Formalising BPMN Service Interaction Patterns
Chiara Muzi, Luise Pufahl, Lorenzo Rossi, Mathias Weske, Francesco Tiezzi

To cite this version:
Chiara Muzi, Luise Pufahl, Lorenzo Rossi, Mathias Weske, Francesco Tiezzi. Formalising BPMN Service Interaction Patterns. 11th IFIP Working Conference on The Practice of Enterprise Modeling (PoEM), Oct 2018, Vienna, Austria. pp.3-20, 10.1007/978-3-030-02302-7_1 . hal-02156461

HAL Id: hal-02156461
https://inria.hal.science/hal-02156461v1
Submitted on 14 Jun 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons Attribution 4.0 International License
Formalising BPMN Service Interaction Patterns

Chiara Muzi¹, Luise Pufahl², Lorenzo Rossi¹, Mathias Weske², and Francesco Tiezzi¹

¹ School of Science and Technology, University of Camerino, Italy
² Hasso-Plattner-Institute, University of Potsdam, Potsdam, Germany

Abstract. Business process management is especially challenging when crossing organisational boundaries. Inter-organisational business relationships are considered as a first-class citizen in BPMN collaboration diagrams, where multiple participants interact via messages. Nevertheless, proper carrying out of such interactions may be difficult due to BPMN lack of formal semantics. In particular, no formal studies have been specifically done to cope with complex BPMN interaction scenarios unified under the name of Service Interaction Patterns. In this work the depiction of the service interaction patterns in BPMN collaboration diagrams is revisited and fully formalised via a direct semantics for BPMN multi-instance collaborations, thus leaving no room for ambiguity and validating the BPMN semantics. To make the formalisation more accessible, a visualisation of the patterns execution by means of a BPMN model animation tool is provided.

Keywords: BPMN · Collaboration · Service Interaction Patterns · Formalisation

1 Introduction

The effective and efficient handling of business processes is a primary goal of organisations. Business Process Management (BPM) provides methods and techniques to support these endeavors [17]. Thereby, the main artefacts are business process models which help to document, analyse, improve, and automate organisation processes [13]. To this aim, nowadays BPMN (Business Process Model and Notation) [16] is the modelling notation most widely applied in industry and academia.

For conducting a successful business, an organisation does not act alone, but it is usually involved in collaborations with other organisations. The importance of interactions has been underlined by many authors [1,10,2] and a lot of effort has been done to identify the most common interaction scenarios from a business perspective, which have been called Service Interaction Patterns [3]. Interactions are considered as a first-class citizen in BPMN collaboration diagrams, where multiple participants cooperate by exchanging messages and sharing thereby data. This motivated the use of BPMN to model service interaction patterns [17], initially defined only in terms of textual descriptions. This effort provided a graphical, more intuitive, description of the patterns and allowed to assess the suitability of BPMN to express common interaction scenarios. However, a severe issue in this study is that the precise behaviour of the BPMN models corresponding to some patterns may result unclear, in particular when multiple
instances of the interacting participants are involved. This problem is mainly due to the fact that the BPMN standard comes without a formal semantics, which is needed in presence of tricky features, like multiple process instantiation.

In this paper, we then aim at formalising the execution semantics of service interaction patterns specified in BPMN. This is a particularly important challenge in the BPM domain, as a precise semantics of the message exchanges as well as their dependencies is a prerequisite to ensure the appropriate carrying out, in practice, of such interactions. To achieve this goal, we resort to a formal semantics for BPMN collaborations including multiple instances introduced in [6]. The operational semantics is directly defined on BPMN elements in terms of Labelled Transition Systems (LTSs), rather than as an encoding into other formalisms. Specifically, for each service interaction pattern we report the related BPMN collaboration model and provide its formalisation in terms of transitions of the corresponding LTS.

A direct formal characterisation is crucial, as it does not leave any room for ambiguity, and increases the potential for formal reasoning. This is especially important when dealing with multiple instances, whose static and compact BPMN representation hides their complex semantics. Moreover, the BPMN formalisation in [6] enables the use of the MIDA animation tool [7] that provides a visualisation, faithfully following the semantics, of patterns execution. This makes the behaviour of patterns easily understandable also to an audience non-familiar with formal methods. Finally, since the service interaction patterns have been used to evaluate different choreography languages [5], their formalisation allows to validate the BPMN semantics itself, both in terms of the considered BPMN elements and of the expected semantic behaviour.

In the remainder of this paper, we start with the introduction of a motivational example in Sec. 2, followed by an overview of the formalisation of BPMN collaboration diagrams in Sec. 3. In Sec. 4 we provide the representation of the service interaction patterns in BPMN and their formalisation. We present the patterns animation in Sec. 5. Finally, related work is discussed in Sec. 6 and the paper is concluded in Sec. 7.

## 2 Motivating Scenario

In this section we introduce an order fulfilment scenario to illustrate BPMN 2.0 collaboration diagrams and the depiction of service interaction patterns. The considered scenario shows an interaction among a Customer, a Retailer, multiple Item Providers and a Logistic Provider (Fig. 1). The processes of the different interacting partners are represented inside rectangles, called pools, and their interaction is given by message edges (dashed connectors) visualising communication flows.

The order fulfilment process is started by the Customer who sends an “Order Request” to the Retailer, via a send task. The arrival of this message starts, via a message start event, a process instance of the Retailer pool. This latter creates a list of needed items and stores this information in the “Item” data object collection. For each item, the Retailer sends out a request to an Item Provider and waits for the response. This interaction is rendered by a multi-instance sub-process communicating with the multi-instance pool of the Item Provider: each message of the Retailer creates a new process instance of the Item Provider pool. Each Provider checks for items availability and decides ei-
Fig. 1: BPMN collaboration diagram of the Order Fulfilment Scenario.

The system will be sent to “Send Item” or not, by following one of the outgoing edges of the XOR split gateway, according to the information in the “Stock” data object. In case the item is not available, the Provider does not respond back to the Retailer. Thus, the Retailer rather stops waiting as soon as enough responses have arrived or a given timeout is expired. This latter behaviour is rendered by the timer event attached to the sub-process, whose activation produces the execution of task “Reduce Invoice by Missing Items”. The Retailer then packs the items and passes the needed information to the Logistic Provider, who is in charge to send the package to the Customer. When the Customer receives the ordered items, via a receiving task, its process completes.

In this scenario, already various interaction patterns can be observed, such as One-to-many send/receive between the Retailer and the Item Provider(s), or the Request with Referral between the Customer, Retailer and Logistic Provider [3]. The service interactions represent different types of dynamic behaviour, ranging from simple message exchanges, to scenarios involving multiple participants and multiple message exchanges, as well as routing behaviour, where information is routed to a new collaboration partner during an interaction (e.g. from the Retailer to the Logistic Provider). As the service interaction patterns are textually provided, a visualisation as well as a formalisation is crucial to precisely render the message exchanges between participants, especially because multiple instances are involved. In particular, the interactions between the Retailer sub-process and the Item Provider may result quite intricate, as both generate multiple instances and, in addition, the sub-process is constrained by a timer event. Without a clear understanding of the interplay between these features, formally provided by the operational semantics, different interpretations may easily arise.

3 Background Notions on the BPMN Formalisation

In this section we provide an overview of the formal semantics of BPMN multi-instance collaborations given in [6]. The formalisation relies on a textual representation of the structure of BPMN collaboration models, defined by the Backus-Naur Form (BNF) grammar in Fig. 2. In the proposed grammar, the non-terminal symbols $C$, $P$ and $A$
represent Collaboration Structures, Process Structures and Data Assignments, respectively. The first two syntactic categories directly refer to the corresponding notions in BPMN, while the latter refers to list of assignments used to specify updating of data objects. The terminal symbols, denoted by the sans serif font, are the typical elements of a BPMN model, i.e. pools, events, tasks and gateways.

Intuitively, a BPMN collaboration model is rendered in this syntax as a collection of pools, each one specifying a process. Formally, a collaboration $C$ is a composition, by means of the $\parallel$ operator, of pools. A pool is either of the form $\text{pool}(p, P)$ (for single-instance pools), or $\text{miPool}(p, P)$ (for multi-instance pools) where $p$ is the name that uniquely identifies the pool, and $P$ is the enclosed process. At process level, $e \in E$ uniquely denotes a sequence edge, while $E \in 2^E$ is a set of edges. For the convenience of the reader, $e_i$ refers to the edge incoming in an element, while $e_o$ to the outgoing edge, and $e_{nb}$ to the (spurious) edge denoting the enabled status of a start event.

To describe the semantics of collaboration models, we enrich the structural information with a notion of execution state, defined by the state of each process instance and the store of the exchanged messages. We call process configurations and collaboration configurations these stateful descriptions. Formally, a process configuration has the form $\langle P, \sigma, \alpha \rangle$, where $P$ is a process structure; $\sigma : E \rightarrow \mathbb{N}$ is a sequence edge state function specifying, for each sequence edge, the current number of tokens marking it ($\mathbb{N}$ is the set of natural numbers); and $\alpha : \mathbb{F} \rightarrow \mathbb{V}$ is the data state function assigning values (possibly null) to data object fields ($\mathbb{P}$ is the set of data fields and $\mathbb{V}$ the set of values).

A collaboration configuration has the form $\langle C, t, \delta \rangle$, where: $C$ is a collaboration structure, $t : P \rightarrow 2^{S_o \times \delta_m}$ is the instance state function mapping each pool name ($P$ is the set of pool names) to a multiset of instance states (ranged over by $I$ and containing pairs of the form $\langle \sigma, \alpha \rangle$), with $S_o$ and $S_m$ the sets of edges and data states, and $\delta : M \rightarrow 2^V$ is a message state function specifying for each message name $m \in M$ a multiset of value tuples representing the messages received along the message edge with the label $m$.

The operational semantics is defined by means of a labelled transition system (LTS), whose definition relies on an auxiliary LTS on the behaviour of processes. The latter is a triple $\langle P, \mathcal{L}, \rightarrow \rangle$ where: $P$ ranged over by $\langle P, \sigma, \alpha \rangle$ is a set of process configurations, $\mathcal{L}$ ranged over by $\ell$, is a set of labels, and $\rightarrow \subseteq P \times \mathcal{L} \times P$ is a transition relation. We will write $\langle P, \sigma, \alpha \rangle \xrightarrow{\ell} \langle P', \sigma', \alpha' \rangle$ to indicate that $\langle \langle P, \sigma, \alpha \rangle, \ell, \langle P', \sigma', \alpha' \rangle \rangle \in \rightarrow$. Now, the labelled transition relation on collaboration configurations formalises the message exchange and the data update according to the process evolution. The LTS is a triple $\langle C, \mathcal{L}_c, \rightarrow_c \rangle$ where: $C$, ranged over by $\langle C, t, \delta \rangle$, is a set of collaboration configurations; $\mathcal{L}_c$, ranged over by $l$, is a set of labels; and $\rightarrow_c \subseteq C \times \mathcal{L}_c \times C$ is a transition relation.
In this section we present and formalise the Service Interaction Patterns [3] supported by BPMN. Since BPMN is not specifically tailored to the needs of service interaction
patterns, the notation cannot completely support all their features. For instance, while the informal and general description of these patterns leaves it open if in an interaction the counter-party is known at design-time or not, in BPMN it is expected to have a priori knowledge of the interacting partners, i.e. the target pool of a message edge cannot be dynamically selected. On the other hand, in case a message is directed to a multi-instance pool, BPMN supports a form of runtime binding of the message with the correct process instance by means of the correlation mechanism [16, Sec. 8.3.2]. Moreover, it is also possible to dynamically specify other model features, such as the number of involved participants and exchanged messages. Each pattern is presented according to the following structure:

**Informal Description** consists of a natural language description, and a graphical representation in terms of a BPMN collaboration fragment.

**Textual Specification** provides the textual notation of the BPMN collaboration model.

**Formal Semantics** describes the operational rules applied to perform each execution step, and shows the results in terms of the execution state functions evolution.

In the following we present first those patterns concerning single transmissions, both bilateral (Sec. 4.1-4.3) and multilateral (Sec. 4.4-4.7), and then the routing patterns involving multiple transmissions (Sec. 4.8-4.9).

### 4.1 Send Pattern

**Informal Description.** A party sends a message to another one. This pattern can be modelled as the BPMN collaboration fragment in Fig. 4. Notably, this is only a way to model it: the send task could be replaced by an intermediate send event or by a message end event. However, up to some technicalities, all cases behave in the same way, thus we report here only one of them.

![Fig. 4: Send pattern.](image)

**Textual Specification.** The collaboration fragment in Fig. 4 is represented in the textual notation as $C = \text{pool}(p, P) \parallel \text{pool}(q, Q)$ with $P = \text{taskSnd}(e_1, \exp_1, \epsilon, m : \exp_2, e_2) \parallel P'$, where $p$ is the sender and $q$ a generic receiver (represented by a black-box pool in the graphical notation, whose process $Q$ is left unspecified in the textual one).

**Formal Semantics.** According to the form of process $P$, and the current state $\langle \sigma, \alpha \rangle$ of pool $p$’s instance, the collaboration can evolve as follows:

- Process $P$ moves by executing Task 1. This execution step takes place by applying rule $P: \text{TaskSnd}$, which requires the incoming edge $e_1$ of the task be marked by at least one token ($\sigma(e_1) > 0$), and the task’s guard $\exp_1$ be satisfied ($\text{eval}(\exp_1, \alpha, \text{true})$). The effects of the task execution are as follows: the message action $!m : \tilde{v}$ is produced, where the message content $\tilde{v}$ results from the evaluation of the expression tuple $\exp_2$ ($\text{eval}(\exp_2, \alpha, \tilde{v})$), and the marking $\sigma$ of the process instance is updated with the movement of one token from $e_1$ to $e_2$, that is $\sigma' = \text{inc}(\text{dec}(\sigma, e_1), e_2)$. Therefore, the application of rule $P: \text{TaskSnd}$ produces the transition $\langle \text{taskSnd}(e_1, \exp_1, \epsilon, m : \exp_2, e_2), \sigma, \alpha \rangle \xrightarrow{!m \triangleright \tilde{v}} \langle \sigma', \alpha \rangle$, where the data state $\alpha$ remains unchanged because no data object is connected...
to Task 1. Hence, the overall process $P$ can evolve according to the interleaving rule $P\text{-Int}_{1}$, that is $\langle P, \sigma, \alpha \rangle \xrightarrow{\text{updI}(\iota, p, \{\langle \sigma', \alpha \rangle \})} \langle \sigma', \alpha \rangle$. Similarly, by applying the rule $C\text{-Deliver}$, and then the interleaving rule at collaboration level, the execution step of the overall collaboration $C$ is represented by the transition $\langle \sigma, \alpha \rangle \xrightarrow{\text{Inc}(\text{dec}(\sigma, e1), e2), \alpha} \langle \text{inc}(\text{dec}(\sigma, e1), e2), \alpha \rangle$ is produced by applying rule $P\text{-InterRcv}$. Then, the process $Q$ evolves by means of the interleaving rule $P\text{-Int}_{1}$. The produced label $\langle \text{Inc}(\text{dec}(\sigma, e1), e2), \alpha \rangle$ indicates the willingness of process $Q$ to consume a message of type $m$ matching the template $e1$. If present, the message is actually consumed by rule $C\text{Receive}$ at collaboration level. Indeed, this rule requires that there is a message in the m’s message queue (i.e., $\delta(m)$) that matches the template $e1$ of the receiving event (match$(e1, \nu) = A$); the assignments $A$ produced by this matching are then applied to the data state $\alpha$ of the q’s instance, and the message is removed from the queue (rm$(\delta, m, \nu)$).

### 4.3 Send/Receive Pattern

**Informal Description.** Two parties, $p$ and $q$, engage in two causally related interactions. In the first interaction, $p$ sends a message (the request) to $q$, while in the second one $p$ receives a message (the response) from $q$. 

---

[Diagram and Textual Specification]
This pattern can be modelled by combining the \textit{Send} and the \textit{Receive} patterns, as shown in Fig. 6.

\textbf{Textual Specification.} The textual representation of the collaboration fragment in Fig. 6 has again the form \( C \) of the previous patterns, with
\[
\begin{align*}
P &= \text{\textit{taskSnd}}(e_1, \text{exp}_1, \epsilon, m_1:\text{exp}_2, e_2) \parallel \text{\textit{interRcv}}(e_2, m_2:t_1, e_3) \parallel P' \\
Q &= \text{\textit{interRcv}}(e_4, m_1:t_2, e_5) \parallel \text{\textit{taskSnd}}(e_5, \text{exp}_3, \epsilon, m_2:\text{exp}_4, e_6) \parallel Q'
\end{align*}
\]

\textbf{Formal Semantics.} The execution steps of this pattern are realised by combining the semantic rules for the \textit{Send} and \textit{Receive} patterns. In detail: let us suppose that there is a token in the incoming edge of Task 1 \((\sigma(e_1) > 0)\) and the other preconditions of rule \( P\text{-}\text{\textit{TaskSnd}} \) are satisfied; by applying this rule we have that \( \langle \text{\textit{taskSnd}}(e_1, \text{exp}_1, \epsilon, m_1:\text{exp}_2, e_2), \sigma, \alpha \rangle \xrightarrow{\text{\textit{inc}}(\text{\textit{dec}}(\sigma, e_1), e_2), \alpha} \langle \text{\textit{inc}}(\text{\textit{dec}}(\sigma, e_1), e_2), \alpha \rangle \). Then, \( P \) evolves by performing a sending action, by means of the interleaving rule \( P\text{-}\text{\textit{Int}}_1 \), that is \( \langle P, \sigma, \alpha \rangle \xrightarrow{\text{\textit{inc}}(\text{\textit{dec}}(\sigma, e_1), e_2), \alpha} \langle \text{\textit{inc}}(\text{\textit{dec}}(\sigma, e_1), e_2), \alpha \rangle \). At the collaboration layer, by applying rule \( C\text{-}\text{\textit{Deliver}} \), the message \( m_1 \) is delivered to \( q \). Now, on the receiving party, assuming that there is a token on \( e_4 \) and that the template \( t_2 \) evaluates to \( e_2 \), by applying rules \( P\text{-}\text{\textit{InterRcv}} \) and \( P\text{-}\text{\textit{Int}}_1 \), we have \( \langle Q, \sigma_2, \alpha_2 \rangle \xrightarrow{\text{\textit{inc}}(\text{\textit{dec}}(\sigma_2, e_4), e_5), \alpha_2} \langle \text{\textit{inc}}(\text{\textit{dec}}(\sigma_2, e_4), e_5), \alpha_2 \rangle \). The observed label indicates the willingness to receive a message of type \( m_1 \). Thus, at collaboration level, rule \( C\text{-}\text{\textit{Receive}} \) can be applied to allow process \( Q \) to actually consume the sent request message. Now, Task 3 is enabled and, by proceeding in a specular way, \( Q \) can send the response message \( m_2 \) and \( P \) can consume it.

\textbf{4.4 Racing Incoming Messages Pattern}

\textit{Informal Description.} 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 can be modelled in BPMN by using in the receiving participant an event-based gateway connected to receiving events. Messages can be expected from one participant (Fig. 7) or they can arrive from different participants (Fig. 8).

\textbf{Textual Specification.} Let us first consider the case in which messages arrive from one participant (Fig. 7). In the textual notation the diagram is rendered as the collaboration of the usual form \( C \), with
\[
\begin{align*}
P &= \text{\textit{xorSplit}}(e_1, \{(e_2, \text{exp}_1), (e_3, \text{exp}_2)\}) \parallel \text{\textit{taskSnd}}(e_2, \text{exp}_3, \epsilon, m_1:\text{exp}_4, e_4) \parallel \\
&\quad \text{\textit{taskSnd}}(e_3, \text{exp}_5, \epsilon, m_2:\text{exp}_6, e_5) \parallel P' \\
Q &= \text{\textit{eventBased}}(e_6, (m_1:t_1, e_7), (m_2:t_2, e_8)) \parallel Q'
\end{align*}
\]

The case in which messages arrive from two different participants (Fig. 8) is rendered in the textual notation as \( C = \text{\textit{pool}}(p, P'') \parallel \text{\textit{pool}}(r, R) \parallel \text{\textit{pool}}(q, Q) \), where
process $Q$ is as the above one, while $P''$ and $R$ are left unspecified (because they are included in black-box pools).

**Formal Semantics.** Let us start with the case in which messages arrive from a single participant, and assume that a token is available in the incoming edge of the XOR split gateway of $P$ ($\sigma(e1) > 0$) and the conditional expression $\exp_1$ is evaluated to true ($\text{eval} (\exp_1, \alpha, \text{true})$). Thus, rule $P$-$\text{XorSplit}_1$ can be applied and the token is moved to the edge $e2$, hence enabling Task 1. Formally, this step corresponds to the transition $\langle P, \sigma, \alpha \rangle \xrightarrow{\epsilon} \langle \text{inc}(\text{dec}(\sigma, e1), e2), \alpha \rangle$, where label $\epsilon$ denotes the movement of the token internally to the process. The next step corresponds to the execution of Task 1, which is as in the case of the $\text{Send}$ pattern. Once the message $m1$ has been sent (hence, there exists a $\tilde{v}$ such that $\tilde{v} \in \delta(m1)$), and assuming that there is a token in $e6$ ($\sigma(e6) > 0$), the event-based gateway can evolve by applying the corresponding rule. This corresponds to the transition $\langle \text{eventBased}(e6, (m1 : t_1, e7), (m2 : t_2, e8)), \sigma', \alpha' \rangle \xrightarrow{?m1 : e_1, \epsilon} \langle \text{inc}(\text{dec}(\sigma', e6), e7), \alpha' \rangle$, with template $e_1$ matching the message $\tilde{v}$. The rule moves the token from the incoming edge to the outgoing edge corresponding to the received message. The produced label enables the application of rule $C$-$\text{Receive}$ at collaboration level, which takes care of consuming the message $\tilde{v}$ of type $m1$ in $\delta$. The case where message $m2$ is selected to be sent is similar.

In the scenario shown in Fig. 8, even if the transitions produced by the collaboration have the same labels, the pattern semantics is quite different. In fact, in the previous case the organisation $p$ internally decides which message will be sent and only one message will be delivered and consumed, while in this case the organisations $p$ and $r$ act independently from each other and it may occur that both $m1$ and $m2$ are sent to $q$. In such a case, one of the two messages will be consumed, depending on their arrival time, and the other message will be pending forever.

### 4.5 One-To-Many Send Pattern

**Informal Description.** A party sends messages to several parties. All messages have the same type (although their contents may be different). The number of parties to whom the message is sent may or may not be known at design time.

In BPMN, this pattern can be modelled as the collaboration fragment in Fig. 9, where each party is represented as an instance of a multi-instance pool and a message is sent to each process instance via a sequential multi-instance send task.

From now on, when a message is sent/received to/by several parties, we will model these parties as a multiple instance pool. This is the most interesting
among various interpretations which are not considered in this work (e.g., representing multiple receiving parties as different single-instance pools). We can have that the number of sent messages is either known at design time (by setting the LoopCardinality attribute of the send task) or it is read from a data object during the process execution.

Textual Specification. Here, to keep the pattern formalisation more manageable, the sequential multi-instance task is rendered as a macro. The macro encloses the task in a \texttt{FOR}-loop expressed by means of a pair of XOR join and split gateways, and an additional data object \( c_1 \) for the loop counter. In the textual notation we have \( C = \text{pool}(p, P) \parallel \text{miPool}(q, Q) \), where process \( Q \) is left unspecified and in \( P \) the attribute LoopCardinality is set to \( n \):

\[
P = \text{xorJoin} \left( \{e_1, e_1'\}, e_1' \right) \parallel \text{taskSnd} \left( e_1', c_1, c \neq \text{null}, c_1, c := c_1, c + 1, m: \text{exp}_1, e_1' \right) \parallel \text{xorSplit} \left( e_1'', \{e_1'', c_1, c \leq n\}, (e_2, \text{default}) \right) \parallel P'
\]

Formal semantics. The execution steps are realised as in the previous cases, by repeatedly applying the semantic rules of the XOR gateway and the send task. It is worth noticing that at each application of rule \( P-\text{TaskSnd} \) the field \( c \) of the data object \( c_1 \) is updated with the assignment \( c_1, c := c_1, c + 1 \). At the end of the pattern execution, the message list \( \delta(m) \) contains \( n \) sent messages.

4.6 One-From-Many Receive Pattern

Informal Description. A party receives several logically related messages arising from autonomous events occurring at different parties. The arrival of messages must be timely so that they can be correlated as a single logical request. The interaction may complete successfully or not depending on the messages gathered. In this pattern the receiver does not know the number of messages that will arrive, and stops waiting as soon as a certain number of messages have arrived or a timeout occurs.

This pattern can be modelled as the collaboration fragment shown in Fig 10.

Textual Specification. Also in this case, to simplify the formal treatment, we rely on a macro for the multi-instance receive task with a timer. In particular, the multi-instance behaviour is represented by enclosing the receive task in a \texttt{FOR}-loop (as for the sequential multi-instance task). The timer attached to the receive task is instead abstracted via a non-deterministic choice, by resorting to a race condition. In detail, the receiving party \( q \) will get, via an event-based gateway, either a message from a sending party (i.e., an instance of \( p \)) or a time-out message from a specific pool \( t \) representing the timer. In the textual notation we have \( C = \text{miPool}(p, P) \parallel \text{pool}(q, Q) \parallel \text{pool}(t, T) \), where
Also, let us consider the case where there are two receiving instances, i.e. correlation information. For example, let us assume that two messages of type \( m \) correct process instance of \( Q \). In this pattern we have that process \( P \) consists of a sequential send task followed by a multi-instance receive task with a timer. We have \( C = \text{pool}(p, P) \parallel \text{miPool}(q, Q) \), where process \( P \) is rendered in terms of macros as already shown in the previous patterns (hence, for the sake of presentation, its specification is omitted), while process \( Q \) is as follows:

\[
Q = \text{interRcv}(e_7, m_1 : t_1, e_8) \parallel \text{taskSnd}(e_8, \exp, A, m_2 : \exp, e_9) \parallel Q'
\]

**Formal Semantics.** In this pattern we have that process \( P \) sends out, by means of rule \( P \cdot \text{TaskSnd} \), several messages of type \( m_1 \) that need to be properly correlated with the correct process instance of \( Q \). The content of the messages themselves provides the correlation information. For example, let us assume that two messages of type \( m_1 \) are sent to \( q \), and that consist of three fields, say \( \langle \text{"foo"}, 5, 1234 \rangle \) and \( \langle \text{"foo"}, 7, 9876 \rangle \). Also, let us consider the case where there are two receiving instances, i.e. \( \nu(q) = \ldots \)
\{\langle \sigma_1, \alpha_1 \rangle, \langle \sigma_2, \alpha_2 \rangle \}, and that template \( \tilde{t} \) of the intermediate receiving event is defined as \( \langle \text{d.f, d.id, ?d.code} \rangle \), meaning that the fields f and id of the data object d identify correlation data while code is a formal field. Now, the correlation takes place according to the data states, which we assume to be as follows: \( \alpha_1(\text{d.f}) = \alpha_2(\text{d.f}) = \text{"foo"} \), \( \alpha_1(\text{d.id}) = 7 \), and \( \alpha_2(\text{d.id}) = 5 \). Therefore, the first message is delivered to the second instance, updating \( \alpha_2 \) with the assignment \( \text{d.code} = 9876 \), while the second message is delivered to the first instance, updating \( \alpha_1 \) with the assignment \( \text{d.code} = 1234 \).

### 4.8 Request with Referral Pattern

**Informal Description.** A party p sends a request to another party q indicating that any follow-up should be sent to another party r. An example of a BPMN collaboration involving the request with referral pattern is shown in Fig. 12, and also in the communication between the Retailer and the Logistic Provider in Fig. 1.

**Textual Specification.** In the textual specification we have \( C = \text{pool}(p, P) || \text{pool}(q, Q) || \text{pool}(r, R) \), where:

\[
P = \text{start}(e_{enb_1, e1}) || \text{taskSnd}(e_1, \text{exp}_1, A_1, m_1: \text{exp}_2, e_2) || \text{interRcv}(e_2, m_2: \tilde{t}_1, e_3) || \text{taskSnd}(e_3, \text{exp}_3, A_2, m_3: \text{exp}_4, e_4) || \text{interRcv}(e_4, m_4: \tilde{t}_2, e_5) || \text{end}(e_5)
\]

and \( Q \) and \( R \) are defined in a similar way.

**Formal Semantics.** The execution steps and their results are simply realised by applying the semantic rules for the different BPMN elements, as already shown for the previous patterns. It is up to the message sent by pool p to pool r to specify in its content the reference to pool q, whose process waits for the routed message.

### 4.9 Relayed Request Pattern

**Informal Description.** A party p makes a request to party q, which delegates the request processing to another party r. This latter party interacts with party p while party q observes a view of the interactions.

This pattern can be rendered as the collaboration fragment in Fig. 13.
**Textual Specification.** In the textual notation the pattern is rendered as follows: $C = \text{pool}(p, P) || \text{pool}(q, Q) || \text{pool}(r, R)$, where:

$$R = \text{startRcv}(m2; t_1, e7) || \text{andSplit}(e7, \{e8, e9\}) || \text{taskSnd}(e8, \exp_1, \epsilon, m4; \exp_2, e10) || \text{taskSnd}(e9, \exp_3, \epsilon, m3; \exp_4, e11) || R'$$

while $P$ and $Q$ are defined in a similar way.

**Formal Semantics.** Similar to the previous pattern, the formal semantics of this pattern is determined by the application of rules for sending and receiving message already described, except for the AND split gateway that simply consumes a token in $e7$ and, simultaneously, produces one token in $e8$ and one token in $e9$.

## 5 Patterns Animation via MIDA

The MIDA (Multiple Instances and Data Animator) tool\(^3\) is a web application written in JavaScript, based on the Camunda bpmn.io modeller. MIDA, whose graphical interface is shown in Fig. 14, is an animator of collaboration models that can involve multiple instances and data objects. MIDA animates process models by means of the visualisation of tokens flow and data evolution. To correctly enact the collaboration behaviour, the implementation of the tool relies on the formalisation presented in Sec. 3.

The core feature of MIDA is the model animation. It results helpful both in educational contexts, for explaining the behaviour of BPMN elements, and in practical modelling activities, for debugging errors. In fact, designers can achieve a precise understanding of the behaviour of processes and collaborations by means of the visualisation of the model execution.

We have exploited MIDA to model and animate the interaction patterns presented in Sec. 4, providing an intuitive knowledge of their behaviour. We have also used MIDA to animate the motivating example in Fig. 1, thus showing how the tool supports the study of more intricate scenarios resulting from the combination of various patterns. These animations are available from [http://pros.unicam.it/service-interaction-patterns/](http://pros.unicam.it/service-interaction-patterns/).

## 6 Related Work

The most common interaction scenarios from a business perspective, named Service Interaction Patterns, have been described in [3,17]. However, they lack of visualisation as well as formal semantics. Since then, effort has been devoted to visualise [5,1,9] and

---

\(^3\) [http://pros.unicam.it/mida](http://pros.unicam.it/mida)
formalise these patterns [10,2,15], as shown in Table 1. This provides a comparison among the state-of-the-art approaches dealing with service interaction patterns with respect to: (i) the language used for patterns specification, (ii) the main contribution of the work and (iii) its limitations. In the following, these works are compared to the contribution of this paper.

| Paper | Year | Language | Contribution | Limitations |
|-------|------|----------|--------------|-------------|
| [2]   | 2005 | ASM      | Formalisation and extension | No models analysis |
| [10]  | 2006 | π-calculus | Formalisation | Ambiguities |
| [15]  | 2007 | CPN      | Formalisation and extension | No correlation |
| [9]   | 2008 | BPMN 2.0 | iBPMN: extension for interaction modelling | No formalisation |
| [1]   | 2009 | Open Nets | Overview of services domain challenges | No formalisation |
| [5]   | 2014 | BPMN 2.0 | Extension of BPMN supported patterns | No formalisation |

Table 1: Review on the Service Interaction Patterns literature.

Considering the patterns specified in BPMN, relevant works are [5,9]. Campagna et al. [5] discuss BPMN 2.0 support for the service interaction patterns and propose a set of enhancements to broaden it. However, they do not formalise these patterns and thus, they do not provide formal validation of the proposed solutions. Decker and Barros [9] introduce iBPMN, a set of extensions to the BPMN standard for interaction modelling. They show that most service interaction patterns can be expressed using iBPMN and present an algorithm for deriving interface behaviour models from simple interaction models. However, they do not aim at providing a formal characterisation of the proposed extensions. Both the above works are more interested in overcoming BPMN lacks for supporting interaction patterns rather than clarifying the semantics of the supported patterns, which is instead a major challenge when using BPMN collaborations [8].

Abstract State Machines (ASM) [4], π-calculus [14] and Petri Nets [12] have been proposed as a solid ground to formalise service interaction patterns. The first formalization of the patterns was given by Barros and Börger [2] proposing a compositional framework and interaction flows. They provide ASM for eight service interaction scenarios and illustrate how, by combinations and refinements of them, one can define arbitrarily complex interaction patterns. The ASMs offer an implementation draft of the patterns, but are less suited for the analysis of collaborations. Decker and Puhlmann [10] provide a formalisation of service interactions via π-calculus as a first step to analyse collaborations. However, their work shows still some ambiguities. For instance, the Racing Incoming Messages pattern allows to receive multiple messages at once, but the work does not clarify how one among the competing messages is chosen for consumption. Moreover, the authors refer to a synchronous communication model, not compliant with the BPMN standard. Mulyar et al. [15] formalise the semantics of the interaction patterns by means of Coloured Petri Nets (CPN). Moreover, they extend the scope of the original service interaction patterns by describing various pattern variants. However, even if Petri Nets provide support for multiple instance patterns, process instances are characterised by their identities, rather than by the values of their data, which are necessary for correlation [11]. Finally, van der Aalst et al. [1] provide an overview of the challenges in the domain of service interaction patterns and they propose to use open nets as a formal framework for addressing these challenges. However, they do not
aim at formalising the interaction patterns, of which they only provide a brief description. Differently from the mentioned works, we focus on BPMN by directly defining the semantics of the supported patterns, thus avoiding the mapping to other formalisms equipped with their own semantics.

7 Concluding Remarks

In this work we focus on service interaction patterns, visualising them in BPMN collaborations and providing a comprehensive formalisation by means of a direct formal semantics for BPMN collaboration diagrams. This allows to validate the semantics in [6], as we show it is suitable to cover the interaction patterns expressed in BPMN. Further, the animation tool MIDA has been exploited to model and animate the BPMN collaborations, allowing an intuitive understanding of the patterns execution.

As a future work, we plan to investigate the formalisation of new BPMN interaction patterns where data objects play a more central role.

References

1. van der Aalst, W.M., Mooij, A.J., Stahl, C., Wolf, K.: Service interaction: Patterns, formalization, and analysis. In: Formal Methods for Web Services. pp. 42–88. Springer (2009)
2. Barros, A., Börger, E.: A compositional framework for service interaction patterns and interaction flows. In: Formal Engineering Methods. pp. 5–35. Springer (2005)
3. Barros, A., Dumas, M., Ter Hofstede, A.H.: Service interaction patterns. In: Business Process Management, pp. 302–318. Springer (2005)
4. Börger, E., Thalheim, B.: Modeling Workflows, Interaction Patterns, Web Services and Business Processes: The ASM-Based Approach. In: ASM, B and Z, pp. 24–38. Springer (2008)
5. Campagna, D., Kavka, C., Onesti, L.: Enhancing BPMN 2.0 support for service interaction patterns. In: Software Engineering and Applications (ICSOFT-EA) (2014)
6. Corradini, F., Muzi, C., Re, B., Tiezzi, F., Rossi, L.: Animating multiple instances in bpmn collaborations: from formal semantics to tool support. In: BPM 2018. LNCS, Springer (2018), to appear
7. Corradini, F., Muzi, C., Re, B., Tiezzi, F., Rossi, L.: Mida: Multiple instances and data animator. In: BPM 2018 (Demo). LNCS, Springer (2018), to appear
8. Cortes-Cornax, M., Dupuy-Chessa, S., Rieu, D.: Choreographies in BPMN 2.0: new challenges and open questions. In: ZEUS. vol. 847, pp. 50–57. CEUR-WS.org (2012)
9. Decker, G., Barros, A.: Interaction modeling using BPMN. LNCS 4928, 208–219 (2008)
10. Decker, G., Puhlmann, F., Weske, M.: Formalizing service interactions. In: International Conference on Business Process Management. pp. 414–419. Springer (2006)
11. Decker, G., Weske, M.: Instance isolation analysis for service-oriented architectures. In: SCC. vol. 1, pp. 249–256. IEEE (2008)
12. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Information and Software Technology 50(12), 1281–1294 (2008)
13. Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A.: Fundamentals of business process management. Springer (2013)
14. Milner, R.: A Calculus of Communicating Systems. Springer, Secaucus, NJ, USA (1980)
15. Mulyar, N., van der Aalst, W.M., Aldred, L., Russell, N.: Service interaction patterns: A configurable framework. management 34, 29 (2007)
16. OMG: Business Process Model and Notation (BPMN V 2.0) (2011)
17. Weske, M.: Business Process Management. Springer (2012)