

# BPMN Profile for Operational Requirements

Conrad Bock<sup>a</sup>    Raphael Barbau<sup>a</sup>    Anantha Narayanan<sup>a</sup>

a. U.S. National Institute of Standards and Technology  
100 Bureau Dr, Stop 8260  
Gaithersburg, MD 20899-8260

**Abstract** An important aspect of systems and products is how they interact with their environment, including how they are operated. Behaviors external to systems usually involve people not trained in the details of how systems are designed and built, but who need to specify or at least understand the procedures they will be performing. People involved in external behaviors prefer different languages for specifying and learning about procedures than languages used by engineers designing the systems themselves. Significant inefficiency arises when these languages are not integrated. An emerging trend is for external behaviors to be defined in the Business Process Model and Notation, especially operational requirements, while system designs are specified in the Unified Modeling Language, or extensions to it, such as the Systems Modeling Language. This paper describes an integration of BPMN and UML as standardized by the Object Management Group, providing a detailed comparison of what the languages imply for physical systems and individual people involved in operation, maintenance, and other activities.

**Keywords** Process modeling; UML extension; BPMN.

## 1 Introduction

An important aspect of systems and products is how they interact with their environment, including how they are operated. For example, operating cars requires them to be fueled and driven under appropriate conditions. Maintaining cars follows procedures particular to each kind of car. These behaviors involve entities outside cars, but are critical to specifying them properly, because car designs must be consistent with their operation, maintenance, and other interactions with their environment. For example, some drivers might want to transport more people, which is an operational behavior that might be accommodated by updates to existing car designs. Changes in car designs might lead to changes in maintenance procedures, and some designs might be ruled out because these procedures become too expensive or time-consuming.

Behaviors external to systems (*external behaviors*) usually involve people not trained in the details of how systems are designed and built, but who need to specify or at least understand the procedures they will be performing. For example, car drivers typically are not car designers or engineers, but still expect to perform particular operational behaviors (*operational requirements*). Maintenance personnel know more about car designs, but also are not designers or engineers themselves, and need to understand maintenance procedures that are consistent with the details of car designs.

People involved in external behaviors prefer different languages for specifying and learning about procedures than languages used by engineers designing the systems themselves. Languages for external behaviors are more helpful when they focus on step-by-step procedures that happen to involve objects, rather than languages focusing on objects that happen to have behaviors, as in engineering. External behavior languages are better when they need little training to start using, even if they have additional depth to be learned over time. This is preferred to languages that require significant up-front training to use, as in engineering.

Languages for external behavior are used for the same systems as engineering languages, and this causes significant inefficiency when the languages are not integrated. Artifacts developed with these languages are part of the same overall system development processes, requiring frequent interaction between people using external behavior languages and engineers using system design languages. This is necessary to ensure external behaviors and system designs are consistent, as described above. When these languages are not integrated, comparison and cross-checking of external behaviors and systems designs is significantly impaired, resulting in errors and confusions leading to costly rework and ineffective discussions.

Integrating languages for external behavior and system design requires detailed comparison of what the languages imply for individual systems and people involved in operation, maintenance, and other activities (*language semantics* or “meaning”). In particular, integration identifies when combinations of elements appear different across the languages, but mean the same thing, and when combinations of elements appear the same across the languages, but mean different things. Fully integrated languages enable reliable comparison and cross-checking, because the implications for physical systems and people are clear regardless of which language is used.

An emerging trend is for external behaviors to be defined in the Business Process Model and Notation (BPMN) [OMG11b], especially operational requirements, while system designs are specified in Unified Modeling Language (UML) [OMG11d], or extensions to it, such as the Systems Modeling Language (SysML) [OMG12b], all standardized by the Object Management Group (OMG) [DPS<sup>+</sup>10]. BPMN is the most widely used modeling standard for enterprise-level processes, including manufacturing enterprises, and is designed for processes of many kinds, including engineering and manufacturing processes, despite the name. It provides a readily understandable notation for subject matter experts, including engineers, and also has a format usable by computers, enabling automated assistance with process design and implementation. UML is the most widely used modeling standard for information systems, and its SysML extension is the most widely used modeling standard for systems engineering.

Other work on integrating BPMN into UML cover only portions of BPMN before its major upgrade (BPMN 2), do not support round-trip translation, do not give access to complete formal transformations, and either do not address collaborating BPMN processes, or translate them incorrectly into UML when they do. The effort with widest coverage of BPMN concerns only common workflow patterns in the first

version of BPMN, before it had a standard computable format, does not provide formal transformations or support for collaborating processes, and does not define a UML extension for BPMN needed for round-trip translation [Whi04]. Other efforts covering narrower portions of BPMN describe only some of the formal transformations and do not provide access to the complete set, and are based on the first version of BPMN [KV06][MR09][AFD10]. The first of these defines a UML extension for BPMN, but does not describe the transformation to it, while the other two do not extend UML, preventing them from supporting round-trip translation. The second of these describes some translations of BPMN elements that require multiple UML elements, while the other two only cover one-to-one translations.

This paper describes an integration of BPMN 2 into UML developed by the authors and adopted as a standard by OMG [OMG13e], to facilitate the use of BPMN notation with UML-based languages, such as SysML. It uses OMG’s mechanisms for extending UML (profiles and transforms), covering most of them in the paper, with the rest available electronically. Section 2 reviews the architecture of BPMN, UML, and the integration presented in the paper. Sections 3 and 4 cover modeling of procedures and interactions between them, respectively, Section 5 outlines transforms between BPMN and the profiled UML, and Section 6 summarizes the paper. Section headings use BPMN terminology.

## 2 OMG Language Architecture and Integration

BPMN and UML are primarily graphical languages, but also have textual formats for automated processing and interchange between modeling tools. Both forms have rules about how they must be constructed (*language syntax*), and are ways of seeing these languages on computer screens or in files (*concrete syntax*). To keep graphical and textual syntax consistent, BPMN and UML are defined in a way that does not depend on how they appear on computer screens or in files (using *abstract syntax*), as illustrated in Figure 1 [Boc03]. These various kinds of syntaxes give the “grammars” of BPMN and UML. Graphical concrete syntax is designed for people (typically called “notation”), textual concrete syntax is designed for information systems (typically called “interchange format”), and abstract syntax is designed for keeping these consistent (typically called “metamodel”).<sup>1</sup> Modeling tools present graphics to users following the rules of notation, generate text files following the rules of interchange formats, and keep these consistent with metamodels.

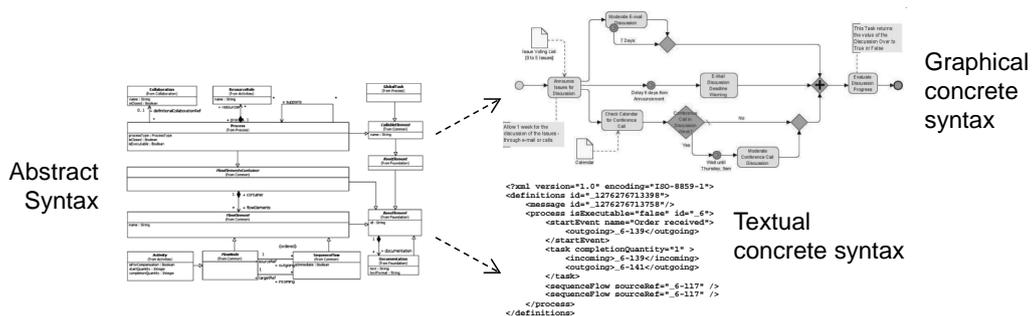


Figure 1 – Syntax

<sup>1</sup>Some graphical modeling languages have human-readable concrete textual syntaxes [ISO04, OMG13a]

BPMN and UML are designed for ease of use, employing syntactic shorthands to express complicated procedures in simple ways. These shorthands are defined by explaining in “long hand” what the shorthands imply for procedures as they are actually carried out (language semantics, see Section 1). The explanation starts with abstract syntax, as illustrated in Figure 2, to ensure graphical and textual concrete syntaxes are given the same meaning. Then the procedures that could be carried out according to models using the syntax are described informally or semi-formally, enabling implementers to see how the models should be interpreted.

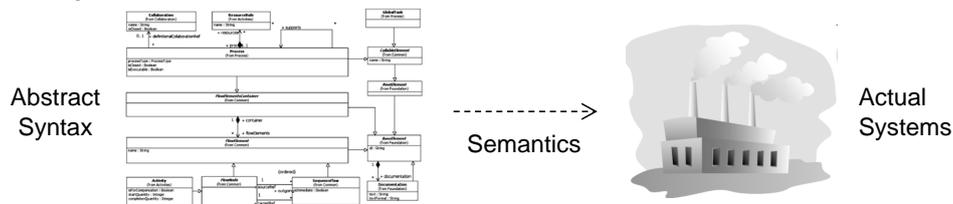


Figure 2 – Semantics

BPMN and UML were developed in parallel with significant communication between the communities involved. They overlap in modeling:

- Step-by-step procedures, where information and objects are passed between steps (BPMN processes and UML activities).
- Interactions between procedures or entities performing procedures (collaborations).

BPMN and UML have similarities and differences in syntax and semantics for the areas above, and the profile and transforms described in this paper address the differences as follows:

- BPMN or UML concrete graphical syntax notation can be used, but not both in the same diagram. Modelers defining external system behaviors will typically use BPMN, while modelers defining systems internals will typically use UML, see Section 1. UML has a standard notation for showing extensions in profiled models.
- UML semantics is extended in profiled models to be equivalent to BPMN semantics when combined with transforms between BPMN and UML. Equivalent semantics ensures that external system behaviors following BPMN process or collaboration diagrams will be carried out the same way whether the diagrams are captured using the BPMN metamodel or the profiled UML metamodel, as illustrated in Figure 3.

The profile and transforms do not change BPMN notation, metamodel, or semantics, and only change the notation, metamodel, and semantics of UML models when the extension is applied, not to all UML models.

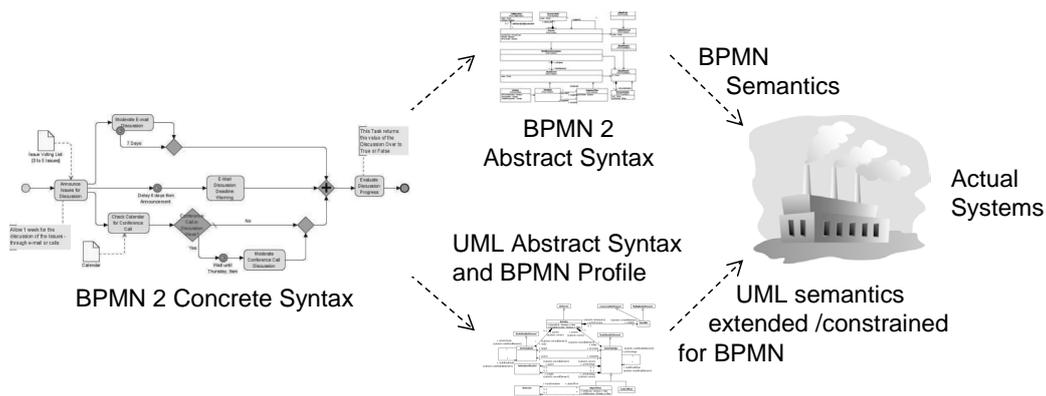


Figure 3 – Equivalent Semantics

BPMN and UML also have areas that do not overlap:

- UML supports detailed modeling of structure (UML class and internal structure), while BPMN does not.
- BPMN and UML support detailed modeling of the time order of interactions between entities without specifying their internal procedures (BPMN choreographies and UML interactions), but their syntaxes are very different.

The first area above is addressed in the profile because it extends UML for BPMN in the context of structure modeling capabilities of UML. The second area is not addressed in the profile.

Within the architecture above, languages can be integrated in at least two ways:

1. Extend the syntax and semantics of one language to accommodate another.
2. Transform models following the syntax of one language into the syntax of another.

The integration described in this paper uses both approaches above. It extends UML abstract syntax to accommodate BPMN concrete graphical syntax (via UML *profiles*), and defines *transforms* between models expressed in the syntaxes of BPMN and the extended UML. UML profiles are a technique for extending the UML metamodel by expanding or restricting UML abstract syntax and semantics as needed, see below in this section. Transforms are specifications of how to take models following the abstract or concrete textual syntax of one language and generate models following the abstract or concrete textual syntax of another. The transforms described in this paper operate in both directions between models following BPMN and profiled UML syntaxes.

The primary constructs in using profiles to extend UML are *stereotypes*, which

1. Add capabilities. For example, BPMN supports the capability to specify whether steps appearing in procedures might happen when the process is carried out, whereas UML does not. To align with BPMN, stereotypes in the profile add this capability when they are applied to UML models.
2. Constrain usage. For example, steps in UML procedures can start just by having their inputs available, whereas BPMN steps also require other steps or events to occur. To align with BPMN, stereotypes in the profile establish this constraint when they are applied to UML models.

3. Make distinctions. For example, BPMN distinguishes between many kinds of events that procedures notice and cause, whereas UML has more general categories. To align with BPMN, stereotypes in the profile are introduced to make these distinctions when applied to UML models.

Stereotypes identify elements in the UML metamodel they are extending (*base* elements, which are always *metaclasses*). Figure 4 shows the notation for extensions in the three examples above. The stereotype on the left is for procedures in BPMN (**BPMNProcess**), extending the element for procedures in UML (**Activity**). It adds the capability to specify whether additional steps might occur when UML activities are carried out that interact with other activities and are not specified in the model (**isClosed**), to align with BPMN. The stereotype in the middle is for steps in BPMN (**BPMNActivity**), extending the element for steps in UML (**Action**). It constrains actions to start after other actions or events occur, not just when inputs are available, to align with BPMN. The stereotypes on the right are for the kinds of events that procedures can notice and bring about in BPMN (**Event Definition** and its specializations), extending the corresponding element in UML (**Event**). These stereotypes are introduced to make distinctions between various kinds of events, to align with BPMN.

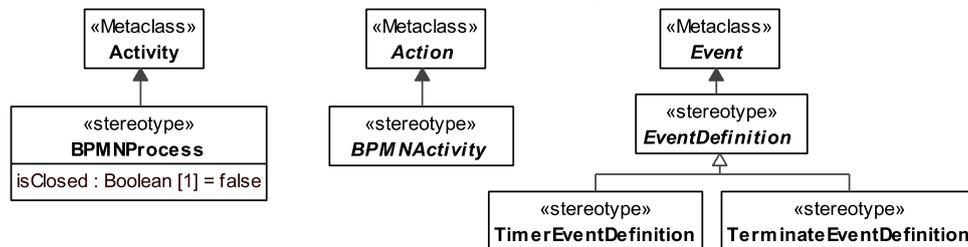


Figure 4 – Stereotypes

### 3 Processes

Procedure models in general determine when each step should start and what its inputs are [Boc99]. BPMN and UML follow traditional approaches by initiating steps according to when others finish (Sections 3.2 and 3.4), when inputs are available (Section 3.3), and when procedures receive things or when particular conditions come about (Section 3.5). BPMN and UML relate procedures and steps to other elements in an overall system, such as external stakeholders or system components (Section 3.6). BPMN and UML also distinguish procedures that specify detailed steps from those that do not, and between models of procedures from carrying out those procedures at particular times (Section 3.1).

#### 3.1 Processes and Global Tasks

BPMN and UML distinguish between procedures that specify steps to be taken in carrying them out (BPMN *processes* and UML *activities*) and procedures that are just given names with no further detail (BPMN *global tasks* and UML *opaque behaviors*). BPMN and UML call these *callable elements* and *behaviors*, respectively. Figure 5 shows how the profile models these correspondences. BPMN identifies various kinds of global tasks, depending on their intended usage (*manual* global tasks performed by people, *script* global tasks performed by computational procedures, *business rule*

global tasks performed by computational rules, and *user* global tasks defined by the modeler). The profile introduces stereotypes extending UML opaque behaviors to distinguish the various kinds of global tasks.

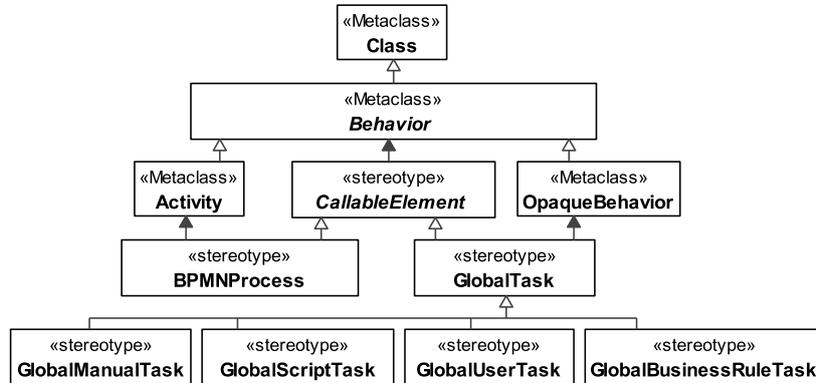


Figure 5 – UML Extension for BPMN Processes and Global Tasks

BPMN and UML distinguish between models of procedures (processes and activities) and procedures as they are carried out at any particular time (*instances* of processes and global tasks, of activities and opaque behaviors). Procedure models can be carried out multiple times, each with potentially different start and end times, and involving different things. For example, a procedure for building a car in a factory is carried out many times on different days, involving different cars, parts, and materials. Activities (and all UML behaviors) are special forms of the primary UML element for specifying instances (*classes*), as shown at the top of Figure 5. Each time a procedure is carried out (each process instance), if it is carried out in a way that follows the model, such as which step comes before which other (see Section 3.2), then the process instance is *valid*, otherwise it is *invalid*.

See Section 4 for other aspects of BPMN processes.

### 3.2 Activities and Sequence Flows

The steps specified by BPMN processes and UML activities are called *activities* and *actions*, respectively. The order in which they occur is specified by links from earlier steps to later ones (BPMN *sequence flows* and UML *control flows*), as illustrated in Figure 6. BPMN and UML have the almost the same notation in this example and both indicate that the first step finishes before the second begins. Figure 7 shows how the profile models these correspondences between BPMN and UML (see below about immediate sequence flows).

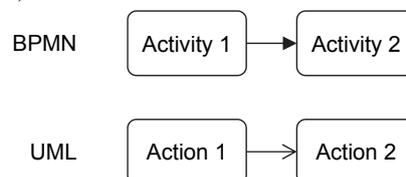


Figure 6 – BPMN activities and sequence flows, UML actions and control flows

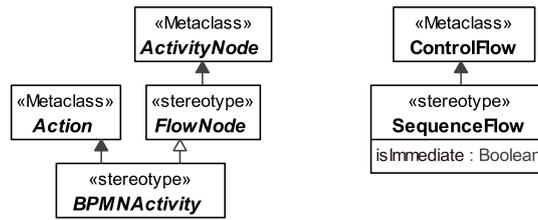


Figure 7 – UML extension for BPMN activities and sequence flows

Sequence and control flows indicate the order in which activities and actions should occur, respectively, but not the amount of time between them. For example, the second step in Figure 6 might need to wait for other inputs before starting, such as particular data or objects. Even if the step does not need to wait for other input, the flow from the first step does not specify whether there is any delay or not between finishing the first step and starting the second.

BPMN and UML distinguish between steps that

- specify further substeps to take and those that do not.
- use separate definitions of what they should do.

These give four combinations, as shown in Table 1. BPMN has a different notation for the two aspects above, and combines them to give four notations, while UML uses the same notation when there are no substeps, regardless of whether they are defined separately. BPMN and UML can show substeps expanded within a procedure, regardless of whether they are defined separately (the table omits these for brevity, except for UML on the upper right, which is the only notation it provides in this case). Figure 8 shows how the profile models these correspondences between BPMN and UML (see below about activity classes). BPMN identifies various kinds of tasks, depending on their intended usage, as it does for global tasks, see Section 3.1.

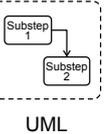
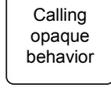
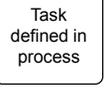
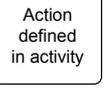
	Separately defined	Not separately defined
Substeps	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Calling separate process                        BPMN                 </div> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Calling separate activity                        UML                 </div> </div> <p>BPMN <i>call activities</i> and UML <i>call behavior actions</i> using separately defined processes and activities, respectively.</p>	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Subprocess defined in process                        BPMN                 </div> <div style="border: 1px dashed black; padding: 5px; text-align: center;">                       UML                 </div> </div> <p>BPMN <i>subprocesses</i> and UML <i>structured activity nodes</i> (a kind of action).</p>
No substeps	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Calling separate global task                        BPMN                 </div> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Calling opaque behavior                        UML                 </div> </div> <p>BPMN call activities and UML call behavior actions using separately defined global tasks and opaque behaviors, respectively.</p>	<div style="display: flex; justify-content: space-around;"> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Task defined in process                        BPMN                 </div> <div style="border: 1px solid black; padding: 5px; text-align: center;">                     Action defined in activity                        UML                 </div> </div> <p>BPMN <i>tasks</i> and UML <i>opaque actions</i>.</p>

Table 1 – BPMN and UML step refinement

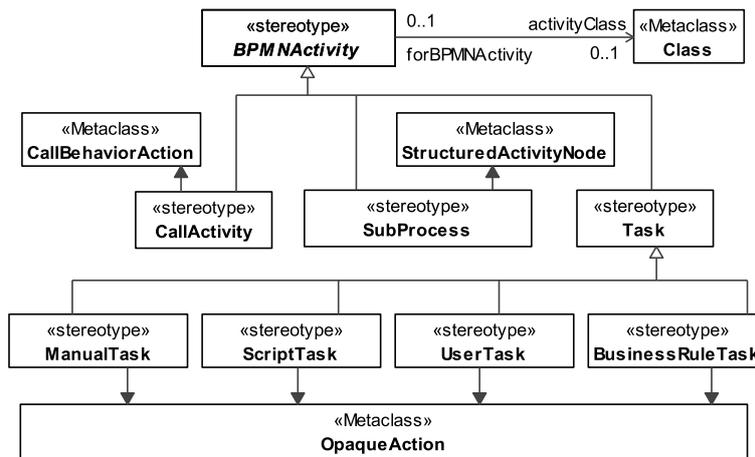


Figure 8 – UML extension for BPMN call activities, subprocesses, and tasks

BPMN distinguishes models of activities from activities as they are carried out at any particular time (*instances* of activities). Activities in process models can be carried out multiple times, each with potentially different start and end times, and involving different things. For example, an activity in a process model for building a car in a factory might specify painting the car. This activity is carried out many times on different days, involving different cars, and different paint. Each instance of the painting activity occurs in an instance of the process for building cars (see Section 3.1 about process instances). The profile supports this on the stereotype for activities using classes, the UML element for specifying instances (*activityClass* in Figure 8, see Section 3.1 about UML classes). Each time an action with the *Activity* stereotype applied is carried out, an instance of the corresponding class is created, providing semantics equivalent to BPMN's.

BPMN supports an option to specify whether activities not appearing in process diagrams may happen during sequence flows when the processes are carried out (*isImmediate*, see Section 3.1 about carrying out processes). In Figure 6 if the modeler indicates the sequence flow is immediate, then no other activities can happen between the first and second steps when the process is carried out, otherwise other activities can happen even though they are not specified in the diagram. This is useful for partially specified processes, such as process diagrams for review outside the organization performing them, see Section 4. Despite the name, immediate sequence flows do not specify whether there is a delay between activities or not. The profile supports this option on the stereotype for sequence flows (*isImmediate* in Figure 7), with an effect equivalent to BPMN on actions taken when activities are carried out.

See Section 4 for other aspects of BPMN activities in general, Section 3.5 for other aspects of BPMN subprocesses in particular, and Section 3.4 for more about steps with multiple flows coming into them.

### 3.3 Data

Steps in BPMN and UML procedures can accept inputs that were output from other steps or that were inputs to the procedure containing the step, and procedures can provide outputs that were outputs of steps they contain. Passing outputs and inputs is specified by links (BPMN *data associations* and UML *object flows*) from and to

elements indicating the kind of things being provided and accepted (BPMN *data objects* and UML *object nodes*), as illustrated in Figure 9. The arrows have an input or output at one end, rather than steps at both ends, indicating they are about things flowing, rather than order of steps (see Section 3.2). Despite the terms “data” and “object”, BPMN and UML support specification of informational and physical inputs and outputs.

Special kinds of elements are used for inputs and output of procedures, as compared to between steps. The top of Figure 9 has a BPMN data input for the process on the left, data output on the right, and data object in the middle. Data inputs and outputs of actions are not shown in BPMN diagrams even though they exist in the underlying model. This means the data object in the middle is neither a data input nor data output, but acts as a buffer for flow between activities. The bottom of Figure 9 has an input UML activity parameter node for the activity on the left, an object node in the middle, and an output activity parameter node on the right. Input and output pins for actions can be shown as small rectangles on actions, but are omitted in the figure to align with BPMN. The object node in the middle is a *data store node*, which is a kind of object node that is neither input or output, to align with BPMN.

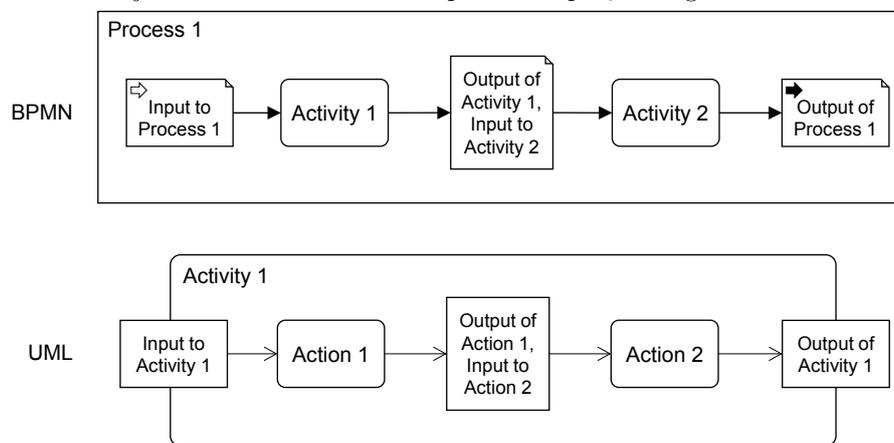


Figure 9 – BPMN and UML inputs and outputs

BPMN and UML specify the kinds of things that are input and output by referring to descriptions of them defined separately from procedures and steps (BPMN *item definitions* and UML *types*). BPMN processes and activities use the same elements to identify which kinds of things are inputs and which are outputs (*data inputs* and *data outputs*), while UML uses different ones (*parameters* on opaque behaviors, *activity parameter nodes* referring to parameters on activities, and *input pins* and *output pins* on actions). BPMN and UML call these *item aware elements* and *typed elements* respectively. Figure 10 shows how the profile models these correspondences between BPMN and UML, as well as those in Figure 9. UML classes are kind of type, and are used for structural as well as behavioral modeling, because structural elements have instances also (see Section 3.1 about UML classes and instances). For example, when a process model for building a car in a factory is carried out, each process instance will take different part instances as input to its assembly steps. BPMN has specialized data associations for inputs and outputs that are omitted from Figure 10 for brevity.

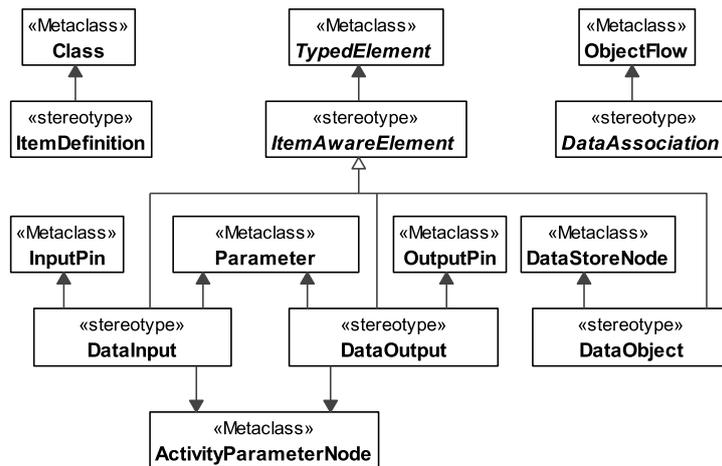


Figure 10 – UML extension for BPMN inputs and outputs

### 3.4 Gateways

The order in which steps occur in BPMN and UML procedures can be specified in more detail using BPMN *gateways* and UML *control nodes*. A step (BPMN activity or UML action) can:

- be followed by
  - multiple other steps at the same time (BPMN *parallel* gateway and UML *fork* node), as in the left column of Table 2 (steps are at the open ends of the arrows, not shown for brevity).
  - one other step among multiple possible steps, based on conditions (BPMN *exclusive* gateway and UML *decision* node), as in the middle column of Table 2.
  - some steps among multiple possible steps, based on conditions (BPMN *inclusive* gateway and UML fork node with conditions), as in the right column of Table 2.

	Followed by multiple	Followed by one among multiple	Followed by some or all among multiple
<b>BPMN</b>	<p>Parallel</p>	<p>Exclusive</p>	<p>Inclusive</p>
<b>UML</b>	<p>Fork</p>	<p>Decision</p>	<p>Fork</p>

Table 2 – BPMN diverging gateways and UML control nodes

- follow multiple other steps causing
  - one execution of the step (BPMN parallel gateway and UML *join* node), as in the left column of Table 3.
  - one execution of the step under modeler-specified conditions (BPMN *complex* gateway and UML *join* node with join specification), as in the middle column of Table 3.
  - multiple executions of the step (BPMN exclusive gateway and UML *merge* node), as in the right column of Table 3.

	Single execution	Single execution under condition	Multiple executions
<b>BPMN</b>	 Parallel	 Complex	 Exclusive
<b>UML</b>	 Join	 Join	 Merge

Table 3 – BPMN converging gateways and UML control nodes

Figure 11 shows how the profile models these correspondences between BPMN and UML.

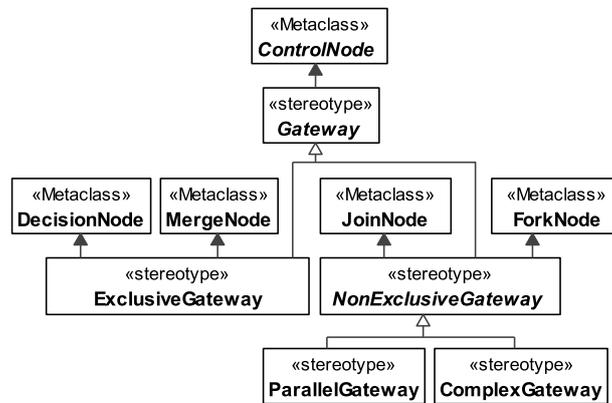


Figure 11 – UML extension for BPMN gateways

Multiple flows coming into the same step have different effects in BPMN and UML, equivalent to using different gateways and control nodes. Multiple sequence flows into the same BPMN activity typically cause the activity to be executed multiple times, once for each sequence flow, as if an exclusive gateway were used. Multiple control flows into the same UML action cause the action to be executed once, after all actions on the other ends of the control flows are finished, as if a join were used. To align BPMN and UML semantics, BPMN and profiled UML models are transformed into each other as illustrated in Figure 12. Multiple sequence flows coming into the same

activity (on the top left of the figure) is equivalent to multiple control flows coming into a UML merge node, followed by another control flow to an action (on the bottom left). The merge node provides semantics equivalent to the BPMN model on top left by starting the action once for each incoming control flow. Similarly, multiple control flows coming into the same action (on the lower right of the figure) is equivalent to multiple sequence flows coming into a BPMN parallel gateway, followed by another sequence flow to an activity (on the top right). The parallel gateway provides semantics equivalent to the UML model on bottom right by starting the activity once when all control flows arrive.

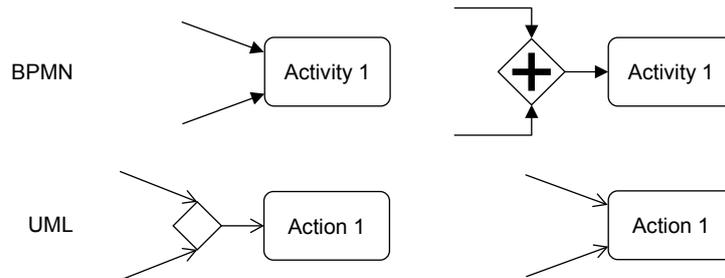


Figure 12 – Transforms for multiple flows into activities and actions

### 3.5 Events

BPMN and UML procedures can detect when something arrives for them or when particular conditions come about, and can send things to other procedures or cause conditions to come about. These capabilities require special kinds of steps (“event steps”) that ask if things have arrived or conditions have come about (BPMN *catch events* and UML *accept event actions*, or UML control nodes in some cases), and send things to other procedures or to themselves and cause conditions to come about that other procedures notice (BPMN *throw events* and UML *call operation actions*). Procedures cannot notice that things arrive or conditions come about unless they have event steps specific to those things or conditions.

The simplest BPMN events and UML control nodes are those for the beginning of a series of steps (BPMN *start events*, a kind of catch event, and UML *initial nodes*) and those for the end (BPMN *end events*, a kind of throw event, and UML *flow final nodes*), as shown by the circular symbols in Figure 13 on the left and right, respectively. End events and flow final nodes only indicate the end of a series of steps, rather than the entire procedure. End events can be augmented to do more than this, see below.

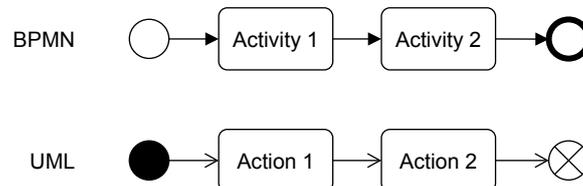


Figure 13 – BPMN start and end events, UML initial and flow final nodes

More complicated BPMN events and UML accept event actions refer to elements defined separately from the procedures, to enable multiple procedures to refer to the same kind of arrival or condition (BPMN *event definitions* and UML *events*, detected by catch events and accept event actions referring to them, respectively). BPMN uses

these same definitions to send things to other procedures and cause conditions to come about that other procedures notice (throw events), whereas the profile uses calls to operations having these effects, which are noticed by accept event actions referring to events identifying calls to the operation being detected (*call* events), except for terminating activities, see below.

Event steps that do more than indicate the beginning and end of a series of steps include those that:

- Stop procedures in which they occur (BPMN *terminate* event definitions and UML activity final nodes, which are kinds of control nodes, see Section 3.4), as shown by the notations in Figure 14. These kinds of events and control nodes in BPMN subprocesses and UML structured activity nodes only stop the subprocesses or structured activity node they are in, rather than the entire process or activity. There are no corresponding event steps in BPMN or UML to detect when procedures are stopped this way (compare to detecting errors, see below).

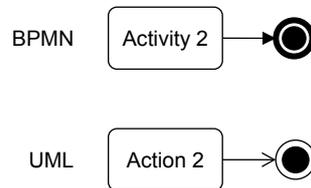


Figure 14 – BPMN termination end event and UML activity final node

- Notify the caller of procedures about situations that might need some special response (BPMN *error* and *escalation* event definitions and UML operations corresponding to these, detected by accept event actions for their call events), as shown by the notations in Figure 15. Modeler-defined objects are sent with the BPMN events, including error and escalation codes if needed, and passed to the corresponding UML operations. If the caller is not concerned with the event (does not have a step asking for it), then the notification goes to the caller of the caller, and so on. For error events, procedures that are not concerned with the event are stopped. In Figure 15, the procedure will continue after the escalation to the second step, but will stop completely after the error at the end, even if other steps are happening concurrently. When these kinds of events occur in BPMN subprocesses, or operations are called in UML structured activity nodes, they first notify the entire process or activity rather than their caller, and for errors stop the subprocesses or structured activity node containing them.

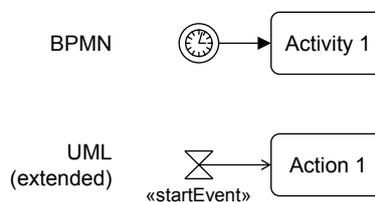


Figure 15 – BPMN escalation and error events, UML call operations

- Specify conditions that might come about, including timing (BPMN *conditional* and *timer* event definitions and UML *change events*). These can be used in the middle of a procedure to wait for conditions to come about. For example, Figure 16 shows a procedure that waits for a particular time or elapsed time after the first step and before the second, and waits for a particular condition after the second step and before the third.

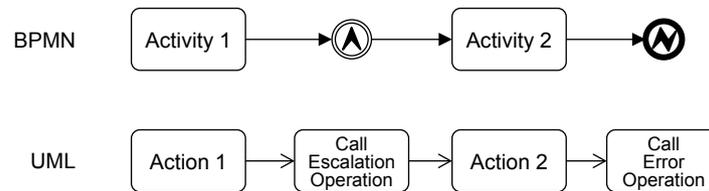


Figure 16 – BPMN timer and condition events, UML accept time and change event actions

- Start procedures and subprocesses based on arrivals or conditions coming about (BPMN start events with event definitions and extended UML accept event actions detecting call events or change events). For example, Figure 17 shows a procedure that starts at a particular time. The profile supports this with extended semantics in the **Process** stereotype (see Section 3.1) that instantiates activities when an event is detected by an accept event action with **StartEvent** applied. Start events can be used this way to start subprocesses in a process already being carried out, with an option to stop the rest of the procedure when this happens. The profile supports this with extended semantics in the **SubProcess** stereotype (see Section 3.1) that terminates the rest of the activity if this option is chosen.

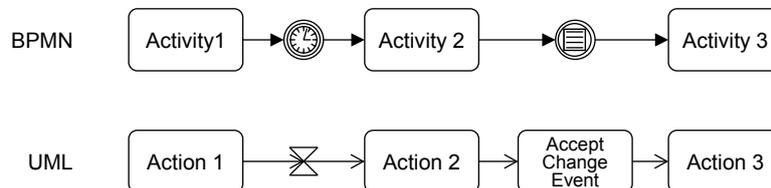


Figure 17 – BPMN start event with event definition, UML extended accept event action

- Send and receive things from other procedures, see Section 4.

Figure 18, Figure 19, and Figure 20 show how the profile models these correspondences between BPMN and UML. Figure 18 is for elements within the flow of procedures, while Figure 19 and Figure 20 are for the elements referred to outside procedures and sharable between them (boundary events and the extending UML accept event actions for BPMN start events are described below). BPMN conditional event definitions and UML change events refer to model-defined expressions of conditions. For BPMN timer event definitions, the expression in UML change events is predefined by the profile to be “BPMNTimerEvent” to indicate that the semantics aligns with BPMN’s by detecting the arrival of a particular time with the `timeDate` property, the passing of a particular amount of time with the `timeDuration` property, and the arrival of a recurring time with the `timeCycle` property, whichever has a value.

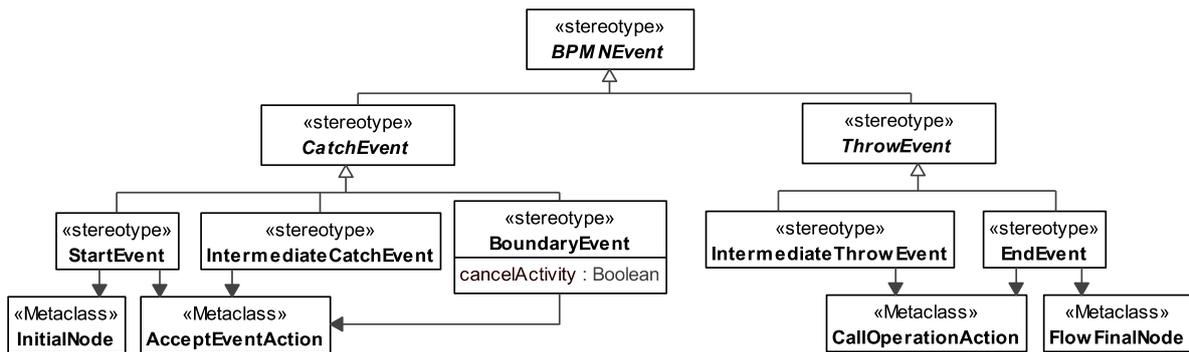


Figure 18 – UML extension for BPMN events

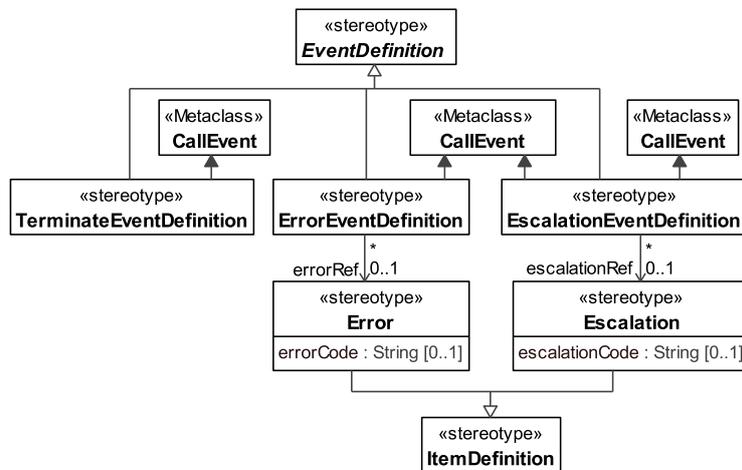


Figure 19 – UML call event extensions for BPMN event definition

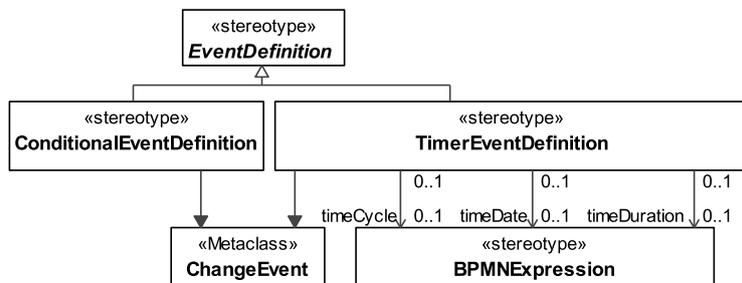


Figure 20 – UML change event extensions for BPMN event definition

BPMN and UML procedures can detect arrivals and conditions while particular steps are occurring, with BPMN providing a compact notation for doing this during a single activity (*boundary* events), as shown at the top of Figure 21. When the process starts the activity, it also begins waiting for the error to occur and the condition to come about, but only for as long as the activity is happening. If the error occurs during that time before the timing condition comes about, the activity is stopped and the process continues along the sequence flow going out of the error. If the timing condition comes about before the error occurs, the activity continues, while the process also continues along the sequence flow going out of the timer. If neither event occurs

during the activity, the procedure stops waiting for the events, and continues along the sequence flow going out of the activity as usual, not along either of the other sequence flows. This is equivalent to the UML construction shown at the bottom of Figure 21. The action is surrounded by a rounded, dashed rectangle (*interruptible* region), with accept event actions for the error and timing condition, and control flows exiting the region, some shown as zigzag lines (*interrupting* edges). The interrupting region starts the accept event actions when the step is started, and the interrupting edges stop the actions in the region when they are traversed. The timer does not have an interrupting edge going out of it because it does not stop the action when it comes about. This construction provides semantics equivalent to the BPMN model on the top.

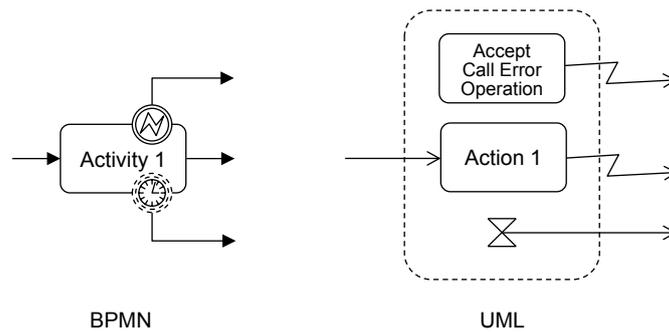


Figure 21 – UML transform for BPMN boundary events

BPMN and UML procedures can take different steps depending on what things arrive for them or which conditions come about, with BPMN providing an expanded graphical notation for this (BPMN *event-based* gateways), as shown at the top of Figure 22. When the process reaches the gateway, it begins waiting for the events after it to occur, in this case, two conditions, one being a timer. When one of the events occurs, the procedure stops waiting for the others, and continues along the sequence flow going out of the error that occurred. This is equivalent to the UML construction shown at the bottom of Figure 22. The accept event actions are in an interruptible region, with a fork starting them all at once. The condition to come about stops the other action, providing semantics equivalent to the BPMN model on the top. Figure 23 shows how the profile adds an additional stereotype to Figure 11 for event-based gateways.

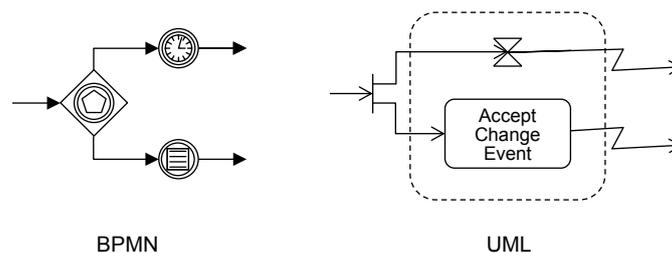


Figure 22 – Transform for BPMN event-based gateways and UML interruptible regions

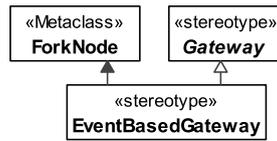


Figure 23 – UML extension for BPMN event-based gateways

### 3.6 Lanes and Resources

Steps in BPMN and UML can be grouped according to relationships with other elements, such as resources needed to perform the steps. Diagrams are divided into rectangular sections (BPMN *lanes* and UML activity *partitions*), indicating steps inside each rectangle have the same relationship to some other element the modeler chooses. Figure 24 shows an example of steps grouped into lanes and partitions. Diagrams can have multiple top-level groups, for example, showing the performers and locations of performance with horizontal and vertical rectangles (BPMN *lane sets* and UML *dimensions*). Figure 25 shows how the profile models these correspondences between BPMN and UML. BPMN uses a metaclass for lane sets, whereas UML uses a metaproperty for dimensions, which has a value of true in the profile for top-level lane sets.

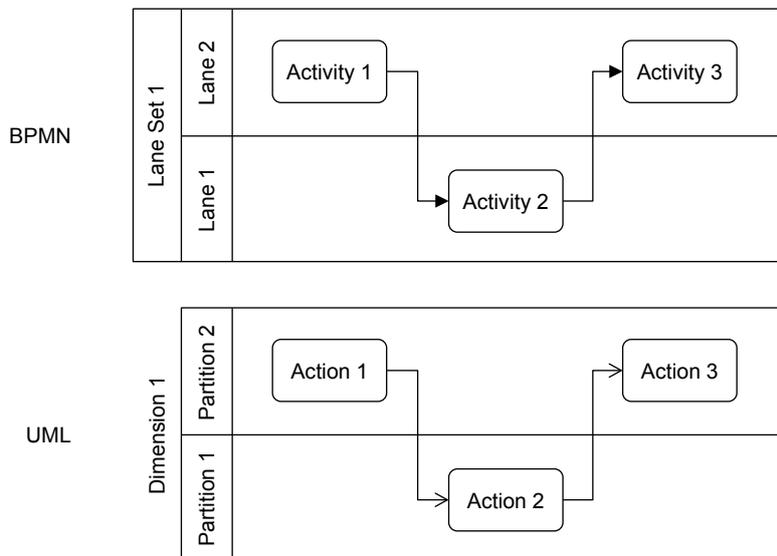


Figure 24 – BPMN lanes and lane sets, UML activity partitions

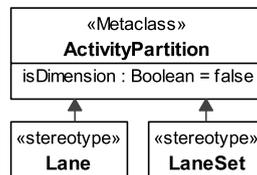


Figure 25 – UML extension for BPMN lanes and lane sets

The relationships between process elements in a lane and other parts of the model are often based on resources used by the process elements. BPMN has a kind of item definition for resources and supports specification of roles they play in processes, global tasks, and activities (*resources* and *resource roles*, respectively). These are supported by the profile as shown in Figure 26. Item definitions are an extension of UML classes (see Section 3.3 about item definitions), and classes can specify elements that have different values in each instance (*properties*, a kind of UML typed element, see Section 3.3 about typed elements and classes as types). Processes and global tasks in the profile are indirect extensions of classes (see Section 3.1), while activities in the profile are used with classes (see Section 3.2 about activity classes), enabling them to have properties, including resource role properties, as well as values in their instances. Lanes can divide activities according to the expected values of resource role properties, as specified by their types or default values. BPMN and the profile support a kind of resource role specifically for those performing processes, global tasks, and activities.

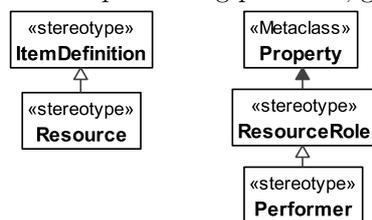


Figure 26 – UML extension for BPMN resources and resource roles

### 3.7 Loop Characteristics

BPMN and UML have elements for repeated performance of the same step without using flows that cycle back to the same step (BPMN *loop characteristics* and UML *loop nodes* and *expansion regions*). BPMN and UML support this in two ways, the first for sequential execution (BPMN *standard* loop characteristics and UML loop node), and the second for parallel execution and other capabilities (BPMN *multi-instance* loop characteristics and UML expansion regions). BPMN loop characteristics apply to a single step, adding repetition semantics to that step, whereas UML loop nodes and expansion regions are structured activity nodes containing separate actions to be repeated. The profile supports this by transformation, as shown in Figure 27 (only multi-instance loop characteristics are shown because UML loop nodes do not have notation). Expansion regions support BPMN's capability to repeat the steps in multi-instance loops on each element of a collection. The expansion region is inside a structured activity node along with an accept event action and activity final node (see Section 3.5), to support BPMN's capability to stop multi-instance loops when some condition comes about. BPMN multi-instance loops can also throw events under certain conditions each time an activity finishes being repeated (*implicit* events). The profile supports this by transformation, with additional actions and control nodes in the expansion region after the action being repeated. Figure 27 shows the case of multiple possible conditions and events after each repetition. Multi-instance loops and expansion regions have the option to repeat their steps in parallel, notated as in Figure 27, or sequentially (notated in BPMN as a marker with horizontal lines and in UML with the «iterative» keyword). Figure 28 shows how the profile models these correspondences between BPMN and UML.

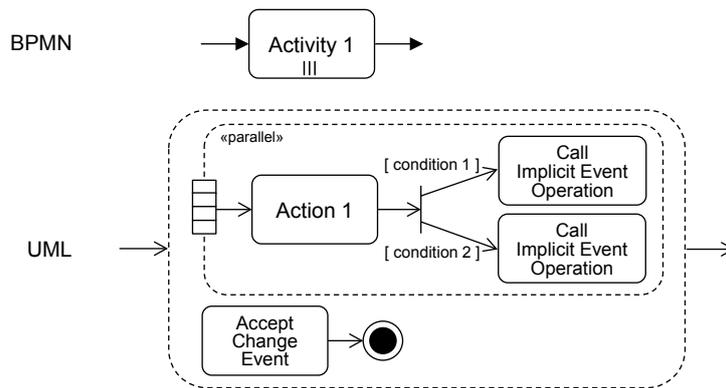


Figure 27 – BPMN loop characteristics and UML expansion regions

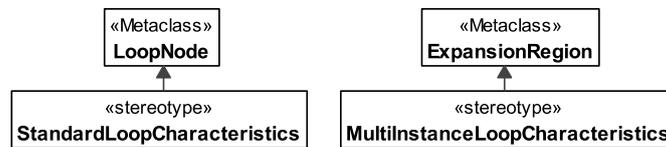


Figure 28 – UML extension for BPMN loop characteristics

## 4 Collaborations

### 4.1 Participants, Messages, and Message Flows

BPMN and UML collaborations specify the roles played by procedures involved in collaborations (BPMN *participants* and UML collaboration *roles*), as shown in Figure 29. BPMN collaborations specify *message flows* sent between participants, while the profile extends UML *information flows* to have equivalent semantics to BPMN’s (BPMN and UML support exchange of physical things, despite the names of these elements). Message flows and information flows refer to elements defined separately from collaborations to specify the kind of things exchanged (BPMN *messages* and UML *information items*). This enables exchanges to send the same kind of thing between participants, possibly in multiple collaborations. Figure 30 shows how the profile models these correspondences between BPMN and UML. Messages are specified by a kind of item definition (see Section 3.3 about item definitions), because they are just things that happen to be sent as messages. See below about closed collaborations, and Sections 4.3 and 4.4 about BPMN participants as extensions of UML properties.

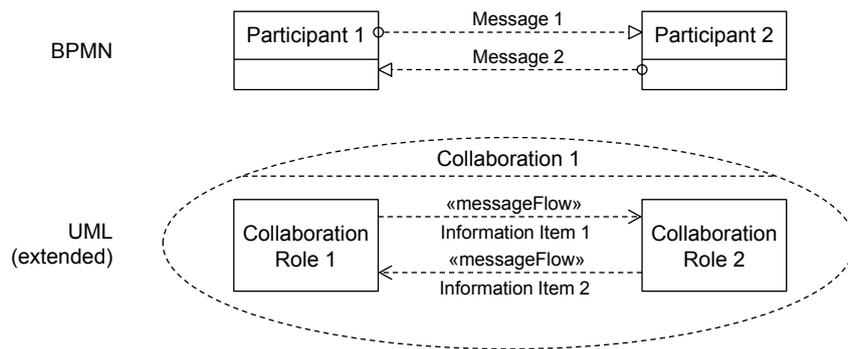


Figure 29 – BPMN and extended UML collaborations

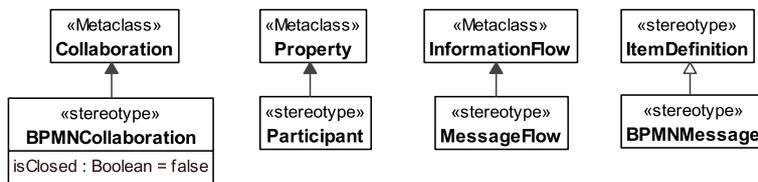


Figure 30 – UML extension for BPMN collaborations

BPMN supports an option to specify whether messages not appearing in collaboration diagrams may happen when the collaborations are carried out (*closed* collaborations). In Figure 29, if the modeler indicates the collaboration is closed, then only the two messages shown can flow between the participants when the collaboration is carried out, otherwise other messages can flow even though they are not specified in the diagram. This is useful for participants that want to review and approve interactions before collaborations are carried out. The profile supports this option on the stereotype for collaborations (*isClosed* in Figure 30), with an effect equivalent to BPMN on information items flowing when collaborations are carried out. See Section 4.5 for more about closed collaborations.

## 4.2 Conversations

BPMN and UML manage complicated exchanges between participants by grouping them together, as shown in Figure 31 (BPMN *call conversations* and UML *collaboration uses*). The two exchanges in Figure 29 are grouped into a single element in between the participants. The messages in Figure 31 are defined in a separate collaboration (not shown for brevity) that is called or used in the original collaboration, by analogy with call activities and call behavior actions, see Section 3.2. BPMN distinguishes collaborations that specify participants and message flows from collaborations that are just given names with no further detail (*global* conversations), while UML does not (the profile extends UML for this, see below).

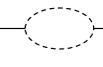
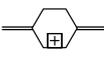
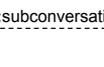
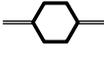
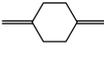
	Separately defined	Not separately defined
<b>Sub-exchanges</b>	 BPMN  UML BPMN <i>call conversations</i> and UML <i>collaboration uses</i> using separately defined collaborations.	 BPMN  UML BPMN <i>subconversations</i> and extended UML <i>information flows</i> .
<b>No sub-exchanges</b>	 BPMN  UML BPMN <i>call conversations</i> and UML <i>collaboration uses</i> using separately defined <i>global conversations</i> .	 BPMN  UML BPMN <i>conversations</i> and extended UML <i>information flows</i> .

Table 4 – BPMN and UML collaboration refinement

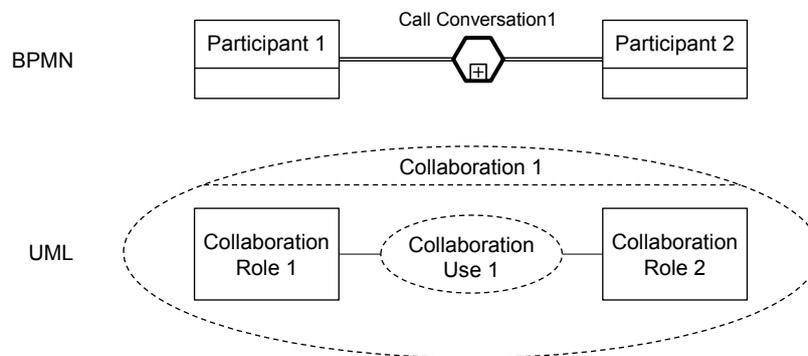


Figure 31 – BPMN call conversations and UML collaboration

BPMN supports other kinds of conversations that do not require separately defined collaborations, by analogy with subprocesses and tasks in processes (*subconversations* and *conversations*), whereas UML does not. The profile supports the other BPMN conversations as extensions of information flows. With the previous option to have detail within a collaboration or not, this gives four combinations, as shown in Table 4, by analogy with the activity combinations in procedures, see Table 1 in Section 3.2. BPMN supports the visual expansion of conversations into their message flows, whereas UML does not. The profile supports this with extended information flows (see Section 4.1).

Figure 32 shows how the profile models these correspondences between BPMN and UML. The profile extends information flows for all conversations, including call conversations, to simplify the model. It also extends information flows and collaborations to identify the message flows and conversations within them, respectively. Global conversations are modeled as a special kind of collaboration that has no participants or message flows, as it is in BPMN by analogy with global tasks.

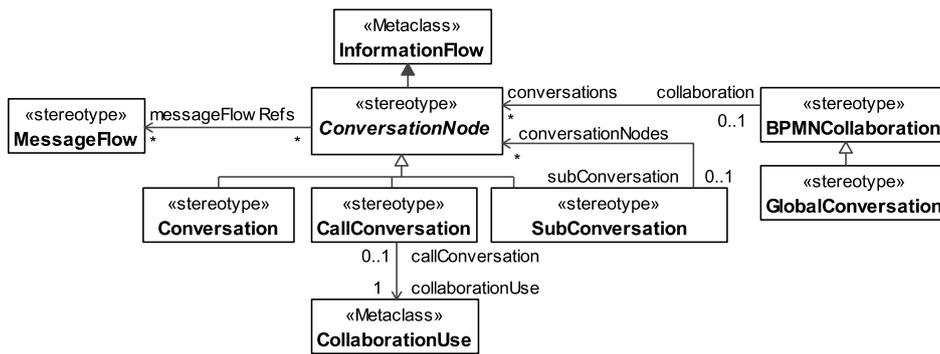


Figure 32 – UML extension for BPMN conversations

### 4.3 Collaborations with Procedures

BPMN activities can be shown in collaborations to specify the order and conditions for flows between processes, using activities and events that send and receive messages (*send* and *receive* tasks, and throw and catch events with *message event* definitions), as in Figure 33. The two message flows in Figure 29 of Section 4.1 follow the order of the send and receive tasks and message events in the processes of the participants. UML concrete syntax does not support showing activities in collaborations, but its abstract syntax does through collaboration roles played by activities, and this is enough to support the integration described in this paper (see Section 2 about syntax and integration). Collaboration roles in UML are properties of collaborations, which can specify that the values of the properties are instances of activities (activity instances playing roles in collaborations, see Section 3.6 about properties and their values, and Section 3.2 about activity instances). The profile supports message flows between activities through information flows (see Section 4.1), which can link any model elements, including actions in activities playing different roles in collaborations.

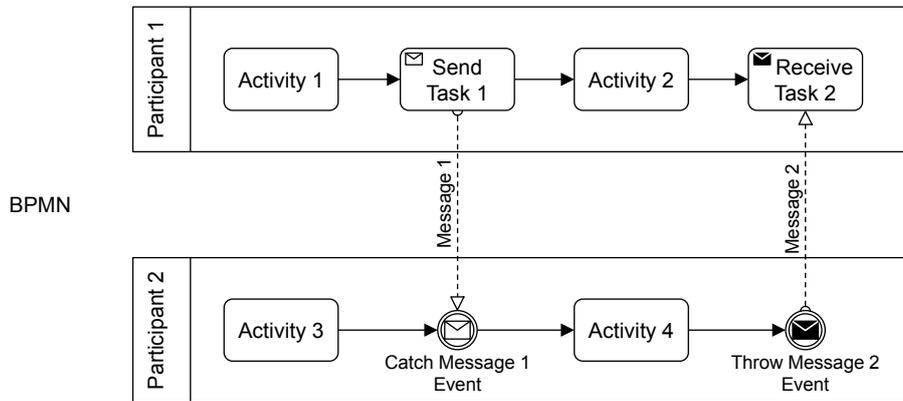


Figure 33 – BPMN collaboration with processes

BPMN conversations can be used in collaborations showing processes, as shown in Figure 34. The two message flows in Figure 33 are grouped into one conversation, as in Figure 31, but with links to the send and receive tasks at the ends of the message flows. Multiple conversations can be used between participants to show that the grouped exchanges are for different purposes. The profile supports this with extended

information flows (see Section 4.2), to identify message flows within conversations, including message flows between actions.

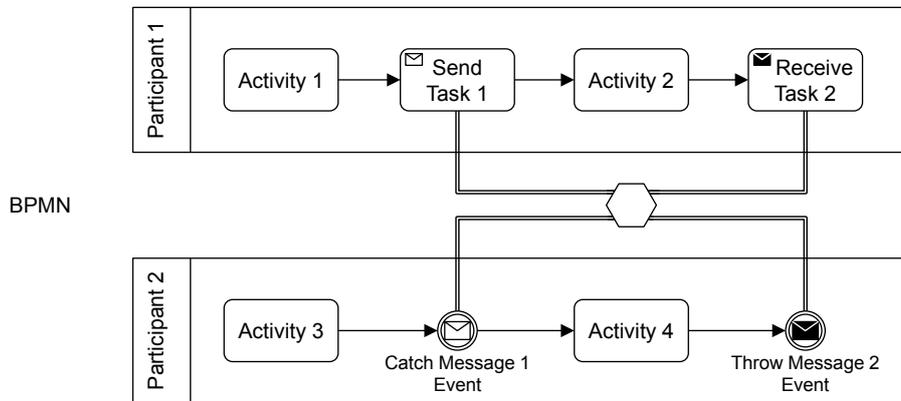


Figure 34 – BPMN collaboration with processes and conversations

Figure 35 shows how the profile models the correspondences between BPMN send and receive tasks, message event definitions, and UML actions. Send and receive tasks are based on the same UML elements as throw and catch events in Figure 18 of Section 3.5, because send and receive tasks use messages to specify the kind of thing being sent or received, similarly to the message sent and received by events with message event definitions.

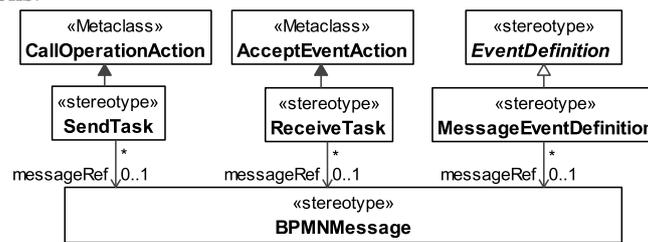


Figure 35 – UML extension for BPMN send and receive tasks and message event definitions

#### 4.4 Partner Roles and Partner Entities

Sometimes roles in collaborations are played by those responsible for the procedures, such as organizations or people performing or supervising the procedures, rather than being played by procedures directly. BPMN and UML collaborations can specify which kinds of things are responsible for the procedures (BPMN *partner roles* and UML classes) or can specify particular individuals or organizations, such as a particular company or person (BPMN *partner entities* and UML *instance specifications*). BPMN and UML do not notationally distinguish these ways of specifying participants and roles, but capture them in their underlying models, see Figure 36. The profile supports participants specifying partner roles as the kinds of things that can be values of the collaboration role properties (play the roles, see Section 4.3 about participants as extensions of UML properties, and compare to activities laying collaboration roles). Participants specifying partner entities are supported through instance specifications as default values of the collaboration role properties.

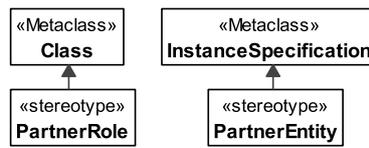


Figure 36 – UML extension for BPMN partner roles and partner entities

#### 4.5 Public and Private Processes

BPMN distinguishes process models that are available to everyone (*public*) from process models that are only available inside organizations carrying them out (*private*, see Section 3.1 about carrying out processes). Public process models usually omit proprietary elements, and often only show activities and events that exchange messages outside the organization. Private process models include proprietary elements, as well as public activities and events. Carrying out private processes will make organizations appear from the outside as if they were carrying out the corresponding public processes. The top of Figure 37 shows a process model that could be a public version of the one carried out by the first participant in Figure 29 in Section 4.1. The bottom of Figure 37 shows another process model that could be private for the public one at the top. Carrying out this private process will have the same external effect as carrying out the public one, because messages are sent and received in the same order and under the same conditions. The profile supports the distinction between public and private process models on the stereotype for processes (see `processType` in Figure 38 below).

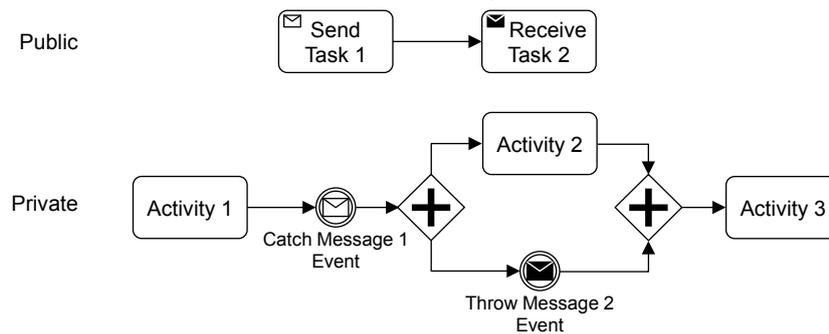


Figure 37 – BPMN public and private processes

BPMN modelers can indicate when private process models are intended to appear from the outside as public process models (*supports*). BPMN does not have a notation for this, but has abstract syntax. When one process model supports others, all the instances valid for the process model are also valid for the others (see Section 3.1 about valid process instances). UML has a construct for linking activities that indicates all valid instances of one activity are valid instances of another (*generalization*), which is equivalent semantics to BPMN's.

Figure 38 shows how the profile models the correspondences for public, private, and closed process models. The profile does not have a stereotype for process model support because BPMN does not have a metaclass for it. It is covered by the transforms.

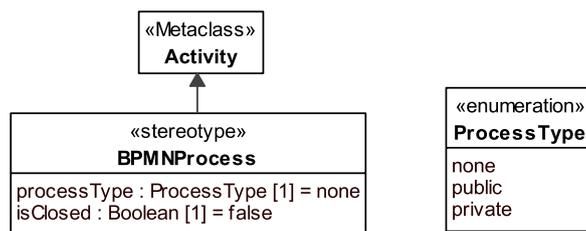


Figure 38 – UML extension for BPMN public, private, and closed processes

## 4.6 Correlation

BPMN supports management of messages when a participant carries out multiple process instances at the same time (*correlation*, see Section 3.1 about process instances). Participants must route messages to the particular process instances that are interacting, based on content of the messages (*correlation keys*). For example, in Figure 33 of Section 4.3, the first participant might be carrying out multiple instances of its process, each sending out its own version of the first message, such as ordering particular parts. When the second participant replies, the second message should go (correlate) to same process instance which sent the first message, rather than other instances being carried out. The first participant identifies the correct process instance from correlation keys in the second message.

Correlation keys are groups of elements (*correlation properties*) that are given particular values in each message. For example, a correlation key in the first message in Figure 33 might have a property for an order number, which is unique to each process instance in the first participant. The second participant includes this key in the second message, and the first participant uses it to identify the process instance that should receive the message. Correlation keys must be shared between messages, because they are used to correlate replies to the original sending process instances.

The profile supports correlation with the stereotypes shown in Figure 39. Classes are extended for correlation keys, using UML properties for correlation properties (see Section 3.1 about UML classes and instances, Section 3.6 about classes and properties). The stereotype for correlation keys is used by those for collaborations and conversations to specify keys for all the messages flowing in particular collaborations or conversations, respectively.

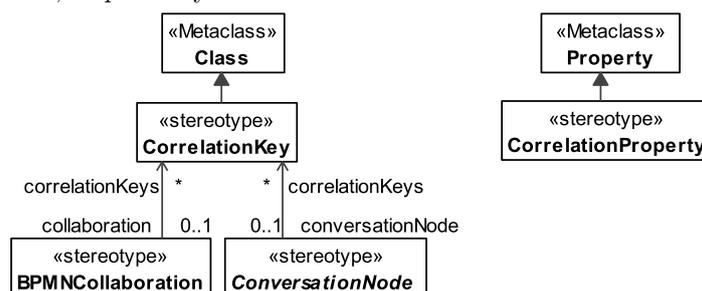


Figure 39 – UML extension for BPMN correlation

## 5 Transforms

The integration of BPMN and UML described in this paper includes transforms between models following the syntax of BPMN and models following the syntax of profiled UML (see Section 2). The transforms cover areas of integration that profiles cannot support, in particular, where BPMN or UML have equivalent semantics, but use combinations of elements that do not match one-to-one, such as the ones in Figure 12 of Section 3.4. Transforms achieve equivalent semantics by adapting models rather than the semantics of the languages involved.

The transforms in the integration operate on BPMN and profiled UML models following abstract and textual concrete syntaxes (see Section 2). Transforms between models following abstract syntaxes have the benefit of being independent of textual concrete syntax, avoiding the complications of textual grammars, but require additional mappings between abstract syntax and textual concrete syntax to produce interchangeable files (mapping from abstract to concrete textual syntax is not addressed in this paper). Transforms between textual concrete syntaxes enable translation of models without mappings from abstract syntax, but are tied to particular textual grammars.

The languages used for the two kinds of transforms reflect the syntaxes of the models they operate on:

- Abstract syntax transforms are expressed in the Query / View / Transformation Relational language (QVT-R) [OMG11c], an OMG specification for transforms between metamodels specified in the Meta-Object Facility [OMG13c], a subset of UML for specifying OMG metamodels, such as BPMN and UML.
- Concrete textual syntax transforms are expressed as eXtensible Stylesheet Language Transformations (XSLT) [W3C99], which operate on text files expressed in the eXtensible Markup Language (XML) [W3C08]. BPMN and UML interchange files are expressed in XML, but they use XML differently. BPMN's interchange files must conform to its XML Schema (BPMN XSD) [W3C12, OMG10b], while UML's interchange files must conform to OMG's XML Model Interchange specification (XMI) when applied to the UML metamodels (UML XMI) [OMG13d, OMG11e].<sup>2</sup>

The integration includes transformations expressed in both QVT-R and XSLT because QVT-R provides more compact, readable transform specifications, but does not currently have tool support for carrying out the transforms automatically, while XSLT has tool support, but is verbose and difficult to read. This is due in part to QVT-R's transforms in each direction being very similar to each other, while XSLT requires very different transforms for each direction. QVT-R is more declarative and consequently more compact, but more difficult for tools to support, whereas XSLT is imperative, but verbose and easier for tools to support. QVT has an imperative variant that some tools support, but also has the same readability disadvantages as XSLT, and is not used in the integration.

Section 5.1 describes example abstract syntax transforms between BPMN and profiled UML using QVT-R, while Section 5.2 covers concrete syntax transforms using XSLT (the complete transforms are available at [OMG13e]).

---

<sup>2</sup>BPMN has an XMI interchange format, but is not used in the integration.

## 5.1 QVT-R Transforms

Transforms expressed in QVT-R consist of *relations* that map between model elements expressed in BPMN and UML. Each relation specifies an element to match in a BPMN or UML model, a new element to create in a new UML or BPMN model, values for attributes in the new elements, and other relations that hold on the new element. Each relation also specifies conditions and other relations that must hold before the relation is applied.

QVT-R specifications designate one relation as the first used during the transformation (*top* relation). In Listing 1 the top relation transforms from BPMN definitions to UML packages. It calls two other relations, which happen to have mutually exclusive conditions. The first of these is defined in Listing 1 (*DefinitionsToPackage*). It matches BPMN definitions, and creates a corresponding package in the UML model. The relation is applied only in the case when the matching BPMN Definition has no extensions in it. This condition is in the *when* clause, expressed in OMG's Object Constraint Language [OMG12a]. The other relation called from the top (*DefinitionsToProfile*, definition not shown) is only applied when the BPMN Definition has extensions, and creates a corresponding profile in the UML model. The rest of the relations from BPMN to UML cascade from the two relations called from the top, because the other model elements are contained in definitions and packages.

```

top relation TopDefinitionsToPackage
{
  checkonly domain bpmn _definitions : Definitions { }
  enforce domain uml _package : uml::Package { }
  where {
    DefinitionsToPackage( _definitions, _package );
    DefinitionsToProfile( _definitions, _package );
  }
}

relation DefinitionsToPackage
//map Definitions to Package if no Extensions are present
{
  checkonly domain bpmn _definitions : Definitions {
    name = _name : String,
    rootElement = _re : RootElement{ },
    extensions = _extensionsSet : Set(Extension) };
  enforce domain uml _package : uml::Package {
    name = _name,
    packageableElement = _pe : PackageableElement{ },
    profileApplication = _profileApplication : uml::ProfileApplication {
      appliedProfile = bpmnprofile } };
  //apply stereotype
  enforce domain uml _umlDefinition : bpmnprofile::Definition {
    basePackage = _package };
  where {
    RootElementToPackageableElement( _re, _pe );
    when {
      _extensionSet->size() = 0;
    }
  }
}

```

Listing 1 – Example QVT-R transform from BPMN to profiled UML

The relation in Listing 2 maps BPMN Send Tasks to UML Call Operation Actions. When a Send Task is matched in the input BPMN model, a new Call Operation Action is created in the output UML model. If a BPMN Operation is referenced, a corresponding Operation is created in the UML model, and attached to the Call Operation Action. The *where* clause in the rule ensures that the appropriate stereotype is attached to the referenced operation.

```

relation SendTaskToCallOperationAction
//map BPMN SendTask to UML CallOperationAction
{ checkonly domain bpmn _sendTask : SendTask{
    name = _name : String,
    operationRef = _operaitonRef : BPMNOperation };
enforce domain uml _callOperationAction : uml::CallOperationAction {
    name = _name,
    operation = _operation : uml::Operation { } };
//apply stereotype
enforce domain uml _bpmnSendTask : bpmnprofile::SendTask {
    base_CallOperationAction = _callOperationAction };
where { OperationToOperation( _operationRef, _operation ); }
}

```

Listing 2 – Example QVT-R transform from BPMN to profiled UML

## 5.2 XSLT Transforms

Transforms expressed in XSLT match portions of XML files and generate new XML files based on the matches. XSLT is also an XML-based language, enabling transforms to look similar to the XML they are matching and generating.

The following challenges arise in XSLT transforms between BPMN and UML:

- Many elements of BPMN and UML interchange files reference other elements using identifiers, resulting in two difficulties:
  - To resolve the reference, XSLT must be written to search the file for an element that has the identifier. This is particularly difficult to write and read when a set of multiple elements with cross references between them must be matched.
  - References in BPMN interchange files can include names with a prefix giving the namespace of the owning file. This enables BPMN models to refer to elements in different files. In UML’s interchange files, it is also possible to refer to elements in other files, but only using the physical location of the file instead of namespaces. BPMN files that associate multiple physical locations to a single namespace cannot be transformed to UML files.
- UML’s interchange file conventions allow the same model to be interchanged with slightly different XML.<sup>3</sup> For example, some data can be represented as an XML attribute or as an XML text node. The transformation from UML to BPMN must account for these differences in case the UML file being transformed was created by a UML modeling tool rather than an earlier transformation from a BPMN file.
- Most BPMN elements transformed into UML require a UML element and a stereotype instance applied to the UML element. UML elements and stereotype instances appear very far apart in the interchange files. As a result, it is necessary to parse the BPMN file twice, once to generate UML elements, and once to generate stereotype instances.

<sup>3</sup>OMG is finalizing new XMI constraints that reduce the allowed differences between UML interchange files. The transformations work with the new constraints, but do not require them [OMG13b].

The example in Listing 3 shows XSLT transforms from BPMN to UML interchange files. The first template (`SendTaskWrapTemplate`) matches send tasks. It creates a node element with parameters from matching the BPMN file (`@id`) and a constant (`uml:CallOperationAction`), generating an ID, UUID, and a type for the XMI element. Then it calls a second template (`SendTaskTemplate`), which adds a reference to an operation if one is present, and it creates an input pin that serves as target for the a call operation action. Adding the reference transforms qualified names from BPMN into unqualified names for internal references or URIs for external references.

```
<xsl:template name="SendTaskWrapTemplate">
  <node>
    <xsl:call-template name="addAttributes">
      <xsl:with-param name="id" select="@id" />
      <xsl:with-param name="type" select="'uml:CallOperationAction'" />
    </xsl:call-template>
    <xsl:call-template name="SendTaskTemplate" />
  </node>
</xsl:template>

<xsl:template name="SendTaskTemplate">
  <xsl:if test="@operationRef">
    <operation>
      <xsl:call-template name="addReference">
        <xsl:with-param name="ref" select="@operationRef" />
      </xsl:call-template>
    </operation>
  </xsl:if>
  <target>
    <xsl:call-template name="addAttributes">
      <xsl:with-param name="type" select="'uml:InputPin'" />
      <xsl:with-param name="suffix" select="'target'" />
    </xsl:call-template>
    <upperBound xmi:type="uml:LiteralUnlimitedNatural">*</upperBound>
  </target>
  <xsl:call-template name="TaskTemplate" />
</xsl:template>
```

Listing 3 – Example XSLT transform from BPMN to profiled UML interchange files

The example in Listing 4 shows XSLT transforms from UML to BPMN interchange files, specifically send tasks from call operation actions. The `basenode` is the call operation action node. When a call operation is matched, the XSLT needs to look for a `SendTask` stereotype instance in the XMI file and check whether it refers to the call operation action. The XSLT function `matchStereotype` can resolve the reference regardless of whether it is serialized as XML element or as XML attribute. If the call operation action has a `SendTask` stereotype applied, various attributes of the stereotype and of the base node are added to the send task. The templates `AddAsAttribute`, `AddStereotypeReferenceAsAttribute`, and `AddReferenceAsAttributes` retrieve values serialized as elements or attributes, and add them as attributes in the BPMN file. Another template (`TaskTemplate`) is called afterwards to add attributes common to any task.

```
<xsl:template name="CallOperationActionTemplate">
  <xsl:param name="basenode" />
  <xsl:param name="packagenode" />
  <xsl:for-each select=
    "/xmi:XMI/BPMNProfile:SendTask[BPMNProfile:matchStereotype(.,
    'CallOperationAction',_,$basenode/@xmi:id)]">
```

```

<bpmn:sendTask>
  <xsl:call-template name="AddAsAttribute">
    <xsl:with-param name="namein" select="'implementation'" />
  </xsl:call-template>
  <xsl:call-template name="AddStereotypeReferenceAsAttribute">
    <xsl:with-param name="namein" select="'messageRef'" />
  </xsl:call-template>
  <xsl:call-template name="AddReferenceAsAttribute">
    <xsl:with-param name="nodein" select="$basenode" />
    <xsl:with-param name="namein" select="'operation'" />
    <xsl:with-param name="nameout" select="'operationRef'" />
  </xsl:call-template>
  <xsl:call-template name="TaskTemplate">
    <xsl:with-param name="basenode" select="$basenode" />
  </xsl:call-template>
</bpmn:sendTask>
</xsl:for-each> ...

```

Listing 4 – Example XSLT transform from profile UML to BPMN interchange files

The XSLT transforms are validated on example BPMN files available from OMG [OMG10a, OMG11a] as follows:

- Concrete syntax check. Valid BPMN interchange files must conform to XML and to the BPMN XSD. Valid UML files, including profiled UML, must conform to XML and to UML XMI. BPMN files can be checked with off-the-shelf software, while UML files can be checked with the NIST Validator [Nat13]. This level of checking ensures the XML elements and attributes have the correct names and are used properly.
- Abstract syntax check. BPMN and UML abstract syntax specifies constraints on the valid models that cannot be represented in XML schema or in XMI. UML specifies these constraints in OCL, and models can be checked against them with the NIST validator.
- Round trip check. Transforming from valid BPMN interchange files to profiled UML files, then transforming these back to BPMN, gives a valid BPMN file. The original BPMN file is at least a subset of the resulting BPMN file, with additional elements possibly added in the UML file, such as identifiers (BPMN identifiers are optional, but aren't in UML, the transformation adds them).<sup>4</sup>

To illustrate the validation, the example in Listing 5 is taken from the “Travel Booking” BPMN file. It shows a send task that has two incoming sequence flows. This fragment passes the BPMN concrete and abstract syntax checks by conforming to the BPMN XSD and to additional syntax constraints in BPMN.

```

<semantic:sendTask messageRef="_1275940518200"
  name="Notify_Customer_to_Start_Again" id="_6-541">
  <semantic:incoming>_6-699</semantic:incoming>
  <semantic:incoming>_6-705</semantic:incoming>
  <semantic:outgoing>_6-592</semantic:outgoing>
</semantic:sendTask>

```

Listing 5 – Example portion of BPMN interchange file

<sup>4</sup>The resulting files might not be completely valid because UML or BPMN require some information that is not present or optional in the other language. The XSLT transforms do not attempt to fill in missing information, but transform at the same level of completeness as the original file.

Applying the XSLT transforms from BPMN to UML to the fragment in Listing 5 results in a portion of UML interchange file shown in Listing 6, as illustrated in Figure 12 of Section 3.4. The send task is transformed to a call operation action. A merge node and control flow are inserted so that the two control flows point to the merge node, and the merge node is connected to the call operation action using a new control flow. At the end of the example, the `SendTask` stereotype is applied to the call operation action. The result in Listing 6 passes UML concrete and abstract syntax checks, according to the NIST Validator.

```

<node xmi:id="_6-541"
  xmi:uuid="http://example.com/definitions/_1275940517919#_6-541"
  xmi:type="uml:CallOperationAction">
  <target xmi:id="d1e100_target"
    xmi:uuid="http://example.com/definitions/_1275940517919#d1e100_target"
    xmi:type="uml:InputPin">
    <upperBound xmi:type="uml:LiteralUnlimitedNatural">*</upperBound>
  </target>
  <name xmi:type="uml:String">Notify Customer to Start Again</name>
  <outgoing xmi:idref="_6-592"/>
  <incoming xmi:idref="_6-541_from_merge"/>
</node>
<node xmi:id="_6-541_merge"
  xmi:uuid="http://example.com/definitions/_1275940517919#_6-541_merge"
  xmi:type="uml:MergeNode">
  <incoming xmi:idref="_6-699"/>
  <incoming xmi:idref="_6-705"/>
  <outgoing xmi:idref="_6-541_from_merge"/>
</node>
<edge xmi:id="_6-541_from_merge"
  xmi:uuid="http://example.com/definitions/_1275940517919#_6-541_from_merge"
  xmi:type="uml:ControlFlow">
  <target xmi:idref="_6-541"/>
  <source xmi:idref="_6-541_merge"/>
  <guard xmi:type="uml:LiteralBoolean">true</guard>
  <weight xmi:type="uml:LiteralUnlimitedNatural">1</weight>
</edge>

```

Listing 6 – Example portion of UML interchange file generated by transforming Listing 5

Applying the XSLT transforms from UML to BPMN to the fragment in Listing 6 produces the original elements in Listing 5, passing the round trip check.

## 6 Summary

This paper describes the integration of commonly used languages for system specification (UML and its extensions, such as SysML) with a language for specifying system environment behaviors, including how systems are operated, maintained, and repaired (BPMN). The integration enables personnel involved in external behaviors to specify and learn these behaviors in a language suited to them (BPMN), and engineers to specify system internals in their preferred languages (UML and its extensions), without incurring the inefficiency of unintegrated languages. It extends UML's underlying model to accommodate BPMN notation, and defines transforms between UML and BPMN in cases where extension is not sufficient. The paper covers the specification of individual processes, such as particular kinds of repair (see Section 3), and of interaction of multiple processes, such as the interaction of maintenance teams (see Section 4). The paper provides a detailed comparison of the semantics of the languages

to ensure that models have the same effect regardless of the language used (see Section 2), and gives examples of transforms between UML and BPMN (Section 5).

## References

- [AFD10] M. Arganaraz, A. Funes, and A. Dasso. An MDA approach to Business Process Model Transformations. *Journal of Informatics and Operations Research*, 9(1):24–48, 2010.
- [Boc99] C. Bock. Three Kinds of Behavior Model. *Journal Of Object-Oriented Programming*, 12(4):36–39, July/August 1999.
- [Boc03] C. Bock. UML without Pictures. *IEEE Software Special Issue on Model-Driven Development*, 20(5):33–35, September/October 2003. doi:10.1109/MS.2003.1231148.
- [DPS<sup>+</sup>10] F. Dandashi, B. Pridemore, S. Semy, J. Valentine, and B. Yost. Operationalizing the tactical edge framework (tef) for service provisioning analysis. Technical report, The MITRE Corporation, March 2010.
- [ISO04] ISO. *The EXPRESS language reference manual*, July 2004. URL: [http://www.iso.org/iso/home/store/catalogue\\_ics/catalogue\\_detail\\_ics.htm?csnumber=38047](http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=38047).
- [KV06] A. Kalnins and V. Vitolins. Use of UML and Model Transformations for Workflow Process Definitions. In *Proceedings of International Baltic Journal Of Object-Oriented Programming Conference on Databases and Information Systems*, July 2006.
- [MR09] O. Macek and K. Richta. The BPM to UML activity diagram activity diagram transformation using XSLT. In *Proceedings of the International Workshop on Databases, Texts, Specifications, and Objects*, pages 119–129, 2009.
- [Nat13] National Institute of Standards and Technology. *NIST Validator*, December 2013. URL: <http://validator.omg.org>.
- [OMG10a] Object Management Group. *BPMN 2.0 by Example*, June 2010. URL: <http://www.omg.org/spec/BPMN/20100601/10-06-02.pdf>.
- [OMG10b] Object Management Group. *BPMN 2.0 XML Schema*, May 2010. URL: <http://www.omg.org/spec/BPMN/20100501/BPMN20.xsd>.
- [OMG11a] Object Management Group. *BPMN 2.0 examples, machine readable files*, January 2011. URL: <http://www.omg.org/spec/BPMN/20100602/2010-06-03>.
- [OMG11b] Object Management Group. *Business Process Model and Notation*, January 2011. URL: <http://www.omg.org/spec/BPMN/2.0>.
- [OMG11c] Object Management Group. *Meta Object Facility 2.0 Query/View/Transformation Specification*, January 2011. URL: <http://www.omg.org/spec/QVT/1.1>.
- [OMG11d] Object Management Group. *OMG Unified Modeling Language, Superstructure*, August 2011. URL: <http://www.omg.org/spec/UML/2.4.1>.

- [OMG11e] Object Management Group. *Unified Modeling Language Superstructure XMI*, July 2011. URL: <http://www.omg.org/spec/UML/20110701/Superstructure.xmi>.
- [OMG12a] Object Management Group. *Object Constraint Language*, January 2012. URL: <http://www.omg.org/spec/OCL/2.3.1>.
- [OMG12b] Object Management Group. *OMG Systems Modeling Language*, June 2012. URL: <http://www.omg.org/spec/SysML/1.3>.
- [OMG13a] Object Management Group. *Action Language for Foundational UML (Alf) Concrete Syntax for a UML Action Language*, March 2013. URL: <http://www.omg.org/spec/ALF/1.0.1/Beta3>.
- [OMG13b] Object Management Group. *Canonical XMI*, August 2013. URL: <http://www.omg.org/spec/XMI/CanonicalXMI/Beta2>.
- [OMG13c] Object Management Group. *Meta Object Facility*, June 2013. URL: <http://www.omg.org/spec/MOF/2.4.1>.
- [OMG13d] Object Management Group. *MOF 2 XMI Mapping Specification*, June 2013. URL: <http://www.omg.org/spec/XMI/2.4.1>.
- [OMG13e] Object Management Group. *UML Profile for BPMN 2 Processes*, June 2013. URL: <http://www.omg.org/spec/BPMNProfile/1.0/Beta1>.
- [W3C99] W3C. *XSL Transformations*, November 1999. URL: <http://www.w3.org/TR/1999/REC-xslt-19991116>.
- [W3C08] W3C. *Extensible Markup Language*, November 2008. URL: <http://www.w3.org/TR/2008/REC-xml-20081126>.
- [W3C12] W3C. *XML Schema*, April 2012. URL: <http://www.w3.org/standards/techs/xmlschema>.
- [Whi04] S. White. *Process Modeling Notations and Workflow Patterns. BPTrends*, March 2004. URL: <http://www.bptrends.com/process-modeling-notations-and-workflow-patterns>.

## About the authors



**Conrad Bock** is a Computer Scientist at the U.S. National Institute of Standards and Technology's Engineering Laboratory, specializing in formal product and process modeling. He was the founding editor for Activity and Action modeling in the Unified Modeling Language and Systems Modeling Language at the Object Management Group, as well as a primary contributor to interaction modeling in the Business Process Model and Notation. He can be reached at [conrad dot bock at nist dot gov](mailto:conrad_dot_bock_at_nist_dot_gov).



**Raphael Barbau** obtained a M.Sc. and a Ph.D. in Computer Science at the University of Burgundy. He is currently a Guest Researcher at the National Institute of Standards and Technology. His research interests include semantic web, product lifecycle management, long-term preservation, and systems engineering. He can be reached at raphael dot barbau at nist dot gov.



**Anantha Narayanan** is a researcher at the University of Maryland, USA. He is currently working as a guest researcher at the National Institute of Standards and Technology (NIST). His research interests are in domain specific modeling, model transformations, systems engineering, product life cycle modeling, and sustainable manufacturing. Anantha has a B.Tech. in Mechanical Engineering from IIT, Madras, India, and an M.S. and Ph.D. in Computer Science from Vanderbilt University. He can be reached at anantha dot narayanan at nist dot gov.

**Acknowledgments** The authors thank Fatma Dandashi for leadership in motivating this work, and comments on this paper.

Commercial equipment and materials might be identified to adequately specify certain procedures. In no case does such identification imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.