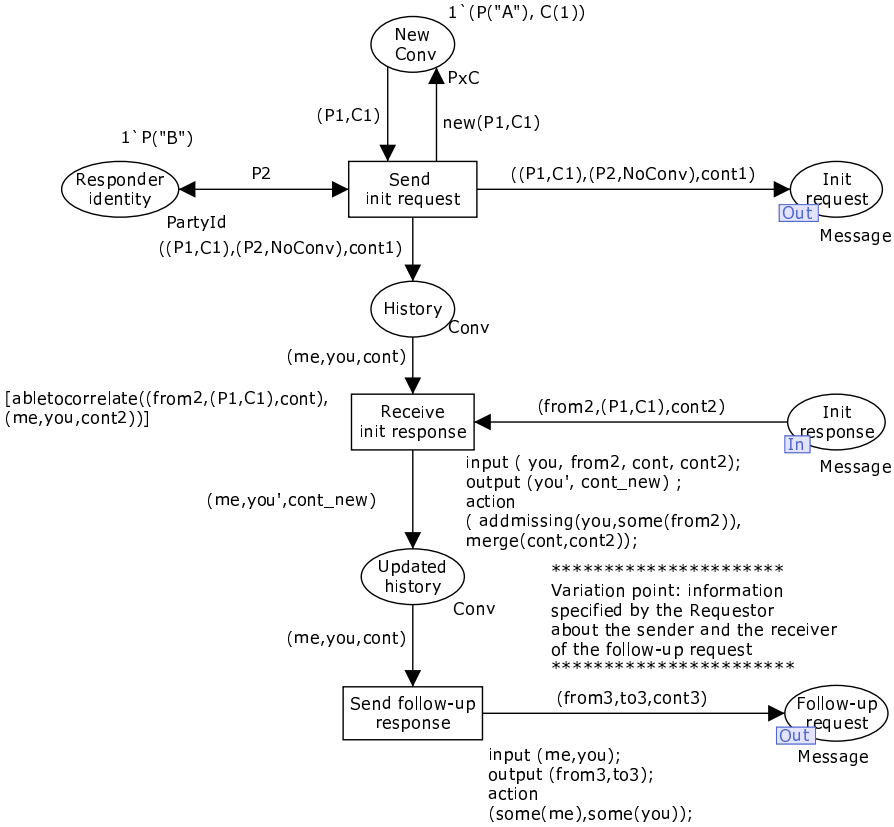


Fig. 42. CPN diagram: The requestor page of Bipartite Conversation Correlation

this function, it is possible to specify that either the party identifier, the conversation identifier or both are forgotten.

In the follow-up request, the requestor may use all or just part of the information stored in place `Updated history` to specify the credentials of the message sender and the message receiver. Function `some` determines how much information will be underspecified in the From- and To- fields of the follow-up response.

The behavior of the responder is depicted in Figure 43. A conversation initiation request sent by the requestor in place `Init request` is consumed by transition `Receive init request`, which creates a new conversation identifier via the function `new()`. The created conversation identifier will be used by the responder in the future interactions with the requestor for correlation purposes. Information contained in the initialization request received is associated with the newly created conversation identifier, and all this information is recorded in the `History` place. When sending the initialization response, the responder may underspecify its identity in the `from2`-field by applying the `some()` function. After the response message has been sent, the conversation history is updated and recorded in the `Updated History` place. The responder expects the requestor to use the

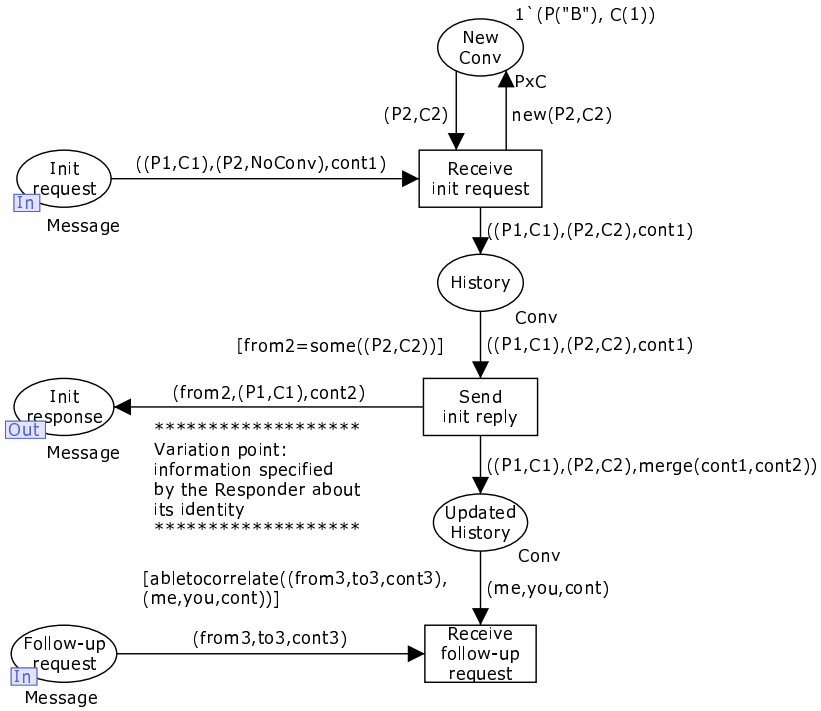


Fig. 43. CPN diagram: The responder page of Bipartite Conversation Correlation

information previously provided by the responder about its identity in follow-up requests. Follow-up requests sent by the requestor to place `Follow-up request` are checked by transition `Receive follow-up request` and consumed only if the function `abletocorrelate()` has identified a conversation with which the message received could be correlated. In this net, all messages for which no matching conversation can be found are ignored. Note that a mechanism for handling correlation failure could be added to this model, in order to specify whether such messages have to be discarded, or whether a new conversation has to be created to process them.

Issues In long-running conversations, the information used for correlation purposes may change. If the parties involved in the conversation underspecify the information about the identity of a requestor or a responder, the correlation might fail. In this pattern family the parties involved in the conversation are assumed to already know each other. However, in some situations the identity of a responding party is not known to the requestor directly. The conversation in such a setting involves an intermediary, who either forwards requests and replies between the involved parties and hides the identify of the parties involved from each other, or brings the parties in contact by providing a reference to their identity. These scenarios are described in the Message Mediation pattern family.

9 Assessment of WS-BPEL v2.0

In this section, we use the service interaction patterns to analyze the Web Services Business Process Execution Language (WS-BPEL) v2.0 [4] and related technologies. For each of the pattern families, we illustrate how the default pattern variants can be realized in practice using Oracle BPEL PM 10.1.3.1.0 (which is a tool based on BPEL) as an implementation vehicle. Furthermore, we analyze which pattern configurations are possible in the context of WS-BPEL.

9.1 Background

Oracle BPEL PM is based on BPEL, therefore it enables interactions across multiple organizations whose processes are deployed as Web services. BPEL is based on the XML schema [15], Simple Object Access Protocol (SOAP) [14], and Web Services Description Language (WSDL) [16]. In order to be accessible by other processes, the process that has been defined in the Oracle Process Designer, must be deployed to Oracle BPEL Server. Once deployed, a BPEL process is published as a Web service, and can be accessed through the client that uses WSDL interface definition of the given process and SOAP as a protocol. The role of the client may be performed by a user initiating the deployed process via the BPEL console, or by another process.

In order to send a message to a process, the client needs to know the custom data types defined in the XML schema of the target process, the message types and the port types declared in the WSDL definition of this process. Types of messages, sent and received by a process, are defined based on the data types declared in the XML schema. The portType element includes a supported set of operations, each including the input and the output messages of the operation. Thus, in order to send a message to a process, in fact an operation on the specific port type has to be called, providing that the type of the message sent coincides with the type of the input message defined for the given port type.

BPEL defines the concept of a partner link, which represents a dependency between two services. A partner link specifies roles played by the services, and the port types supported by each of the roles. In order to represent an interaction of a process with another service, a valid partner link needs to be defined. In Oracle BPEL PM, in order to define the partner link, one has to look up the required service in a Universal Description, Discovery, and Integration (UDDI) browser. UDDI is a specification for maintaining standardized Web-based distributed directory containing information about Web services, i.e. their capabilities, location and requirements in a universally recognized format [13].

Two types of interactions can be implemented in Oracle BPEL PM: synchronous and asynchronous. In a synchronous interaction, a client sends a request to a service, and immediately receives a reply. In an asynchronous interaction, a client sends a request and waits until a service replies. When describing the implementation of a pattern variant, we will explicitly indicate the type of interaction modeled.

Having introduced the main technologies and concepts, the knowledge of which is necessary for understanding the details of implementing services in Oracle BPEL PM, we proceed with pattern evaluations. For each of the pattern families, first, we illustrate how a selected pattern variant can be implemented in Oracle BPEL PM. For this, we describe the mapping between the configuration parameters and corresponding settings in the implementation presented. Thereafter, we discuss the support for other pattern variants by analyzing each of the configuration parameters in a detail.

9.2 Multi-party Multi-message Request-Reply Conversation

Figure 44 shows the notation for the pattern variant where all configuration parameters are set to the default values. In this pattern variant, a `Requestor` sends one message to a `Responder`, who responds with a reply message. Messages received by the `Requestor` are not queued and consumed immediately (one message is required for the requestor to start the consumption and utilization of the messages received). The `Requestor` consumes a message from the queue only once, and the rest of the messages are discarded. Below we describe what steps have to be taken in order to realize this example in Oracle BPEL PM.

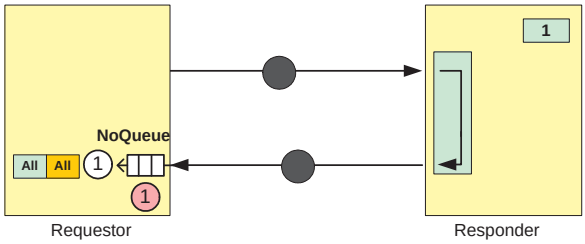


Fig. 44. Notation for the default pattern variant of
the Multi-party Multi-message Request-Reply Conversation pattern family

We implement the considered example as an asynchronous interaction, because there is a possibility that the responder service will not reply.

Each of the processes we consider, i.e. the requestor and the responder, needs to define the other process as a `partnerLink`. Since the requestor process needs to send a message to one responder process, only one partner link needs to be defined. In order to send a message, the requestor process needs an `<invoke>` activity, and in order to receive a message, one of the Inbound Message Activities (IMA), i.e. `<receive>`, `<pick>`, and `onEvent` is required. The responder process needs a `<receive>` activity to accept the incoming request, and a `<reply>` activity to return either the requested information or an error message (a fault).

The requestor and responder processes, implemented in Oracle BPEL PM, are illustrated in Figures 45(a) and (b) respectively. Figure 45(a) shows an asynchronous process which upon an initiation by a client performs an invocation of a

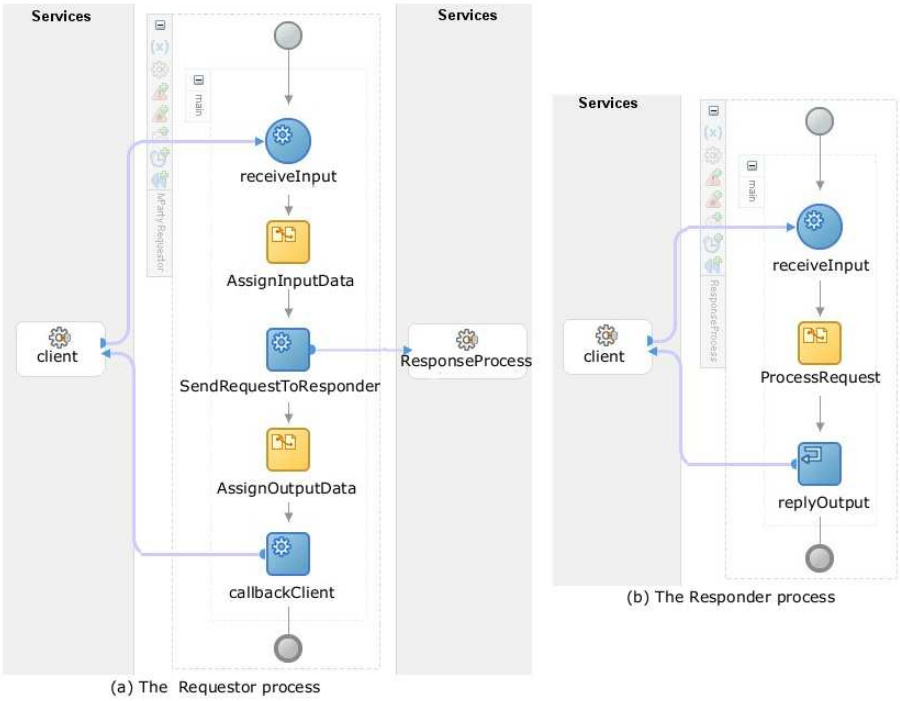


Fig. 45. Implementation in Oracle BPEL PM: the default pattern variant of the Multi-party Multi-message Request-Reply Conversation

synchronous service `ResponseProcess` presented in Figure 45(b) using an invoke activity `SendRequestToResponder`. The request message sent by the requestor process is specified in the `RequestorInputVariable` input variable of the invoke activity as shown below. Note that the type of the message sent by one process to another has to be the same in the input variable of the invoke activity and in the output variable of the reply activity. The message types are defined in the WSDL definition of each of the interacting processes. In this case, the input and output variables are of the String type (the source code of the XML, WSDL and BPEL files can be found in []).

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

In order to specify the content of the message, prior to the invoke activity the value of its input variable has to be set. For this, the `<assign>` activity `AssignInputData` in Figure 45(a) is used.

The `ResponseProcess` process is initiated by a message received from the requestor process. The message received via the `<receive>` activity is processed by an <assign> activity `ProcessRequest` in Figure 45(b), and a response is

sent back to the requestor process using a `replyOutput` reply activity. The response message sent by the responder process is assigned to an output variable `ObtainedOutputVariable` of the `SendRequestToResponder` invoke activity. Note that the <invoke> activity has no attribute for message queueing, therefore response messages are not queued and are consumed and processed as soon as they arrive. This maps to consumption and utilization indexes whose value is set to one, and the sorting of messages configuration parameter whose value is set to `NoQueue`. Messages only of a specific type can be consumed by the `<invoke>` activity, other messages are ignored. In this implementation, as soon as the corresponding message is received by the `<receive>` activity of the requestor process, the flow of control proceeds. This maps onto the consumption frequency configuration parameter whose value is set to one.

For each of the configuration parameters listed below, we describe how different values from the predefined range can be realized.

- *N (number of sub-requests in a message)*: to include multiple sub-requests in a message, the type of the message has to be of composite form. Since WS-BPEL is XML-based, it is possible to define custom data types in the XML-schema and use them afterwards for message definition in the corresponding WSDL-source. Sending a message of the defined type to another service can be done via an `<invoke>` activity as has been described earlier.
- *M (number of responders involved in a conversation)*: by definition, an invocation activity may call operations only on a single service represented as a partner link. In order to involve multiple responding parties in the conversation, every party has to be defined as a separate `PartnerLink`. To send the same message to the set of defined partner links, a `<flow>` construct must be introduced with as many parallel branches as there are partner links, providing that a separate `<invoke>` activity is placed in each of the branches. The model resulting from the addition of a separate branch for interacting with each responding process may become complex. As an alternative to this implementation, a new variant of the `<invoke>` activity associated with a set of partner links can be defined using an `<extensionActivity>`.
- *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 process, the `<invoke>` activity is blocked until the response is received. This however does not guarantee that the service invoked will respond. A fault response to an `<invoke>` activity can be generated if the request sent is undeliverable (fault types and how they are handled can be explicitly defined in the `<invoke>`). The definition of such a fault is handled in the WSDL operation.
- *Possibility of missing replies*: in WS-BPEL inbound message activities may complete only after they have received a matching message. However, in some situations an *orphaned IMA* occurs when an inbound message activity remains to be blocked for which no matching message has been received. 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*: WS-BPEL defines that a receiving activity needs at most one message to proceed. A message is processed as soon as it has been received by the matching target activity. This corresponds to the pattern variants, where NoQueue is specified. However, in situations where a receiving activity is not ready for consumption and multiple messages arrive simultaneously, 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. In order to support scenarios where multiple messages are required in order to make a decision, the receiving activity has to be included in a loop. During each iteration, an array can be used to aggregate data from the messages received. When sufficient number of messages have been received, the array data can be examined and the required data can be extracted from it. FIFO, LIFO and PRIO ordering of messages received can be realized via the extension construct `<extensionAssignOperation>`. These newly defined operations should be used for assigning data from the received messages to the array. Note, that the inability to queue incoming messages limits the number of service interaction pattern variants supported by WS-BPEL. For instance, scenarios where non-consumed messages have to be kept in the queue for future use are not possible.
- *Enabling condition*: an IMA becomes enabled as soon as a matching message has been received by a process instance (i.e. a message of a specific type). This corresponds to an enabling condition where the number of messages required is set to one, i.e. K=1. If queuing of the messages is realized by a receiving activity inserted in a loop, the enabling condition examining properties of the messages received can be specified as part of the loop termination condition. To realize enabling for message consumption based on the timeout, a `<wait>` construct can be used.
- *Consumption index*: only one message at a time can be consumed by an inbound message activity. In WS-BPEL, due to the absence of the message queueing facilities, it is not possible to specify that a subset of the messages received has to be consumed for processing. The consumption index of a queue that is associated via the receiving activity embedded in the loop, is set to `All` for consumption of all messages received in the loop.
- *Utilization index*: inbound message activities can only consume one message at at a time, therefore the message consumed is also the one used for processing, and is set to 1.
- *Consumption Frequency*: in WS-BPEL, it is possible to specify that a party may consume messages multiple times if an IMA is placed in a `<while>` or `<repeatUntil>` loop construct. The consumption frequency in this case is defined by the evaluation of the Boolean condition defined in these repetitive constructs.

To conclude, the majority of pattern variants belonging to the *Multi-party Multi-message Request-Reply Conversation* pattern family can be realized in WS-BPEL through the entities it offers for defining a business process and some programmatic extensions. However, since support for queues in inbound message activities

is missing, pattern variants based on queueing and consumption of multiple messages are hard or even im possible to realize.

9.3 Renewable Subscription

Figure 46 shows a graphical notation for the Provider-initiated Customer-renewed subscription. We will use this subscription type as an example for realizing in Oracle BPEL PM.

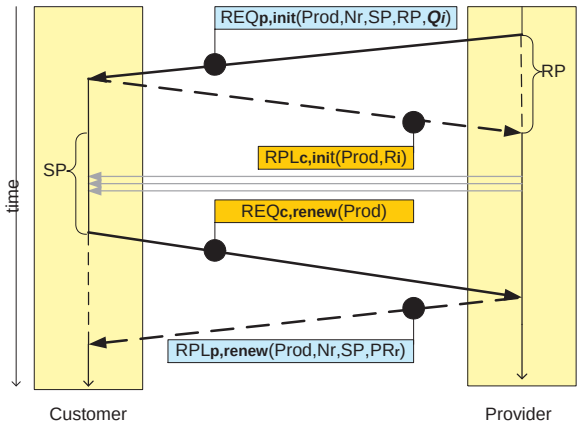


Fig. 46. Implementation in Oracle BPEL PM: Provider-initiated Customer-renewed subscription

In this example, two communicating parties, i.e. a customer and a provider, are involved in a conversation that consists of two phases: a subscription initiation and a subscription renewal. The customer and the provider have to be implemented as two processes. Each of these processes has to declare the other process as a `PartnerLink`, with whom it will interact. Figures 47 and 48 show the implementation of the provider process, and figures 49 and 50 show the implementation of the customer process respectively.

The processes presented describe the logic of the subscription renewal, where the provider sends to the customer via an `<invoke>` activity an offer for the subscription initiation. This offer specifies the product offered, the subscription period, the period within which the customer has to respond, and the expected initiation confirmation. The messages exchanged in this example are of composite data types, which have to be defined in the XML-schema shared between both the provider and customer processes, since no predefined messages of subscription type are available. The XML-schema shown below declares four message types that map directly on the graphical notation in Figure **??**. These are `Request, Response, RenewRequest` and `RenewResponse` defined for the `RespondingCustomerProcess` customer process to represent an initiation request issued by the provider, an initiation response issued by the customer, a renewal request issued by the customer,

and a renewal response issued by the provider respectively. These message types are used in invocation and inbound message activities of both processes.

```
<schema attributeFormDefault="unqualified" elementFormDefault="qualified"
        targetNamespace="http://xmlns.oracle.com/RespondingCustomer"
        xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="RespondingCustomerProcessRequest">
  <complexType>
   <sequence>
    <element name="Prod">
     <simpleType>
      <restriction base="string"/>
     </simpleType>
    </element>
    <element name="Nr" type="int"/>
    <element name="SP" type="int"/>
    <element name="RP" type="int"/>
    <element name="Q">
     <simpleType>
      <restriction base="string">
       <enumeration value="Yes"/>
       <enumeration value="No"/>
       <enumeration value="YesNo"/>
      </restriction>
     </simpleType>
    </element>
   </sequence>
  </complexType>
 </element>
 <element name="RespondingCustomerProcessResponse">
  <complexType>
   <sequence>
    <element name="Prod" type="string"/>
    <element name="R">
     <simpleType>
      <restriction base="string">
       <enumeration value="Yes"/>
       <enumeration value="No"/>
      </restriction>
     </simpleType>
    </element>
   </sequence>
  </complexType>
 </element>
 <element name="RespondingCustomerProcessRenewRequest">
  <complexType>
   <sequence>
    <element name="Prod" type="string"/>
   </sequence>
  </complexType>
```

```
 </element>
 <element name="RespondingCustomerProcessProviderRenewResponse">
  <complexType>
   <sequence>
    <element name="Prod" type="string"/>
    <element name="Nr" type="int"/>
    <element name="SP" type="int"/>
    <element name="PR">
     <simpleType>
      <restriction base="string">
       <enumeration value="Accept"/>
       <enumeration value="Reject"/>
      </restriction>
     </simpleType>
    </element>
   </sequence>
  </complexType>
 </element>
</schema>
```

In general, in a renewable subscription process both the customer and the provider of a service can play a role of the subscription initiator. For this, an initiating party has to execute an `<invoke>` activity, while the invoked party has to respond tothe message received via a `<receive>` or `<pick>` activity, where the `createInstance` attribute is set to "yes" (the latter is needed to create a new process instance which can be correlated with a given subscription). In this specific example, the provider process plays the role of the subscription initiator.

The top level of the provider process is shown in Figure 47. After a process has been initiated, the `AssignVariationPoints` activity is executed that sets configuration parameters in the initiation request to a value from the defined range. The product identifier, subscription period, response period, an expected response period are the values specified to the `InvokeInitCustomerInputVariable` input variable of the `InvokeInitCustomer` invocation activity. This activity is of the synchronous type, which means that the flow of control in the provider process is blocked until a response from the customer process arrives.

Figure 48 shows the details of processing the renewal request sent by the customer when the accepted subscription is about to expire. The `<switch>` construct is used to model that either no response is sent by the provider or a response approving or rejecting the subscription renewal request is sent to the customer. Figure 49 illustrates the top view of the customer process, that is initialized upon receival of a subscription offer from the provider process via the `<receiveInput>` activity. The assignment activity is used to set static attributes and configuration parameters of the customer response, i.e. whether the subscription offer will be accepted, rejected or ignored.

The details of processing the offer received by the customer are shown in Figure 50. Several `switch` cases are used to define the content of the initiation response message depending on the type of the request posted by the Provider. For instance, for the subscription offer whose confirmation attribute Qi="No", no

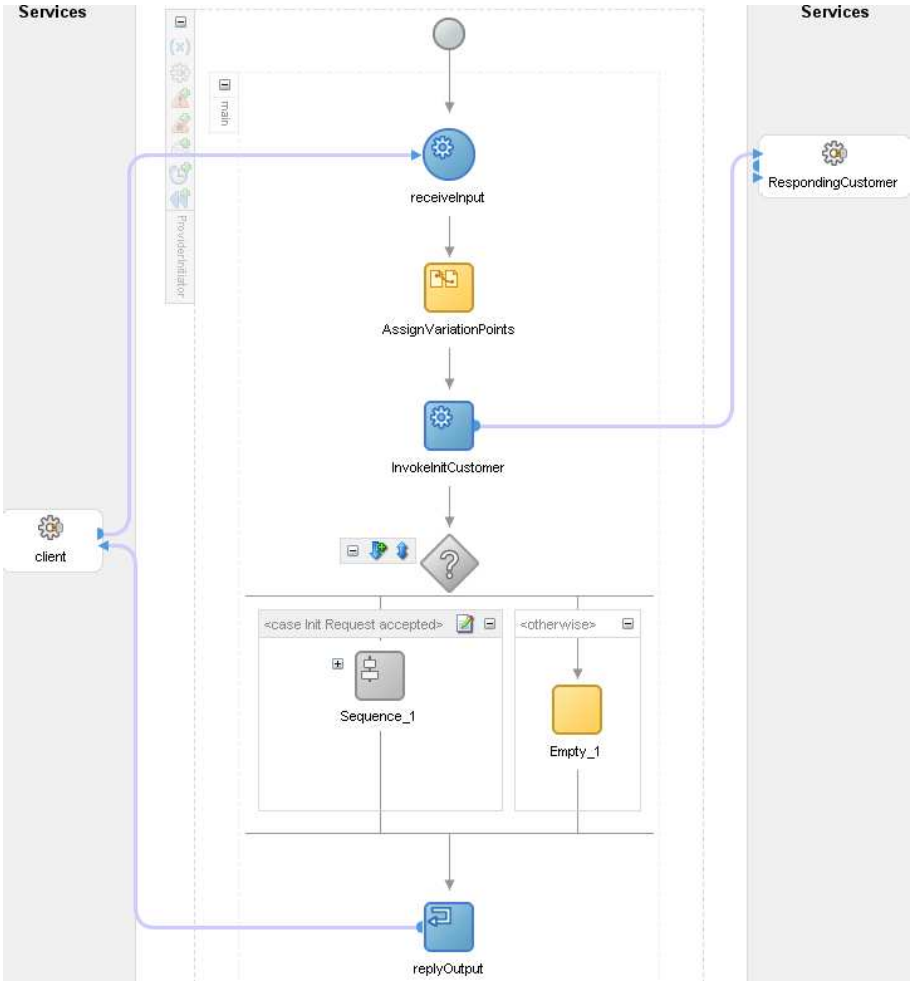


Fig. 47. Provider process: top level

response is required to accept the subscription, while for Qi="Yes/No" an explicit response needs to be sent.

Figure 49(b) shows the logic for renewing the subscription on the initiative of the customer. The decision of the customer process to renew or not to renew the subscription is implemented via the `<switch>` construct with two branches. If the subscription does not need to be renewed, an `<empty>` activity is executed, after which the customer process is terminated, otherwise the invocation activity is executed to send a message of the `RenewRequest` type to the provider process.

In this example, only one configuration parameter (`Qi`) needs to be set in order for a specific pattern variant to be realized. The expected initiation confirmation

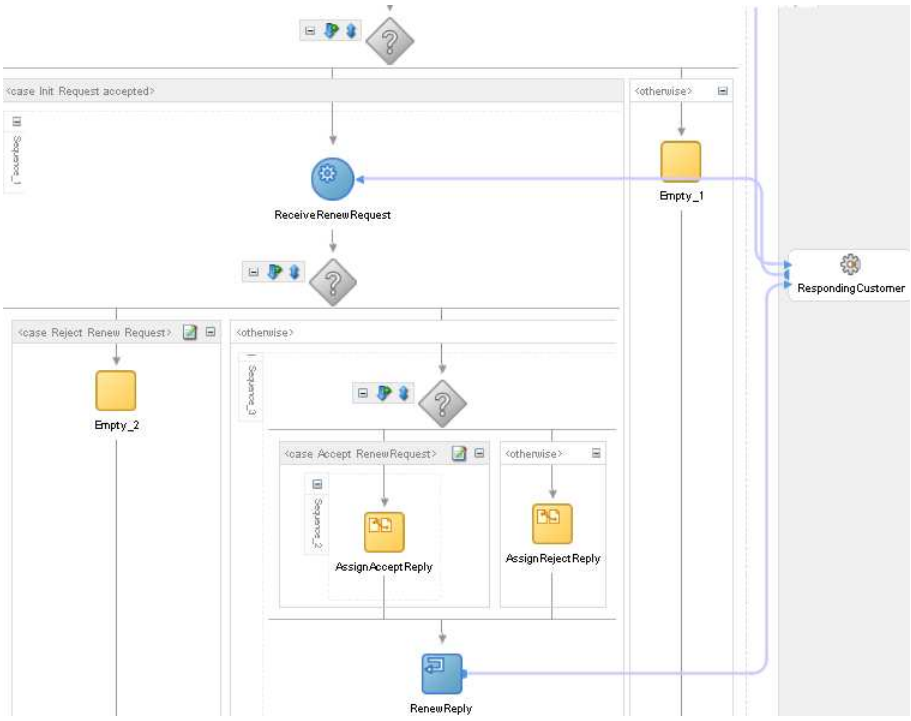


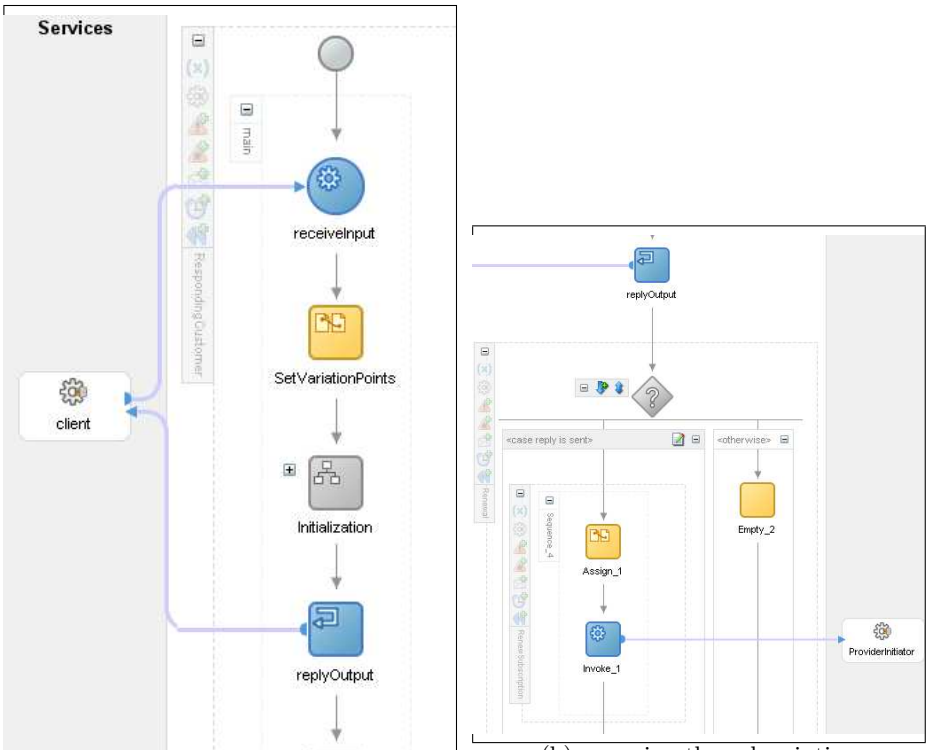
Fig. 48. Provider process: processing of the renewal request

parameter is set in the `<assign>` activity before the subscription initiation offer is sent to the customer process.

We have shown how different pattern variants for the Provider- initiated Customer- renewed subscription type can be realized in Oracle BPEL PM. To illustrate what other pattern configurations can be realized in WS-BPEL, we discuss each of the configuration parameters below.

- *Subscription renewal type*: six subscription renewal types can be realized in the similar manner as it has been done for the Provider-initiated Customer-renewed subscription earlier. In these subscription renewal types, one of the processes, either a customer or a provider, has to take an initiative for the subscription initiation. For this, an initiating party has to execute an `<invoke>` activity, while the invoked party has to react on the message received via a `<receive>` or `<pick>` activity, where the `createInstance` attribute is set to "yes" (the latter is needed to create a new process instance which can be correlated with a given subscription).

  The subscription renewal can be done as at the initiative of the customer, the provider or automatically. For automatically renewed subscriptions, in the customer process the renewal phase has to be realized as a `<while>` construct that contains an activity for receiving the product subscriptions from



(a) top level

(b) renewing the subscription

Fig. 49. Customer process

the provider process. The termination condition of this `<while>` construct determines whether the subscription should be cancelled or not. If a decision to terminate the subscription is taken, a message of the cancellation type has to be sent to the provider process.

Different message types, i.e. initiation, renewal, and cancelation requests and replies issued by the customer and the provider, must be defined for each of the subscription renewal types. As it has been shown, the data type declarations are made in the XML-schema that has to be included in both the provider and the customer processes. The data types have to include of the configuration parameters and static attributes, which when instantiated represent the pattern configuration for a specific pattern variant.

- *Expected initiation confirmation*: this configuration parameter constitutes a part of the initiation request issued by the provider process. It has to be defined as an element with three possible values: `Yes`, `No`, `YesNo`. For a specific

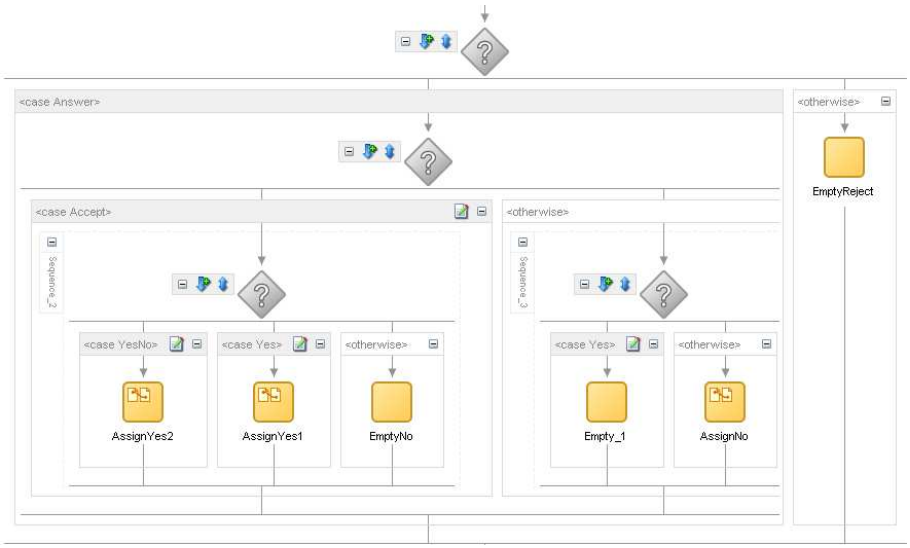


Fig. 50. Customer process: processing the subscription initiation request by Customer

pattern variant, this parameter needs to be set to one of these values depending on what confirmation the provider process expects. In order to accept the subscription offer whose `Q` field is set to `Yes` or `YesNo` the customer process has to send `Yes` in the response message, or not to send any response if the `Q` field is set to `No`.

- *Expected renewal confirmation*: this confirmation parameter constitutes a part of the subscription renewal request issued by the provider process, thus it is applied only to the provider-renewed subscriptions. The value of this configuration parameter has to be set to `Yes`, `No`, `YesNo` in the provider-renewal request. The message of such type can be defined as it has been described for the expected initiation confirmation parameter.

  The decision of the customer to accept, reject or ignore the offer can be encoded using the `<switch>` construct with three branches. Depending on the type of the confirmation expected by the provider, either a message accepting or rejecting the offer is returned or no message is sent at all.

WS-BPEL provides basic primitives to describe behavior of a business process based on interactions with other processes through web service interfaces. Although it concentrates on message exchange between processes, WS-BPEL does not define mechanisms for conversation-oriented scenarios like renewable subscriptions because they are considered to be operate at a higher level of abstraction and they are typically managed at application level. Nevertheless, WS-BPEL provides business logic to realize all pattern variants of renewable subscription types via predefined modeling entities and extension constructs.

9.4 Message Correlation

An example of the Message Correlation pattern variant where all configuration parameters are set to the default value is presented in Figure 51. In this example, a customer sends an order request to the provider, where it specifies its address (`Ps`), the order id (`Cs`) and the address of the customer `Pr`. The provider receives the order request and records the id of the order request received to use it for the future correlations. We will use this example for illustrating an implementation in Oracle BPEL PM.

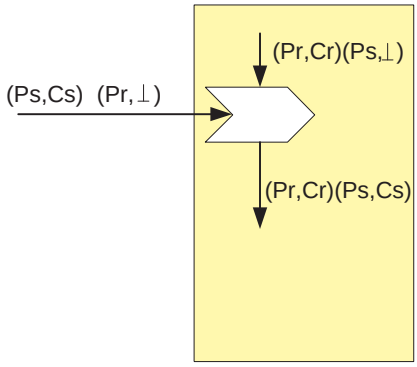


Fig. 51. Default notation: Message Correlation

There are two approaches for realizing message correlation in Oracle BPEL PM. Either of the two approaches WS-Addressing [19], BPEL correlation sets or their combination can be used for this purpose. Oracle BPEL PM uses WS-Addressing to automatically set location and correlation information associated with the client role. WS-Addressing defines two concepts: endpoint references and message information headers. Endpoint references convey the information providing addresses for individual messages sent to and from Web services [19]. The endpoint reference format shown below includes the `<Address>` that could be used for specifying of the address of the message sender or the message receiver. Furthermore, it includes `<ReferenceProperties>` or `<ReferenceParameters>` fields suitable for including the conversation identifier used by the party for correlation purposes.

```
<wsa:EndpointReference>
    <wsa:Address>xs:anyURI</wsa:Address>
    <wsa:ReferenceProperties>... </wsa:ReferenceProperties> ?
    <wsa:ReferenceParameters>... </wsa:ReferenceParameters> ?
    <wsa:PortType>xs:QName</wsa:PortType> ?
    <wsa:ServiceName PortName="xs:NCName"?>xs:QName</wsa:ServiceName> ?
    <wsp:Policy> ... </wsp:Policy>*
</wsa:EndpointReference>
```

The message information headers convey message attributes including addresses of source and destination endpoints, and the message identity. The message

information header format shown below specifies the source address via the `From` field and the destination address via the `To` field. The address of the message destination must be explicitly specified, while the source address may be omitted (in this case the message sender may stay anonymous). These requirements directly map to the `From` and `To` configuration parameters of the *Message Correlation* pattern family.

```
<wsa:MessageID> xs:anyURI </wsa:MessageID>
<wsa:RelatesTo RelationshipType="..."?>xs:anyURI</wsa:RelatesTo>
<wsa:To>xs:anyURI</wsa:To>
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:From>endpoint-reference</wsa:From>
<wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
<wsa:FaultTo>endpoint-reference</wsa:FaultTo>
```

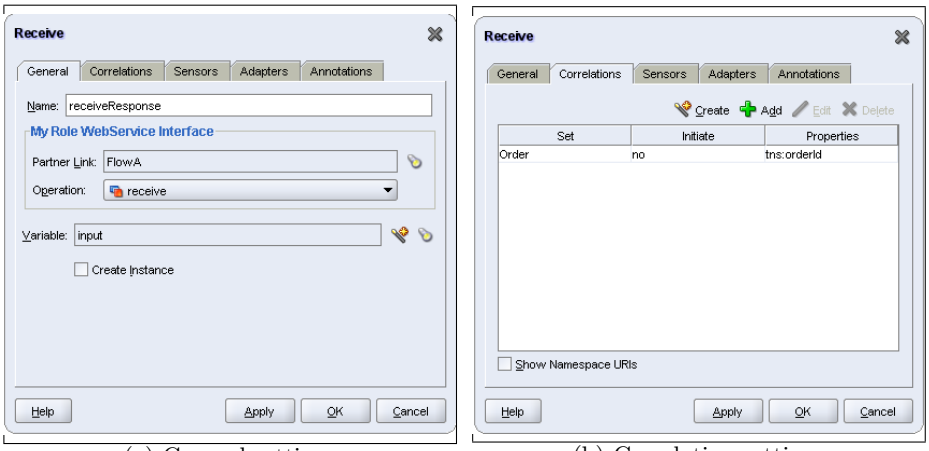
Endpoint references must contain addressing information identifying an endpoint on the mandatory basis. Reference properties and reference parameters specify individual properties and parameters required to properly interact with the endpoint. This information is optional and may be omitted. When address and reference properties supplied in a message are compared with actual address and reference properties of the endpoint, these are checked for equivalence. Not only addresses of endpoints must match, but also the endpoints must contain the same number of individual properties and for each reference property there must exist an equivalent reference property in the other endpoint. If the sender address appears to be anonymous, other information specified in the message information header about the message id or about the relation of this message to another message can be used to identify the destination where reply messages can be sent to. This combines both the key-matching and property-based analysis correlation methods.

In addition to WS-Addressing, Oracle BPEL PM supports the specification of correlation sets, i.e. a set of properties, that are used amongst web-services to uniquely identify a conversation. To use correlation sets for message correlation, custom data types have to be defined in the XML-schema of the communicating processes. An example below shows the declaration of the order type that will be used to uniquely identify orders by their `orderID` identifiers.

```
<element name="order" type="tns:orderType"/> <complexType name="orderType">
        <sequence>
            <element name="orderId" type="string"/>
         </sequence>
</complexType>
```

The actual definition of correlation sets is done in the BPEL-source of both sender and receiver processes as follows:

```
<correlationSets>
      <correlationSet name="Order" properties="tns:orderId"/>
</correlationSets>
```



(a) General settings (b) Correlation settings

Fig. 52. `Correlation settings of the <Receive>` activity

A receiving activity processes a message, whose property `orderID` matches with the property specified in the correlation set (as shown in Figure 52).

When a message with a specified correlation set is sent to a process for the first time, the correlation set has to be initialized. For this, its `initiate` attribute is set to `yes` as shown in the following code fragment.

```
<receive name="receiveInput"
            partnerLink="client"
            portType="tns:Customer"
            operation="initiate"
            variable="input"
            createInstance="yes">
            <correlations>
                <correlation set="Order" initiate="yes"/>
            </correlations>
 </receive>
```

We have shown how a Message Correlation pattern variant can be implemented in Oracle BPEL PM. In this implementation, both the facilities of the WS-Addressing and correlation of BPEL have been used. The combination of WS-Addressing and correlation sets allows for the specification of the majority of message correlation pattern variants except the ones where the identity of the target process is underspecified by the sender process. Below we analyze which pattern variants can be realized using WS-BPEL v.2.0. The relevant configuration parameters are:

- *Message Sender field*: instead of the party identifier and the conversation identifier representing the credentials of the message sender, WS-BPEL provides a mechanism to specify a set of properties shared by all messages in the correlated group of operations within a process instance known as a correlation set.

  `<correlationSets>?`

```
    <correlationSet name="NCName" properties="QName-list">+
</correlationSets>
```

Uniquely named correlation sets together with the partner link can be used to represent the credentials of the message sender.

- *Message Receiver field*: in WS-BPEL, the credentials of the message receiver are not included in the message explicitly. To send a message to a certain party, a `PartnerLink` is defined to whom the message is targeted. Thus the destination address is specified in the process definition or it may be set dynamically, however it is not possible to send a message to an arbitrary party. In order to specify the conversation identifier used by the message receiver, a correlation set initiated by the message receiver in the previous interaction has to be specified.
- *Credentials of the message sender before message correlation*: parties with whom the message receiver has been or will be involved in a conversation with, have to be defined as partner links. Since a single dynamic partner link can be used by different parties, messages sent by these parties have to be accompanied by uniquely named correlation sets in order for these parties to be unambiguously distinguished. It is possible that initially the message receiver does not have any knowledge about the correlation set used by the message sender.
- *Credentials of the message sender after message correlation*: If a message received by a party contains a correlation set that has not been initialized yet, the message receiver has to initiate it. After this, the initiated correlation set can be used in follow-up interactions. If at the moment of the message receival, the correlation set has been already initiated, no information is gained.

Although WS-BPEL claims to support message correlation, in fact it offers a work-around solution based on correlation sets. Instead of using a party identifier and a conversation identifier in a message to unambiguously identify the series of previously exchanged messages relevant to the conversation, it mandates the obligation for all processes to adopt and utilize the same set of identifiers in the form of correlation sets. The drawback of such an approach for correlation is that it narrows the range of message interactions that can be successfully correlated. As such, WS-BPEL is not capable of correlating messages where incomplete credentials are specified for the message sender. In WS-BPEL, an incomplete or incorrect specification of the correlation set would be considered as an exceptional situation necessiating that either a new process instance be created for handling of the message or that message be discarded.

In a more general case, despite the underspecified information about the message sender credentials, it should be possible to identify the related conversation by analyzing other properties of the message received (for instance, as can be done in WS-Adressing by analyzing the reference properties and reference parameters). WS-BPEL also does not handle scenarios where the receiving party forgets to retrieve some of the correlation-related information specified in the message. If

incomplete information about the message sender credentials is used in the follow-up response, the response message would not be successfully correlated (unless property-based analysis of this message is performed).

9.5 Message Mediation

In this section, we illustrate how a Mediated Introduction pattern variant whose configuration parameters are set to the default value can be implemented in Oracle BPEL PM. We omit the implementation of the Mediated Interaction pattern variant, since the realization of the default pattern variant is very similar to the example of the Bipartite Conversation Correlation presented in this next section the only difference being with that instead of two parties, three parties are involved in the conversation, and for correlation of messages two kinds of correlation sets are used: one shared between the customer, mediator, and provider, and the other shared only between the mediator and the provider.

Figure 53 illustrates the graphical notation of the Message Introduction pattern variant where all pattern attributes are set to the default value. In this pattern variant, the customer sends a request to the mediator who in turn has to delegate it to the provider. The customer does not specify credentials of the provider (the `Them1` field is empty), therefore the mediator is responsible for its selection. The customer expects a response to be received back. In order for the provider to contact the customer directly, the mediator passes the reference to the credentials of the customer in the `Them2` field. When sending the response message to the customer, the provider specifies all information about its credentials in the `From5` field, and also the reference to the mediator in the `Them5` field.

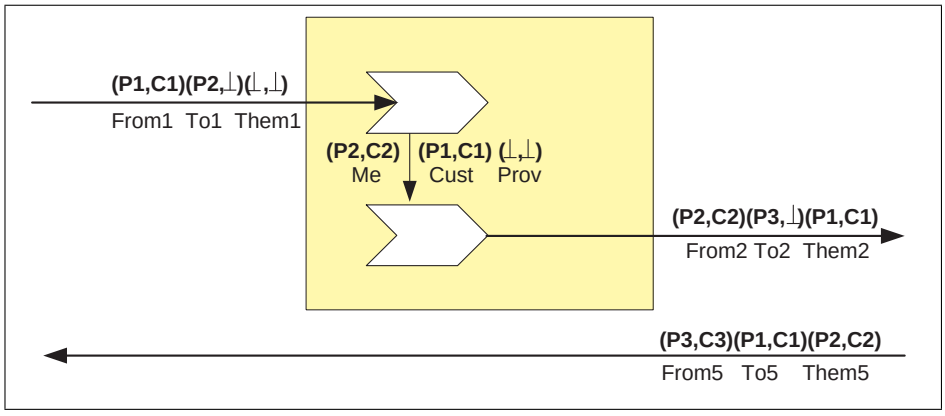


Fig. 53. Default notation: Message Introduction

In the default Mediated Introduction pattern variant, a customer process named `FlowA` sends a request to the mediator process `FlowB`. The mediator process forwards the request to the provider process `FlowC`, who based on the information provided by the mediator dynamically binds with the customer process and sends

the response message directly to it (note that the customer process does not know the provider process initially). The customer, mediator and provider processes are shown in Figures 54, 55 and 56 respectively.

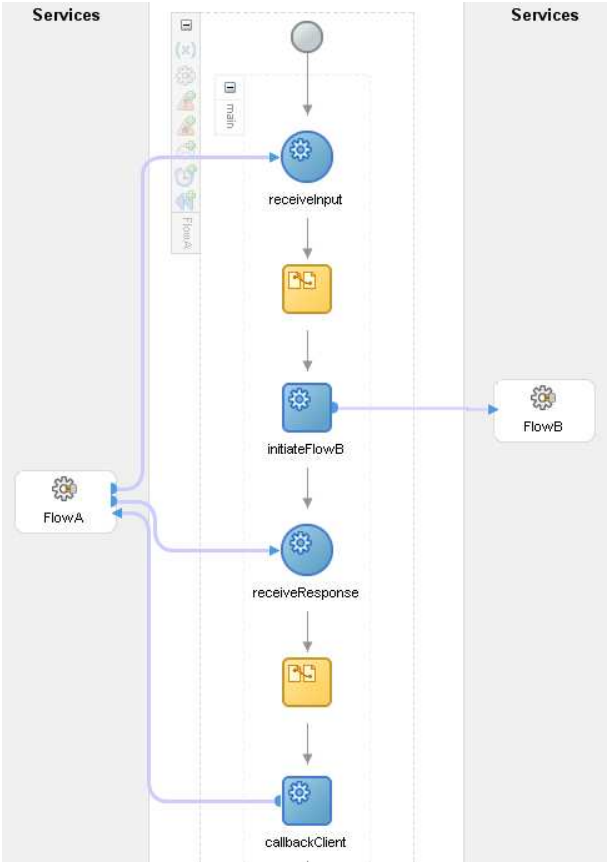


Fig. 54. Customer process: FlowA

To correlate message exchange between these three processes, message invocation and receival activities have to be associated with a shared correlation set. In this example, an order identifier is used as a property for correlation set `Order`.

```
<correlationSets>
    <correlationSet name="Order" properties="tns:orderId"/>
</correlationSets>

<complexType name="orderType">
        <sequence>
            <element name="orderId" type="string"/>
         </sequence>
```

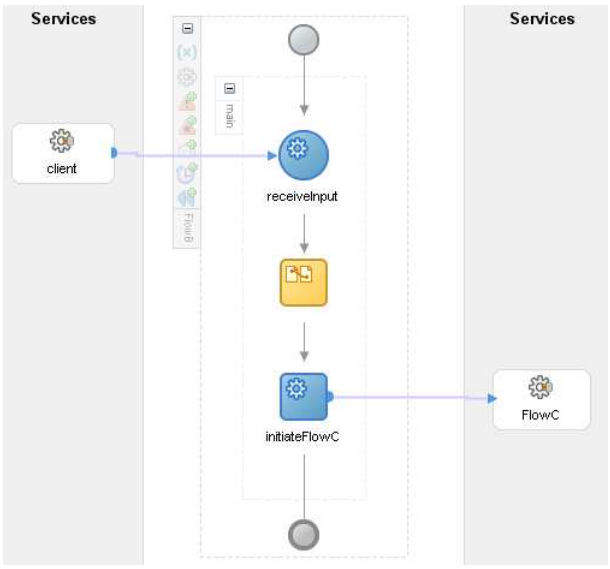


Fig. 55. Mediator process: FlowB

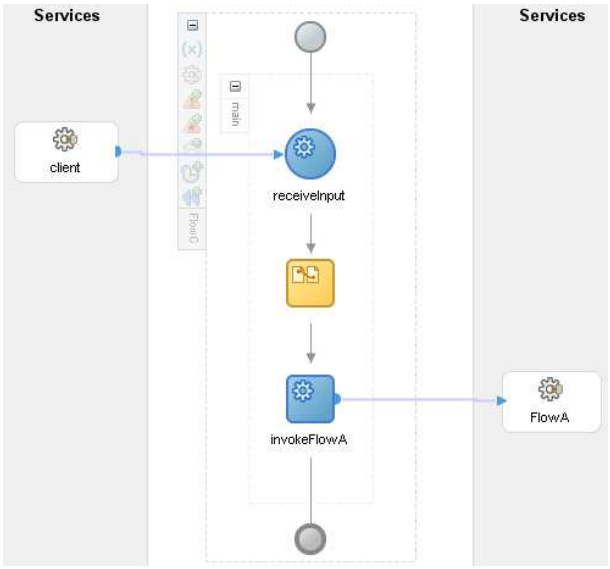


Fig. 56. Provider process: FlowC

```
</complexType>

<bpws:property name="orderId" type="xsd:string"/>
```

```
<bpws:propertyAlias propertyName="tns:orderId"
      messageType="tns:ABCARoutingFlowRequestMessage" part="payload"
      query="/tns:order/tns:orderId"/>
```

After the customer process has sent a request, it awaits an asynchronous callback. In this example, the callback location and correlation id is transparently handled using WS-Addressing. In particular, WSDL-file of the deployed `FlowA` process contains the fields specifying that the `ReplyTo` field of the message information header is used to utilize the address to which the response message has to be sent. This corresponds to the configuration parameter `Them2`. Furthermore, the address location is set dynamically by the caller.

```
<mesage name = "WSAReplyToHeader">
  <part name = "ReplyTo" element = "wsa:ReplyTo"/>
</message>

<service name = "FlowACallbackService">
  <port name = "FlowACallbackPort" binding="tns:FlowACallbackBinding">
    <soap:address location = "http://set.by.caller" />
  </port>
</service>
```

To support all Message Mediation pattern variants, it should be possible to specify in a message the credentials of a third-party process; the message receiver should be able to dispatch the information about the credentials of the third party and dynamically bind with it. These requirements are met by WS-Addressing using the concepts of endpoints and message information headers, whose format has been described in Section 9.4.

An endpoint reference is the information needed to identify a process. The endpoint reference must contain an address of the process and may contain other properties and parameters that are necessary to properly interact with the process.

Message information headers convey information about the source and destination endpoints of a given message in the `From` and `To` fields, and `ReplyTo` address that can be used for as a destination address for the follow-up responses.

```
<wsa:MessageID> xs:anyURI </wsa:MessageID>
<wsa:RelatesTo RelationshipType="..."?>xs:anyURI</wsa:RelatesTo>
<wsa:To>xs:anyURI</wsa:To>
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:From>endpoint-reference</wsa:From>
<wsa:ReplyTo>endpoint-reference</wsa:ReplyTo>
<wsa:FaultTo>endpoint-reference</wsa:FaultTo>
```

The only mandatory field in the message information header is the address of the message destination `To`. This means that the message sender may stay anonymous by omitting the detailed specification of the `From` field, while at the same time it may provide additional information in the `ReplyTo` field, which will be used as the address to which to send the response message. The `RelatesTo` field may contain information specifying how the given message relates to other messages. In addition, the `MessageID` field may contain a unique message identifier that could be used for correlation purposes. In the considered example, the `ReplyTo` field of

the request sent by the mediator to the provider contains the credentials of the customer process (`Them2` field in Figure **??**). The provider uses this information to specify in the `To5` field representing the address of the party to whom the response message will be sent, i.e. the address of the customer.

We have shown an example of how the Mediated Introduction pattern variant can be implemented in Oracle BPEL PM. In this implementation both facilities of WS-Addressing and BPEL were used, and it is interesting to see what other pattern variants can be implemented in WS-BPEL v.2.0. For this, we analyze each of the configuration parameters below:

- *Customer request for specific provider*: in order to implement pattern variants, where the customer specifies the credentials of the provider, the partner link of the provider has to be set dynamically.
  In order to send a request to the provider after the mediator has been triggered by a request received from a customer, the mediator should be able to dynamically bind to the reference provided. In WS-BPEL, this can be done via assignment of end-point references using the `<Assign>` activity. The reference provided is copied from the `<from>` field to the `<to>` field:

  ```
  <from partnerLink="NCName" endpointReference="myRole|partnerRole"/>
  <to partnerLink="NCName"/>
  ```

  Note that types for values specified in the from/to-assignment clause is of the `<sref:service-ref>`.
  One could underspecify the `PartnerLink` by setting it to a generic service, and assign its value during run-time using the endpoint reference of WS-Addressing. Note that for this, a variable of the `EndpointReference` type needs to be declared:

  ```
  <variable name = "partyReference" element="wsa:EndpointReference">
  ```

  The schema for the `EndpointReference` type is provided by the WS-Addressing standard:

  ```
  <xlmns:wsa = "http://schema.xmlsoap.org/ws/2003/03/addressing">
  ```

  The value of this variable has to be assigned to the partner link that has to be configured dynamically.
- *Information about the message sender in the request from mediator to provider*: to receive a response message back, the mediator has to specify a correlation set that can be shared with the provider, otherwise the correlation set initiated by the customer service has to be passed.
- *Information about the message sender in the response from provider to customer*: the identity of the provider may be enclosed in the new correlation set (note that correlation sets have to be shared) or via end-point references as has been described earlier. An endpoint reference represents data required to describe a partner service endpoint as a service reference container `<sref:service-ref>`. This reference can be used to dynamically determine a partner service that is not known at the moment of process instantiation.

- *Information about mediator exposed by provider to customer*: in contrast to WS-Addressing where the `RelatesTo` field is used to pass the reference to related process, in WS-BPEL this can only be done in the content of the message. The message in this case has to be based on the data type defined in the XML-schema of the corresponding process.

To realize all *Message Introduction* and *Message Interaction* pattern variants, processes involved in a bilateral interaction should be able to provide references to other parties as well as to use the references provided for dynamic binding. WS-BPEL acknowledges the importance of supporting such scenarios via a notion of endpoint references.

In the *Message Introduction* and *Message Mediation* pattern variants there exists the possibility that a customer process will provide the mediator with an explicit reference to the provider process. As has been already mentioned in the `Message Correlation` section, the credentials of parties corresponding to the `From` and `To` message attributes are not used for correlation in WS-BPEL (correlation sets are used instead). This sets limitations on the number of pattern variants supported. All information related to identification of the party in interactions with other parties is not explicitly exposed and is irrelevant for correlation of messages in each of the interactions belonging to the corresponding Message Mediation pattern variant. This however, does not prohibit the inclusion of information about the party credentials in the content of messages and specification of whether the information provided should be disclosed to the third party involved in the conversation. In order to include such information in the message content, the definition of data types in the XML-schema of the corresponding processes is required.

Message Mediation pattern variants, where some of the information related to the party credentials is underspecified, cannot be realized in WS-BPEL, mainly because it has been deliberately chosen not to use this information for correlation purposes. The rest of the pattern variants can be realized, providing that instead of message sender credentials and message receiver credentials, correlation sets are used. All limitations of using correlation sets for message correlation (described in the analysis of *Message Correlation* pattern family) also apply to the *Message Mediation* pattern family.

9.6 Bipartite Conversation Correlation

Figure 57 presents the graphical notation for one of the bipartite conversation pattern variants where all configuration parameters are set to their default values. The requestor and the responder retrieve all information from the message received, and specify complete information about the identity of the message sender and the message receiver. In the follow-up interaction, the requestor uses information provided by the responder as a token of correlation. We use this example as a basis for implementation in Oracle BPEL PM.

Two processes, `Buyer` and `Seller`, one requesting a service and another offering it, are directly related and thus must be declared as partner links in the

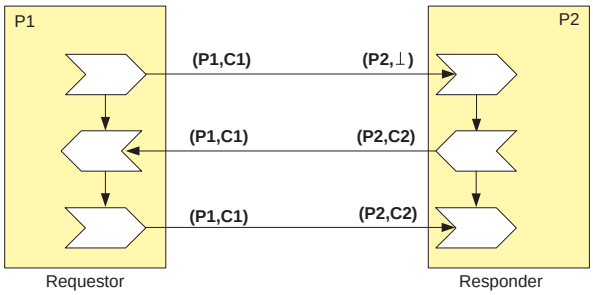


Fig. 57. Default notation: Bipartite Conversation Correlation

corresponding processes. To illustrate correlation information changing during the course of a conversation between these processes let's consider their implementations shown in Figures 58 and 59 respectively.

The `Buyer` sends a request to the Seller and specifies the correlation set that has to be used in the response message. Note that the correlation set has been initialized before the invocation of the Seller process, therefore its `initiate` attribute is set to `no`. Furthermore, the `pattern` attribute is set to `out` indicating that the correlation set applies to the outbound message.

```
<invoke partnerLink="Seller" portType="seller:Seller"
        inputVariable="input"
        name="SendRequest" operation="AsyncPurchase">
      <correlations>
        <correlation set="PurchaseOrder" initiate="no" pattern="out"/>
      </correlations>
</invoke>
```

In order to be used in the process, this correlation set must be declared in the BPEL-specification of the `Buyer` process. The correlation sets shown below are the ones used by the Buyer and Seller processes during the conversation. Each correlation set contains two properties, representing the identity of the party and the order identifier.

```
<correlationSets>
    <correlationSet name="PurchaseOrder"
                    properties="cor:customerID cor:orderNumber"/>
    <correlationSet name="Invoice" properties="cor:vendorID
                    cor:invoiceNumber"/>
</correlationSets>
```

The `Seller` process, receives an initiation request from the Buyer process, and uses the same correlation set `PurchaseOrder` for correlation. Note that this message not only creates a new process instance, but also initiates the correlation set within this process instance:

```
<receive partnerLink="Buyer" portType="seller:Seller"
  operation="AsyncPurchase" variable="input" createInstance="yes"
            name="ReceiveInitRequest">
   <correlations>
```

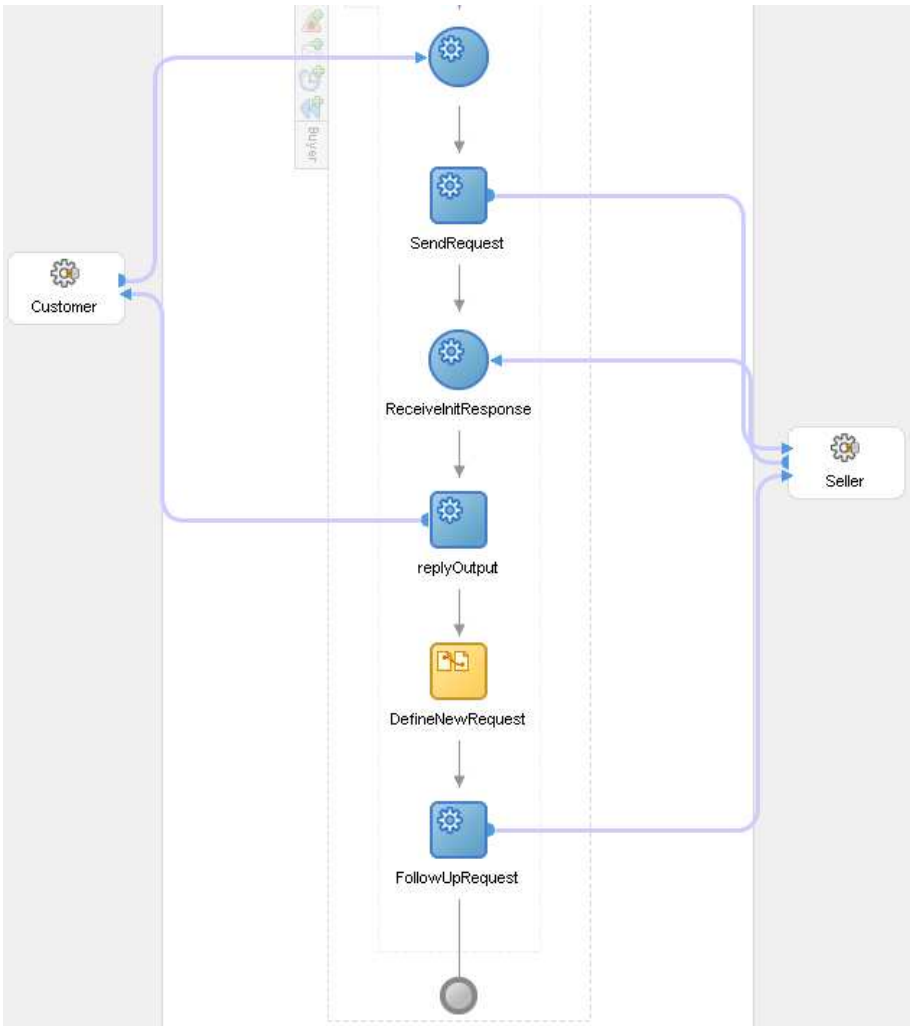


Fig. 58. Requestor process: Buyer

```
        <correlation set="PurchaseOrder" initiate="yes"/>
   </correlations>
</receive>
```

After processing an initiation request, the `Seller` invokes the `Buyer` process and next to the `PurchaseOrder` correlation set required by the `Buyer` also specifies its own correlation set `Invoice` that has to be used in the follows-up responses.

```
<invoke partnerLink="Buyer" portType="seller:Buyer"
        operation="AsyncPurchaseResponse" inputVariable="output"
        name="SendInitResponse">
   <correlations>
```

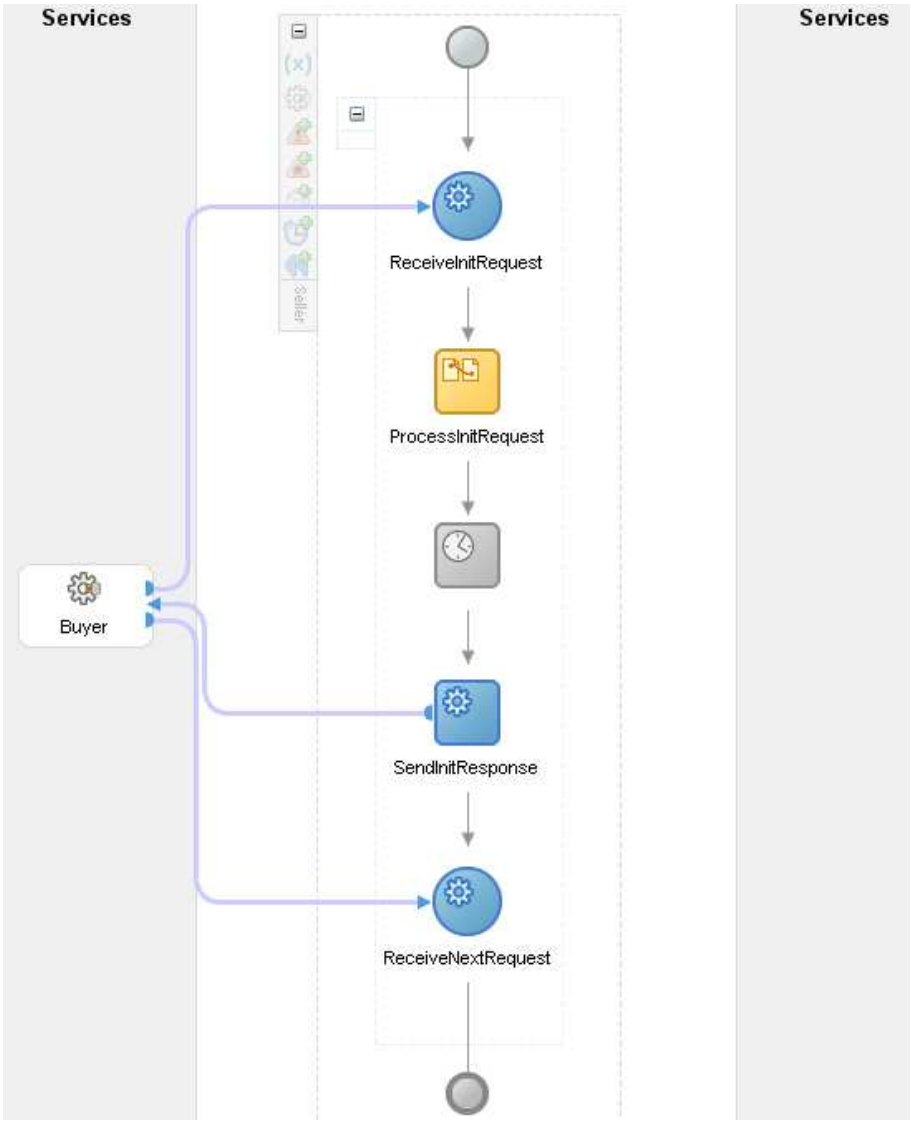


Fig. 59. Responder process: Seller

```
      <correlation set="PurchaseOrder" initiate="no" pattern="out"/>
      <correlation set="Invoice" initiate="yes" pattern="out"/>
   </correlations>
</invoke>
```

The `Buyer` process receives the initiation response from the `Seller`, and initializes the `Invoice` correlation set:

```
<receive partnerLink="Seller" portType="seller:Buyer"
```

```
        operation="AsyncPurchaseResponse" variable="output"
        createInstance="no" name="ReceiveInitResponse">
   <correlations>
      <correlation set="PurchaseOrder" initiate="no"/>
      <correlation set="Invoice" initiate="yes"/>
   </correlations>
</receive>
```

The follow-up request sent by the `Buyer` process to the `Seller` process specifies both correlation sets as shown below.

```
<invoke name="FollowUpRequest" partnerLink="Seller"
        portType="seller:Seller"
        operation="AsyncNextPurchase" inputVariable="nextinput">
   <correlations>
     <correlation initiate="no" set="PurchaseOrder"/>
     <correlation initiate="no" set="Invoice"/>
   </correlations>
</invoke>
```

Realization of pattern variants, where only part of the correlation information required by the message receiver is specified by the message sender is possible by excluding corresponding correlation set details from the correlation sets of the invocation activity. To deal with such a situation, the receiver must have a separate inbound message activity whose correlation sets match with the ones specified by the message sender. In general, such messages would not be processed, thus the corresponding conversation will not be identified.

There is an alternative means of supporting message correlation via WS-Addressing, where message information headers contain information both about the message sender in the `From` field and the message receiver in the `To` field, thus it should be possible to realize pattern variants where the message sender underspecifies its credentials. If in general, the absence of the message sender credentials makes it impossible to deliver the response message to the right destination, in WS-Addressing message information headers may contain additional information about a message id or relationship of this message to another message. By analyzing this information, a corresponding process instance to which the response message has to be sent can be found.

In order to define what other pattern variants can be realized using WS-BPEL v.2.0, we analyze configuration parameters of the *Bipartite Conversation Correlation* pattern configuration below:

- *Responder credentials in the initial response*: All correlation sets specified by the requestor process in a message initiating a conversation with the responder process must be initiated, i.e. the `initiate` attribute of every correlation set must be set to "yes". Once initiated by the requestor, this correlation set should be used by the responder process in the follow-up responses. However, next to the correlation set provided by the requestor, the responder may initiate another correlation set that should be used in follow-up messages issued by the requestor. This new correlation set may represent the provider credentials. However in order to correlate a message in which this correlation set

is specified, the receiver process must have a corresponding inbound message activity sharing the same correlation set.

- *Responder credentials in the follow-up request*: Depending on the context of the conversation, the requestor may choose to use its own correlation set or the one provided by the responder. In order for the follow-up request issued by the requestor to be successfully processed, the responder process must have a pending activity whose correlation sets match, otherwise such a message will raise the `bpel:correlationViolation` fault. In general, the correlation set specified by the responder may be combined with a new or already existing correlation set initiated by the requestor.
- *Requestor credentials in the follow-up request*: The correlation set specified by the responder process in the initiation response must be used by the requestor in the follow-up request. It may only be omitted if the responder has a pending activity whose correlation sets match, otherwise such a message will raise the `bpel:correlationViolation` fault.

The *Bipartite Conversation Correlation* pattern family addresses scenarios where knowledge used by a party for message correlation may change during the course of a long-running conversation with another party. Obviously, WS-BPEL allows parties involved in the conversation to define what information to use for correlation of messages and correlation sets may change for different activities. However, due to the limitations of the approach employed by WS-BPEL to realize message correlation (as described in the *Message Correlation* pattern family section), not all pattern variants of the Bipartite Conversation Correlation can be realized in WS-BPEL. In particular, such scenarios where a party uses only a subset of credentials provided by another party as correlation information specified in the follow-up messages or scenarios where a party specifies additional information to that requested by the other party can not be realized in WS-BPEL.

10 Conclusions

In this paper we presented a framework for generating pattern variants in the context of service orientation. The core of the framework is composed of five pattern families: Multi-party Multi-message Request-Reply Conversation, Renewable Subscriptions, Message Correlation, Message Mediation and Bipartite Conversation Correlation. These pattern families are closely related to each other by means of the concepts defined in the meta-model. In each pattern family, multiple pattern variants can be identified. Pattern variants belonging to different pattern families can be combined to solve more complex problems.

The framework presented in this paper is useful for a variety of purposes. First of all, the framework allows for generation of different pattern variants. The graphical notation proposed for each pattern family allows different patterns to be visualized and distinguished in an intuitive way. The CPN models formalize the patterns, thus minimizing the possibility for misinterpretation. Furthermore, complex service orientation scenarios can be classified by means of this framework

by relating them to the pattern families identified. Consequently, this framework can be used as a source of reference when comparing service-oriented languages and standards. In this paper we applied the pattern families to analyze the capabilities of WS-BPEL v2.0. In future we plan to use the proposed framework for testing the pattern support of a selection of WFM tools.

Acknowledgments

We would like to thank Christian Stahl and Jan Martijn van der Werf for their involvement in the clarification of message correlation scenarios.

References

1. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.
2. L. Aldred, W.M.P. van der Aalst, M.Dumas, and A.H.M. ter Hofstede. Understanding the Challenges in Getting Together: The Semantics of Decoupling in Middleware. BPM Center Report BPM-06-19, BPMcenter.org, 2006.
3. C. Alexander. *Timeless Way of Building*. Oxford University Press, 1979.
4. A. Alves and A. Arkin et al. Web Services Business Process Execution Language, Version 2.0. OASIS TC Available via http://www.oasis-open.org, Last accessed on May 18, 2007, 2007.
5. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
6. A. Arkin, S. Askary, S. Fordin, and W. Jekel et al. Web Service Choreography Interface (WSCI) 1.0. Standards proposal by BEA Systems, Intalio, SAP, and Sun Microsystems, 2002.
7. A. Arkin et al. Business Process Modeling Language (BPML), Version 1.0, 2002.
8. A. Barros, G. Decker, M. Dumas, and F. Weber. Correlation Patterns in Service-Oriented Architectures. In *Proceedings of the 9th International Conference on Fundamental Approaches to Software Engineering (FASE)*, Braga, Portugal, 2007.
9. A. Barros, M. Dumas, and A.H.M. ter Hofstede. 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. A. Barros, M. Dumas, and A.H.M. ter Hofstede. Service Interaction Patterns. In *In Proceedings of the 3rd International Conference on Business Process Management*, volume 3716/2005, pages 302–318, Nancy, France, 2005.
11. A.P. Barros and E. Borger. A Compositional Framework for Service Interaction Patterns and Interaction Flows. In *ICFEM*, pages 5–35. Springer Verlag, 2005.
12. M.A. Beedle. *Enterprise Architecture Patterns*. Cambridge University Press, 1998.
13. T. Belwood and et al. UDDI Version 3.0. http://uddi.org, 2000.
14. D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (SOAP) 1.1. http://www.w3.org/TR/soap, 2000.
15. T. Bray, J. Paoli, C.M. Sperberg-McQueen, and E. Maler. eXtensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, 2000.
16. E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl, 2001.
17. D. Cooney, M. Dumas, and P. Roe. GPSL: A Programming Language for Service Implementation. In *Proceedings of the 8th International Conference on Fundamental Approaches to Software Engineering*, Vienna, Austria, 2006.
18. CPN Group, University of Aarhus, Denmark. CPN Tools Home Page. http://wiki.daimi.au.dk/cpntools/.
19. D.Box, E.Christensen, and D.Ferguson et al. Web Services Addressing. W3C Available via http://www.w3.org/Submission/ws-addressing/, Last accessed on November 19, 2007, 2004.

20. G. Decker, F. Puhlmann, and M. Weske. Formalizing service interactions. In In S. Dustdar, J.L. Fiadeiro, and A. Sheth (Eds.), editors, *Proc. of the 4th International Conference on Business Process Management (BPM 2006)*, volume 4102 of *LNCS*, pages 414–419, Vienna, Austria, 2006. Springer Verlag.
21. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison Wesley, Reading, MA, USA, 1995.
22. G. Hohpe and B. Woolf. *Enterprise Integration Patterns*. Addison-Wesley Professional, Reading, MA, 2003.
23. K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. EATCS monographs on Theoretical Computer Science. Springer-Verlag, Berlin, 1992.
24. N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, and Y. Lafon. Web Services Choreography Description Language Version 1.0(W3C Working Draft 17 December 2004). http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/, 2004.
25. Oracle BPEL Process Manager. Developers Guide 10g Release 2 (10.1.2). Available via http://download.oracle.com/otndocs/products/bpel/bpeldev.pdf, Last accessed on November 19, 2007, 2005.
26. C. McDonald. Orchestration, Choreography, Collaboration and Java Technology-based Business Integration. Last accessed on May 18, 2007. http://weblogs.java.net/blog/caroljmcdonald/archive/2003/10, , 2007.
27. C. Peltz. Web services orchestration: a review of emerging technologies, tools and standards. *Hewlett Packard, Co.*, 2003.
28. W. Pree. *Framework patterns*. SIGS Books, 1996.
29. N. Russell, A.H.M. ter Hofstede, W.M.P. van der Aalst, and N. Mulyar. Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22, BPM-center.org, 2006.
30. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns. QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.
31. N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Resource Patterns. BETA Working Paper Series, WP 127, Eindhoven University of Technology, Eindhoven, 2004.
32. S. Thatte. XLANG Web Services for Business Process Design, 2001.
33. A. van Dijk. Contracting Workflows and Protocol Patterns. In *Business Process Management*, pages 152–167. Springer Verlag, 2003.
34. Workflow Patterns Home Page. http://www.workflowpatterns.com.
35. WPHP. Workflow Patterns Home Page. http://www.workflowpatterns.com.
36. J.M. Zaha, A.P. Barros, M. Dumas, and A.H.M. ter Hofstede. Let's Dance: A Language for Service Behavior Modeling. In *OTM Conferences (1)*, Vienna, Austria, pages 145–162, 2006.

Appendix A: Glossary

Conversation Communication of a set of contextually related messages between two or more parties.

Configuration parameters Parameters that have to be set to a specific value from the defined range in order to for a specific pattern variant to be configured.

Correlation A mechanism used by a party to identify a process instance for processing a received message in the context of a particular conversation.

Customer A party who is a recipient of a service or a product. A customer may be represented by various people and processes in different roles, the principal roles being those of user, buyer, payer, etc.

Dynamic attribute A pattern attribute whose value is derived from other pattern attributes once all configuration parameters characterizing a specific pattern variant have been set to a specific value.

Mediator A party who acts as a link between two parties involved in a conversation.

Message A unit of information that may be composed of one or more data fields.

Pattern A solution for a recurring problem encountered in a certain context. Conceptually similar patterns that share a set of common pattern attributes and differ only by values these pattern attributes take are also termed as pattern variants.

Pattern attributes A set of attributes describing characteristics of a pattern by means of the static attributes, dynamic attributes and configuration parameters.

Pattern configuration A set of configuration parameters defined for a given pattern family to differentiate between different pattern variants. By setting configuration parameters to different values, from the defined range, various different pattern variants can be generated.

Pattern variant An instance of a pattern configuration for a particular pattern family whose configuration parameters are set to a specific value from the defined range.

Pattern family A set of pattern variants sharing the same concepts grouped together.

Party An entity involved in communication with other entities by means of sending/receiving messages. For instance, a process, a service, a business unit, etc.

Provider A party that provides services/products to a customer.

Product A good or a service that is produced.

Reply A reply, answer, or additional message that is returned to a requestor.

Renew policy A policy specifying which of the two parties involved in a subscription is responsible for renewing the subscription.

Responder A party producing a reply message to respond to a on the request message received.

Response period A period of time within which a reply to a previously sent message is expected.

Request A message sent by a requestor party to another party.

Requestor A party issuing request messages to (a set of) other parties involved in a conversation.

Static attribute A pattern attribute whose value is fixed for all pattern variants derived from the pattern configuration.

Subscription A special kind of a conversation between two parties, a provider and a customer, who are both capable of initiating and renewing the process of subscription which aims to deliver a certain product under accepted subscription terms.

Subscription initiation A conversation held for the purpose of establishing a subscription.

Subscription renewal A conversation held for the purpose of renewing an established subscription.

Subscription terms A set of rules or constraints characterizing a given subscription, which includes the subscription period, the renewal policy and the product of subscription.

Appendix: CPN models

The CPN model of the Customer-Initiated Customer-Renewable Subscription scenario is presented in Figure 60, where both Customer and Provider are substitution transitions unfolding to the nets presented in Figure 61 and Figure 62 respectively.

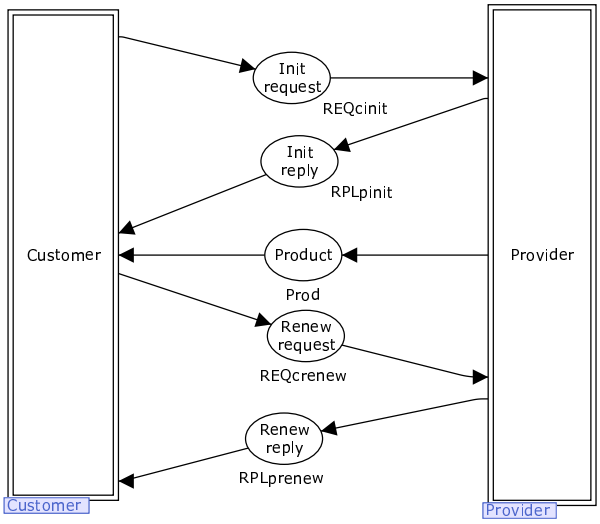


Fig. 60. CPN diagram: The top view of Customer-Initiated Customer-Renewable Subscription

The CPN model of the Provider-Initiated Customer-Renewable Subscription scenario is presented in Figure 63, where both Customer and Provider are substitution transitions unfolding to the nets presented in Figure 64 and Figure 65 respectively.

The CPN model of the Provider-Initiated Provider-Renewable Subscription scenario is presented in Figure 66, where both Customer and Provider are substitution transitions unfolding to the nets presented in Figure 67 and Figure 68 respectively.

The CPN model of the Provider-Initiated Automatically-Renewable Subscription scenario is presented in Figure 69, where both Customer and Provider are substitution transitions unfolding to the nets presented in Figure 70 and Figure 71 respectively.

The CPN model of the Customer-Initiated Automatically-Renewable Subscription scenario is presented in Figure 72, where both Customer and Provider are substitution transitions unfolding to the nets presented in Figure 73 and Figure 74 respectively.

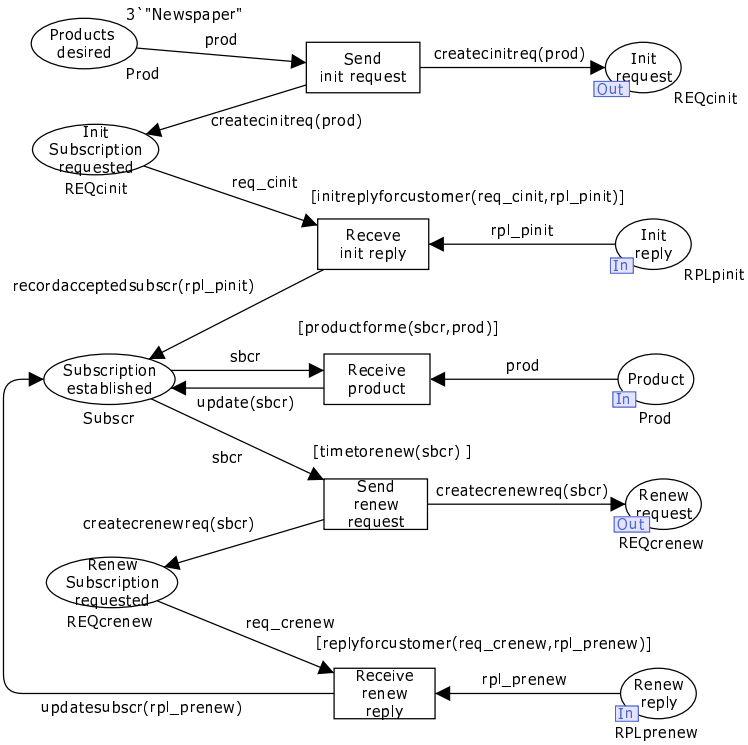


Fig. 61. CPN diagram: The `Customer` page of Customer-Initiated Customer-Renewable Subscription

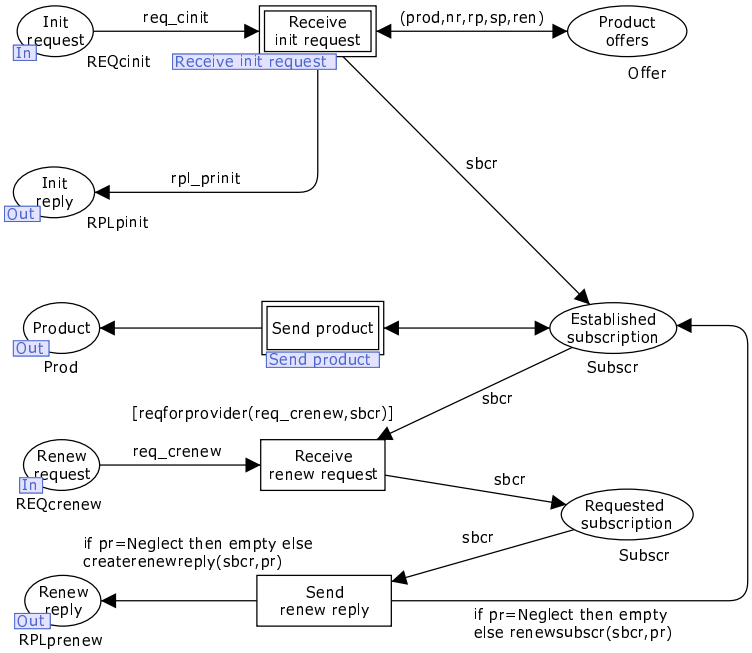


Fig. 62. CPN diagram: The `Provider` page of Customer-Initiated Customer-Renewable Subscription

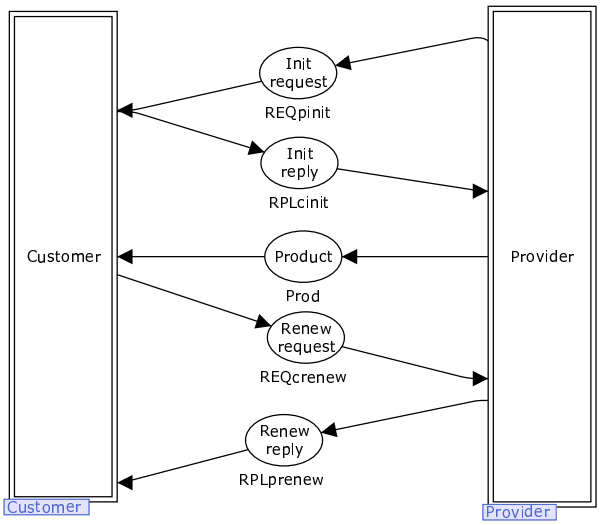


Fig. 63. CPN diagram: The top view of Provider-Initiated Customer-Renewable Subscription

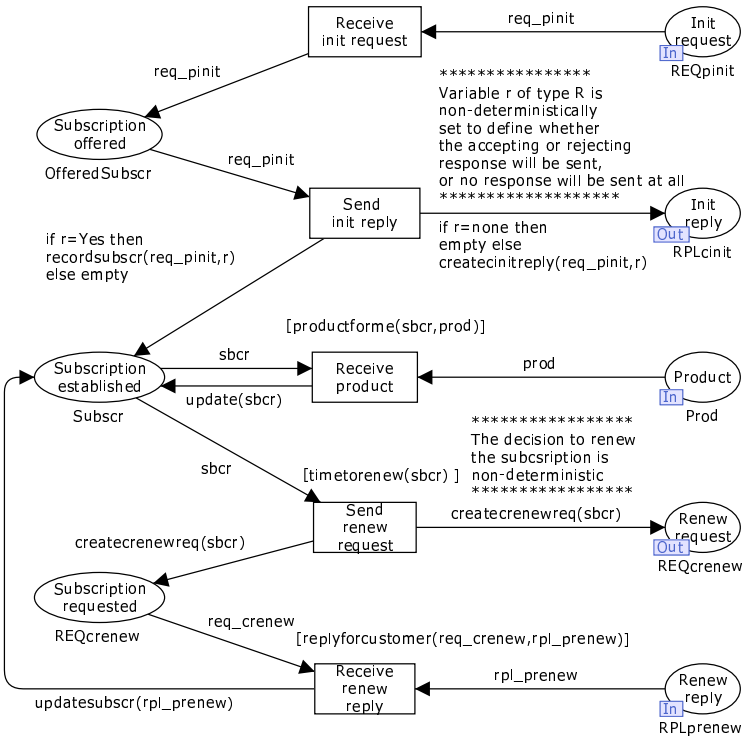


Fig. 64. CPN diagram: The `Customer` page of Provider-Initiated Customer-Renewable Subscription

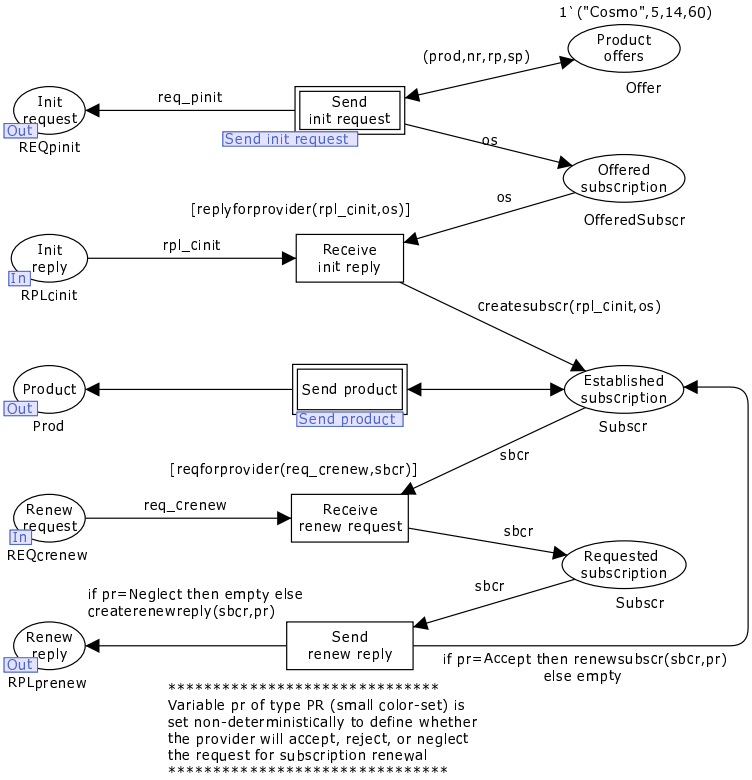


Fig. 65. CPN diagram: The `Provider` page of Provider-Initiated Customer-Renewable Subscription

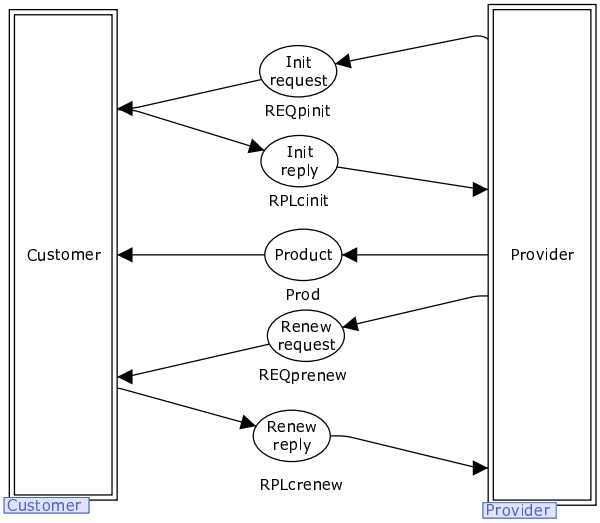


Fig. 66. CPN diagram: The top view of Provider-Initiated Provider-Renewable Subscription

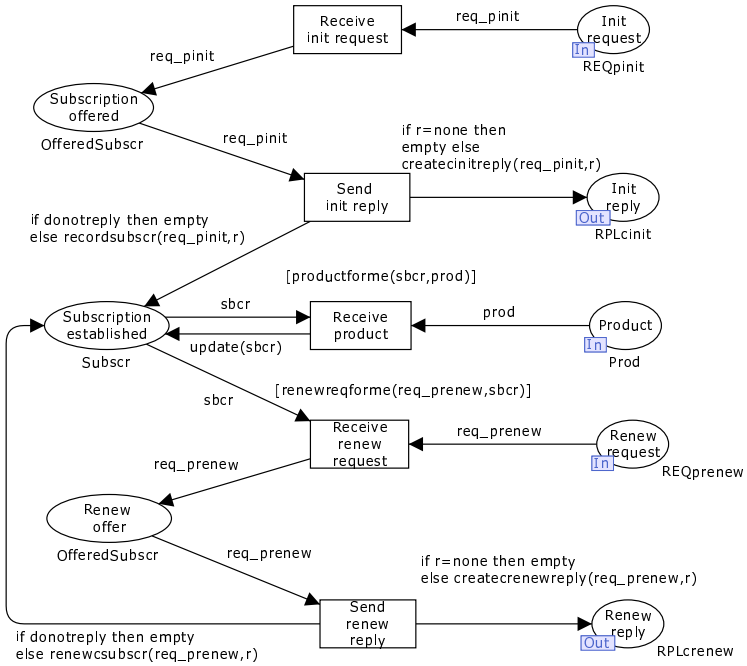


Fig. 67. CPN diagram: The `Customer` page of Provider-Initiated Provider-Renewable Subscription

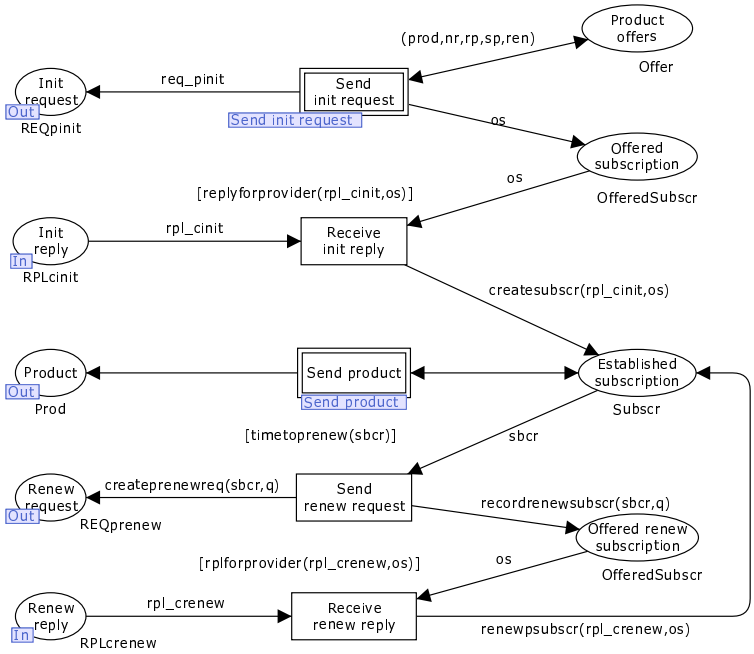


Fig. 68. CPN diagram: The `Provider` page of Provider-Initiated Provider-Renewable Subscription

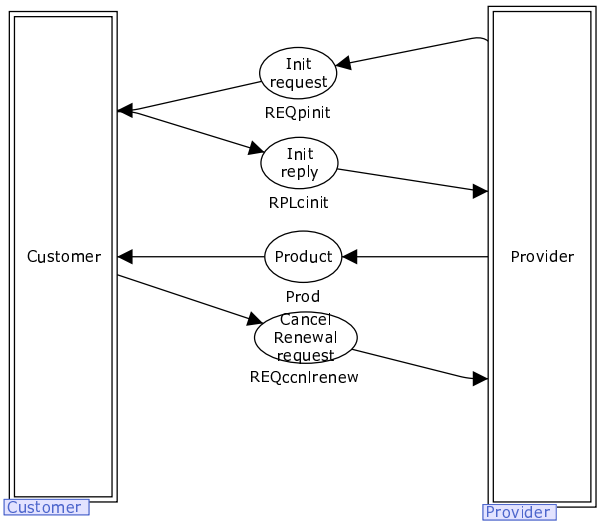


Fig. 69. CPN diagram: The top view of Provider-Initiated Automatically-Renewable Subscription

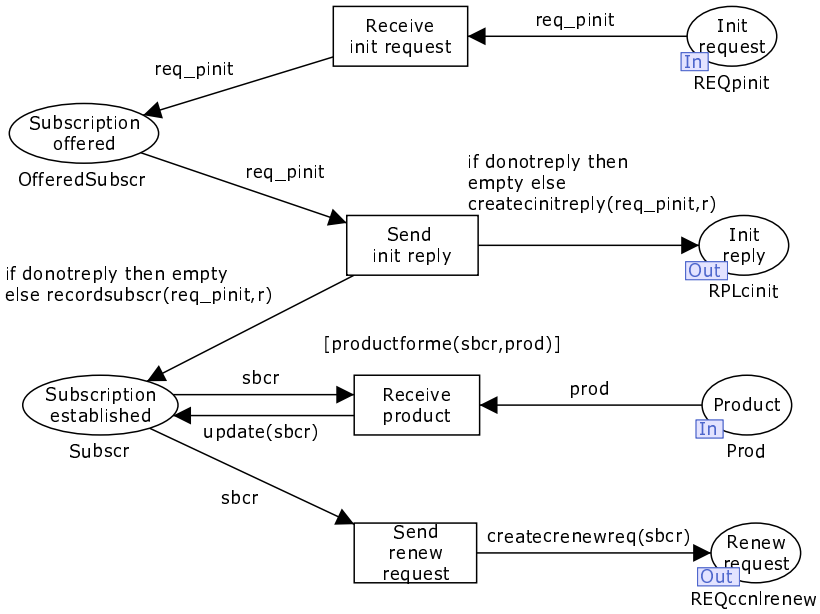


Fig. 70. CPN diagram: The `Customer` page of Provider-Initiated Automatically-Renewable Subscription

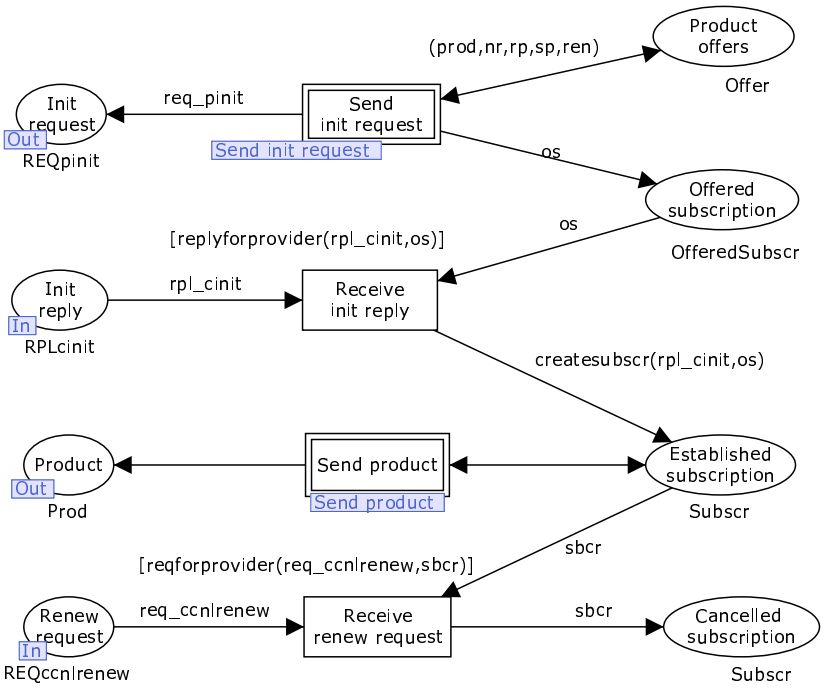


Fig. 71. CPN diagram: The `Provider` page of Provider-Initiated Automatically-Renewable Subscription

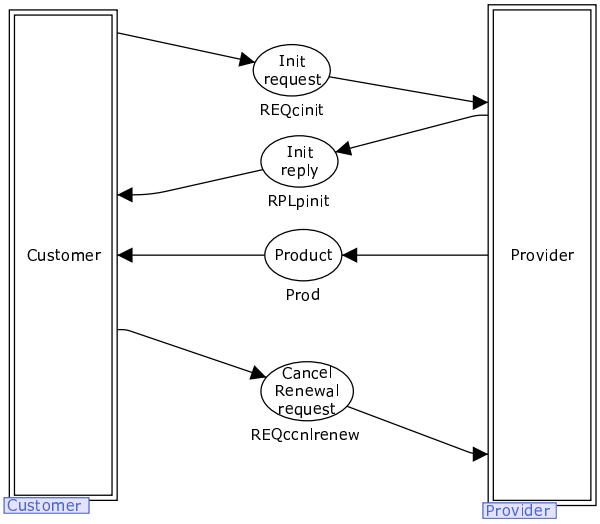


Fig. 72. CPN diagram: The top view of Customer-Initiated Automatically-Renewable Subscription

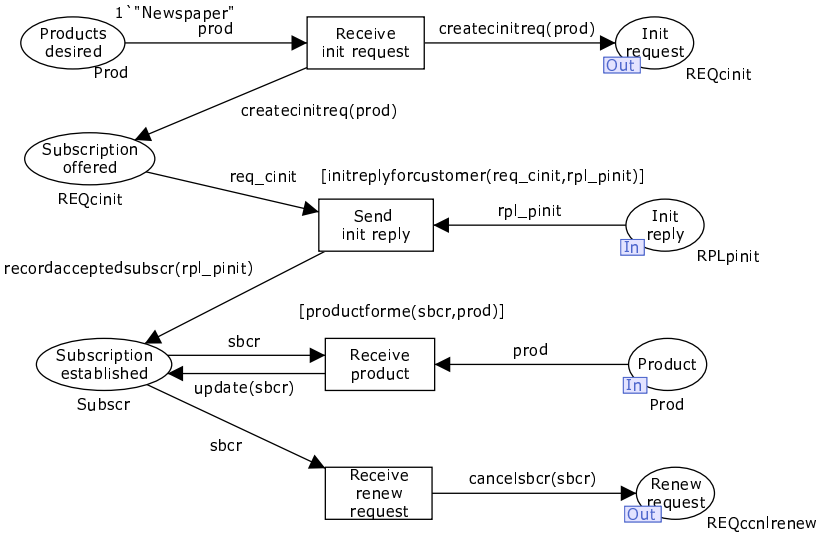


Fig. 73. CPN diagram: The `Customer` page of Customer-Initiated Automatically-Renewable Subscription

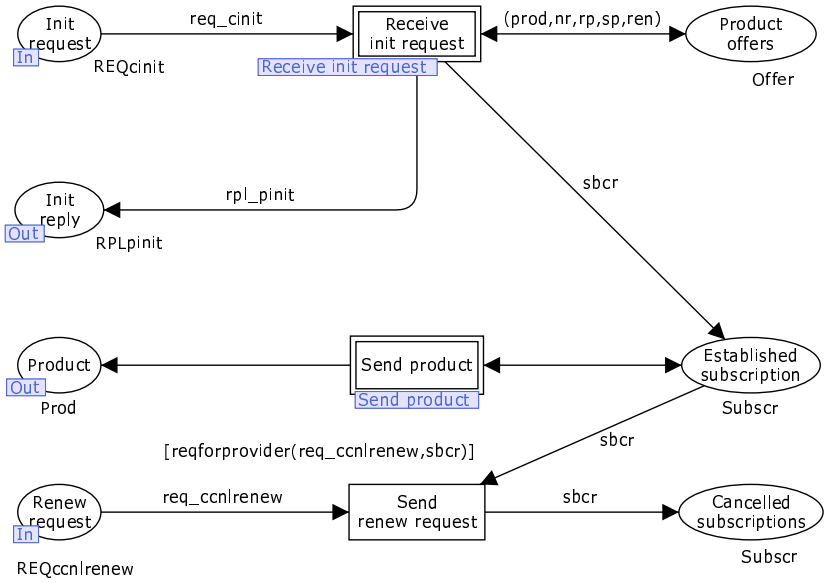


Fig. 74. CPN diagram: The `Provider` page of Customer-Initiated Automatically-Renewable Subscription