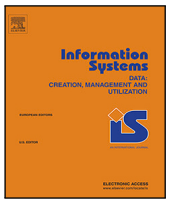
Information Systems 144 (2027) 102806



Contents lists available at ScienceDirect

Information Systems

journal homepage: www.elsevier.com/locate/is



On the interplay between multi-instance BPMN collaborations and the environment

Flavio Corradini [a], Luca Mozzoni [a], Jessica Piccioni [a,*], Barbara Re [a], Lorenzo Rossi [a], Francesco Tiezzi [b]

[a] University of Camerino, Computer Science Division, Camerino, Italy
[b] University of Florence, Dipartimento di Statistica, Informatica, Applicazioni, Firenze, Italy

ARTICLE INFO

Keywords:
BPMN
Physical environment
Meta-model
Formal semantics
Multiple instances
Time
Animation

ABSTRACT

Nowadays, business processes are increasingly influenced by information from the surrounding physical environment and vice versa. This is especially evident in the presence of multi-instance participants able to move in and interact within a physical environment. In this regard, BPMN is a widely accepted standard for representing multi-party business processes in terms of collaboration diagrams, offering an expressive and understandable notation. However, BPMN, as well as its extensions, overlooks the support for the physical environment. In this paper, we describe the interplay between BPMN collaborations, including multiple instances of the same participant, and the physical environment. Specifically, we extend the BPMN meta-model with environmental concepts, defining environmental BPMN collaboration models. To facilitate a deeper understanding of the dynamics of these models, we provide a formal account of their semantics, addressing the handling of multiple instances of the same participant and incorporating time as a factor influencing process execution during interplay with the environment. Beyond defining a novel formalization, we propose BEAR, a tool that implements the formal semantics and supports modeling, animation, and debugging of environmental BPMN collaborations, extended to support multi-instance participants and time progression. We illustrate our findings through a fire-extinguishing multi-instance collaborative scenario.

1. Introduction

Nowadays, modern organizations continually seek efficient and effective ways to model, analyze, and optimize their business processes. In this regard, the Business Process Model and Notation (BPMN) [1] has gained widespread adoption as a standard to model business processes in an intuitive and expressive manner. In particular, the so-called BPMN collaboration diagrams specify how different participants (e.g., humans, software systems, robots) behave, exchange information, and work together in the same environment to reach a common goal.

In modeling business processes, it is crucial to consider the context in which participants can operate [2]. According to [3], a context is defined as "the minimum of variables containing all relevant information that affect the design and execution of a business process". In this work, we focus on the environmental context, which includes all information beyond the process control flow that influences the process [4], such as (i) the *space* where different process participants act and move, and (ii) the domain *attributes* that characterize it. For the sake of presentation, we henceforth use the term *environment* to denote the union of space and its associated attributes. Awareness of the environment plays an important role in determining the outcome of a business process [5]. Process participants are often part of the environment [6,7]: they can hold a position, move, react to a particular situation, or even make environmental changes. Therefore, activities, decisions, and events can depend on specific environmental conditions and influence them. More precisely, the status of the environment can (i) constrain the execution of an activity, e.g., a machine starts only if the item to work on is in place; (ii) drive decision points, e.g., if a charge station is occupied, an autonomous forklift in a warehouse decides to wait in a safety zone; (iii) trigger events, e.g., a fire that occurs in an area triggers the activation of the alarms located in the nearby. At the same time, a process activity can influence the state of the environment by (iv) involving the movement of a participant in the space, e.g., a truck that moves and reaches the loading area; (v) altering the environment, e.g., a human that closes a door or moves a box in the middle of a passage.

Such models often involve multiple instances of the same participant. Therefore, coordinating actions and interactions in such settings is essential. For example, in a scenario with multiple instances of

* Corresponding author.
E-mail address: jessica.piccioni@unicam.it (J. Piccioni).

https://doi.org/10.1016/j.is.2026.102806
Received 10 September 2025; Received in revised form 28 March 2026; Accepted 11 September 2026
Available online 23 September 2026
0306-4379/© 2026 The Authors. Published by Elsevier Ltd. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

a fire-fighting robot, if the closest instance to a fire is temporarily occupied with another task, the control system may delegate the fire-extinguishing task to another available instance. Beyond the need for multi-instance participants to coordinate their actions in a shared environment, the time required to perform these actions is also a crucial aspect to consider. In particular, temporary constraints may prevent an action from being executed immediately and may require waiting for a specific amount of time before the action becomes feasible, e.g., a robot may need to wait two minutes before retrying a failed connection. Moreover, performing a specific action in the environment may take more or less time depending on the type of action to do, e.g., moving to a given location from a distant position may take significantly longer than reaching the same destination from a nearby point. Furthermore, while some actions are instantaneous and do not consume time, e.g., registering a position, others may be time-consuming, e.g., extinguishing a fire. Additionally, in certain situations, actions may be temporarily blocked due to unmet conditions, such as an unreachable path to the destination. In these cases, the task remains stalled, but after some time, the process might perform an alternative action that unlocks the stalled activity, such as asking others to reach the destination.

As a result, accurate modeling of business processes requires explicit consideration of the environment in which participants can influence and be influenced by environmental aspects. However, the literature does not fully address the interplay between BPMN business processes and the environment. Existing works on the concept of the environment in BPMN [6–8] have limitations that underscore the need for a more comprehensive and integrated approach to modeling this interplay. Indeed, they often propose extensions to the BPMN graphical notation. Still, none of them explicitly represents and provides a formalization of the environment and how BPMN elements relate to it. Moreover, they tend to overlook the combined consideration of multiple participants interacting within the environment and of the temporal perspective.

To fill this gap, in this paper, we present an approach to bridge BPMN collaborations including multi-instance participants, the environment, and the time perspective. This work is based on the paper "On the interplay between BPMN collaborations and the physical environment" [9] and the demo paper "BEAR: BPMN and Environment AnimatoR" [10] published in the proceedings of the 22nd International Conference on Business Process Management. Besides describing our approach in greater detail, this article extends the scope of both the original conference paper and demo paper as reported.

1. We have further extended the metamodel in [9] by introducing two new task types related to the environment and the movements performed on it. They are, the *environmental task* and the *movement task*. The former focuses on the interaction and modification of the environment; the latter describes a movement on it. This extension aims to enhance model readability and provide a more accurate representation of the interaction between BPMN elements and the environment.
2. We have enriched the proposed formalization to support multi-instance pools, timer events, task durations, and timer boundary events. We have hence extended and streamlined the formal definitions of our syntax and semantics.
    (a) We handle data coming from multi-instance pools using data object collections, and we adopt the Friendly Enough Expression Language (FEEL) [11] to standardly manipulate such data.
    (b) We have defined a set of auxiliary FEEL functions to facilitate the modeling of environmental BPMN collaborations, thereby improving the readability and maintainability of BPMN models.
3. We have extended the BEAR tool to support the novel elements introduced in the formalization.
4. We have enriched the running example with a collaborative fire-extinguishing scenario involving multiple robot instances, in order to illustrate better the effectiveness and potential of our formalization and tool, and to facilitate the understanding of the concepts as mentioned above.

The rest of this paper is organized as follows. Section 2 shows the interplay of BPMN collaborations and the environment. Section 3 explores the modeling and management of multi-instance pools and sequence tasks, data objects, and time. Section 4 introduces the formal syntax of environmental BPMN collaborations while Section 5 describes their semantics. Section 6 and Section 7 illustrate our formalization at work through the running example and how BEAR practically realizes such formal concepts, respectively. Section 8 discusses related work, and Section 9 concludes the paper by outlining the limitations of our approach and potential future work.

2. Bridging BPMN and environment

In this section, we present our approach to combine the BPMN meta-model with the environment. We also introduce a collaborative scenario, which we use throughout the paper as a running example.

Running example. We consider a fire-extinguishing collaboration in a student dormitory. The dormitory, inhabited by students, comprises a kitchen, an entrance, three bedrooms, two bathrooms, and a living room connected by a corridor. Each room has a door and a fire sensor, except for the corridor, which has three fire sensors. The collaboration is performed by eleven sensors, a fire control system, and three fire-fighter autonomous robots. The sensors are in charge of detecting possible fires in dormitory rooms and notifying the fire control system about it; the fire control system manages possible fire notices by informing the closest robot fire-fighter, which is in charge of extinguishing the fire. The collaboration starts when one of the sensors detects a fire presence and notifies it to the fire control system. In this case, the fire control system activates the alarm, calculates the closest robot, and notifies the fire position to it. The robots wait at their base station for a message from the control system. As soon as the closest robot receives a notification, it moves towards the fire and, after extinguishing it, sends feedback to the control system, which turns off the alarm.

Modeling the environment. To consider the environment in the modeling and execution of a business process, we need a way to represent it. Over the years, several models have been proposed to represent the environment [12]. In particular, two main classes of environmental models are defined: *geometric* and *symbolic*. The former mainly comprises cell-based [13,14] and boundary-based [15] geometrical representations. These models typically focus on quantitative aspects of space, such as room dimensions or road lengths, providing precise information. The latter uses topological-based structures [16] or graphs [17], where it is possible to define an ad-hoc level of abstraction by capturing the connectivity and reachability between two locations in the space. Since our target is to combine the environment to the BPMN notation, we chose place graphs as they can abstract the space topology at will, like also BPMN does [18,19]. Indeed, place graphs are a particular kind of symbolic model, where places represent a predefined area, like buildings and rooms, and the edges between two places represent connections that make it possible to move between them [12,20]. Thus, they provide a direct way to represent movements between places and planning routes to destinations. Moreover, place graphs can be enriched with attributes related to places, e.g., the illumination status, and edges, e.g., the distance.

Referring to the running example, Fig. 1 depicts the place graph of the dormitory on top of its floor plan, which comprises 44 places and 168 edges; for the sake of presentation in the figure, we use bi-directional edges, while the place graph is directed, meaning that two places may be connected by two distinct edges, one in each direction,

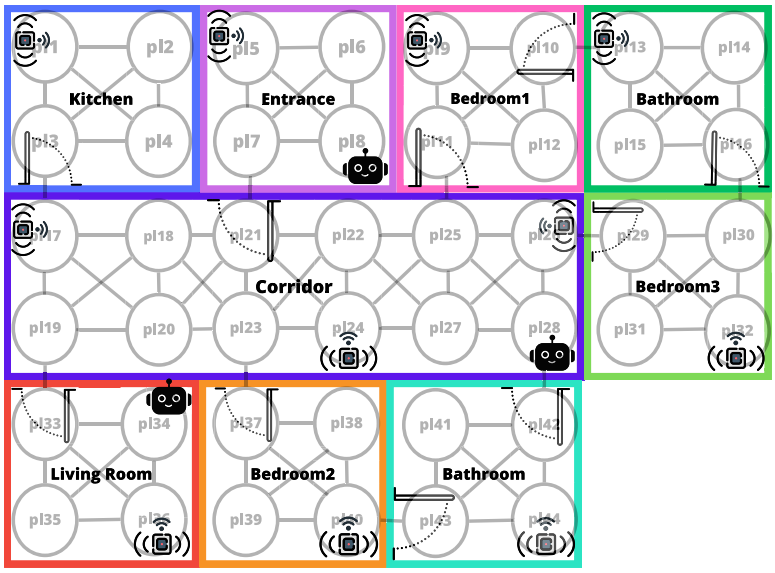


Fig. 1. Floor plan and place graph of the dormitory.

representing possible movement from one place to the other and vice versa. To cover the entire dormitory, we spread the places on the floor plan homogeneously, i.e., adjacent places are equidistant and connected through edges, except for places divided by a wall or a closed door. Besides representing the environment topology, the place graph includes the attributes necessary to handle the fire-extinguishing collaboration. In particular, we include in each place attributes indicating the presence of fire and the status of the alarm.

Extending the bpmn meta-model. To combine BPMN collaborations with the environment, we extend the BPMN meta-model using its extensibility mechanism [21] as described in Fig. 2.

For what concerns the environment, we introduce the place graph via two new classes, called *Place* and *Edge*, which respectively correspond to places and edges of the place graph. Places refer to edges (and vice-versa) through two attributes *sourcePlace* and *targetPlace* (*incoming* and *outgoing*). Each place can be linked to many source/target places, permitting the representation of the entire place graph. Each place also has a list of *placeAttributes*, which refers to its environment characteristics. The extension point between the environment and BPMN collaboration happens through the attribute *position* that links the BPMN classes *Participant* and *Place*. This relationship is used to indicate the current position of participants in the environment. Concerning BPMN process tasks, we introduced two new types of tasks: *Environmental Tasks* and *Movement Tasks*. Both tasks extend the BPMN class *Task* and its associated attributes. Environmental tasks are primarily focused on interacting with and modifying the environment. To enable this functionality, we introduce two new attributes, attached to the BPMN class *Task*, called *Guard* and *Assignment*. The former is used to constrain the execution of a task to an environmental status, the latter is used to modify the environment topology or its attributes. Graphically, environmental tasks are represented by a world icon, indicating their close relationship with the environment, as shown in Fig. 3(a). Although the environmental task inherits all the properties of the BPMN task, its introduction mainly aims to improve the readability of diagrams by clearly indicating that the task has effects on the environment.

In contrast, movement tasks involve the movement of a participant within the environment. In addition to supporting guard and assignment attributes inherited by BPMN task, they introduce a specific attribute called *destination*, specifying a target place to reach. Movement tasks are visually denoted by an arrow icon as is depicted in Fig. 3(b). Both environmental and movement tasks also include the *duration* attribute, inherited from the *Task* class, which specifies the interval time required to execute the task.

The guard, assignment, and destination attributes are defined using formal expressions, so they are a composition of the class $FormalExpression$, which extends the class *Expression*. The $Expression$ class specifies expressions that are not executable and are considered underspecified. The $FormalExpression$ class, instead, is used to specify executable expressions using a given expression language, whose instantiation is left to the designer to deal with specific concrete uses of the notation. In this paper, we rely on FEEL, the expression language defined in the DMN standard [11] and formalized in [11,22,23]. FEEL provides a declarative language that enhances the efficiency and readability of the model, providing the necessary expressiveness to handle the real-world complexity often encountered in business processes, without sacrificing the ease of use. Most of all, the use of FEEL permits to keep the decision logic of models independent from the expression language of the underlying programming language, thus making the model shareable, interpretable, and maintainable across different tools and platforms. In this work, FEEL is used to define guards, assignments, and auxiliary functions to support the management and analysis of place graphs and BPMN processes, such as *getPlaces()*, which returns all places in the place graph, or *getInstance(name, idinst)*, which retrieves the information of a specific instance (identified by its ID) of a BPMN process (identified by its name). The complete account of FEEL auxiliary functions is provided in Section 4.

To give a concrete view of the combination between a BPMN collaboration and the environment, we provide in Fig. 4 the collaboration diagram representing the behavior of the running example related to the environmental model in Fig. 1. The collaboration comprises a multi-instance pool encapsulating the behavior of sensors, a single instance pool that is related to the fire control system, and a multi-instance pool referring to fire-fighting robots. To improve the comprehension of the running example, we report in Table 1 the description of the attributes contained in the data objects.

The fire-extinguishing collaboration process starts when a sensor instance, after sensing the environment, detects fire in the environment through the conditional intermediate event *Fire detection*. This event continuously monitors the environment for any changes. When the condition associated with this event evaluates to true (i.e., when the value of the place attribute `fire` linked to the surrounding places of the sensor becomes true), the process moves to the send event *Fire Position Sent*. This event sends a message to the fire control system with the position

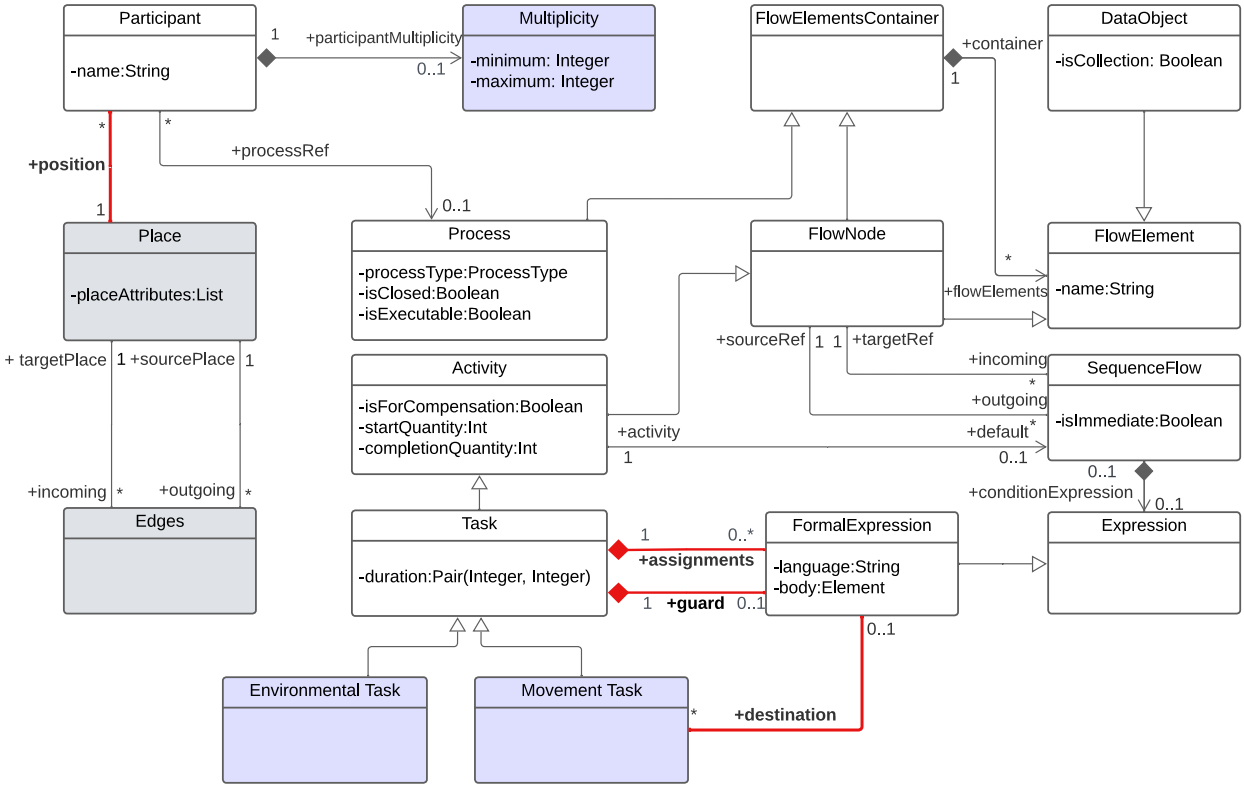


Fig. 2. BPMN meta-model extended with the environment.

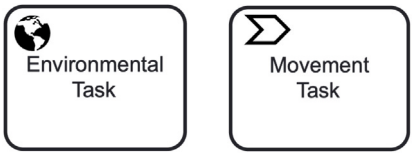


Fig. 3. Environmental task (a) and Movement task (b).

Table 1
A description of the data object attributes.

Data object	Attributes	Description
Fire pos	*place*	The *place* at fire.
Robot positions	*list*	A *list* of couples of id and place.
Robot position	*id*, *place*	The *id* of the robot and the *place* it holds.
Fire counter	*count*	The *count* of detected fires.
Fire position	*place*	The *place* at fire.
Closest robot	id	The *id* of the robot.
Fire intervention	*place*	The *place* where to intervene.
Base station	*place*	The *place* of the base station.
Fire	*place*	The fire *place* assigned to the Fire-fighter.

of the detected fire. Meanwhile, the fire control system, after receiving the positions of all fire-fighting robot instances in the environment, waits for a message from one of the sensors indicating a fire to be extinguished. Upon receiving a fire position, the system activates the alarm if it is not already active. Then, the fire control system checks the position of the fire and updates the value of the data object *Fire counter*, which keeps track of the number of active fires and is incremented each time the sensor reports a new fire to be handled. Next, it calculates the closest fire-fighting robot and notifies it by sending a *presence fire* message. After this, the fire control system resumes managing other potential fires while concurrently waiting for a response confirming that the reported fire has been extinguished or not. When the instance of the closest robot receives the notification about the fire to extinguish, it performs the task *Move to fire position*. This activity involves the movement of the process participant in the environment. In particular, the robot will try to reach the destination (corresponding to the fire position) from its current location (corresponding to its base station). If the robot manages to reach the destination, it extinguishes the fire, changes the value of the place attribute `fire` to false, and sends feedback to the fire control system, which updates the data object *Fire counter*. Each notification is sent by the robot dispatched to handle a specific fire. In scenarios where multiple robots are activated, the fire control system may receive multiple fire extinguished messages; however, the order in which these notifications are received is irrelevant, as the *Fire counter* is simply decremented for each successfully extinguished fire. The fire control system turns off the alarm only when *Fire counter* indicates zero active fires. Then, the robot returns to the base station. Regarding the *Move to fire position* task, the timer boundary event is used to trigger exceptional behaviors if the robot cannot reach a destination. The unreachability could be caused by the absence of a path from the source location to the destination. In this case, the boundary event attached to the *Move to fire position* task leads the robot to notify its blocked condition to the fire control systems and return to the base station without extinguishing the fire. In this case, the fire is extinguished by humans with the task *Intervention by human*.

3. Dealing with multi-instances, data objects and time

Business processes often involve multiple participants who must share the same environment. In such cases, their actions and decisions need to be carefully coordinated. In addition, managing communication between multiple participants is essential for the correct execution of actions and coordinating activities within the environment. Indeed, communication plays a key role in determining, for example, which

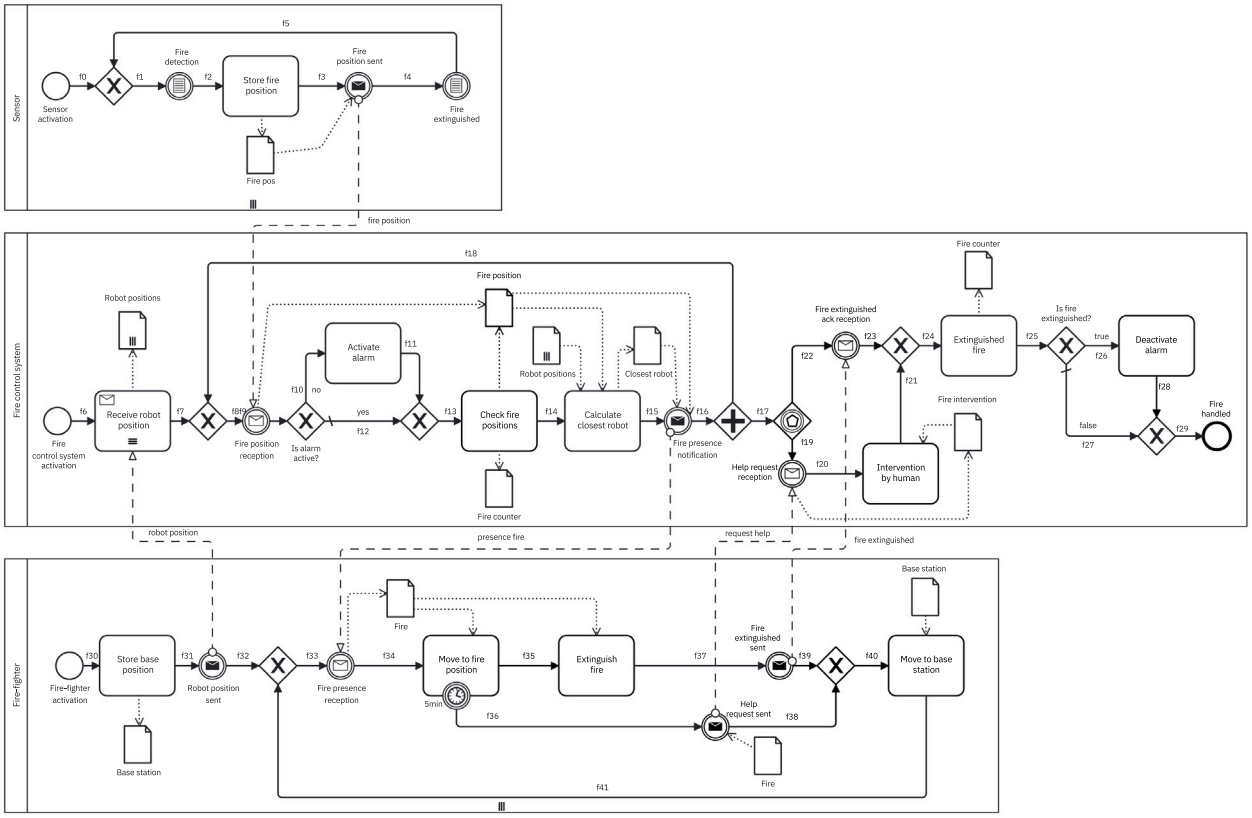


Fig. 4. Fire-extinguishing collaboration model.

specific instance of a participant should carry out a given action in a shared environment, or in ensuring that all relevant spatial information has been correctly received and recorded. This becomes particularly challenging in complex scenarios involving collaboration models with both single-instance and multi-instance participants.

To address these challenges, we use sequential multi-instance message tasks. In this regard, to keep the formal treatment concise and simplify the tool implementation, we treat the multi-instance tasks as macros. Such macros represent the communication between: (i) a single-instance pool sending multiple sequential messages to a multi-instance pool, and (ii) a multi-instance pool sending a message to a single-instance pool that contains a multi-instance receiving task. In the first case, as depicted in Fig. 5, the single-instance pool repeatedly sends messages to the multi-instance pool through a loop of activities that evaluate guard conditions, send messages, and perform assignments until the intended number of messages is reached. The second case, shown in Fig. 6, adopts a similar logic, but focuses on handling multiple incoming messages within a single multi-instance receiving task. Notably, a comprehensive treatment of multi-instance sequence and parallel message tasks can be found in the work of Corradini et al. [24]. For the sake of readability and clarity, in Figs. 5 and 6 we omit to represent data objects, as their structure and usage may vary depending on the specific application scenario. An example of the treatment of data objects within the macros is provided in Fig. 7 referring to our running example.

The fire-extinguished collaboration model, initially presented in Fig. 4, has been redefined using the proposed macros. In particular, we updated the fire control system pool as illustrated in Fig. 7 by transforming the multi-instance task *Receive robot position* in Fig. 4 into a loop following the macro described in Fig. 6. In this case the single instance pool *Fire control system* receives the position of the first robot and stores it in the data object *Robot position* which serves as a storage for one instance at a time. The task *Receive robot position assignment* then appends this information on the data object collection *Robot Positions*, which collects the positions of all robots, and decrements the counter in the *Loop cardinality* data object by one. The fire control system subsequently returns back to receive the position information from the second robot, overwriting the content of *Robot position* data object with the newly received data. This information is again appended to the collection *Robot Positions*, and the procedure is repeated until the counter if the loop cardinality reaches 0.

In addition to managing multi-instance participants, it is essential to properly handle the information exchanged and manipulated during process execution. To store environmental information, we rely on data objects and assume that everything manipulated by the process must be explicitly represented, ensuring full visibility into the data flow. Therefore, all data objects and data associations are included in the collaboration model; an exception was only made for those data associations linked to XOR gateways, which are omitted for the sake of presentation. Moreover, in accordance with the guidelines of the BPMN standard [1, page 207], we represent the same data object instance multiple times within the model to improve diagram readability. In scenarios involving multi-instance pools or tasks, data object collections are used to represent sets of data objects effectively. The contents of both individual data objects and data object collections are expressed in JSON format, which aligns with the structure used for data inputs in FEEL expressions. This ensures consistency in data handling and facilitates integration with FEEL expressions.

Concerning time, we assume that each BPMN element requires a certain duration to be executed. Regarding the work of Hennessy et al. [25], we distinguish between elements that require time to execute and those that occur instantaneously, independently of time. We also adopt the principle of maximal progress [26], which asserts that internal actions, such as message exchanges, must be executed as soon as they are enabled, preventing unnecessary delays and ensuring that time does not advance while instantaneous actions are still possible. Consequently, in our paper, we assume that all BPMN events and gateways

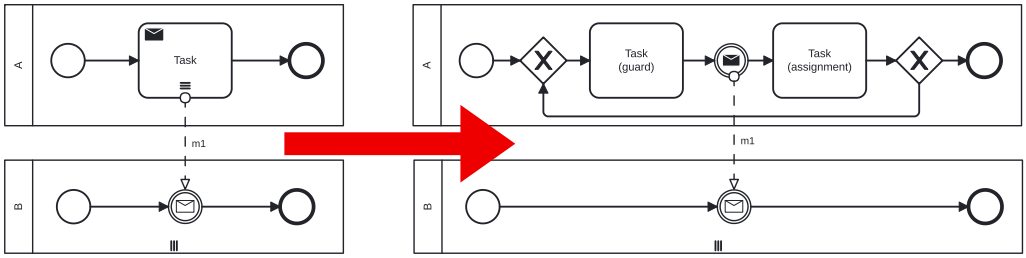


Fig. 5. Macro for multi-instance sending task.

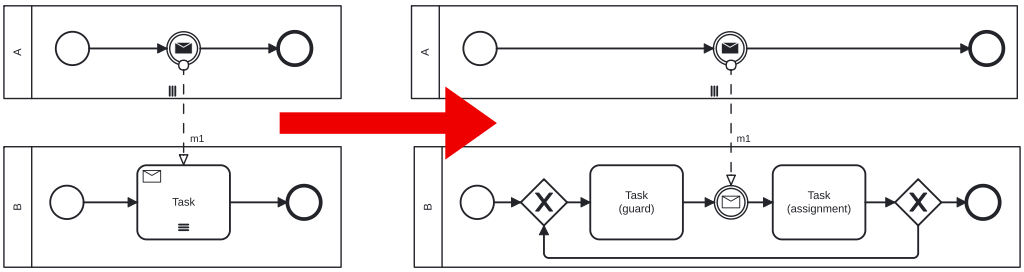


Fig. 6. Macro for multi-instance receiving task.

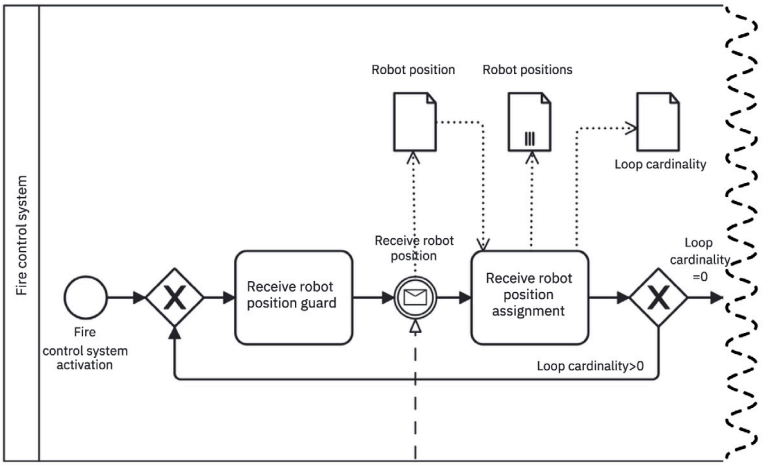


Fig. 7. An excerpt of the Fire control system with macro.

are instantaneous and do not consume time, except for timer events, which have a predefined duration, and tasks, which have a duration time. In particular, tasks and environmental tasks must be assigned a predefined, explicit duration, whereas the movement tasks have an implicit duration equal to the time needed to reach the destination, which depends on spatial factors such as the distance to the destination and environmental conditions. For instance, the execution time may vary if participants encounter an obstacle or cannot reach a destination due to a blocked path. Moreover, referring to the work of Aceto et al. [27], we assume that the execution time for BPMN tasks is discrete, progressing in distinct steps. By adopting a discrete notion of time, we can more easily represent task durations, synchronize concurrent activities, and evaluate the progression of system states in a controlled manner.

4. Formal account of environmental BPMN collaborations: syntax

To enable precise reasoning about BPMN collaborations and their environment, we formally define environmental BPMN collaboration models. To keep the formalization manageable and understandable, we focus on elements that are strictly needed to define meaningful collaborations, the environment where the collaboration happens, and the relations between them.

The formalization extends the one proposed in [9] and resorts to a textual representation that uses a Backus–Naur Form (BNF) syntax for representing the new BPMN elements and the extension of the environment. The motivation for using a textual representation rather than the usual graphical notation is that the former is more manageable for writing operational semantic rules than the latter. In addition, the textual notation makes explicit those technical details of collaboration models that are not part of the graphical representation (e.g., message payloads, assignments, guard conditions), but are part of the low-level XML characterization of the model. This information is needed to properly define the execution semantics of the environmental BPMN collaboration models. The correspondence between the graphical notation of BPMN and the textual representation used here is straightforward, as highlighted by the description associated to each syntactical term in Fig. 8. For a more detailed account of the correspondence, the interested reader can refer to Appendix.

C ::=		*(Collaboration Structures)*
	$\mathsf{pool}(\mathsf{p}, P, \mathsf{k})$	(pool)
|	$C \parallel C$	(pool composition)
P ::=		*(Process Structures)*
	$\mathsf{start}(\mathsf{f}, \mathsf{f}')$ | $\mathsf{end}(\mathsf{f})$	(start event, end event)
|	$\mathsf{sndEvent}(\mathsf{f}, \mathsf{m}:\tilde{\mathsf{exp}}, \mathsf{f}')$	(msg sending intermediate event)
|	$\mathsf{rcvEvent}(\mathsf{f}, \mathsf{m}:\tilde{\mathsf{t}}, \mathsf{f}')$	(msg receiving intermediate event)
|	$\mathsf{condEvent}(\mathsf{f}, \mathsf{exp}, \mathsf{f}')$	(conditional catching interm. event)
|	$\mathsf{timerEvent}(\mathsf{f}, \mathsf{n}, \mathsf{k}, \mathsf{f}')$	(intermediate timer event)
|	$\mathsf{andSplit}(\mathsf{f}, F)$ | $\mathsf{andJoin}(F, \mathsf{f})$	(AND split/join gateway)
|	$\mathsf{xorSplit}(\mathsf{f}, (\mathsf{exp}_1, \mathsf{f}_1) \ldots (\mathsf{exp}_n, \mathsf{f}_n))$	(XOR split gateway)
|	$\mathsf{xorJoin}(F, \mathsf{f})$	(XOR join gateway)
|	$\mathsf{eventBased}(\mathsf{f}, (\mathsf{m}:\tilde{\mathsf{t}}, \mathsf{f}_1)^*, (\mathsf{exp}, \mathsf{f}_2)^*, (\mathsf{n}, \mathsf{k}, \mathsf{f}_3)^?)$	(event-based gateway)
|	$\mathsf{task}(\mathsf{f}, \mathsf{n}, \mathsf{exp}_g, A, M, \mathsf{d}, B, \mathsf{f}')$	(task/environmental task)
|	$\mathsf{taskM}(\mathsf{f}, \mathsf{n}, \mathsf{exp}_g, A, M, \mathsf{exp}_d, B, \mathsf{f}')$	(movement task)
|	$P \mid P$	(process composition)
A ::=	$\mathsf{assign}(\mathsf{exp}_f, \mathsf{do.a})^*$ $\mathsf{assign}(\mathsf{exp}_f, \mathsf{pl.a})^*$ $\mathsf{assign}(\mathsf{exp}_f, \mathsf{exp}_t, \mathsf{a})^*$ $\mathsf{append}(\mathsf{exp}_f, \mathsf{do.a})^*$	*(Assignments, Appends)*
M ::=	$\mathsf{connect}(\mathsf{e})^*$ $\mathsf{disconnect}(\mathsf{e})^*$	*(Place Graph Modification)*
B ::=	$\mathsf{bCondEvent}(\mathsf{exp}, \mathsf{f})^*$ $\mathsf{bTimerEvent}(\mathsf{n}, \mathsf{k}, \mathsf{f})^?$	*(Boundary Events)*
G ::=	(P_l, E) with $E \subseteq P_l \times P_l$	*(Place Graphs)*

(Notation)

Pool name: p	Data object name: do
Sequence flow: f	Place name: pl
Sequence flow set: F	Natural number: k
Message flow: m	Task and timer name: n
Expression: exp	Place set: P_l
Value: v	Edge name: e
Data Object and Place attribute: a	Edge set: E
Task duration: d	Template: $\tilde{\mathsf{t}}$

Fig. 8. BNF syntax of environmental BPMN collaboration models.

Fig. 8 reports the BNF syntax defining the textual notation describing the structure of environmental BPMN collaboration models. Specifically, the upper part of the table reports the grammar productions defining the syntax of collaboration models, the middle part defines place graphs, and the bottom part reports the notation for the generic elements of the syntactic categories. The symbol $*$ stands for (possibly empty) sequences, the symbol ? stands for 0 or 1 elements, and the notation $\tilde{\ }$ denotes tuples (e.g., $\tilde{\mathsf{exp}}$ stands for a tuple of expressions $\langle \mathsf{exp}_1 \ldots \mathsf{exp}_n \rangle$). As usual for BPMN models, we assume that message flows, sequence flows, and task names are unique in the model.

Notably, even if this syntax would allow to write collaboration models that cannot be expressed in BPMN, here are considered only those terms of the syntax that can be derived from BPMN models. Intuitively, an environmental BPMN collaboration is the union of the term C, indicating the structure of the collaboration, and the term G, indicating the environmental model. In other words, a collaboration is rendered in the presented syntax as a collection of pools coupled with an environmental model in the form of a place graph. Formally, a collaboration C is a composition, by means of the operator $\parallel$, of pool elements $\mathsf{pool}(\mathsf{p}, P, \mathsf{k})$ uniquely identified by a pool name p and enveloping process structures of the form P. Similarly, a process P is a composition of process elements (denoted by the sans serif font) by means of the operator |. We identify collaborations (resp. processes) up to commutativity and associativity of pool (resp. process element)

collection. Thus, e.g., $C_1 \parallel C_2$ and $C_2 \parallel C_1$ are identified as the same collaboration due to the commutativity property, while $P_1 \mid (P_2 \mid P_3)$ and $(P_1 \mid P_2) \mid P_3$ are identified as the same process due to the associativity property. Finally, a place graph G is a tuple (P_l, E) where P_l is the set of places of interest for the collaboration and E is the set of edges linking couples of places. To support a compositional approach, in the textual notation each sequence/message flow of the graphical notation is split into two parts: the part outgoing from the source element and the part incoming into the target element. The two parts are correlated by a unique sequence/message flow name.

More specifically, the elements of the notation are as follows.

- pool(p, P, k) denotes a pool identified by a name p, enveloping a process structure P, and characterized by k instances (k = 1 for single-instance pools, and k > 1 for multi-instance pools). The number k of instances is the minimum and maximum number of instances in a given pool, as our model does not consider dynamic instantiation. Thus, the number of instances set at design time remains fixed: no new instances can be created or removed during execution. This is because process instances represent participants that are present in the environment since the beginning of the execution and that cannot disappear at runtime (like robots and sensors in a building). This choice reflects the assumption that all participants are known and predefined in the model and also in the environment.
- start(f, f′) denotes a start event that contains an incoming (spurious) sequence flow f, named enabling flow, used to activate the start event, and the outgoing sequence flow f′.
- end(f) denotes an end event and f is the incoming sequence flow.
- sndEvent(f, m : e$\tilde{\text{x}}$p, f′) denotes a message intermediate sending event that is characterized by an incoming sequence flow f, a pair m : e$\tilde{\text{x}}$p, where m is the (unique) message name and e$\tilde{\text{x}}$p is a tuple of expressions whose evaluation will return a tuple of values to be sent, and f′ is the outgoing sequence flow.
- rcvEvent(f, m : $\tilde{\text{t}}$, f′) denotes a message intermediate receiving event that is characterized by an incoming sequence flow f, an outgoing sequence flow f′, and a pair m : $\tilde{\text{t}}$ where $\tilde{\text{t}}$ denotes a template used to select matching messages m received by the pool. Formally, a template $\tilde{\text{t}}$ is a sequence of terms $\text{t}_1, \ldots, \text{t}_n$ each of which can be an expression or a formal field, i.e., t ::= exp | ?do.a | ?pl.a. Formal fields are data objects/place attributes fields identified by the ?-tag and are used to bind fields to values.
- condEvent(f, exp, f′) denotes a conditional event characterized by an incoming and an outgoing sequence flow (f and f′), and by a conditional expression in the form exp, which is evaluated to trigger the event.
- timerEvent(f, n, k, f′) denotes a timer event characterized by an incoming and an outgoing sequence flow (f and f′), a name n, and a duration k which represents the discrete amount of time to wait until the timer event is triggered.
- andSplit(f, F) denotes an AND split gateway characterized by an incoming sequence flow f and a non-empty set of outgoing sequence flows F.
- andJoin(F, f) denotes an AND join gateway characterized by a non-empty set of incoming sequence flows F and an outgoing sequence flow f.
- xorSplit(f, $(\text{exp}_1, \text{f}_1) \ldots (\text{exp}_n, \text{f}_n)$) denotes an XOR split gateway characterized by an incoming sequence flow f and a list of outgoing sequence flows, each one specifying a condition exp to decide which sequence flow to activate.
- xorJoin(F, f) denotes XOR join gateways characterized by a set of incoming sequence flows F and an outgoing sequence flow f.
- eventBased(f, $(\text{m} : \tilde{\text{t}}, \text{f}_1)^*, (\text{exp}, \text{f}_2)^*, (\text{n}, \text{k}, \text{f}_3)^?$) denotes an event-based gateway which is similar to the XOR-split gateway, but its outgoing sequence flow activation depends on whether catching event takes place. In this paper, we consider three catching events: receiving message events $(\text{m} : \tilde{\text{t}}, \text{f}_1)$, conditional events (exp, f_2) and timer events $(\text{n}, \text{k}, \text{f}_3)$.
- task(f, n, exp_g, A, M, d, B, f′) denotes a task, or an environmental task, that has an incoming and an outgoing sequence flow, f and f′, respectively, a name n, a guard condition exp_g defined over attributes, a list of attribute assignments A, and a list of spatial modifications M. The task also has a duration d that represents the amount of time it takes to execute, and possibly it can have an attached boundary event B. Note that, when there is no guard to evaluate, its value is set to *true*, while if there is no assignment to apply, its value is set to the empty list (denoted by ϵ). Moreover the duration d can be expressed in different ways; in this paper, it is considered as couple $(\text{k}_{min}, \text{k}_{max})$. When $\text{k}_{min} = \text{k}_{max}$ the interval collapses to a single value. We use the same syntactic element to denote both task and environmental task; indeed, as explained in Section 2, the distinction between the two is introduced solely to improve model readability and to highlight that the guard conditions and assignments associated with environmental tasks have a direct impact on the environment.
- taskM(f, n, exp_g, A, M, exp_d, B, f′) denotes a movement task, which is a type of task that may involve a movement in the place graph; it is similar to the previous case, but it has an additional expression exp_d that represents the destination to reach. Although in the metamodel in Fig. 2 the movement task inherits the attribute duration, this attribute is not explicitly defined in its syntax, because it is assumed that the duration of a movement task corresponds to the time it takes to move from one place to another to reach the destination. Therefore, in the formal syntax, a movement task does not explicitly include the term d.
- assign(exp_f, do.a) and assign(exp_f, pl.a) denote an assignment that assigns the value resulting from the evaluation of the expression exp_f to the data object attribute do.a or the place attribute pl.a, respectively.
- assign(exp_f, exp_t, a) denotes an assignment where the value resulting from evaluating the expression exp_f is assigned to the attribute a of each element in the evaluation result of the expression exp_t.
- append(exp_f, do) denotes the insertion of new items, resulting from evaluating the expression exp_f, to the data object collection do.
- connect(e) and disconnect(e) denote spatial modifications that modify the space topology of the environment by removing or adding a connection between two places, respectively.
- bCondEvent(exp, f) denotes a conditional boundary event that can be attached to tasks.
- bTimerEvent(n, k, f) denotes a timer boundary event that can be attached to tasks.

We do not provide a direct syntactic representation of data objects and place attributes, the evolution of their state during the model execution is a semantic concern described later in this section. As well, attributes are not typed and we do not assume that the expressions or the assignments are well-formed; so conflicts can happen in case of wrong modeling. Nevertheless, all expressions are specified in FEEL, whose syntax and semantics are formally defined in [11,22,23]; we adopt a JSON structure for data objects and place attributes. In this context, we have also defined a set of FEEL auxiliary functions to support the management and analysis of place graphs and BPMN processes. These functions are designed to be reusable across various contexts, providing flexible and robust operations for handling place graphs and process instances. Fig. 9 presents an overview of the defined auxiliary functions, all of which were tested using the online tool Feel Playground.[1] FEEL also allows developers to define custom functions, which can be tailored to meet specific requirements in a

[1] https://github.com/nikku/feel-playground.

Function getPlaces() returns all places of the place graph. `getPlaces: function() {for p in places return p}` where `places` stands for the set of physical places P_l.
Function getEdges() returns all edges of the place graph. `getEdges: function() {for e in edges return e}` where `edges` stands for the set of edges E.
Function getInstances(name) returns all the information of all the instances of a BPMN process pool by passing the pool name. `getInstances: function(name) {for i in name return i}` where `name` stands for a pool name p.

Function getInstance(name,idinst) returns all the information of a specific instance of a given BPMN process.
`getInstance: function(name, idinst)`
`{getInstances(name)[item.id=idinst]}`
where `name` stands for a pool name p and `idinst` stands for the number of instance whose information is being retrieved.

Function getPlace(name,idinst) returns the place of a specified instance of a BPMN process,
`getPlace: function(name,idinst)`
`{getInstance(name,idinst).location}`

Function isConnected(place1,place2) returns true/false if there is an edge that connects the two given places.
`isConnected: function(place1,place2)`
`{some e in getEdges() satisfies`
`(e.sourcePlace=place1 and e.targetPlace=place2) or`
`(e.sourcePlace=place2 and e.targetPlace=place1)}`
where `place1` and `place2` stand for physical place names belonging to P_l.

Function neighbour(place) returns all adjacent places to the one that is passed as the argument.
`neighbour: function(place)`
`{getPlaces()[isConnected(place, item.id)]}`
where `place` stands for a physical place name pl.

Function meAndNeighbour(place) returns all the places adjacent to the one passed as an argument, including the place itself.
`meAndNeighbour: function(place)`
`{getPlaces()[isConnected(place, item.id) or place=item.id]}`

Fig. 9. Auxiliary FEEL functions.

scenario. Custom functions can also interact with other FEEL functions or external libraries, expanding the scope of operations available within the FEEL environment; e.g., the function *Calculate_closest_robot (process_instances.place, place.fire)* is a custom function that uses external libraries to calculate the closest robot to a fire, based on the position of all process instances and the position of the fire to be extinguished.

In Fig. 10, we provide the textual representation of the environmental BPMN collaboration model of the running example depicted in Fig. 4, which uses as a place-graph the one depicted in the background of Fig. 1. The textual representation is given after applying the macro defined in Fig. 7, so it corresponds to the expanded form of the running example. This exemplifies the correspondence between the BPMN graphical notation and the proposed BNF syntax. In Section 7 we exploit the textual representation of the example to discuss the semantics of the running scenario.

5. Formal account of environmental BPMN collaborations: semantics

To formally describe the semantics, the structural information introduced in the previous section is enriched with a notion of execution state, given by the marking of sequence flows with tokens [1, p. 27], the value of data object, the value of place attributes, the status of tasks, the position of the participants in the place graph, the payloads stored in the message queues, and the value of timers.

These stateful descriptions are called process configurations and collaboration configurations. A *process configuration* (resp. *collaboration configuration*) has the form $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle$ (resp. $\langle C, G, \sigma_i, \sigma_{apl}, \sigma_p, \sigma_m \rangle$) where P is a process structure (resp. C is a collaboration structure); G is the place graph; p is the pool name, and j is the instance of the pool's process identified by a natural number. Moreover, the configurations include the following state functions.

- $\sigma_f : \mathbb{F} \to \mathbb{N}$ is a *sequence flow state function* (where $\mathbb{F}$ is the set of sequence flows and $\mathbb{N}$ is the set of natural numbers) specifying for each sequence flow the current number of tokens marking it; σ_f^0 denotes the state where all sequence flows are unmarked except for the spurious flows, each one marked by one token.
- $\sigma_{ado} : \mathbb{ADO} \to \mathbb{V}$ is a *data object state function* (where $\mathbb{ADO}$ is the set of data object attributes and $\mathbb{V}$ is the set of values) assigning values (possibly null) to data object attributes. σ_{ado}^0 denotes the state where all data object attributes are set to null except for those data objects (the so-called *data inputs*) whose initial value is already provided by the modeler.
- $\sigma_{apl} : \mathbb{APL} \to \mathbb{V}$ is a *place attribute state function* (where $\mathbb{APL}$ is the set of place attributes) assigning values (possibly null) to place

$C_d = \mathsf{pool}(\mathsf{p}_s, P_s, 11) \parallel \mathsf{pool}(\mathsf{p}_f, P_f, 1) \parallel \mathsf{pool}(\mathsf{p}_r, P_r, 3)$

$P_s = \mathsf{start}(\mathsf{f}'_0, \mathsf{f}_0) \mid \mathsf{xorJoin}((\mathsf{f}_0, \mathsf{f}_5), \mathsf{f}_1) \mid \mathsf{condEvent}(\mathsf{f}_1, \mathsf{exp}_1, \mathsf{f}_2) \mid \mathsf{task}(\mathsf{f}_2, \mathsf{Store_fire_position}, \mathsf{true}, \mathsf{A}_1, \varepsilon, 0, \varepsilon, \mathsf{f}_3) \mid$
$\mathsf{sndEvent}(\mathsf{f}_3, \mathsf{fire_pos}: \mathsf{Fire_pos.place}, \mathsf{f}_4) \mid \mathsf{condEvent}(\mathsf{f}_4, \mathsf{exp}_2, \mathsf{f}_5)$
$\mathsf{exp}_1 = \texttt{some i in Fire_status() satisfies i=true}$
$\mathsf{exp}_2 = \texttt{every i in Fire_status() satisfies i=false}$
where $\texttt{Fire_status: function() for pl in neighbour(myPlace()) return pl.fire}$
$A_1 = \mathsf{assign}(\mathsf{myPlace}(), \mathsf{Fire_pos.place})$

$P_f = \mathsf{start}(\mathsf{f}'_6, \mathsf{f}_6) \mid \mathsf{xorJoin}((\mathsf{f}_6, \mathsf{f}_{11}), \mathsf{f}_7) \mid \mathsf{task}(\mathsf{f}_7, \mathsf{Receive_robot_position_guard}, \mathsf{true}, \varepsilon, \varepsilon, 0, \varepsilon, \mathsf{f}_8) \mid$
$\mathsf{rcvEvent}(\mathsf{f}_8, \mathsf{robot_position}: \langle ?\mathsf{Robot_position.id}, ?\mathsf{Robot_position.place} \rangle, \mathsf{f}_9) \mid \mathsf{xorJoin}((\mathsf{f}_{12}, \mathsf{f}_{22}), \mathsf{f}_{13}) \mid$
$\mathsf{task}(\mathsf{f}_9, \mathsf{Receive_robot_position_assignment}, \mathsf{true}, \mathsf{A}_{2,3}, \mathsf{M}, 0, \varepsilon, \mathsf{f}_{10}) \mid \mathsf{xorSplit}(\mathsf{f}_{14}, (\mathsf{exp}_3, \mathsf{f}_{15}), (\mathsf{exp}_4, \mathsf{f}_{16})) \mid$
$\mathsf{xorSplit}(\mathsf{f}_{10}, (\mathsf{Robot_counter.count} = 3, \mathsf{f}_{12}), (\mathsf{Robot_counter.count} < 3, \mathsf{f}_{11})) \mid \mathsf{xorJoin}((\mathsf{f}_{17}, \mathsf{f}_{16}), \mathsf{f}_{18}) \mid$
$\mathsf{rcvEvent}(\mathsf{f}_{13}, \mathsf{fire_position}: \langle ?\mathsf{Fire_pos.place} \rangle, \mathsf{f}_{14}) \mid \mathsf{task}(\mathsf{f}_{15}, \mathsf{Activate_alarm}, \mathsf{true}, \mathsf{A}_4, \varepsilon, 0, \varepsilon, \mathsf{f}_{17}) \mid$
$\mathsf{task}(\mathsf{f}_{18}, \mathsf{Check_fire_position}, \mathsf{true}, \mathsf{A}_5, \varepsilon, 0, \varepsilon, \mathsf{f}_{19}) \mid \mathsf{task}(\mathsf{f}_{19}, \mathsf{Calculate_closest_robot}, \varepsilon, \mathsf{A}_6, \varepsilon, 0, \varepsilon, \mathsf{f}_{20}) \mid$
$\mathsf{sndEvent}(\mathsf{f}_{20}, \mathsf{presence_fire}: \langle\ \mathsf{Closest_robot.id}, \mathsf{Fire_position.place} \rangle, \mathsf{f}_{21}) \mid \mathsf{andSplit}(\mathsf{f}_{21}, (\mathsf{f}_{22}, \mathsf{f}_{23})) \mid$
$\mathsf{eventBased}(\mathsf{f}_{23}, (\mathsf{request_help}: \langle ?\mathsf{Fire_intervention.place} \rangle, \mathsf{f}_{25}), (\mathsf{fire_extinguished}: \langle \rangle, \mathsf{f}_{28}) \mid$
$\mathsf{task}(\mathsf{f}_{25}, \mathsf{Intervention_by_human}, \mathsf{true}, \mathsf{A}_7, \varepsilon, 0, \varepsilon, \mathsf{f}_{26}) \mid \mathsf{xorJoin}((\mathsf{f}_{26}, \mathsf{f}_{28}), \mathsf{f}_{29}) \mid$
$\mathsf{task}(\mathsf{f}_{29}, \mathsf{Extinguished_fire}, \mathsf{true}, \mathsf{A}_8, \varepsilon, 0, \varepsilon, \mathsf{f}_{30}) \mid \mathsf{task}(\mathsf{f}_{31}, \mathsf{Deactivate_alarm}, \mathsf{true}, \mathsf{A}_9, \varepsilon, 0, \varepsilon, \mathsf{f}_{32}) \mid$
$\mathsf{xorSplit}(\mathsf{f}_{30}, (\mathsf{Fire_counter.count} = 0, \mathsf{f}_{31}), (\mathsf{Fire_counter.count} > 0, \mathsf{f}_{33})) \mid \mathsf{xorJoin}((\mathsf{f}_{32}, \mathsf{f}_{33}), \mathsf{f}_{34}) \mid \mathsf{end}(\mathsf{f}_{34})$
$\mathsf{exp}_3 = \texttt{every i in getAlarms() satisfies i=false}$
$\mathsf{exp}_4 = \texttt{some i in getAlarms() satisfies i=true}$
$\texttt{getAlarms: function() for i in getPlaces() return i.alarm}$
$A_2 = \mathsf{append}(\{\mathsf{id}: \mathsf{robot_position.id},\ \mathsf{place}: \mathsf{robot_position.place}\}, \mathsf{Robot_positions.list})$
$A_3 = \mathsf{assign}(\mathsf{Robot_counter.count} + 1, \mathsf{Robot_counter.count})$
$A_4 = \mathsf{assign}(\mathsf{true}, \mathsf{getPlaces}(), \mathsf{alarm})$
$A_5 = \mathsf{assign}(\mathsf{Fire_counter.count} + 1, \mathsf{Fire_counter.count})$
$A_6 = \mathsf{assign}(\mathsf{Calculate_closest_robot}(\mathsf{Robot_positions}, \mathsf{Fire_position}), \mathsf{Closest_robot.id})$
$A_7 = \mathsf{assign}(\mathsf{false}, \mathsf{meAndNeighbour}(\mathsf{Fire_intervetion.place}), \mathsf{fire})$
$A_8 = \mathsf{assign}(\mathsf{Fire_counter.count} - 1, \mathsf{Fire_counter.count})$
$A_9 = \mathsf{assign}(\mathsf{false}, \mathsf{getPlaces}(), \mathsf{alarm})$

$P_r = \mathsf{start}(\mathsf{f}'_{35}, \mathsf{f}_{35}) \mid \mathsf{task}(\mathsf{f}_{35}, \mathsf{Store_base_position}, \mathsf{true}, \mathsf{A}_{10}, \varepsilon, 0, \varepsilon, \mathsf{f}_{36}) \mid$
$\mathsf{sndEvent}(\mathsf{f}_{36}, \mathsf{robot_position}: \langle \mathsf{myId}(), \mathsf{myPlace}() \rangle, \mathsf{f}_{37}) \mid \mathsf{rcvEvent}(\mathsf{f}_{38}, \mathsf{presence_fire}: \langle\ \mathsf{myId}(), ?\mathsf{Fire.place} \rangle, \mathsf{f}_{39}) \mid$
$\mathsf{xorJoin}((\mathsf{f}_{37}, \mathsf{f}_{46}), \mathsf{f}_{38}) \mid \mathsf{taskM}(\mathsf{f}_{39}, \mathsf{Move_to_fire\ position}, \mathsf{true}, \varepsilon, \varepsilon, \mathsf{Fire.place}, \mathsf{bTimerEvent}(\mathsf{t1}, 5, \mathsf{f}_{43}), \mathsf{f}_{40}) \mid$
$\mathsf{sndEvent}(\mathsf{f}_{43}, \mathsf{request_help}: \mathsf{Fire.place}, \mathsf{f}_{44}) \mid \mathsf{task}(\mathsf{f}_{40}, \mathsf{Extinguish_fire}, \mathsf{true}, \mathsf{A}_{11}, \varepsilon, 4, \varepsilon, \mathsf{f}_{41}) \mid$
$\mathsf{sndEvent}(\mathsf{f}_{41}, \mathsf{fire_extinguished}: \langle \rangle, \mathsf{f}_{42}) \mid \mathsf{xorJoin}((\mathsf{f}_{42}, \mathsf{f}_{44}), \mathsf{f}_{45}) \mid$
$\mathsf{taskM}(\mathsf{f}_{45}, \mathsf{Move_to_base\ station}, \mathsf{true}, \varepsilon, \varepsilon, \mathsf{Base_station.place}, \varepsilon, \mathsf{f}_{46})$
$A_{10} = \mathsf{assign}(\mathsf{myPlace}(), \mathsf{Base_station.place})$
$A_{11} = \mathsf{assign}(\mathsf{false}, \mathsf{meAndNeighbour}(\mathsf{Fire.place}), \mathsf{fire})$

$G_d = (P_{ld}, E_d) \quad P_{ld} = \{\mathsf{pl}_j \mid 1 \le j \le 44\} \quad E_d = \{(\mathsf{pl}_1, \mathsf{pl}_2), (\mathsf{pl}_1, \mathsf{pl}_3), (\mathsf{pl}_1, \mathsf{pl}_4), (\mathsf{pl}_2, \mathsf{pl}_1), (\mathsf{pl}_2, \mathsf{pl}_3), (\mathsf{pl}_2, \mathsf{pl}_4), \ldots\}$

Fig. 10. Textual representation of the running example.

attributes; σ^0_{apl} denotes the state where all place attributes are set to null or the initial value provided by the modeler.

- $\sigma_s : \mathbb{T} \to \{enab, dis\}$ is a *task and timer state function* (where $\mathbb{T}$ is the set of task and timer names) used to keep track of the number of active task instances and timers; the status of a task/timer depends on its evolution from enabled (*enab*) to disabled (*dis*) states and vice versa; σ^0_s denotes the state where all tasks/timers are disabled.
- $\sigma_p : \mathbb{P} \times \mathbb{N} \to P_l$ is the *place state function* (where $\mathbb{P}$ is the set of pool names) mapping each pool's process instance to a place; σ^0_p denotes the state where all places are empty except for those containing the initial positions of the instances provided by the modeler.
- $\sigma_m : \mathbb{M} \to 2^{\mathbb{V}^n}$ is a *message state function* (where $\mathbb{M}$ is the set of message flows) that assigns to each message flow name m a list of values representing the messages received along m; σ^0_m denotes the state where all message multisets are empty.
- $\sigma_t : \mathbb{T} \to \mathbb{N}$ is a *timer state function* specifying for each timer the time units to wait until it expires; σ^0_t denotes the state where all timers are set to $undef$.
- $\sigma_i : \mathbb{P} \times \mathbb{N} \to \mathbb{S}_{\sigma_f} \times \mathbb{S}_{\sigma_{ado}} \times \mathbb{S}_{\sigma_s} \times \mathbb{S}_{\sigma_t}$ is an *instance state function* (where $\mathbb{S}_\sigma$ is the set of states of kind σ) mapping each pool's process instance to a quadruple of the form $\langle \sigma_f, \sigma_{ado}, \sigma_s, \sigma_t \rangle$; σ^0_i denotes the state mapping each instance to the tuple of the corresponding initial states.

Therefore, we denote with $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma^0_f, \sigma^0_{ado}, \sigma^0_{apl}, \sigma^0_s, \sigma^0_p, \sigma^0_m, \sigma^0_t \rangle$ (resp. $\langle C, G, \sigma^0_i, \sigma^0_{apl}, \sigma^0_p, \sigma^0_m \rangle$) the initial process (resp. collaboration) configuration.

The state obtained by updating in σ_f the number of tokens of the sequence flow f to n, written as $\sigma_f \cdot [\mathsf{f} \mapsto \mathsf{n}]$, is defined as follows: $(\sigma_f \cdot [\mathsf{f} \mapsto \mathsf{n}])(\mathsf{f}')$ returns n if $\mathsf{f}' = \mathsf{f}$, otherwise it returns $\sigma_f(\mathsf{f}')$. The updates of the other state functions are defined similarly. It is worth noting that the definition of two distinct state functions, one for data object attributes and one for place attributes, stems from the need to keep the environment separate from the process model, thus enabling their reuse in other scenarios.

The operational semantic is defined employing *Labeled Transition Systems* (LTSs). The LTS of the process behavior is a triple $\langle \mathscr{P}, \mathscr{L}, \to \rangle$ where: $\mathscr{P}$ is a set of processes configurations; $\mathscr{L}$, ranged over by λ, is a set of labels; and $\to \subseteq \mathscr{P} \times \mathscr{L} \times \mathscr{P}$ is a transition relation. As a simplification, $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle \xrightarrow{\lambda} \langle P, G, \mathsf{p}, \mathsf{j}, \sigma'_f, \sigma'_{ado}, \sigma'_{apl}, \sigma'_s, \sigma'_p, \sigma'_m, \sigma'_t \rangle$ indicates that $(\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle, \lambda, \langle P, G, \mathsf{p}, \mathsf{j}, \sigma'_f, \sigma'_{ado}, \sigma'_{apl}, \sigma'_s, \sigma'_p, \sigma'_m, \sigma'_t \rangle) \in \to$, and says that the process in the configuration $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle$ can do a transition labeled by λ and evolve to the process configuration $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma'_f, \sigma'_{ado}, \sigma'_{apl}, \sigma'_s, \sigma'_p, \sigma'_m, \sigma'_t \rangle$. The LTS describing the behavior of environmental BPMN collaborations is a triple $\langle \mathscr{C}, \mathscr{L}, \to \rangle$ where: $\mathscr{C}$ is a set of collaboration configurations, and $\to \subseteq \mathscr{C} \times \mathscr{L} \times \mathscr{C}$ is a transition relation. The labels used by the transition relations $\to$ are generated by the grammar $\lambda ::= \checkmark \mid \tau$, where $\checkmark$ denotes the passing of a unit of time (due to, e.g., movements in the place graph), and τ denotes a silent transition (due to, e.g., token flow in the processes).

Function $inc(\sigma_f, \mathsf{f}) = \sigma_f \cdot [\mathsf{f} \mapsto \sigma_f(\mathsf{f}) + 1]$, resp. $dec(\sigma_f, \mathsf{f}) = \sigma_f \cdot [\mathsf{f} \mapsto \sigma_f(\mathsf{f}) - 1]$, increments, resp. decrements, by one the number of tokens marking the sequence flow f in the state σ_f. They extend in a natural way to sets F of sequence flows; they are inductively defined as follows: $inc(\sigma_f, \emptyset) = dec(\sigma_f, \emptyset) = \sigma_f$, $inc(\sigma_f, \{\mathsf{f}\} \uplus F) = inc(inc(\sigma_f, \mathsf{f}), F)$, $dec(\sigma_f, \{\mathsf{f}\} \uplus F) = dec(dec(\sigma_f, \mathsf{f}), F)$. Function $dec(\sigma_t, N)$, which decrements the timer value for each timer in N, is defined similarly.

Function $enab(\sigma_s, \mathsf{n}) = \sigma_s \cdot [\mathsf{n} \mapsto enab]$, resp. $dis(\sigma_s, \mathsf{n}) = \sigma_s \cdot [\mathsf{n} \mapsto dis]$, activates, resp. deactivates, the task/timer n in the state σ_s.

Function $eval(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{exp}) = \mathsf{v}$ states that v is the value resulting from the evaluation of the expression exp on the data and place attributes in σ_{ado} and σ_{apl} and on the place state σ_p, considering that the expression is evaluated for the instance j of the pool p. The latter information permits basing the evaluation on the current position of the instance, i.e., the place $\mathsf{pl}{=}\sigma_p(\mathsf{p}, \mathsf{j})$. The function *eval* naturally extends to the evaluation of tuples of expressions. Moreover, we assume two particular applications of the eval function in the case the expression exp is one of the special attribute names myId and $\mathsf{myPlace}$. Specifically, $eval(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{myId}) = getInstance(p, j)$ and $eval(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{myPlace}) = getPlace(p, j)$. The evaluation of a formal field always returns the formal field itself. For the sake of presentation, we omit the state functions, p, and j when they are clear from the context, thus writing $eval(\mathsf{exp})$. Notably, the definition of the *eval* function depends on the used expression language; in this paper we assume to use the language FEEL, hence, *eval* relies on the standard evaluation of FEEL.

Function $updAdo(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, A)$, resp. $updApl(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, A)$, performs the attribute assignments/appends A in the state σ_{ado}, resp. σ_{apl}. The relevant cases for the inductive definition of *updAdo* are as follows (the other cases are straightforward):
- $updAdo(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{assign}(\mathsf{exp}_f, \mathsf{do.a})) = \sigma_{ado} \cdot [\mathsf{do.a} \mapsto \mathsf{v}]$ if $eval(\mathsf{exp}_f) = \mathsf{v}$; otherwise it returns σ_{ado};
- $updAdo(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{assign}(\mathsf{exp}_f, \mathsf{pl.a})) = \sigma_{ado}$;
- $updAdo(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{assign}(\mathsf{exp}_f, \mathsf{exp}_t, \mathsf{a})) = \sigma_{ado} \cdot [\mathsf{do}_1\mathsf{.a} \mapsto \mathsf{v}] \cdot \ldots \cdot [\mathsf{do}_n\mathsf{.a} \mapsto \mathsf{v}]$ if $eval(\mathsf{exp}_f) = \mathsf{v}$, $eval(\mathsf{exp}_t) = \{\mathsf{do}_1, \ldots, \mathsf{do}_n\}$; otherwise it returns σ_{ado};
- $updAdo(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{append}(\mathsf{exp}_f, \mathsf{do.a})) = \sigma_{ado} \cdot [\mathsf{do.a} \mapsto \langle \sigma_{ado}(\mathsf{do.a}), \mathsf{v} \rangle]$ if $eval(\mathsf{exp}_f) = \mathsf{v}$.

The definition of the function *updApl* is similar. For instance, $updApl(\sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{assign}(\mathsf{exp}_f, \mathsf{exp}_t, \mathsf{a})) = \sigma_{apl} \cdot [\mathsf{pl.a} \mapsto \mathsf{v}] \cdot \ldots \cdot [\mathsf{pl}_n\mathsf{.a} \mapsto \mathsf{v}]$ if $eval(\mathsf{exp}_f) = \mathsf{v}$, $eval(\mathsf{exp}_t) = \{\mathsf{pl}_1, \ldots, \mathsf{pl}_n\}$; otherwise it returns σ_{apl}.
For the sake of presentation, we omit the terms clear from the context, thus writing $updAdo(A)$, resp. $updApl(A)$.

Function $updI(\sigma_i, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_s, \sigma_t) = \sigma_i \cdot [(\mathsf{p}, \mathsf{j}) \mapsto \langle \sigma_f, \sigma_{ado}, \sigma_s, \sigma_t \rangle]$ replaces an existing instance j of the pool p in the state σ_i.

Function $mod(G, M)$ performs the modifications M in the place graph G. The relevant cases of the definition are as follows: $mod((P_l, E), \mathsf{connect(e)}) = (P_l, E \cup \{\mathsf{e}\})$, resp. $mod((P_l, E), \mathsf{disconnect(e)}) = (P_l, E \setminus \{\mathsf{e}\})$.

Function $next(G, \sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{pl})$ returns the set of places that are next in the paths in G from the current place $\sigma_p(\mathsf{p}, \mathsf{j})$ of the instance to the destination pl; the function returns $\emptyset$ when there is no path to the destination. Notably, different definitions can be provided for this function to specify different navigation strategies; in this paper, the *next* function computes the shortest paths.

Function $move(\sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{pl}) = \sigma_p \cdot [(\mathsf{p}, \mathsf{j}) \mapsto \mathsf{pl}]$ assigns to the instance j of the pool p the place pl in the state σ_p.

Function $set(\sigma_t, \mathsf{n}, \mathsf{k}) = \sigma_t \cdot [\mathsf{n} \mapsto \mathsf{k}]$ sets the timer identified by the name n to a natural number k.

Function $reset(\sigma_t, \mathsf{n}) = \sigma_t \cdot [\mathsf{n} \mapsto undef]$ resets the timer identified by the name n.

Function $add(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}) = \sigma_m \cdot [\mathsf{m} \mapsto \sigma_m(\mathsf{m}) \uplus \{\tilde{\mathsf{v}}\}]$ adds the tuple of values $\tilde{\mathsf{v}}$ for the message flow m in the state σ_m.

Function $rm(\sigma_m, \mathsf{m}, \tilde{\mathsf{v}}) = \sigma_m \cdot [\mathsf{m} \mapsto \sigma_m(\mathsf{m}) \setminus \{\tilde{\mathsf{v}}\}]$ removes the tuple of values $\tilde{\mathsf{v}}$ for message m from the state σ_m.

Function $match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$ performs pattern-matching on structured data thus determining if an evaluated template $\tilde{\mathsf{et}}$ matches a tuple of values $\tilde{\mathsf{v}}$. A successful matching returns a list of assignments updating the formal fields in the template; otherwise, the function is undefined. Function *match* is inductively defined as follows:
– $match(\mathsf{v}, \mathsf{v}) = \varepsilon$;
– $match(?\mathsf{a}, \mathsf{v}) = \mathsf{assign}(\mathsf{v}, \mathsf{a})$;
– $match((\mathsf{et}', \tilde{\mathsf{et}}), (\mathsf{v}', \tilde{\mathsf{v}})) = match(\mathsf{et}', \mathsf{v}')\ match(\tilde{\mathsf{et}}, \tilde{\mathsf{v}})$.

Function $getTimer(B)$ returns the set of timer names in B. The relevant cases of the definition are as follows: $getTimer(\mathsf{bCondEvent(exp, f)}) = \emptyset$ and $getTimer(\mathsf{bTimerEvent(n, k, f)}) = \{\mathsf{n}\}$.

Function $triggered(\sigma_t, \sigma_{ado}, \sigma_{apl}, \sigma_p, \mathsf{p}, \mathsf{j}, B)$ returns the set of sequence flows of the active boundary events in B. For timer boundary events, we have: $triggered(\sigma_t, \sigma_{ado}, \sigma_{apl}\sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{bTimerEvent(n, k, f)}) = \{\mathsf{f}\}$ if $\sigma_t(\mathsf{n}) = 0$, otherwise $\emptyset$; while for conditional boundary events we have: $triggered(\sigma_t, \sigma_{ado}, \sigma_{apl}\sigma_p, \mathsf{p}, \mathsf{j}, \mathsf{bCondEvent(exp, f)}) = \{\mathsf{f}\}$ if $eval(\mathsf{exp}) = true$, otherwise $\emptyset$. For the sake of presentation, we omit the terms clear from the context, thus writing: $triggered(B)$.

Function $getD(\mathsf{d})$ returns the time units corresponding to the duration specification d. Specifically, since d is considered as couple $(\mathsf{k}_{min}, \mathsf{k}_{max})$, $getD(\mathsf{d})$ returns a uniformly distributed natural number in the interval $[\mathsf{k}_{min}, \mathsf{k}_{max}]$.

Function $start(\sigma_t, B)$ sets the timers of the boundary events at their initial values. In case of timer boundary events we have: $start(\sigma_t, \mathsf{bTimerEvent(n, k, f)}) = set(\sigma_t, \mathsf{n}, \mathsf{k})$; instead for conditional boundary events we have $start(\sigma_t, \mathsf{bCondEvent(exp, f)}) = \sigma_t$.

Function $notTimedEl(P)$ returns $false$ when P is a timer event, a task, a movement task, or an event-based gateway with timer events, i.e., $notTimedEl(\mathsf{timerEvent(f, n, k, f')}) = notTimedEl(\mathsf{task(f, n, exp}_g, A, M, \mathsf{d}, B, \mathsf{f}')) = notTimedEl(\mathsf{taskM(f, n, exp}_g, A, M, \mathsf{exp}_d, B, \mathsf{f}')) = notTimedEl(\mathsf{eventBased(f, ..., (n, k, f')}))) = false$. For all other elements, this function returns $true$. Finally $notTimedEl(P_1 \mid P_2) = notTimedEl(P_1) \wedge notTimedEl(P_2)$.

Fig. 11. Auxiliary functions and relations.

The semantics we propose follows a Structural Operational Semantics [28] approach, hence it is a 'small-step semantics' that formalizes how the individual steps of an environmental BPMN collaboration take place. Therefore, the application of one rule at process level will lead to infer one single transition in the LTS describing the behavior of the environmental BPMN collaboration.

To improve the readability of the operational rules, instead of the extended notation we adopt the following simplifications.

(i) We omit, in the source configuration of the transition, the place graph, the instance information, and the states functions; and

(ii) we omit, in the target configuration, the process structure, the place graph, the instance information, and the states not affected by the transition.

For instance, the transition $\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle \xrightarrow{\lambda} \langle P, G, \mathsf{p}, \mathsf{j}, \sigma'_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t \rangle$ will be written as $P \xrightarrow{\lambda} \langle \sigma'_f \rangle$ when the operational rule simply affects the sequence flow state function. The same simplification holds for collaboration configurations. Moreover, to simplify the definition of the operational rules, in Fig. 11 we define *auxiliary functions and relations* that update states of process configurations and act on the place graph topology.

$$\mathsf{start}(\mathsf{f},\mathsf{f}') \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}')\rangle \quad \sigma_f(\mathsf{f}) > 0 \quad (P\text{-}Start)$$

$$\mathsf{end}(\mathsf{f}) \xrightarrow{\tau} \langle dec(\sigma_f,\mathsf{f})\rangle \quad \sigma_f(\mathsf{f}) > 0 \quad (P\text{-}End)$$

$$\mathsf{rcvEvent}(\mathsf{f},\mathsf{m}:\tilde{\mathsf{t}},\mathsf{f}') \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}'),updAdo(A),updApl(A),rm(\sigma_m,\mathsf{m},\tilde{\mathsf{v}})\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge \tilde{\mathsf{v}} \in \sigma_m(\mathsf{m}) \wedge eval(\tilde{\mathsf{t}}) = \tilde{\mathsf{et}} \wedge match(\tilde{\mathsf{et}},\tilde{\mathsf{v}}) = A \quad (P\text{-}Rcv)$$

$$\mathsf{condEvent}(\mathsf{f},\mathsf{exp},\mathsf{f}') \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}')\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge eval(\mathsf{exp}) = true \quad (P\text{-}Cond)$$

$$\mathsf{sndEvent}(\mathsf{f},\mathsf{m}:\tilde{\mathsf{exp}},\mathsf{f}') \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}'),add(\sigma_m,\mathsf{m},\tilde{\mathsf{v}})\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge eval(\tilde{\mathsf{exp}}) = \tilde{\mathsf{v}} \quad (P\text{-}Snd)$$

$$\mathsf{timerEvent}(\mathsf{f},\mathsf{n},\mathsf{k},\mathsf{f}') \xrightarrow{\tau} \langle dec(\sigma_f,\mathsf{f}),enab(\sigma_s,\mathsf{n}),set(\sigma_t,\mathsf{n},\mathsf{k})\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge \sigma_s(\mathsf{n}) = dis \quad (P\text{-}Timer_1)$$

$$\mathsf{timerEvent}(\mathsf{f},\mathsf{n},\mathsf{k},\mathsf{f}') \xrightarrow{\checkmark} \langle dec(\sigma_t,\mathsf{n})\rangle \quad \sigma_s(\mathsf{n}) = enab \wedge \sigma_t(\mathsf{n}) > 0 \quad (P\text{-}Timer_2)$$

$$\mathsf{timerEvent}(\mathsf{f},\mathsf{n},\mathsf{k},\mathsf{f}') \xrightarrow{\checkmark} \langle\rangle \quad \sigma_s(\mathsf{n}) = dis \quad (P\text{-}Timer_3)$$

$$\mathsf{timerEvent}(\mathsf{f},\mathsf{n},\mathsf{k},\mathsf{f}') \xrightarrow{\tau} \langle inc(\sigma_f,\mathsf{f}'),dis(\sigma_s,\mathsf{n}),reset(\sigma_t,\mathsf{n})\rangle \quad \sigma_s(\mathsf{n}) = enab \wedge \sigma_t(\mathsf{n}) = 0 \quad (P\text{-}Timer_4)$$

$$\mathsf{andSplit}(\mathsf{f},F) \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),F)\rangle \quad \sigma_f(\mathsf{f}) > 0 \quad (P\text{-}AndSplit)$$

$$\mathsf{xorSplit}(\mathsf{f},(\mathsf{exp}_1,\mathsf{f}_1)\ldots(\mathsf{exp}_n,\mathsf{f}_n)) \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}')\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge \exists i \in [1,n]\ s.t.\ eval(\mathsf{exp}_i) = true \wedge \forall j \in [1,i),\ eval(\mathsf{exp}_j) = false \quad (P\text{-}XorSplit)$$

$$\mathsf{andJoin}(F,\mathsf{f}) \xrightarrow{\tau} \langle inc(dec(\sigma_f,F),\mathsf{f})\rangle \quad \forall \mathsf{f}' \in F\ .\ \sigma_f(\mathsf{f}') > 0 \quad (P\text{-}AndJoin)$$

$$\mathsf{xorJoin}(\{\mathsf{f}\}\cup F,\mathsf{f}') \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}')\rangle \quad \sigma_f(\mathsf{f}) > 0 \quad (P\text{-}XorJoin)$$

$$\mathsf{eventBased}(\mathsf{f},(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{f}_1)^*,(\mathsf{exp}',\mathsf{f}')(\mathsf{exp},\mathsf{f}_2)^*) \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}')\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge eval(\mathsf{exp}') = true \quad (P\text{-}EventB_C)$$

$$\mathsf{eventBased}(\mathsf{f},(\mathsf{m}':\tilde{\mathsf{t}},\mathsf{f}')(\mathsf{m}:\tilde{\mathsf{t}},\mathsf{f}_1)^*,(\mathsf{exp},\mathsf{f}_2)^*) \xrightarrow{\tau} \langle inc(dec(\sigma_f,\mathsf{f}),\mathsf{f}'),updAdo(A),updApl(A),rm(\sigma_m,\mathsf{m}',\tilde{\mathsf{v}})\rangle \quad \sigma_f(\mathsf{f}) > 0 \wedge \tilde{\mathsf{v}} \in \sigma_m(\mathsf{m}') \wedge eval(\tilde{\mathsf{t}}) = \tilde{\mathsf{et}} \wedge match(\tilde{\mathsf{et}},\tilde{\mathsf{v}}) = A \quad (P\text{-}EventB_M)$$

Fig. 12. BPMN process semantics: events and gateways.

The operational rules defining the transition relation of the process semantics are given by the inference rules in Figs. 12, and 13. We now briefly comment on the rules dealing with events and gateways in Fig. 12.

- Rule *P-Start* starts the execution of a process when it has been activated. To denote the enabled status of start events we have included in their syntactical definition an incoming (spurious) sequence flow, named enabling sequence flow. Thus, the process is activated when the enabling sequence flow of a start event is marked. The effect of the rule is to increment the number of tokens in the sequence flow outgoing from the start event and to decrease the marking of the enabling sequence flow. To show how the auxiliary functions and relations update process configurations, we exemplify their outcome in the application of rule *P-Start*, without applying the transition simplifications. Let us assume that $\sigma_f(\mathsf{f})$ is 1 for $\mathsf{f} = \mathsf{f}_1$ and 0 for any other sequence flow f; thus $\langle \mathsf{start}(\mathsf{f}_1,\mathsf{f}_2), G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t\rangle \xrightarrow{\tau} \langle \mathsf{start}(\mathsf{f}_1,\mathsf{f}_2), G, \mathsf{p}, \mathsf{j}, \sigma'_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t\rangle$ where $\sigma'_f(\mathsf{f})$ is 1 for $\mathsf{f} = \mathsf{f}_2$ and 0 for any other f. Indeed, $dec(\sigma_f,\mathsf{f}_1) = \sigma_f \cdot [\mathsf{f}_1 \mapsto \sigma_f(\mathsf{f}_1) - 1] = \sigma_f \cdot [\mathsf{f}_1 \mapsto 0] = \sigma''_f$ and, hence, $\sigma'_f = inc(dec(\sigma_f,\mathsf{f}_1),\mathsf{f}_2) = inc(\sigma''_f,\mathsf{f}_2) = \sigma''_f \cdot [\mathsf{f}_2 \mapsto \sigma''_f(\mathsf{f}_2) + 1] = \sigma''_f \cdot [\mathsf{f}_2 \mapsto 1]$; thus, $\sigma'_f = \sigma_f \cdot [\mathsf{f}_1 \mapsto 0] \cdot [\mathsf{f}_2 \mapsto 1]$.

$$\mathsf{task}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\tau} \langle dec(\sigma_f,\mathsf{f}), enab(\sigma_s,\mathsf{n}), set(start(\sigma_t,B),\mathsf{n},getD(\mathsf{d}))\rangle \quad \sigma_f(\mathsf{f})>0 \wedge \sigma_s(\mathsf{n})=dis \wedge eval(\mathsf{exp}_g)=true \quad (P\text{-}Task_{Act})$$

$$\mathsf{task}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\checkmark} \langle dec(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B)\cup\{\mathsf{n}\} \wedge \forall \mathsf{n}'\in N\,.\,\sigma_t(\mathsf{n}')>0 \wedge triggered(B)=\emptyset \quad (P\text{-}Task_{Elps1})$$

$$\mathsf{task}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\checkmark} \langle\rangle \quad \sigma_s(\mathsf{n})=dis \quad (P\text{-}Task_{Elps2})$$

$$\mathsf{task}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\tau} \langle inc(\sigma_f,\mathsf{f}^b), dis(\sigma_s,\mathsf{n}), reset(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B)\cup\{\mathsf{n}\} \wedge \mathsf{f}^b\in triggered(B) \quad (P\text{-}Task_{B})$$

$$\mathsf{task}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\tau} \langle mod(G,M), inc(\sigma_f,\mathsf{f}'), upd(\sigma_{ado},A), upd(\sigma_{apl},A), dis(\sigma_s,\mathsf{n}), reset(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B)\cup\{\mathsf{n}\} \wedge \sigma_t(\mathsf{n})=0 \quad (P\text{-}Task_{C})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{exp}_d,B,\mathsf{f}') \xrightarrow{\tau} \langle dec(\sigma_f,\mathsf{f}), enab(\sigma_s,\mathsf{n}), start(\sigma_t,B)\rangle \quad \sigma_f(\mathsf{f})>0 \wedge \sigma_s(\mathsf{n})=dis \wedge eval(\mathsf{exp}_g)=true \quad (P\text{-}TaskM_{Act})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{exp}_d,B,\mathsf{f}') \xrightarrow{\checkmark} \langle move(\sigma_p,\mathsf{p},\mathsf{j},\mathsf{pl}), dec(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B) \wedge \forall \mathsf{n}'\in N\,.\,\sigma_t(\mathsf{n}')>0 \wedge triggered(B)=\emptyset \wedge \mathsf{pl}\in next(G,\sigma_p,\mathsf{p},\mathsf{j},eval(\mathsf{exp}_d)) \quad (P\text{-}TaskM_{Elps1})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{exp}_d,B,\mathsf{f}') \xrightarrow{\checkmark} \langle\rangle \quad \sigma_s(\mathsf{n})=dis \quad (P\text{-}TaskM_{Elps2})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{exp}_d,B,\mathsf{f}') \xrightarrow{\checkmark} \langle dec(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B) \wedge \forall \mathsf{n}'\in N\,.\,\sigma_t(\mathsf{n}')>0 \wedge triggered(B)=\emptyset \wedge next(G,\sigma_p,\mathsf{p},\mathsf{j},eval(\mathsf{exp}_d))=\emptyset \wedge \sigma_p(\mathsf{p},\mathsf{j})\neq eval(\mathsf{exp}_d) \quad (P\text{-}TaskM_{Elps3})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{exp}_d,B,\mathsf{f}') \xrightarrow{\tau} \langle inc(\sigma_f,\mathsf{f}^b), dis(\sigma_s,\mathsf{n}), reset(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B) \wedge \mathsf{f}^b\in triggered(B) \quad (P\text{-}TaskM_{B})$$

$$\mathsf{taskM}(\mathsf{f},\mathsf{n},\mathsf{exp}_g,A,M,\mathsf{d},B,\mathsf{f}') \xrightarrow{\tau} \langle mod(G,M), inc(\sigma_f,\mathsf{f}'), upd(\sigma_{ado},A), upd(\sigma_{apl},A), dis(\sigma_s,\mathsf{n}), reset(\sigma_t,N)\rangle \quad \sigma_s(\mathsf{n})=enab \wedge N=getTimer(B) \wedge \sigma_p(\mathsf{p},\mathsf{j})=eval(\mathsf{exp}_d) \quad (P\text{-}TaskM_{C})$$

Fig. 13. BPMN process semantic: tasks and movement tasks.

- Rule *P-End* is enabled when there is at least one token in the incoming sequence flow of the end event, which is then consumed.
- Rule *P-Rcv* is enabled when there is at least a token in the incoming sequence flow and there is a message in the queue that matches the evaluated template. As a result, the rule moves the token that has enabled its application (i.e., it decrements the number of tokens in the incoming sequence flow and increments the number of tokens in the outgoing one), possibly updates the data object and places attributes with the received values, and removes the first matching message from the queue [29].
- Rule *P-Cond* is enabled when there is at least a token in the incoming sequence flow and when the conditional expression is evaluated as true. As a result, the token moves from the incoming sequence flow to the outgoing one.
- Rule *P-Snd* is enabled when there is at least a token in the incoming sequence flow; as a result, the tuple of values resulting from the evaluation of the message expressions is added to the message queue and the token is moved.
- The behavior of the timer event is defined through a set of four rules, each one captures a specific stage in the timer lifecycle during the execution of the process. Rule $P\text{-}Timer_1$ is enabled when there is at least a token in the incoming sequence flow and the timer is disabled. As a result, it decrements the token from the incoming sequence flow, enables the timer, and sets the timer delay to the specified value k. Rule $P\text{-}Timer_2$ is enabled when the timer is enabled and not expired. As a result, the delay of the timer is decreased by one. Rule $P\text{-}Timer_3$ allows the time to elapse when the timer is disabled. In this case the empty tuple $\langle\rangle$ indicates that the state functions remains unchanged while time

$$P \xrightarrow{\checkmark} \langle\rangle \quad notTimedEl(P) = true \quad (P\text{-}Elps)$$

$$\frac{P_1 \xrightarrow{\tau} \langle\sigma\rangle}{P_1 \mid P_2 \xrightarrow{\tau} \langle\sigma\rangle} \ (P\text{-}\tau) \qquad \frac{P_1 \xrightarrow{\checkmark} \langle\sigma'\rangle \quad \langle P_2, \sigma'\rangle \xrightarrow{\checkmark} \langle\sigma''\rangle \quad P_1 \mid P_2 \xrightarrow{\tau}}{P_1 \mid P_2 \xrightarrow{\checkmark} \langle\sigma''\rangle} \ (P\text{-}\checkmark)$$

$$\frac{\langle P, G, \mathsf{p}, \mathsf{j}, \sigma_f, \sigma_{ado}, \sigma_{apl}, \sigma_s, \sigma_p, \sigma_m, \sigma_t\rangle \xrightarrow{\lambda} \langle G', \sigma'_f, \sigma'_{ado}, \sigma'_{apl}, \sigma'_s, \sigma'_p, \sigma'_m, \sigma'_t\rangle}{\mathsf{pool}(\mathsf{p}, P, \mathsf{k}) \xrightarrow{\lambda} \langle G', updI(\sigma_i, \mathsf{p}, \mathsf{j}, \langle\sigma'_f, \sigma'_{ado}, \sigma'_s, \sigma'_t\rangle), \sigma'_{apl}, \sigma'_p, \sigma'_m\rangle} \ \begin{array}{l} 1 \le \mathsf{j} \le \mathsf{k} \wedge \\ \sigma_i(\mathsf{p}, \mathsf{j}) = \langle\sigma_f, \sigma_{ado}, \sigma_s, \sigma_t\rangle \end{array} \ (C\text{-}Int)$$

$$\frac{C_1 \xrightarrow{\tau} \langle\sigma\rangle}{C_1 \parallel C_2 \xrightarrow{\tau} \langle\sigma\rangle} \ (C\text{-}\tau) \qquad \frac{C_1 \xrightarrow{\checkmark} \langle\sigma'\rangle \quad \langle C_2, \sigma'\rangle \xrightarrow{\checkmark} \langle\sigma''\rangle \quad C_1 \parallel C_2 \xrightarrow{\tau}}{C_1 \parallel C_2 \xrightarrow{\checkmark} \langle\sigma''\rangle} \ (C\text{-}\checkmark)$$

Fig. 14. BPMN process and collaboration semantic: interleaving.

progresses. Rule $P\text{-}Timer_4$ is enabled when the timer expires. As a result, it increments the token in the outgoing sequence flow, and disables and resets the timer. It is worth noting that, to ensure a clearer interpretation of the timer, we assign the value $undef$ to a reset timer and the value 0 to an expired timer, as defined in Fig. 11.

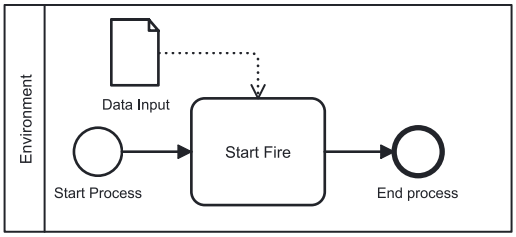
- Rule *P-AndSplit* is enabled when there is at least one token in the incoming sequence flow of an AND split gateway. As a result of its application, the rule decrements the number of tokens in the incoming sequence flow, and increments the tokens in each outgoing sequence flow.
- Rule *P-XorSplit* is enabled when a token is available in the incoming sequence flow of an XOR split gateway and the conditional expression of at least one of its outgoing sequence flows is evaluated to true. If more flows have their condition satisfied, the first one is selected. As a result, the rule decrements the token in the incoming sequence flow and increments the token in the selected outgoing sequence flow.
- Rule *P-AndJoin* is enabled when there is a token in all incoming sequence flows of an AND split gateway. As a result, the rule decrements the number of tokens in the incoming sequence flows, and increments the token in the outgoing sequence flow.
- Rule *P-XorJoin* is enabled every time there is a token in one of the incoming sequence flows, which is then moved to the outgoing sequence flow.
- Rules $P\text{-}EventB_C$ and $P\text{-}EventB_M$ are enabled when there is a token in the incoming sequence flow of an event-based gateway, and respectively a conditional expression is evaluated to true or a matching tuple of values is present in a message queue. The application of the rules move the token from the incoming sequence flow to the outgoing sequence flow corresponding to the true condition ($P\text{-}EventB_C$) or the triggered message event ($P\text{-}EventB_M$). In the latter case, the involved data object and place attributes are updated, and the message tuple is removed from the queue. Note that, in the notations $(\mathsf{m}' : \tilde{\mathsf{t}}, \mathsf{f}')(\mathsf{m} : \tilde{\mathsf{t}}, \mathsf{f}_1)^*$, and $(\mathsf{exp}', \mathsf{f}')(\mathsf{exp}, \mathsf{f}_2)^*$ the pairs $(\mathsf{m}' : \tilde{\mathsf{t}}, \mathsf{f}')$ and $(\mathsf{exp}', \mathsf{f}')$ refer to an arbitrary outgoing sequence flow that triggers the gateway.

Rules in Fig. 13 deal with task, environmental task, and movement task execution; notice that the rules labeled as *P-Task* are valid for both tasks and environmental tasks, while those labeled *P-TaskM* specifically refer to movement tasks. We comment on them.

- Rules $P\text{-}Task_{Act}$ and $P\text{-}TaskM_{Act}$ activate tasks, environmental tasks, and movement tasks when they are disabled, there is a token in the incoming sequence flow, and the guard is satisfied. As result, they remove the token in the incoming sequence flow, enable the tasks, and start the timer of the task (not for the movement task) and the possible timer of the attached boundary event.
- Rule $P\text{-}Task_{Elps1}$ decrements the delay time from the timer of the task when it is enabled, no boundary events are triggered, and the timer is not expired.
- Rules $P\text{-}Task_{Elps2}$ and $P\text{-}TaskM_{Elps2}$ enable tasks, environmental tasks, and movement tasks to evolve with temporal transitions when they are disabled.
- Rules $P\text{-}Task_B$ and $P\text{-}TaskM_B$ deal with the boundary events attached to the tasks, environmental tasks and movement tasks. If they are enabled and one of the boundary events is triggered, the rules add a token in the outgoing sequence flow of the triggered boundary event, disable the task, and reset the timer.
- Rule $P\text{-}Task_C$ deals with the completion of tasks and environmental tasks. Such tasks complete when their timer expires; as result, the rule possibly updates the place graph, increments the outgoing sequence flow, performs the assignments, disables the task, and resets the timer.
- Rule $P\text{-}TaskM_{Elps1}$, when a movement task is enabled, no boundary event is triggered, and the next place (towards the specified destination) is reachable, performs a movement for the participant that is executing the task to the next place, and decrements the delay time from the boundary event timer.
- Rule $P\text{-}TaskM_{Elps3}$ decrements the delay time from the boundary event timer of the movement task when it is enabled, no boundary event is triggered, the timer is not expired, and the next place is not reachable and is not the destination.
- Rule $P\text{-}TaskM_C$ deals with the completion of the movement task. If the task is enabled and the destination has been reached, the rule increments the outgoing sequence flow, performs the assignments, disables the task, and resets the timer.

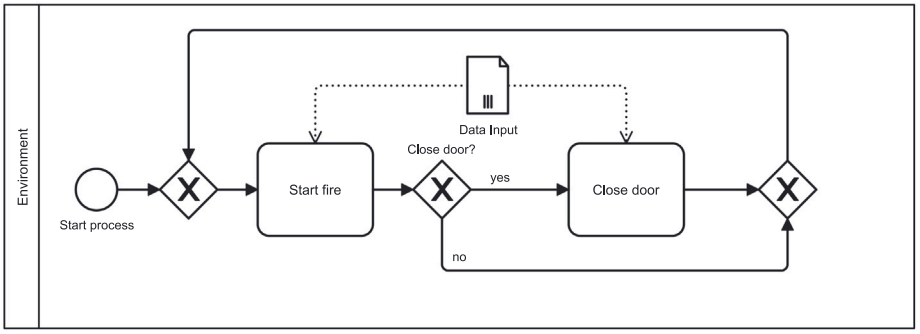
The operational rules dealing with the interleaving of process and collaboration elements are reported in Fig. 14. We briefly comment on them.

- Rule *P-Elps* enables non-timed process elements to evolve the process configuration with temporal transitions, without changing its state functions.
- Rule $P\text{-}\tau$ permits to interleave the execution of τ actions performed by process elements of the same pool, so that if a part of a process evolves, the whole process evolves accordingly.
- Rules $P\text{-}\checkmark$ enable process elements to evolve the process configuration with temporal transitions ✓if no τ actions may occur (maximal progress rule [26]).



$$\begin{array}{|l|}\hline \mathsf{pool}(\mathsf{p}_e, P_e, 1) \\ \quad P_e \quad = \quad \mathsf{start}(\mathsf{f}'_{48}, \mathsf{f}_{48}) \mid \mathsf{task}(\mathsf{f}_{48}, \mathsf{Start\ fire}, \varepsilon, \mathsf{A}_1, \varepsilon, 0, \varepsilon, \mathsf{f}_{49}) \mid \mathsf{end}(\mathsf{f}_{49}) \\ \qquad\quad A_1 = \mathsf{assign}(\mathit{true}, \mathsf{Data_Input.place}, \mathsf{fire}) \\ \hline \end{array}$$

Fig. 15. Base environment pool: graphical and textual representation.



$$\begin{array}{|l|}\hline \mathsf{pool}(\mathsf{p}_e, P_e, 1) \\ \quad P_e \quad = \quad \mathsf{start}(\mathsf{f}'_{48}, \mathsf{f}_{48}) \mid \mathsf{xorJoin}((\mathsf{f}_{48}, \mathsf{f}_{54}), \mathsf{f}_{49}) \mid \mathsf{task}(\mathsf{f}_{49}, \mathsf{Start\ fire}, \varepsilon, \mathsf{A}_1, \varepsilon, 0, \varepsilon, \mathsf{f}_{50}) \mid \\ \qquad\quad \mathsf{xorSplit}(\mathsf{f}_{50}, (\mathsf{DataInput.door} = \mathsf{true}, \mathsf{f}_{51}), (\mathsf{DataInput.door} = \mathsf{false}, \mathsf{f}_{52})) \mid \\ \qquad\quad \mathsf{task}(\mathsf{f}_{51}, \mathsf{Close\ door}, \varepsilon, \varepsilon, \mathsf{M}_1, 0, \varepsilon, \mathsf{f}_{53}) \mid \mathsf{xorJoin}((\mathsf{f}_{52}, \mathsf{f}_{53}), \mathsf{f}_{54}) \\ \qquad\quad A_1 = \mathsf{assign}(\mathit{true}, \mathsf{Data_Input.place}, \mathsf{fire}) \\ \qquad\quad M_1 = \mathsf{disconnect}(\mathsf{Data_Input.placeDoorSource}, \mathsf{Data_Input.placeDoorTarget}), \\ \qquad\qquad\quad \mathsf{disconnect}(\mathsf{Data_Input.placeDoorTarget}, \mathsf{Data_Input.placeDoorSource}) \\ \hline \end{array}$$

Fig. 16. Complex environment pool: graphical and textual representation.

- Rule *C-Int* propagates the transition from process to collaboration configurations.
- Rules *C*-τ and *C*-✓ behave as *P*-τ and *P*-✓, respectively, but at collaboration level. Notably, rules *P*-τ and *C*-τ allow single process elements to evolve the collaboration configuration, while *P*-✓ and *C*-✓ impose a synchronous triggering of rules producing a temporal transition.

In conclusion, given an environmental BPMN collaboration structure, our formalization relies on the application of the presented rules to produce the LTS, from the initial state to all these states without outgoing transitions, which we consider as final. Our formalization admits the presence of non-deterministic behaviors, which are reflected in the LTS as states with multiple outgoing transitions with the same label. Concerning final states, they can represent not only the proper completion of the environmental BPMN model (which occurs when every token has been consumed by an end event), but also they can represent deadlock situations where no more semantic rules are enabled, and one or more tokens are still marking some of the sequence flows. It is worth noticing that our LTS-based semantics is consistent with the token-based semantics of BPMN [30].

6. Formalization at work

In this section, we put our formalization into practice to guarantee a more precise understanding of the interplay between BPMN collaborations and the environment. We rely on the previously introduced fire-extinguishing scenario.

The influence of the environment. To demonstrate how changes in the environment can significantly impact the execution of the entire collaboration, a new pool, named *Environment*, was added to the collaboration model. This pool, shown in Fig. 15, illustrates a base case involving a specific environmental impact: a power outlet causes a short circuit, which then leads to a fire. This simple scenario is modeled using a start event, a task that triggers a fire at a specified location (as defined in the data input), and an end event. However, the environment may undergo more complex changes, as shown in Fig. 16. In this scenario, while the environment initiates a fire, other elements may interact with it by choosing whether to close a door or not, further impacting the process execution. Indeed, the process execution affects the environment (closing a door corresponds to disabling an edge between two places in the place graph) and, vice-versa, the environment itself influences the process execution (the disabling of an edge makes a place no longer reachable and may potentially cause a deadlock). Note that this pool is responsible for making changes in the environment. We assume that during the real enactment, which is out of the scope of this work, these changes are driven by actual fire sensors.

Formalization at work for the running example. The initial configuration of the fire-extinguishing collaboration is the following: $\langle C_d, G_d, \sigma_i^0, \sigma_{apl}^0, \sigma_p^0, \sigma_m^0 \rangle$. In detail, the state function σ_i^0 maps each process instance with its initial configuration, which in its own turn marks with a token the spurious sequence flow of the start events in the process. For example, $\sigma_i^0(p_s, 1)$ maps the first sensor instance to the initial process configuration $\langle \sigma_f^0, \sigma_{ado}^0, \sigma_s^0, \sigma_t^0 \rangle$, where σ_f^0 contains a token in the sequence flow f', i.e., $\sigma_f^0(\mathsf{f}') = 1$. Whereas, σ_{apl}^0 is the state

function of the environmental attributes where the *fire* and the *alarm* attributes are *false*: $\forall \mathsf{pl} \in P_l \,.\, \sigma^0_{apl}(\mathsf{pl}.fire) = false \wedge \sigma^0_{apl}(\mathsf{pl}.alarm) = false$. Then, σ^0_p is the state function of the participants' position, which is initialized with the initial place held by the participants as follows: $\sigma^0_p(\mathsf{p}_s, 1) = \mathsf{pl}_1, \ldots, \sigma^0_p(\mathsf{p}_s, 11) = \mathsf{pl}_{44}, \sigma^0_p(\mathsf{p}_f, 1) = \mathsf{pl}_{22}, \sigma^0_p(\mathsf{p}_r, 1) = \mathsf{pl}_8, \sigma^0_p(\mathsf{p}_r, 2) = \mathsf{pl}_{28}, \sigma^0_p(\mathsf{p}_r, 3) = \mathsf{pl}_{34}$.

In the following, we describe two different execution outcomes of the running example: one considering the base case of the environment and the other considering the extended case of the environment. Note that we focus on the execution steps that better show the interaction between BPMN and the environment. The collaboration starts with the enactment of all its instance pools, as tokens are initially present in f'_0, f'_6, f'_{35}, f'_{48}. Each of these tokens enables rule *P-Start*, which consumes the token from the enabled sequence flow and produces a new token on the respective outgoing sequence flow. In the running example, we have 11 tokens for the sensors pool, 1 token for the fire control system, 3 tokens for the fire-fighting robots, and 1 token for the environment. Tokens move according to the operational rules induced by our formal semantics. When a task has a non-zero duration, the token remains on the task in the BPMN model until it is completed. Similarly, since movement tasks duration is determined by the places to traverse, the token in the BPMN model stays on the task until the corresponding token in the environment model reaches the destination.

The tokens in f_0 related to the multi-instance sensor pool will execute the rule *P-XorJoin* and then wait at the conditional event *Fire detection* until the condition exp_1 is triggered. The condition exp_1 is triggered when one of the sensors detects a fire in one of its adjacent places, specifically, when the fire attribute of a neighboring place is set to true by the Environment pool. After that, only the sensor instance that detects the fire executes the task *Store fire position*, which assigns to the field place of the data object *Fire_pos* its current position. For instance, if a fire was detected in the kitchen, the active sensor instance would be the number 1, whose position is place pl_1, and the formalization of the assignment would be the following: assign(pl_1, Fire_pos.place). Once the fire position is stored, the sensor sends it to the fire control system by executing the rule *P-Snd*; it then waits until the fire is extinguished, that is, when the condition exp_2 of the conditional event *Fire extinguished* becomes true. The condition exp_2 is satisfied when none of the attributes fire of the neighboring places of the fire position sent by the sensor is set to true.

The pool responsible for changing the value of fire place attributes is the *Environment*. As discussed previously, we consider two types of environment pools. In the simple case, the environment, after executing the start event with flow f_{48}, proceeds to execute the *Start fire* task, simulating the ignition of a fire, formalized through the assignment assign(*true*, Data_Input.place, fire), which sets as true the attribute fire in the place specified in the Data Input, corresponding to pl_2 in the case study. After that, it concludes with the rule *P-End*. On the other hand, in the complex case, beyond executing the *Start Fire* task, the environment, based on the content of the *Data input*, may execute (or not) the *Close door* task. This task alters the environment's topology by removing specific edges between places. The removal of the edges is formalized as described in M_1 in Fig. 16, which in our case study corresponds to the removal of an edge from place pl_3 to place pl_{17} and vice versa. In the meantime, the three tokens in the fire fighter robot pool, execute the start event with flow f_{35} and perform the *Store base position* task to record their respective positions, by using the rules $P\text{-}Task_{Act}$ and $P\text{-}Task_C$. This task performs an assignment in the field place of the *Base station* data object. After that, they send their positions to the pool *Fire control system* with the rule *P-Snd*. Afterward, the three tokens will wait to receive a message about the fire position they have to reach at the *Fire position reception* message catch event.

The token in f'_6 related to the *Fire control system* pool, after executing the *P-Start*, executes a series of tasks in a loop, repeating as many times as the number of robot instances (i.e., 3). This loop corresponds to the macro shown in Fig. 6, which is used when a single instance pool has to receive and register messages coming from a multi-instance pool. In this case, the fire control system executes the task *Receive robot position guard* with the rules $P\text{-}Task_{Act}$ and $P\text{-}Task_C$; then it executes the rule *P-Rcv* for the receive event *Receive robot position*, which registers the content of the received message in the *Robot position* data object. After that, it executes the *Receive robot position assignment* task which performs two assignments: the first increments by one the count field into the *Robot counter* data object, while the second appends the id of the robot and its position to the *Robot positions* data object collection, defined as append({id : Robot_position.id, place : Robot_position.place}, Robot_positions.list).

Once the fire control system has iterated for the number of robot instances (Robot_counter.count = 3 is true), its token will wait to receive fire positions from the fire-detecting sensors. When the fire control system receives a fire position, it then controls if the alarm is already active and, if it is not, it proceeds to activate it. Afterward, it executes the *Check fire position* task, which performs an assignment incrementing by one the number of the fire field in the *Fire_counter* data object. Next, it executes the task *Calculate closest robot*, which assigns to the data object *Closest robot* the id of the closest robot to the fire location. This assignment is determined using a custom function expressed in FEEL, as detailed in Section 4. After identifying the closest fire-fighting robot, the fire control system sends it the position of the fire to extinguish. The robot then executes the movement task rules ($P\text{-}TaskM_{Act}$, $P\text{-}TaskM_{Elps1}$, $P\text{-}TaskM_C$) to travel from its initial position to the destination. Meanwhile, the fire control system resumes waiting for notifications from the fire-fighting robots to manage other fires in the environment.

In the base case of the environment pool, the fire-fighting robot reaches its destination and extinguishes the fire by executing the task *Extinguish fire*, which assigns the value false to all fire attributes of the places near the fire position that is expressed as: assign(false, meAndNeighbour(Fire.place), fire). After extinguishing a fire, the robot notifies the fire control system, which then decrements the value of the data object Fire_counter. If no other fires are present (i.e., Fire_counter.count = 0), the system deactivates the alarm and the fire is considered managed. Otherwise, if Fire_counter.count > 0, the alarm remains active.

In the more complex case of the environment pool, if a connection between two places does not exist, the robot cannot reach the destination, and the timer boundary event is triggered by the rule $P\text{-}TaskM_B$. The robot then sends a message containing the fire position to the fire control system, which executes the task *Intervention by humans*. This task assigns the value false to all fire attributes of the places near the reported fire position. Afterward, the process continues as described above.

Summing up, this example demonstrates how interactions with the environment can affect the execution and outcome of an environmental BPMN collaboration. By considering two different scenarios, we can obtain two distinct outcomes, as the environment exerts different influences on the process, and vice versa. This paves the way for reasoning on different scenarios where the physical environment can influence the process execution and vice versa.

7. Modeling, animating, and debugging environmental BPMN collaborations

In this section, we present an extended version of BEAR (BPMN Environmental AnimatoR), a tool we developed for modeling, animating, and debugging environmental BPMN collaborations. We extend the version of the tool introduced in [10] to support the features and aspects of environmental BPMN collaborations defined in this paper, i.e., multi-instance behaviors and time progression. The tool is released under the MIT license and its source code, user guide and demonstration video can be accessed on-line.[2]

[2] https://pros.unicam.it/bear-v1/.

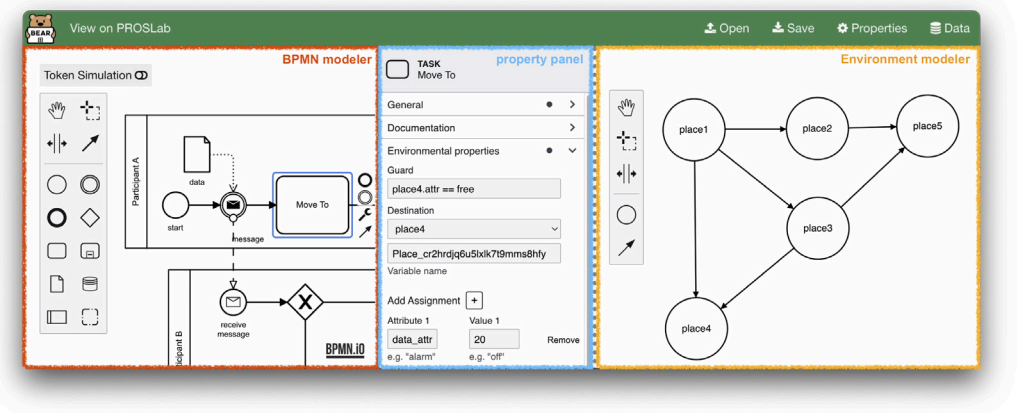


Fig. 17. BEAR Modeler GUI.

Modeling. BEAR provides a visual modeler for designing environmental BPMN collaborations. Its interface, shown in Fig. 17, is divided into two main parts: the **BPMN modeler** on the left and the **Environment modeler** on the right.

Users can leverage the BPMN modeler to create a collaboration diagram with BPMN elements, and the latter to design a place graph representing the environment. By means of a **property panel**, information related to a selected element can be added, then used to drive the animation. Concerning the environment, each place and edge in the model can be characterized by defining a list of attributes. These attributes are *key–value* pairs, e.g., `alarm = false`, accessible by processes in the collaboration with the following notation: `place2.alarm`. Similarly, the property panel shows additional environment-related information for each BPMN element. By selecting a *process pool*, users can add a **position** property indicating a participant's initial position. As for *data objects*, they can define and instantiate process attributes. Such data are only accessible by those pool process elements, such as tasks, events, and gateways, enabling the storage, manipulation, and sharing of information. For *multi-instance process pools* and *multi-instance data objects*, multiple values can be set, indicating the initial positions and attributes of each instance. This is particularly useful when different instances maintain distinct values for the same attribute. Concerning *tasks*, users can also specify their **type**, either **movement** or **environmental**. The former includes a **destination** attribute, specifying a place to reach from the participant's current position. The latter allows the specification of **assignments** and **guards**, which respectively update and predicate over environmental attributes. Regarding *events*, users can define message events (start, intermediate, end) that support value passing. Send events can carry explicit values, e.g., `true`, or attribute references, e.g., `place1.fire`, while receive events specify where the payload should be stored. Conditional events (start and boundary) include a **condition** predicating over environmental attributes, which triggers the event. For conditional boundary events attached to a movement task, the `unreachable` keyword can be used to handle situations in which a participant is unable to reach the specified destination. Similarly, conditions can be attached to *XOR gateway* flows.

To dynamically reference environmental elements, such as place and attribute identifiers, BEAR exploits FEEL. With this integration, BEAR enables runtime evaluation of tasks, events, and gateways, based on the current state of the process. For instance, a movement task destination may be specified through a FEEL expression whose evaluation ultimately resolves to a place name. The same holds for environmental attributes targeted by assignments, guards and conditions.

The BEAR modeler, permits the user to export/load an environmental BPMN collaboration as a `.bpmn` file of the BPMN collaboration and an `.xml` file of the environment model compressed into a `.zip` file. Notably, the `.bpmn` file is compatible with other BPMN modelers, except for specific aspects of our approach like custom task markers, destinations, guards, and assignments, that are not rendered in the visual modeler.

Animation. BEAR supports process token-animation, showing the evolution of the BPMN collaboration with its environment. By clicking the top-left `Token Animation` toggle, a play button appears over each start event in the collaboration, allowing to start the animation of the corresponding process. Upon that, BEAR displays colored tokens in both models, offering an intuitive view of the collaboration dynamics and the participants' positions, as shown in Fig. 18. Then, tokens automatically start moving within the models according to the operational rules induced by our formal semantics. Each process token and its corresponding counterpart in the environment model are annotated with a number representing the instance identifier. When a task has a non-zero duration, a token remains on the task in the BPMN model until it is completed. Similarly, since movement tasks duration is determined by the places to traverse, the token in the BPMN model stays on the task until the corresponding token in the environment model reaches the destination. Notably, when multiple elements are enabled in parallel, the animator handles this non-determinism by adopting a random selection strategy. That is, one of the enabled elements is randomly chosen and executed, allowing the animation to progress, without requiring additional user intervention. This differs from tools like [31], which adopts a step-by-step simulation driven by the user. The animation terminates once no tokens can move forward; however, it can be paused at any time to check the distribution of tokens in the environment and in the BPMN collaboration.

Over the course of the animation, users can monitor which places catch fire during the animation by opening the data panel on the right, which lists the values of all attributes. Moreover, changes to the environment's topology are directly visible in the model, as removed edges disappear and newly added ones appear at runtime, indicating doors closing or opening. These features allow users to observe changes in the environment and understand how the collaboration affects it.

Debugging. Throughout the process animation, some undesired situations may arise, preventing it from continuing. When this happens, BEAR halts the animation and freezes the tokens, marking both the BPMN model and the place graph, allowing them to be inspected. Additionally, as shown in Fig. 19, error messages are displayed when some specific errors are detected.

An *unreachable destination* is reported during a movement task when no path exists between the destination place and the participant's current position. A *violated guard* occurs when the boolean condition associated with an environmental task or an XOR gateway evaluates to false, thus preventing the process from progressing. A *syntax error* is raised when inconsistencies or malformed definitions are found in the process model, preventing its correct interpretation. Finally, a *missing*

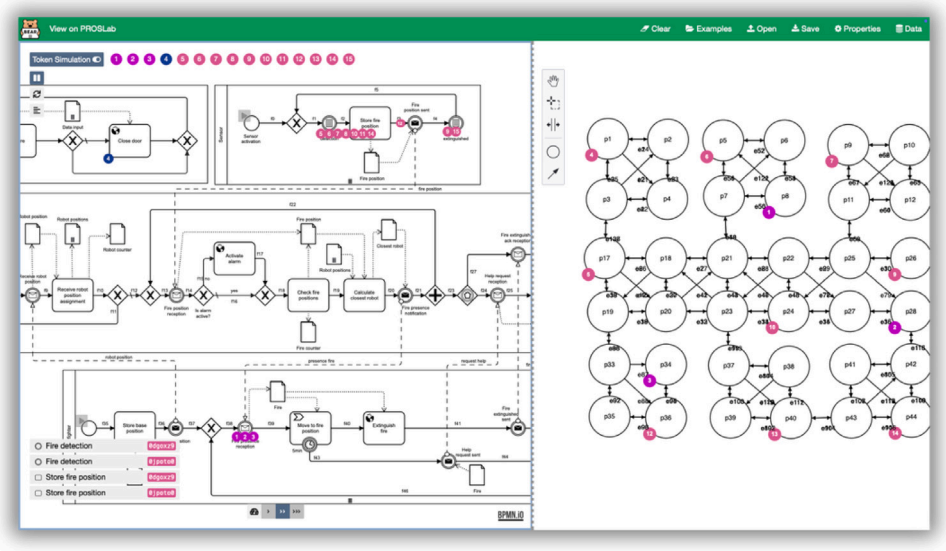


Fig. 18. BEAR Animator GUI.

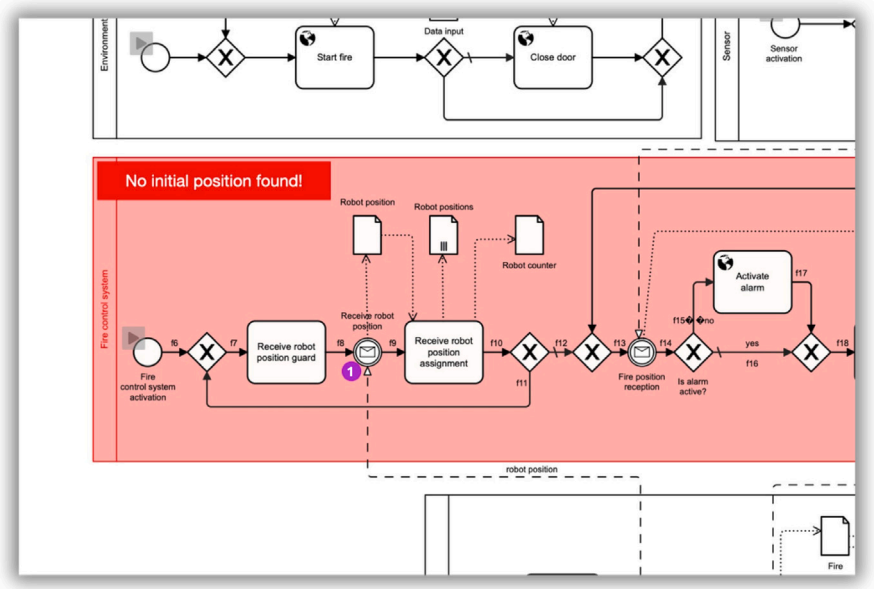


Fig. 19. BEAR error messages.

initial position is reported when a participant pool is not associated with any place in the environment model. From a practical perspective, together with the data panel and the ability to pause the animation at any time, these features support debugging of environmental BPMN collaboration models. By enabling users to inspect the current execution state and identify the causes of halting, both the process and environment models can be revised before rerunning the animation.

8. Related work

In this section, we examine the literature on BPMN formalization and its interaction with the environment. We first discuss approaches that provide a formal execution semantics for BPMN elements. Then, we discuss works that extend BPMN with environmental aspects, highlighting in a comparison table how our approach goes beyond the state of the art.

8.1. BPMN formalization

Several works in the literature have attempted to formalize the core elements of BPMN using different approaches. Some of them map BPMN elements into already established formal models, allowing existing semantics and analysis techniques to be reused. For instance, Dijkman et al. [32] define a formal semantics for BPMN through Petri nets, enabling the verification of properties such as soundness. Khelldoun et al. [33] extend this approach by using High-Level Petri Nets to also capture data objects within BPMN models. Decker et al. [34] instead, propose a transformation from BPMN models into YAWL nets, allowing BPMN models to benefit from the formal execution semantics provided by the YAWL language.

On the other hand, other works provide a direct formalization of the BPMN semantics. Wong and Gibbons [35] propose a compositional semantics based on process algebra, specifically using techniques derived from the Calculus of Communicating Systems, which supports

reasoning about process behavior through algebraic operators such as sequencing, choice, parallel composition, and communication. Börger and Thalheim [36] formalize BPMN elements using Abstract State Machines, representing each element within a state and defining execution rules that specify how process states evolve over time. Another formalization is the one proposed by Dijkman and Van Gorp [37], who formalize the execution semantics of BPMN using graph rewrite rules. In their approach, BPMN models are interpreted as graphs, and the execution of process elements is captured through transformation rules that describe how the graph evolves over time. In a subsequent work, Van Gorp and Dijkman [30] propose a visual token-based formalization using in-place transformations. This formalization specifies how tokens are created, consumed, and propagated through BPMN elements, providing an operational semantics that captures the behavior of tokens in BPMN models. Corradini et al. [38] propose a novel formalization of BPMN collaborations based on a BNF syntax and a semantics defined in terms of a Labelled Transition System. Their approach formalizes a subset of BPMN elements, including pools, start events, end events, tasks, XOR/AND/OR split and join gateways, sequence flows, and message flows. This initial formalization has been extended in subsequent works by the same authors. In [39], the formalization is enriched to include event-based gateways, terminate end events, and intermediate events. In [40], the authors further extend the framework to formalize receive and send tasks, as well as subprocesses. Finally, in [41], data objects and multiple-instance pools and tasks are also formalized.

However, none of these formalizations takes the environment into account. So far, the only formalization that considers environmental aspects is the one by Saddem-Yagoubi et al. [7], which formalizes BPMN elements and their semantics using Linear Temporal Logic, integrating also environmental constraints. Our work goes further by adopting an LTS semantics that accounts for the environment while also formalizing a wider range of BPMN elements, including multiple-instance pools and data objects.

Summing up, the literature proposes two main approaches to formalize BPMN: either mapping BPMN to established formal models or providing direct formalizations using different techniques. However, these approaches largely neglect environmental aspects. Our work addresses this limitation by introducing an LTS-based semantics that incorporates the environment while supporting a broader set of BPMN constructs.

8.2. BPMN and the environment

Concerning the literature that discusses the interplay between BPMN and the physical environment, Decker et al. [42] and Zhu et al. [43] propose BPMN extensions that allow modeling location-based tasks whose execution is constrained by the participant's location; in addition, the latter introduce location-dependent gateways whose conditions are based on environmental information. Dorndorfer et al. [44] propose an extension to the BPMN standard where spatial information can trigger conditional boundary events attached to tasks to handle exceptional behaviors. Chiu et al. [45] introduce a new type of event, called location event, marked with a pin icon that can describe either the current position or the movement of devices or physical entities, while Poss et al. [46] adopt location-based events and, in addition, define location data, i.e., data objects used to store and retrieve environmental information, and introduce the possibility of allocating resources to tasks based on the location. Tomas Kozel [8] does a step forward, proposing, in addition to location-based events, the possibility to use specific markers for participants that can move in the environment and track their position. Nevertheless, these works focus solely on BPMN processes, adding constraints and handlers based on environmental aspects. Differently from our work, none of them provides a model that explicitly represents the environment and its evolution during process execution, nor do they offer a formalization of these concepts.

Considering time progression, despite the extensive literature on modeling time in BPMN [47,48] and defining it formally [25,27,49,50], none of the above approaches considers time as a dynamic factor interacting with the environment. Notable exceptions are the works of Mazhar et al. [51] and Grefen et al. [52], who consider environment and time together. Both represent the environment by grouping task activities within defined areas using the BPMN process grouping symbol, which denotes physical spaces relevant to the process. Mazhar et al. additionally model time as an attribute of a location, used to represent waiting periods before an activity can be performed. Similarly, Grefen et al. associate time with location, but distinguish between absolute time (a fixed interval with a defined start and end) and relative time (a specific time window). In their work, time serves as a constraint for performing specific activities within locations.

A work that provides a formalization and considers time and environment together with BPMN is the one of Saddem-Yagoubi et al. [7]. They define a formal temporal logic that extends BPMN elements, such as gateways and conditional sequence flows, to incorporate location information and enable location-dependent decision-making. In this case, the time is handled through a non-deterministic and abstract semantics. Concerning the spatial model, they describe a place graph only formally, without a graphical representation. They also provide a model checker for the verification of behavioral properties on the produced models. Nevertheless, differently from our approach, the interaction between process activities and the environment is modeled through standard BPMN tasks, rather than introducing a dedicated task. Furthermore, although they support formal verification and counter-example animation, they do not provide animation capabilities such as those enabled by our tool.

However, all these works, differently from ours, merely use time as a constraint rather than being formally treated or modeled as a cause-and-effect factor influencing the execution of tasks in relation to the environment. Moreover, none of these approaches considers the presence of multi-instance pools interacting with the environment.

Summing up, even though the literature lately is increasingly dealing with the concept of environment within BPMN, our approach advances the current state of the art for different aspects, as resumed in Table 2. To the best of our knowledge, ours is the only formal approach that allows altogether to: model the physical environment and its attributes explicitly, represent environmental tasks with environmental constraints (guards) and modifications (assignments), model tasks that involve a movement in the space, and keep track of participants' position during the process execution, since we associate a position in the environment with each participant. All of this while accounting for scenarios with multi-instance pools. Moreover, our approach allows for the definition of environmental conditions on decision gateways and events, and also considers time as a factor that can influence the execution of activities in relation to the environment. In terms of tool support, only our work and that of [7] provide an implementation. Our tool enables the animation (A) and debugging (D) of BPMN collaborations integrated with the environment, while the tool in [3] is used to conduct formal verification (V). While the aforementioned approaches incorporate environmental aspects into BPMN models in different ways, they all lack one crucial component: an execution semantics that can be observed and understood through proper animation tools. Such animator tools play an important role in understanding the behavior of business processes, as they offer a concrete step by step visualization of their execution semantics. In this regard, several animator tools have been proposed in the literature to support the animation of BPMN models [53]. Allweyer and Schweitzer [54] and Signavio [31] provide tools to animate the control flow of BPMN processes; Corradini et al. [24] propose the tool MIDA to animate BPMN collaborations with multi-instance participants and data, while Abdul et al. [55] present UBBA to create and then animate a virtual 3D representation of BPMN collaborations. In this field of animator tools, ours represents a novel contribution, as it is the only one that implements all the proposed environmental features, providing a comprehensive understanding of the environmental BPMN behavior through its animation and debugging features.

Table 2
Resuming BPMN and the environment related work table. (V) Verification, (A) Animation, (D) Debugging.

Features\Papers		[42]	[43]	[44]	[45]	[46]	[8]	[7]	[51]	[52]	Ours
Formal semantics								•			•
Environment	Environment model							•			•
	Environmental attributes										•
BPMN model	Environmental task	•	•					•	•	•	•
	Movement task										•
	Participant position						•	•			•
	Multi-instance pool										•
	Location-based gateway		•					•			•
	Location-based events			•	•	•	•			•	•
Time									•	•	•
Type of analysis								V			A/D
Analysis tool support								•			•

9. Concluding remarks

This work discusses the interplay between BPMN collaborations involving multi-instance participants and their surrounding environment. In particular, we prescribe the use of place graphs and place attributes as an abstraction of the physical environment where the collaborations occur. We link these concepts incorporating the environment within the BPMN meta-model, resulting in the definition of environmental BPMN collaborations. We provide a formal operational semantics of the environmental BPMN collaboration models, including multi-instance pools and the time perspective with timer events, task durations, and timer boundary events. We illustrate and assess our proposal on a running example.

Finally, we present BEAR, an innovative BPMN animator designed to enhance understanding of interactions between BPMN multi-instance collaborations and the environment.

Discussion. We merge business processes and the environment into BPMN collaborations and semantically enriched place graphs, respectively. The choice of adopting the BPMN notation depends on its large application [56]. In particular, we use collaboration diagrams to model process aspects of interest when dealing with the environment. Reasoning on multiple participants amplifies the interaction with the environment, while communication enables the sharing of environmental information. At the same time, we intentionally left out of the discussion those BPMN elements that draw away the reader from the addressed problem, e.g., sub-processes and signal events. Moreover, our contribution extends the BPMN notation with two novel graphical elements and their related semantics. Although extending standard notation is generally discouraged due to potential issues with compliance and interoperability [57], we chose to introduce new BPMN elements to provide a clearer, more intuitive representation of the environment's influence on BPMN collaborations. Regarding the representation of time, we have decided to include it in our approach since it is a variable closely linked to the environment, especially for movements. However, to maintain formal clarity and focus on environmental aspects, we chose to use discrete time. Naturally, adopting continuous time would improve the realism of our models, but for now, our priority is to ensure a comprehensible, structured representation of environmental BPMN collaboration models.

Concerning data handling, in our formalization, we do not introduce data types to avoid increasing the complexity, as they become important when implementing the model for its enactment in the real world, while we aim to focus on modeling. As well, we do not make any assumption about the well-formedness of assignments that may lead to models containing deadlocks, as in more usual cases, e.g., where the outgoing flows of a XOR Split merge into an AND Join.

Concerning the environment, we use place graph models to provide an abstract representation of the space, rather than geometric models, which represent the space with faithful measurements. In particular, place graphs represent space at varying levels of abstraction tailored to specific domains; for instance, a place may correspond to a room or a building. This flexibility enables the modeling of varying degrees of spatial detail. Moreover, the choice of adopting a place graph enables a certain degree of dynamism. This is made possible by modifying the topology of the place graph, as edges can be added or removed by the process execution, c.f. connect and disconnect. Moreover, dynamism is also supported by the possibility of mimicking the behavior of the environment through a specific process that modifies the attributes characterizing the places. Indeed, this allows the other processes to react to specific environmental conditions (e.g., a sensor that is triggered by a fire). At the same time, some aspects of the environment are assumed to be static. Differently from the set of edges, the set of places is fixed, and our approach does not allow the addition or removal of places. This choice is driven by our intention to keep a trade-off between the complexity and the expressiveness of our formalization. For example, in the modeling of a scenario, one can assume that the environmental model contains all the places that are necessary for the process, while the connection between them can change over time, e.g., a door closed. Furthermore, we currently assume that places are equidistant, as no weight function is defined on edges to represent environmental concepts such as distance or gradients. Another assumption we made is that participants in the environmental BPMN collaboration move at the same speed and possess a priori knowledge of the environment in which they can move. These choices are driven by the intention of keeping the proposed formalization comprehensible. Nonetheless, this aspect becomes more crucial when dealing with verification, which we plan to address in future work. Nevertheless, place graphs can be extended to support these omitted features by adding new spatial assignments to the semantics that act on places. Moreover, they could support weight functions on individual edges to accommodate more diverse and flexible spatial scenarios.

Concerning BEAR, usability may be limited when simulating large collaborations with many concurrent process instances. At any given moment, a significant number of tokens may appear on screen, corresponding both to participants in the process and their positions in the environment. This can make it challenging to follow the animation and understand the process's current state. Furthermore, BEAR does not perform formal verification. Consequently, situations such as livelocks or other forms of non-termination are not automatically detected or resolved by the tool. While the animation supports pausing, it may get stuck in a loop if the process enters a state where execution can proceed indefinitely without completing.

Future work. In future work, we intend to continue investigating the interplay between BPMN collaborations and the environment by enriching the environment model and its formal account. We plan to incorporate new environmental properties, such as size and occupancy, which are important in scenarios with multiple participants who must collaborate. Additionally, we aim to extend our approach with additional BPMN elements, enabling a broader range of modeling scenarios,

including the terminate event, additional multi-instance markers, exceptions, and sub-processes. Finally, we plan to develop verification techniques to ease understanding of our environmental BPMN models and enable formal reasoning.

CRediT authorship contribution statement

Flavio Corradini: Funding acquisition. **Luca Mozzoni:** Writing – original draft, Software, Conceptualization. **Jessica Piccioni:** Writing – original draft, Investigation, Conceptualization. **Barbara Re:** Writing – review & editing, Conceptualization. **Lorenzo Rossi:** Writing – review & editing, Methodology, Conceptualization. **Francesco Tiezzi:** Writing – review & editing, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Italian Ministry of University and Research (MUR) under D.M. 118/2023 “Doctoral programs of national interest in Blockchain & Distributed Ledger Technology”.

Appendix. One-to-one correspondence between the BPMN graphical notation, the textual notation and the BPMN XML format

In this appendix, we show the one-to-one correspondence between the BPMN graphical notation, the textual notation, and the BPMN XML format. Table A.3 regards pool and events, Table A.4 regards gateways, Table A.5 regards tasks and environmental tasks, and Table A.6 regards movement tasks.

Data availability

No data was used for the research described in the article.

Table A.3
Correspondence between graphical, textual and BPMN XML notation: pool and events.

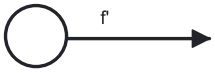
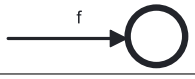
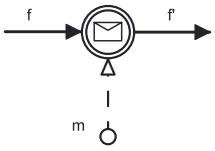
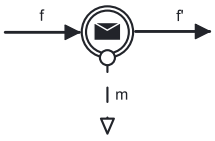
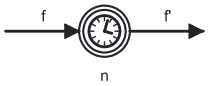
BPMN elements	BPMN XML syntax	Textual notation
	`<participant name="p" ... />`	pool(p, P, 1)
	`<participant name="p" ... /> <participantMultiplicity k=3 />`	pool(p, P, 3)
	`<startEvent ...> <outgoing> f' </outgoing> </startEvent>`	start(f, f′)
	`<endEvent ...> <incoming> f </incoming> </endEvent>`	end(f)
	`<intermediateCatchEvent ...> <incoming> f </incoming> <outgoing> f' </outgoing> <messageEventDefinition .../> <bpmn:extensionElements> <bpmnenv:field field= <t_1 ,... , t_n> /> </bpmn:extensionElements> </intermediateCatchEvent>`	rcvEvent(f, m : ⟨$t_1, \ldots, t_n$⟩, f′)
	`<intermediateThrowEvent ...> <incoming> f </incoming> <outgoing> f' </outgoing> <messageEventDefinition .../> <bpmn:extensionElements> <bpmnenv:field field= <expr_1 ,... , expr_n> /> </bpmn:extensionElements> </intermediateThrowEvent>`	sndEvent(f, m : ⟨$\exp_1, \ldots, \exp_n$⟩, f′)
	`<intermediateCatchEvent ...> <incoming>f</incoming> <outgoing>f'</outgoing> <conditionalEventDefinition ...> <condition> exp </condition> </conditionalEventDefinition> </intermediateCatchEvent>`	condEvent(f, exp, f′)
	`<intermediateCatchEvent name=n> <incoming> f </incoming> <outgoing> f' </outgoing> <timerEventDefinition ..." > <timeDuration>k</timeDuration> <timerEventDefinition /> </intermediateCatchEvent>`	timerEvent(f, n, k, f′)

Table A.4
Correspondence between graphical, textual and BPMN XML notation: gateways.

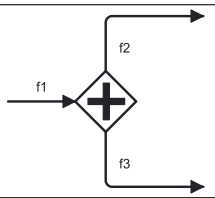
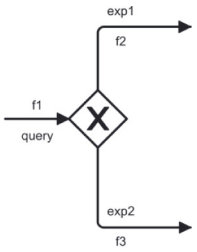
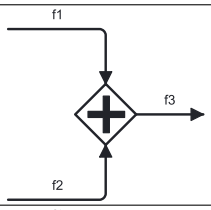
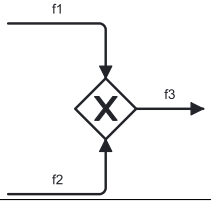
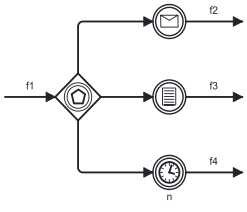
Graphical notation	BPMN XML syntax	Textual notation
	`<parallelGateway ... > <incoming>f1</incoming> <outgoing>f2</outgoing> <outgoing>f3</outgoing> </parallelGateway>`	andSplit(f1, {f2, f3})
	`<exclusiveGateway> <incoming>f1</incoming> <outgoing>f2</outgoing> <outgoing>f3</outgoing> </exclusiveGateway> ... <sequenceFlow id="f2"> <conditionExpression ... > exp_1 </conditionExpression> </sequenceFlow> ...`	xorSplit(f1, {(f2, exp_1), (f3, exp_2)})
	`<parallelGateway ... > <incoming>f1</incoming> <incoming>f2</incoming> <outgoing>f3</outgoing> </parallelGateway>`	andJoin({f1, f2}, f3)
	`<exclusiveGateway ... > <incoming>f1</incoming> <incoming>f2</incoming> <outgoing>f3</outgoing> </exclusiveGateway>`	xorJoin({f1, f2}, f3)
	`<eventBasedGateway ... > <incoming>f1</incoming> ... </eventBasedGateway> <intermediateCatchEvent ... > ... <outgoing> f2 </outgoing> <messageEventDefinition .../> <bpmn:extensionElements> <bpmnenv:field field=t /> </bpmn:extensionElements> </intermediateCatchEvent> <intermediateCatchEvent ... > ... <outgoing> f3 </outgoing> <conditionalEventDefinition.... > <condition>exp</condition> </conditionalEventDefinition> </intermediateCatchEvent> <intermediateCatchEvent ... > ... <outgoing> f4 </outgoing> <timerEventDefinition..." > <timeDuration>k</timeDuration> <timerEventDefinition /> </intermediateCatchEvent>`	eventBased(f1, (m : $\tilde{t}$, f2), (exp, f3), (n, k, f4))

Table A.5
Correspondence between graphical, textual and BPMN XML notation: tasks and environmental tasks.

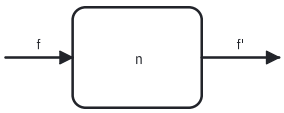
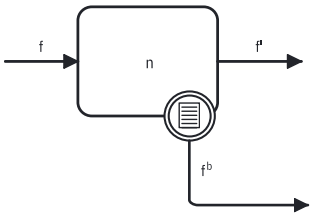
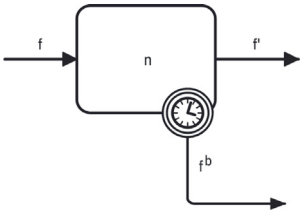
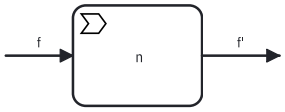
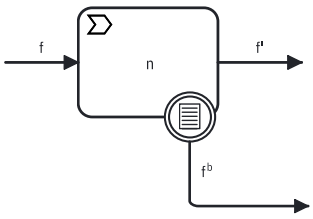
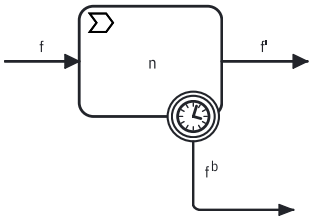
Graphical notation	BPMN XML syntax	Textual notation
	<task name="n" ... > <incoming>f</incoming> <outgoing>f'</outgoing> <bpmn:extensionElements> <bpmnenv:guard> exp_g </bpmnenv:guard> <bpmnenv:assignments> <bpmnenv:assignment> A </bpmnenv:assignment> </bpmnenv:assignments> <bpmnenv:duration> d </bpmnenv:duration> </bpmn:extensionElements> </task>	task(f, n, $\exp_g$, A, M, d, ϵ, f′)
	<task name="n".... > <incoming>f</incoming> <outgoing>f'</outgoing> ... </task> <boundaryEvent ... > <outgoing>f^b</outgoing> <conditionalEventDefinition.... > <condition ... >exp</condition> </conditionalEventDefinition> </boundaryEvent>	task(f, n, $\exp_g$, A, M, d, bCondEvent(exp, f^b), f′)
	<task name="n" ... > <incoming>f</incoming> <outgoing>f'</outgoing> </task> <boundaryEvent ... > <outgoing>f^b</outgoing> <timerEventDefinition..." > <timeDuration>k</timeDuration> <timerEventDefinition /> </boundaryEvent>	task(f, n, $\exp_g$, A, M, d, bTimerEvent(n, k, f^b), f′)

Table A.6
Correspondence between graphical, textual and BPMN XML notation: movement tasks.

Graphical notation	BPMN XML syntax	Textual notation
	<movementTask name="n" ... > <incoming>f</incoming> <outgoing>f'</outgoing> <bpmn:extensionElements> <bpmnenv:guard> exp_g </bpmnenv:guard> <bpmnenv:destination> exp_d </bpmnenv:destination> </bpmn:extensionElements> </movementTask>	taskM(f, n, $\exp_g$, ϵ, ϵ, $\exp_d$, ϵ, f′)
	<movementTask name="n".... > <incoming>f</incoming> <outgoing>f'</outgoing> </task> <bpmnenv:guard> exp_g </bpmnenv:guard> <bpmnenv:destination> exp_d </bpmnenv:destination> <boundaryEvent ... > <outgoing>f^b</outgoing> <conditionalEventDefinition.... > <condition ... >exp</condition> </conditionalEventDefinition> </boundaryEvent> </movementTask>	taskM(f, n, $\exp_g$, ϵ, ϵ, $\exp_d$, bCondEvent(exp, f^b), f′)

(*continued on next page*)

Table A.6 (*continued*).




```
<movementTask name="n".... >
  <incoming>f</incoming>
  <outgoing>f'</outgoing>
</task>
  <bpmnenv:guard> exp_g
  </bpmnenv:guard>
  <bpmnenv:destination> exp_d
  </bpmnenv:destination>
  <boundaryEvent ... >
   <outgoing>f^b</outgoing>
   <timerEventDefinition..." >
    <timeDuration>k</timeDuration>
   <timerEventDefinition />
  </boundaryEvent>
</movementTask>
```

$\text{taskM}(f, n, \text{exp}_g, \epsilon, \epsilon, \text{exp}_d, \text{bTimerEvent}(n, k, f^b), f')$

References

[1] OMG, Business process model and notation, 2011, BPMN V2.0.

[2] N. Aoumeur, J. Fiadeiro, C. Oliveira, Towards an architectural approach to location-aware business process, in: Enabling Technologies: Infrastructure for Collaborative Enterprises, vol. 13, IEEE, 2004, pp. 147–152.

[3] M. Rosemann, J. Recker, Context-aware process design: Exploring the extrinsic drivers for process flexibility, in: Business Process Modeling, Development, and Support Proceedings of the Workshops and Doctoral Consortium, Namur University Press, 2006, pp. 149–158.

[4] M. Rosemann, J. Recker, C. Flender, Contextualisation of business processes, Int. J. Bus. Process. Integr. Manag. 3 (1) (2008) 47–60.

[5] L. Poss, S. Schönig, A generic approach towards location-aware business process execution, in: Enterprise, Business-Process and Information Systems Modeling, in: LNCS, vol. 479, Springer, 2023, pp. 103–118.

[6] L. Poss, L. Dietz, S. Schönig, LABPMN: Location-aware business process modeling and notation, in: Cooperative Information Systems, vol. 14353, Springer, 2024, pp. 198–216.

[7] R. Saddem-Yagoubi, P. Poizat, S. Houhou, Business processes meet spatial concerns: the sBPMN verification framework, in: Formal Methods, in: LNCS, vol. 13047, Springer, 2021, pp. 218–234.

[8] T. Kozel, BPMN mobilisation, in: International Conference on Mobile Business and Global Mobility Roundtable, World Scientific and Engineering Academy and Society, 2010, pp. 307–310.

[9] F. Corradini, J. Piccioni, B. Re, L. Rossi, F. Tiezzi, On the interplay between BPMN collaborations and the physical environment, in: International Conference on Business Process Management, in: LNCS, vol. 14940, Springer, 2024, pp. 93–110.

[10] F. Corradini, L. Mozzoni, J. Piccioni, B. Re, L. Rossi, F. Tiezzi, BEAR: BPMN and environment animator, in: Proceedings of the Best Dissertation Award, Doctoral Consortium, and Demonstration & Resources Forum at BPM, vol. 3758, 2024, pp. 121–125.

[11] OMG, Decision model and notation, 2015, DMN V1.0.

[12] I. Afyouni, C. Ray, C. Christophe, Spatial models for context-aware indoor navigation systems: A survey, J. Spat. Inf. Sci. 1 (4) (2012) 85–123.

[13] H. Ledoux, Modelling Three-Dimensional Fields in Geoscience with the Voronoi Diagram and its Dual, University of South Wales (United Kingdom), 2006.

[14] M. Mekni, Automated Generation of Geometrically-Precise and Semantically-Informed Virtual Geographic Environments Populated with Spatially-Reasoning Agents, Universal-Publishers, 2010.

[15] R. Chatila, J. Laumond, Position referencing and consistent world modeling for mobile robots, in: Proceedings of the International Conference on Robotics and Automation, vol. 2, IEEE, 1985, pp. 138–145.

[16] M. Bhatt, F. Dylla, J. Hois, Spatio-terminological inference for the design of ambient environments, in: International Conference on Spatial Information Theory, Springer, 2009, pp. 371–391.

[17] E. Remolina, J.A. Fernandez, B. Kuipers, J. Gonzalez, Formalizing regions in the spatial semantic hierarchy: An ah-graphs implementation approach, in: International Conference on Spatial Information Theory, Springer, 1999, pp. 109–124.

[18] F. Corradini, A. Polini, B. Re, L. Rossi, F. Tiezzi, Consistent modelling of hierarchical BPMN collaborations, Bus. Process. Manag. J. 28 (2) (2022) 442–460.

[19] C. Monsalve, A. April, A. Abran, Business process modeling with levels of abstraction, in: Conference on Communication and Computing, IEEE, 2015, pp. 13–15.

[20] C. Jensen, H. Lu, B. Yang, Graph model based indoor tracking, in: Mobile Data Management: Systems, Services and Middleware, ACM, 2009, pp. 122–131.

[21] L.J.R. Stroppi, O. Chiotti, P.D. Villarreal, Extending BPMN 2.0: Method and tool support, in: Business Process Model and Notation, in: LNBIP, Springer, 2011, pp. 59–73.

[22] D. Calvanese, M. Dumas, U. Laurson, F. Maggi, M. Montali, I. Teinemaa, Semantics, analysis and simplification of DMN decision tables, Inf. Syst. 78 (2018).

[23] D. Calvanese, M. Dumas, F.M. Maggi, M. Montali, Semantic DMN: Formalizing and reasoning about decisions in the presence of background knowledge, 2018, arXiv:1807.11615.

[24] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, Formalising and animating multiple instances in BPMN collaborations, Inf. Syst. 103 (2022) 101459.

[25] M. Hennessy, T. Regan, A process algebra for timed systems, Inform. and Comput. 117 (2) (1995) 221–239.

[26] W.P.d. Roever, J. Hooman, Design and verification in real-time distributed computing: an introduction to compositional methods, in: International Symposium on Protocol Specification, Testing and Verification, North-Holland Publishing, 1989, pp. 37–56.

[27] L. Aceto, A. Ingólfsdóttir, K.G. Larsen, J. Srba, Reactive Systems: Modelling, Specification and Verification, Cambridge University Press, 2007.

[28] G.D. Plotkin, A structural approach to operational semantics, J. Log. Algebraic Methods Program. 60–61 (2004) 17–139.

[29] R. Pugliese, F. Tiezzi, A calculus for orchestration of web services, J. Appl. Log. 10 (1) (2012) 2–31.

[30] P. Van Gorp, R. Dijkman, A visual token-based formalization of BPMN 2.0 based on in-place transformations, Inf. Softw. Technol. 55 (2) (2013) 365–394.

[31] Signavio, Signavio, 2018, http://www.signavio.com/.

[32] R.M. Dijkman, M. Dumas, C. Ouyang, Semantics and analysis of business process models in BPMN, Inf. Softw. Technol. 50 (12) (2008) 1281–1294.

[33] A. Kheldoun, K. Barkaoui, M. Ioualalen, Formal verification of complex business processes based on high-level Petri nets, Inform. Sci. 385 (2017) 39–54.

[34] G. Decker, R. Dijkman, M. Dumas, L. García-Bañuelos, Transforming BPMN diagrams into YAWL nets, in: International Conference on Business Process Management, in: LNCS, vol. 5240, Springer, 2008, pp. 386–389.

[35] P.Y. Wong, J. Gibbons, A process semantics for BPMN, in: International Conference on Formal Engineering Methods, in: LNCS, vol. 5256, Springer, 2008, pp. 355–374.

[36] E. Börger, B. Thalheim, A method for verifiable and validatable business process modeling, in: Advances in Software Engineering, in: LNCS, vol. 5316, Springer, 2008, pp. 59–115.

[37] R. Dijkman, P. Van Gorp, BPMN 2.0 execution semantics formalized as graph rewrite rules, in: International Workshop on Business Process Modeling Notation, in: LNCS, Springer, 2010, pp. 16–30.

[38] F. Corradini, A. Polini, B. Re, F. Tiezzi, An operational semantics of BPMN collaboration, in: C. Braga, P.C. Ölveczky (Eds.), Formal Aspects of Component Software - 12th International Conference, FACS 2015, Niterói, Brazil, October 14-16, 2015, Revised Selected Papers, in: Lecture Notes in Computer Science, vol. 9539, Springer, 2015, pp. 161–180.

[39] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A formal approach to modeling and verification of business process collaborations, Sci. Comput. Program. 166 (2018) 35–70.

[40] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, A. Vandin, A formal approach for the analysis of BPMN collaboration models, J. Syst. Softw. 180 (2021) 111007.

[41] F. Corradini, C. Muzi, B. Re, L. Rossi, F. Tiezzi, BPMN 2.0 OR-join semantics: Global and local characterisation, Inf. Syst. 105 (2022) 101934.

[42] M. Decker, H. Che, A. Oberweis, P. Stürzel, M. Vogel, Modeling mobile workflows with BPMN, in: Mobile Business and Mobility Roundtable, IEEE, 2010, pp. 272–279.

[43] X. Zhu, J. Recker, G. Zhu, F. Maria Santoro, Exploring location-dependency in process modeling, Bus. Process. Manag. J. 20 (6) (2014).

[44] J. Dörndorfer, C. Seel, A framework to model and implement mobile context-aware business applications, in: Modellierung 2018, Gesellschaft für Informatik, 2018, pp. 23–38.

[45] H.-H. Chiu, M.-S. Wang, Extending event elements of business process model for internet of things, in: Proceedings of International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing, IEEE, 2015, pp. 783–788.
[46] L. Poss, S. Schönig, Location-aware business process modeling and execution, Softw. Syst. Model. 24 (3) (2024) 37–67.
[47] S. Cheikhrouhou, S. Kallel, N. Guermouche, M. Jmaiel, Toward a time-centric modeling of business processes in BPMN 2.0, in: Proceedings of IIWAS, Association for Computing Machinery, 2013, pp. 154–163.
[48] S. Houhou, S. Baarir, P. Poizat, P. Quéinnec, L. Kahloul, A first-order logic verification framework for communication-parametric and time-aware BPMN collaborations, Inf. Syst. 104 (2022) 101765.
[49] F. Moller, C. Tofts, A temporal calculus of communicating systems, in: CONCUR Theories of Concurrency: Unification and Extension, vol. 458, Springer, 1990, pp. 401–415.
[50] F. Corradini, D. D'Ortenzio, P. Inverardi, On the relationships among four timed process algebras, Fund. Inform. 38 (4) (1999) 377–395.
[51] S. Mazhar, P. Wu, M. Rosemann, Designing complex socio-technical process systems – the airport example, Bus. Process. Manag. J. 25 (2018).
[52] P. Grefen, N. Brouns, H. Ludwig, E. Serral, Co-location specification for IoT-aware collaborative business processes, in: Information Systems Engineering in Responsible Information Systems, vol. 350, Springer, 2019, pp. 120–132.
[53] I. Maslov, S. Poelmans, Facilitating the comprehension of business process models for unexperienced modelers using token-based animations, Inf. Manage. 61 (5) (2024) 103967.
[54] T. Allweyer, S. Schweitzer, A tool for animating BPMN token flow, in: International Workshop on BPMN, in: LNBIP, vol. 125, Springer, 2012, pp. 98–106.
[55] B.M. Abdul, F. Corradini, B. Re, L. Rossi, F. Tiezzi, UBBA: unity based BPMN animator, in: Conference on Advanced Information Systems Engineering Forum, in: LNBIP, vol. 350, Springer, 2019, pp. 1–9.
[56] C. Muzi, L. Pufahl, L. Rossi, M. Weske, F. Tiezzi, Formalising BPMN service interaction patterns, in: The Practice of Enterprise Modeling, in: LNBIP, vol. 335, Springer, 2018, pp. 3–20.
[57] R. Braun, W. Esswein, Classification of domain-specific BPMN extensions, in: The Practice of Enterprise Modeling, in: LNBIP, vol. 197, Springer, 2014, pp. 42–57.