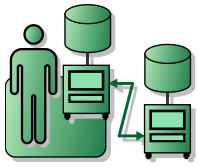
Unified Architecture Method

Technical Perspective Language UML Profile

Contents

Ottawa, Ontario, Canada

Introduction



Similar to the Business and Logical Perspectives, the Technical Perspective is used to describe, at the technical level, *how* the system or (or enterprise or business domain) operates (activities and processes), *what* is involved in these Processes (entities), *where* the system has a presence (locations), and *who* interacts with the business processes (roles and actors). It optionally includes technical level specifications such as server technology, network equipment, storage solutions and desktop makes and models—in other words, the technical standards for the system (or enterprise). It is an adaptation of the logical level models to more accurately represent the specific technology and structural choices made.

The human interaction, the technical roles defined and their interaction with the system, are associated with specific sets (classes) of people having specific positions within the system or enterprise along with prescribed accesses to applications and data. Security solutions, such as RBAC and other policy enforcement solutions are defined based upon these technical roles.

The Technical Perspective is used by stakeholders and system designers to understand how the systems currently work (if in an *as-is* form) including the technology used, and to analyze the effect of changes to the technical level of the business (if in a *to-be* form). The IT architect is responsible for the structure and integrity of the model, while system designers and other IT specialists may assist in detailing technical elements, patterns, and standards within the models.

The Technical Perspective builds upon the Logical Perspective through the addition of more elements as well as additional attributes of these elements. Specific technologies or families of technologies may be specified; however, the Technical Perspective is not intended to be a design level set of models, it is still an abstraction albeit (depending upon the architectural context and objectives) a minor abstraction (especially if applied at the project level). Enterprise level architectures may only define classes of technologies and patterns for their use—the enterprise technology standards for example.

Again there are the four aspects defined, namely the Technical Entity, Technical Process, Technical Locations and Technical Roles Viewpoints. The models within these viewpoints describe the Data, Activity, Location, and People aspects of the IT architecture, respectively. (Figure 1)

These viewpoints show how people interact with processes at various locations within the business, and the things they handle and use (Technical Entities). The viewpoints also show how these different models and the elements within them relate to one another, both statically (i.e., the static view of a process as shown in a process diagram) and dynamically (i.e., a dynamic view of a process as shown in a sequence diagram), to produce the desired business results. This perspective places emphasis on the structure of Entities, Processes, Locations, and Roles, and their responsibilities and interactions. The models show all collaborations (relationships) needed to support business activities, and all classes (the static view of entities) from which objects (operational instances) will be instantiated to obtain the desired business results for the various Roles. Computational aspects, such as *storage* and *server* and *database* are included at the technical level, with the additional ability to specify makes and models for this technology.

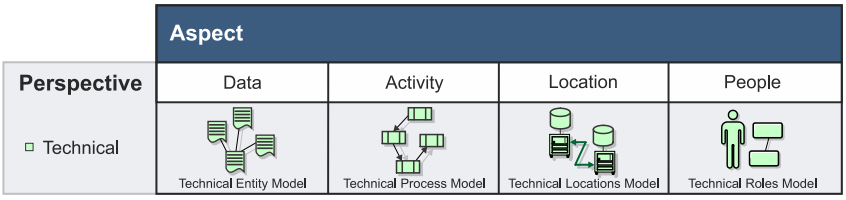


Figure 1 – Technical Perspective Viewpoints

The Technical Perspective is a set of physical level models with the level of detail being dictated

by the architectural context and objectives. The models may still be relatively abstract if one is describing a complete enterprise, or they may be very detailed if describing a small system or component. Either way, specific technologies and patterns for their usage are optionally defined. At the enterprise level these will define the corporate technology standards and optionally patterns for their use. At the small system level, the models will be very close to an implementation level view, but they will still be an abstraction from the implementation level—some details will not be provided.

The UML profiles described here define the modelling language for the perspectives and viewpoints in UAM. They define the *structural* and *behavioural component signatures* for the viewpoint languages. The perspective language is summarized followed by detailed descriptions of each of the four viewpoints and their language elements.

Note: OMG was just receiving proposals for UML Profiles for BPMN when this book was written. Therefore the profiles defined for the Business, Logical and Technical Perspectives will eventually migrate to the officially approved OMG UML profiles for BPMN.

Technical Perspective

The Technical Perspective in UAM conforms to the **BPMN Analytic Conformance** profile. This conformance profile is used for technical-level business process modelling in UAM, as required by the context and objective of the architecture effort. The Analytic Conformance sub-class contains all the elements of the Descriptive Conformance sub-class (i.e. the Logical Perspective, see the Logical Perspective Language UML Profile document) plus the elements shown in Table 1. This conformance profile is used for more detailed technical level business process modelling in UAM, as required by the context and objective of the architecture effort; however, the UAM methodology has some specific recommendations on how to use this profile.

Each element is described below, including a definition of the metamodel—the structure and relationship rules for the Technical Perspective Language (i.e. the grammar). An overview of the main set of language components and relationships is show in Figure 2.

Table 1 – BPMN Analytic Conformance Sub-Class Elements and Attributes (OMG 2013)

Element	Attributes
sequenceFlow (conditional)	id, name, sourceRef, targetRef, conditionExpressiona
sequenceFlow (default)	id, name, sourceRef, targetRef, defaultb
sendTask	id, name
receiveTask	id, name
Looping Activity	standardLoopCharacteristics
MultiInstance Activity	multiInstanceLoopCharacteristics
exclusiveGateway	Add default attribute
inclusiveGateway	id, name, eventGatewayType
eventBasedGateway	id, name, eventGatewayType
Link catch/throw Intermediate Event	Id, name, linkEventDefinition
signalStartEvent	id, name, signalEventDefinition
signalEndEvent	id, name, signalEventDefinition
Catching message Intermediate Event	id, name, messageEventDefinitionBusiness
Throwing message Intermediate Event	id, name, messageEventDefinition
Boundary message Intermediate Event	id, name, attachedToRef, messageEventDefinition
Non-interrupting Boundary message Intermediate Event	id, name, attachedToRef, cancelActivity=false, messageEventDefinition
Catching timer Intermediate Event	id, name, timerEventDefinition
Boundary timer Intermediate Event	id, name, attachedToRef, timerEventDefinition
Non-interrupting Boundary timer Intermediate Event	id, name, attachedToRef, cancelActivity=false, timerEventDefinition
Boundary error Intermediate Event	id, name, attachedToRef, errorEventDefinition
errorEndEvent	id, name, errorEventDefinition
Non-interrupting Boundary escalation Intermediate Event	id, name, attachedToRef, cancelActivity=false, escalationEventDefinition
Throwing escalation Intermediate Event	id, name, escalationEventDefinition

Element	Attributes
escalationEndEvent	id, name, escalationEventDefinition
Catching signal Intermediate Event	id, name, signalEventDefinition
Throwing signal Intermediate Event	id, name, signalEventDefinition
Boundary signal Intermediate Event	id, name, attachedToRef, signalEventDefinition
Non-interrupting Boundary signal Intermediate Event	id, name, attachedToRef, cancelActivity=false, signalEventDefinition
conditionalStartEvent	id, name, conditionalEventDefinition
Catching conditional Intermediate Event	id, name, conditionalEventDefinition
Boundary conditional Intermediate Event	id, name, conditionalEventDefinition
Non-interrupting Boundary conditional Intermediate Event	id, name, cancelActivity=false, conditionalEventDefinition
message[c]	id, name, add messageRef attribute to messageFlow

a. `ConditionExpression`, allowed only for **Sequence Flow** out of **Gateways**, MAY be null.
b. `Default` is an attribute of a `sourceRef` (exclusive or inclusive) **Gateway**.
c. Note that `messageRef`, an attribute of various message **Events**, is optional and not in the sub-class.

BPMN forms the heart of the Technical (and Logical) Perspective, with additional elements added in support of UAM essentially through the BPMN extension mechanisms to define the complete Technical Perspective Language. BPMN is very much an implementation-oriented set of structures and definitions. UAM needs slightly more abstract concepts and structures, which is done through simple adjustments in definitions along with specific recommendations on how to use this profile.

Each element is described below, including a definition of the metamodel—the structure and relationship rules for the Technical Perspective Language (i.e. the grammar). An overview of the main set of language components and relationships is show in Figure 2.

BPMN references and source material:

OMG: http://www.omg.org/spec/BPMN/
BPMN: http://www.bpmn.org/

In addition to these elements defined for the **Analytic Conformance** profile for BPMN V2.0, UAM has added the following BPMN elements to the Technical Process Viewpoint language:

Activity – Ad Hoc Subprocess, Compensation;
Cancel Events – Boundary Interrupting, and End;
Compensation Events – Event Sub-Process Interrupting, Boundary Interrupting, Throwing, End;
Escalation Event – Boundary Interrupting;
Error Event – Event Sub-Process Interrupting;
Signal Events – Event Sub-Process Interrupting;
Tasks – Manual, Business Rule and Script tasks;
Transaction – Transaction Sub-process (activities that logically belongs together);
Untyped Event – Throwing.

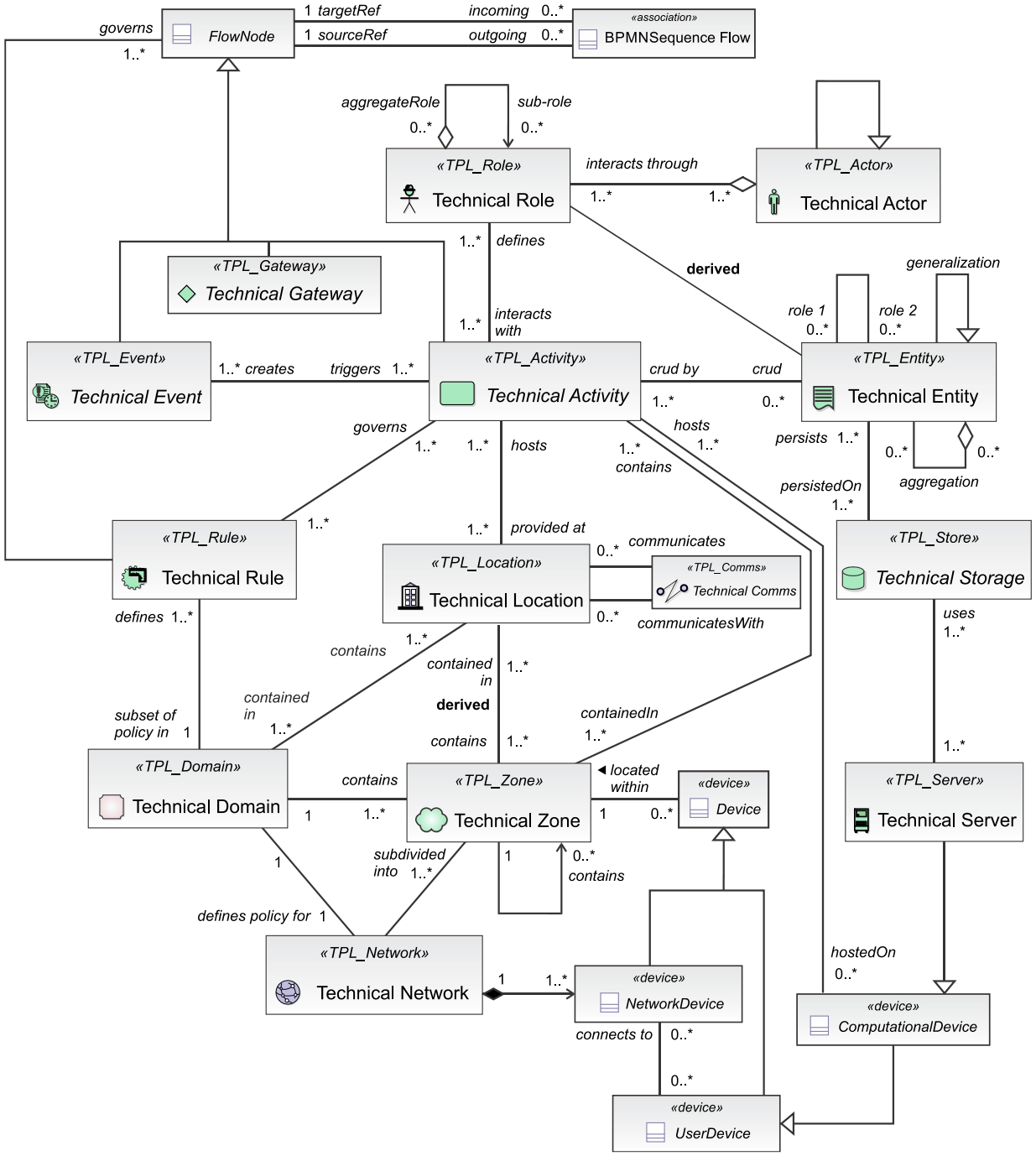


Figure 2 – Technical Perspective Language Overview in UML

BPMN references and source material:

http://www.omg.org/spec/BPMN/

These BPMN elements are integrated into a complete Technical Perspective language through the addition of the UAM elements defined within the Technical Entity, Location, and Role Viewpoint Languages.

A very much simplified view of the Technical Perspective language is shown in Figure 2. The main structures and concepts are shown, which is almost identical to the Logical Perspective. The main differences from the Logical level are in the details; the attributes of elements as well as the addition of more types of Events and Activities. This metamodel for the Technical Perspective language says:

- A Technical Actor aggregates (has) one or more associated Technical Roles;
- Technical Actors interact with the system through Technical Roles;
- A Technical Role interacts with one or more (abstract) Technical Activities. There are a number of types of Activities and Tasks defined in the Technical Process Viewpoint language for use in documenting the architecture;
- A Technical Activity (what it does and the interactions) defines a Technical Role. Roles are a representation of the *interface* to the activity along with IT security aspects;
- A Technical Role may be composed with other Roles;
- A Technical Activity may have a hierarchical (includes) structure, with high-level Activities hiding more detailed representations;
- A Technical Activity is triggered by one or more (abstract) Technical Events. There are a large number of types of Events defined in the Technical Process Viewpoint language for use in documenting the architecture;
- A Technical Activity may create Events;
- A Technical Activity may Create, Read, Update or Delete (CRUD) Technical Entities;
- Technical Gateways, Technical Events, and Technical Activities are all types for `FlowNodes` and are therefore interconnected into a sequence that defines a Process;
- The rules governing these `FlowNodes` are defined by the Technical Domain and Technical Rules;
- Technical Entities may have complex relationships and structures including specialization/generalization and aggregation/composition;
- Technical Entities have an (important and useful) derived relationship with Technical Roles (and therefore also Actors);
- A Technical Domain defines: the policy and rules used within Technical Rules (and the Gateways or Activities and Processes they relate to), the policy that governs Technical Locales, the policy that governs Technical Networks and Technical Zones contained within these locales;
- Technical Locales communicate with each other through Technical Comms (which may be anything from physical transport of good to electronic communications);
- Technical Networks are subdivided into Technical Zones, which are contained within Technical Locations;
- Technical Zones may be contained within other Technical Zones;
- Technical Servers and Storage are types of `Devices` and are located within Technical Zones;
- Technical Networks compose `NetworkDevices` which are types of `Devices` and are also contained within Technical Zones;
- Technical Storage persist Technical Entities and are used by Technical Servers.

This is just a simple overview of the Technical Perspective Language; each of the four Technical Perspective Viewpoint languages (metamodels) is defined in detail below.

Technical Entity Viewpoint

The Technical Entity viewpoint is not an information or data design model, but it should be fairly close. The focus is still *architecture*; therefore definitions, structures and relationships are the main focus however much more detail is provided in this model than in the Logical level model.

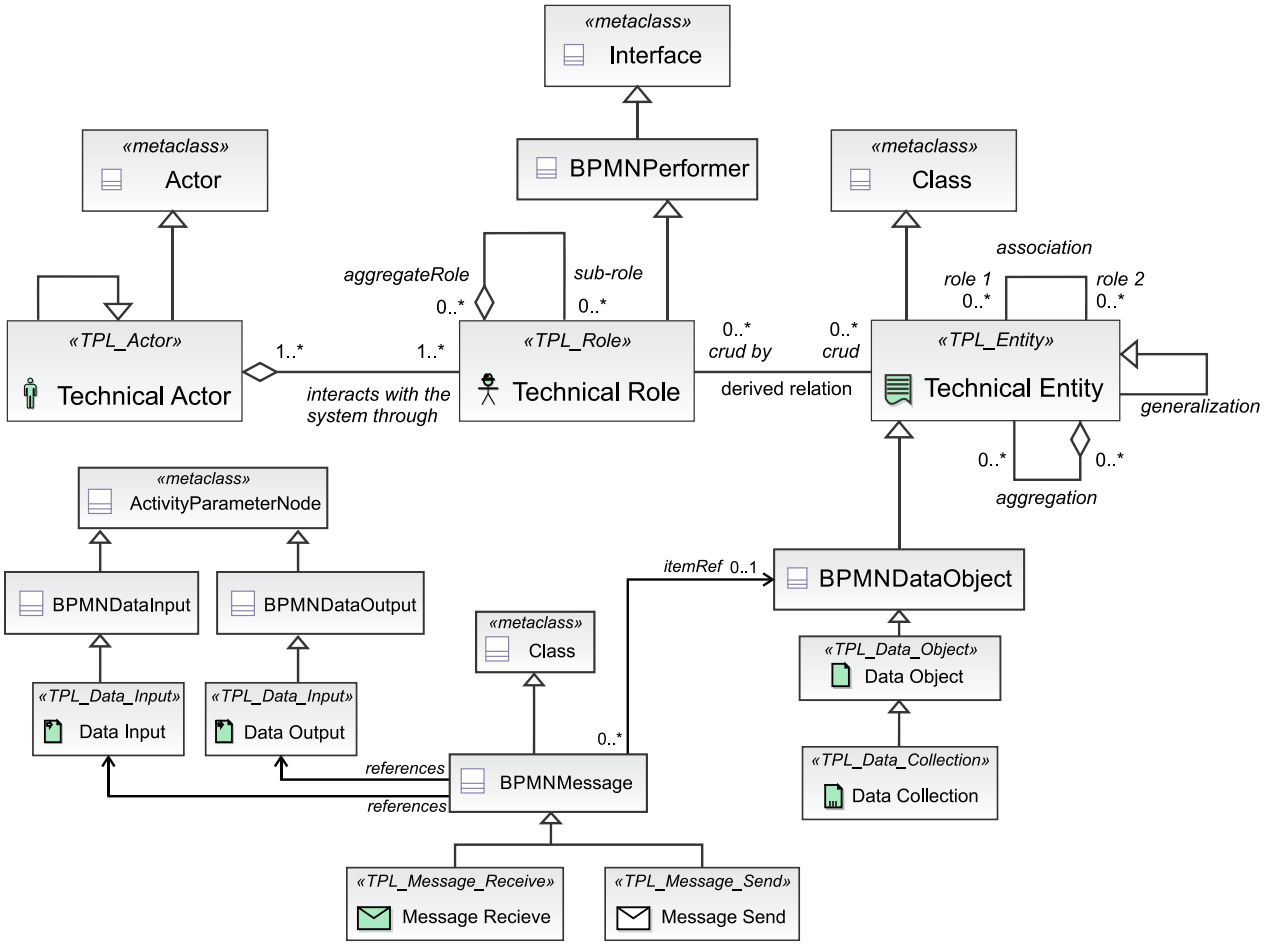


Figure 3 – Technical Entity Viewpoint Language

The important things to specify are the system-wide or corporate-wide definitions for the elements within this model. Element relationships are very important since they related directly to how the entities are used and need to be used to support the business. Likewise the structure of the information is important—it must also support the business both present and future.

The Technical Roles Model has at this level flushed out some important structures and relationships as well. See the definition for the Technical Roles Model below and how it supports Processes and their required access to Entities.

The UAM methodology provides guidance and advice on defining this viewpoint.

More information on the Technical Entity Model, its recommended structure and content along with how-to advice:

Guidance > Guidelines > Technical Entity Model

«stereotype» TPL_Entity

Extends

«metaclass» Class

Semantics

A Technical Entity is a representation of the information and data used within the system being modelled. At the technical level a very complete view of the data entities are defined, including their attributes and relationships. If desired, primary keys and other structural information may also be included. Technical Entities are used to define a physical data model in support of the processes and activities of the system, but perhaps not to the level of detail required for implementation (depending upon the context and objectives of the architecture effort). This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

The properties of Technical Entities, including attributes and other characteristics, are defined as required by the context and subject that they represent. In the Technical Perspective the entities are used to develop a fairly complete physical data model for the architectural context and supporting the system under analysis. The basic attributes of an entity are:

Name	Description
id: string	This attribute is used to uniquely identify Entities.
name: string	A descriptive name for the Entity.
owner: string	Defines the owner of the information—the organizational element that makes usage and access decisions about the information. Normally defined as a specific organizational position within the enterprise or business line (e.g., COO).
attribute: variable	The many defined attributes of the entity, including the type. As many attributes are added as necessary to properly model the entity.

Notation

Constraints

- May have relationships with only Technical Roles or with other Entities (all relationships types are allowed, such as aggregations, generalizations, specializations, etc.);
- Primary and other keys may be identified in support of structured persistent storage.

«stereotype» TPL_EntityAssociation

Extends

«metaclass» Association

Semantics

Used to define Technical level associations between two Entities in the model. Roles are assigned if desired to indicate how the Entities are related or used. At this level there are three possible types of associations: simple, aggregations and generalization.

Properties

Name	Description
id: string	This attribute is used to uniquely identify model elements.
name: string	A descriptive name for the association.
sourceRef: entity	The `Entity` that the Association is connecting from
targetRef: entity	The `Entity` that the Association is connecting to
sourceRole: string	The role that the source entity has with the target entity for simple and aggregation/composition associations.
targetRole: string	The role that the target entity has with the source entity for simple and aggregation/composition associations.
sourceMult: integer = 1 {0..1 | 1 | 0..* | 1..*}	The multiplicity that the source entity has with the target entity.
targetMult: integer = 1 {0..1 | 1 | 0..* | 1..*}	The multiplicity that the target entity has with the source entity.
type: AssociationType = Simple {Simple | Aggregation | Generalization}	**Simple:** a simple association (with or without Roles and Multiplicities) with perhaps a defined meaning; **Aggregation**: represents a part-whole or part-of relationship; **Generalization**: a specialized form of the other (the *super type*) and super-class is considered as '***Generalization'*** of subclass (without Roles and without Multiplicities).
associationDirection: AssociationDirection = None {None | One | Both}	Defines whether or not the Association shows any directionality with an arrowhead. The default is `None` (no arrowhead). A value of `One` means that the arrowhead shall be at the Target Entity. A value of `Both` means that there shall be an arrowhead at both ends of the association line.

Notation

Simple associations:  or or

Aggregation association: 

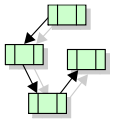
Composition association: 

Generalization association: 

Constraints

- May define simple relationships between any two `TPL_Entity` elements;

Technical Process Viewpoint



The Technical Process viewpoint defines the Activities and Processes involved in the system or enterprise, as defined by the scope of the modelling effort. Depending upon the context and scope of the modelling effort, the level of detail may be quite extensive or be very broad and conceptual. Technical solutions may be defined, such as middleware solutions or other hardware or software choices.

The scope and impact of these choices and decisions depends upon the context, scope and objectives of the architecture. At the enterprise level, these choices define the technical standards used within the enterprise, and to some extent the patterns of their use.

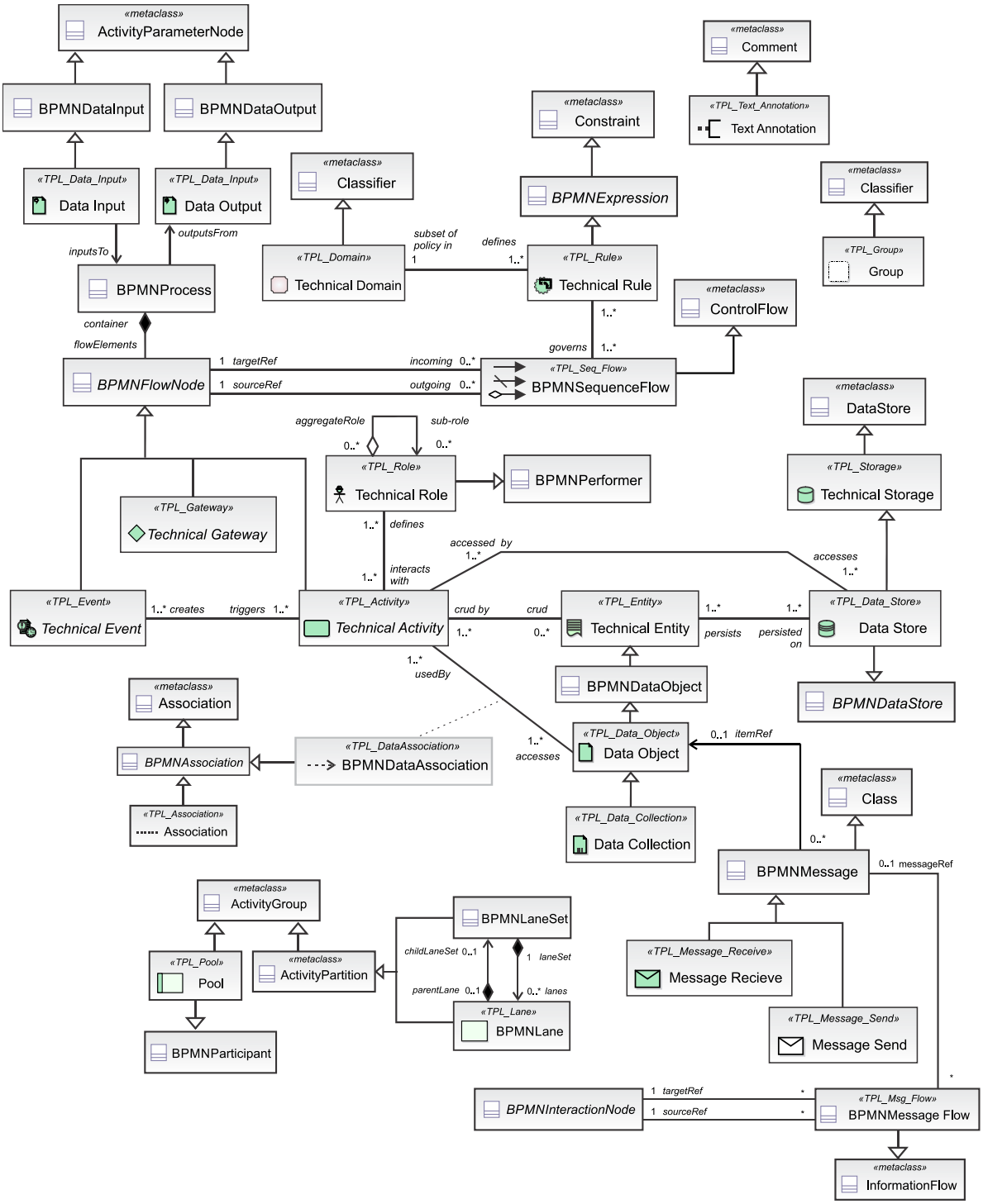


Figure 4 – Technical Process Viewpoint Language (part 1)[1]

In the Figure 4, and those below, the elements of the Technical Process Viewpoint language are defined along with some elements from BPMN V2.0 to provide context. It should be noted that eventually

[1] Note that only a small subset of the BPMN V2.0 metamodel is shown for context; see the BPMN V2.0 definition for the complete set of definitions.

this metamodel will conform to the BPMN UML Profile as sanctioned by OMG. The elements (stereotypes) forming the Technical Process viewpoint language are defined in detail, including relationships and constraints.

In BPMN there are five basic categories of elements defined that are used to specify business processes. These categories, also adopted in the UAM metamodel, for defining Processes are:

1. **Flow Objects;**
2. **Data;**
3. **Connecting Objects;**
4. **Swimlanes;**
5. **Artefacts.**

Flow Objects are the main elements used to define the behaviour within a Process. There are three Flow Objects:

1. **Events** – message, timer, conditional, and other events that result in activity;
2. **Activities** – where actual business benefit is realised, arranged into processes;
3. **Gateways** – decision points where workflow is combined, split or otherwise controlled.

All Processes and Activities require Data, which is represented through four elements:

1. **Data Objects** – basic data;
2. **Data Inputs** – inputs to Processes;
3. **Data Outputs** – outputs from Processes;
4. **Data Stores** – data object persistence.

The three types of Flow Objects are connected together in order to define the Process flow. These Flow Objects may also be associated to each other or to other information. There are four Connecting Objects used for this purpose:

1. **Sequence Flows** – used to define the sequence of Events, Activities, and Gateways in a business process;
2. **Message Flows** – used to define interactions between Processes and Activities that cross Pool and Lane boundaries;
3. **Associations** – use to associate information, text annotations, and other artefacts with BPMN elements;
4. **Data Associations** – use to associate Data (Objects, Inputs, Outputs, and Stores) with BPMN elements.

Processes are grouped and organized using two mechanisms through the notion of swim lanes (similar to the notion within UML), specifically:

1. **Pools;**
2. **LaneSets** and **Lanes.**

Artefacts are used to provide additional information in or about BPMN processes. There are two standardized Artefacts, but UAM has added two addition artefacts: Technical Rule and Technical Domain. The current BPMN / UAM set of Artefacts includes:

- **Group** – a grouping of graphical elements that are within the same category (a user defined classification of BPMN elements);
- **Text Annotation** – a mechanism for a modeller to provide additional text information for the reader of the viewpoint;
- **Technical Rule** – the (business) rules associated with an Event, Activity or Gateway;
- **Technical Domain** – defines the complete set of policies and rules that apply to the system under study (and often to a wider corporate scope). At the technical level these rule are defined in a way that is closer to that required for implementation.

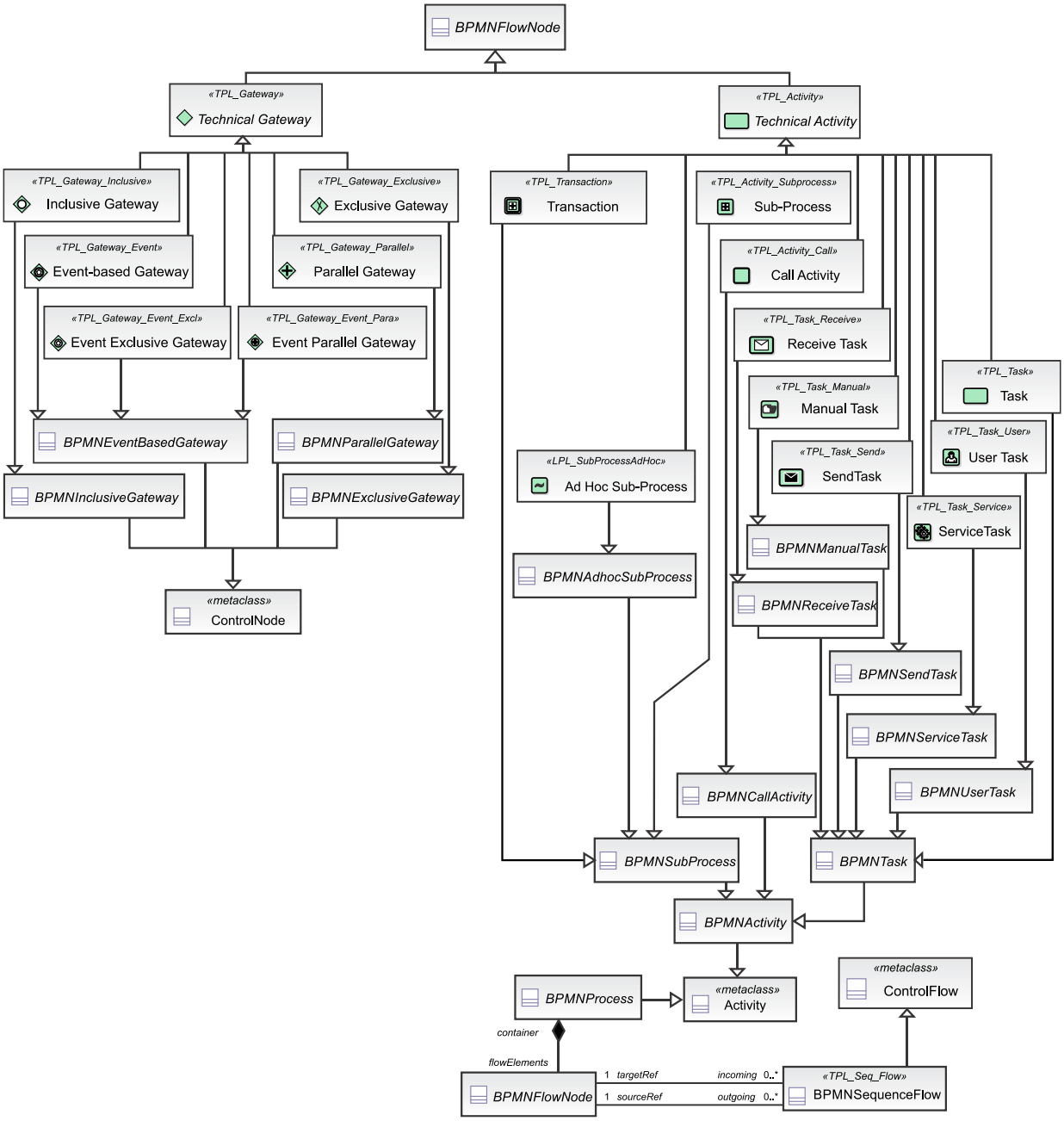


Figure 5 – Technical Process Viewpoint Language (part 2)

From Figure 4 it can be seen that *FlowNodes*, namely Events, Gateways or Activities, are linked

together through *Sequence Flow* associations to form Processes. The Technical Domain defines all policies and rules that control this workflow along with other required policies and rules for the system under study (or the Domain of which the system is part), with Technical Rules being associated with specific *FlowNode* elements.

Technical Roles interact with Technical Activities to execute portions of business processes, which may result in the creation, reading, update or deletion of Technical Entities. A Data Object is a type of Technical Entity that is accessed or received by a Technical Activity. Technical Entities may be persisted on Data Stores, which are Technical Stores that support business processes and access to Data Objects.

Technical Events and Activities are types of *InteractionNodes* which are sources and targets for messages flows, as are *Pools* and *Lanes*. This is the mechanism defined in BPMN for illustrating communications between Processes that are segregating into different Pools or Lanes. Messages are sent to Events or Activities and from Events and Activities to Events and Activities in other Pools. Messages may contain Data Objects.

The rest of the elements and structure of the Technical Process language are shown in Figure 5 and Figure 6 where the numerous types of Technical Events, Technical Gateways, and Technical Activities are defined.

The portion of the Technical Process Language metamodel shown in Figure 5 adds the elements Data Input and Data Output, which are used to illustrate the Data Objects (or Collections) that are inputs or outputs to a Process, and which may also be referenced by Messages. Send Message and Receive Message are also shown on this model for context. The complete set of Technical Gateway and Technical Activity *FlowNodes* is defined, which is much more extensive than in the Logical Process Viewpoint Language. Note that Technical Event, Technical Gateway and Technical Activity are abstract (as illustrated by their names being in italics) and as such they cannot be instantiated in a process model description. Generic or "un-typed" Events, Gateways and Activities (Tasks) are defined that allow for the creation of high-level models if desired.

The portion of the Technical Process Language metamodel shown in Figure 6 and Figure 7 add definitions for the complete set of Technical Events in the Technical Process Viewpoint language. The *FlowNode* element is also shown on this model for context. The Event elements in Figure 6 are:

- **Signal** – Start and End, Catching, Throwing, Boundary, and Non-interrupting Boundary;
- **Message** – Catching, Throwing and Boundary;
- **Timer** – Catching, Boundary and Non-interrupting Boundary;
- **Escalation** – Throwing, Non-interrupting Boundary and End;
- **Conditional** – Start, Catching, and Non-interrupting Boundary;

In summary, the elements for the Technical Process language are:

- **Activity** – the various types of activities and tasks defined in Figure 5, which are used in process definitions;
- **Association / DataAssociation** – used to associate text annotations, Data Objects, and Data Inputs and Outputs with Groups, Tasks or Sequence Flows;
- **Data Input** and **Data Output** – used to identify the input to and outputs from a Process or the contents of Messages;
- **Data Object** – a Technical Entity used within a Process or Processes or part of a *Sequence Flow*;
- **Data Store** – persistence for Data Objects and other information used or required by a Process or set of Processes;

- **dataStoreReference** – a reference used to associate Data Stores with Sub-Process or Tasks that use them;
- **Events** – the various types of events defined in Figure 6, which are used in process definitions;
- **Exclusive Gateway** – the flow goes to only one outgoing branch, or when merging any input branch can trigger the flow;
- **Inclusive Gateway** – the flow goes to one or more outgoing branches, or when merging all input branches must complete before triggering the outgoing flow;
- **Parallel Gateway** – the flow goes to all outgoing branches, or when merging all input branches must complete before triggering the outgoing flow(s);
- **Event-based Gateway** – always followed by catching events or receive tasks, the flow goes to the subsequent Event or Task that happens first;
- **Group** – used to group together tasks or other process elements;
- **Lane** – used to define and structure Processes;
- **LaneSet** – used to define and structure Lanes;
- **Technical Domain** – the set of policy and rules governing the system under study;
- **Technical Role** – the role assumed by an actor when interacting with an activity;
- **Technical Rule** – rule governing the activities of a specific *FlowNode*;
- **Technical Storage** – persistence for any information within the system;
- **Message Flow** – connections and flows of messages between two *Pools / Lanes*;
- **Pool** – used to define and structure Processes;
- **Receive Message** – used to communicate (data objects etc.) between Pools (Processes);
- **Send Message** – used to communicate (data objects etc.) between Pools (Processes);
- **Sequence Flow** – connections and flows within a process and between two *FlowNodes*.

Note that the following high-level elements are provided in the viewpoint language as a way for dealing with complexity through the use of hierarchically structured models—these elements define something at a high-level, with the detail provided in lower level detailed models:

- **Sub-Process** – the highest level of abstraction for an Activity or Process definition;
- **Data Object** – the highest level of abstraction for a data definition;
- **Technical Store or Data Store** – the highest level of abstraction for persistence;
- **Untyped Events** – the highest level of abstraction for events;
- **Technical Domain** – the highest level of abstraction for the system;
- **Pool** – the highest level of abstraction for grouping of Activities and sub-processes.

The UAM methodology provides guidance and advice on defining this viewpoint.

More information on the Technical Process Model, its recommended structure and content along with how-to advice:

Guidance > Guidelines > Technical Process Model

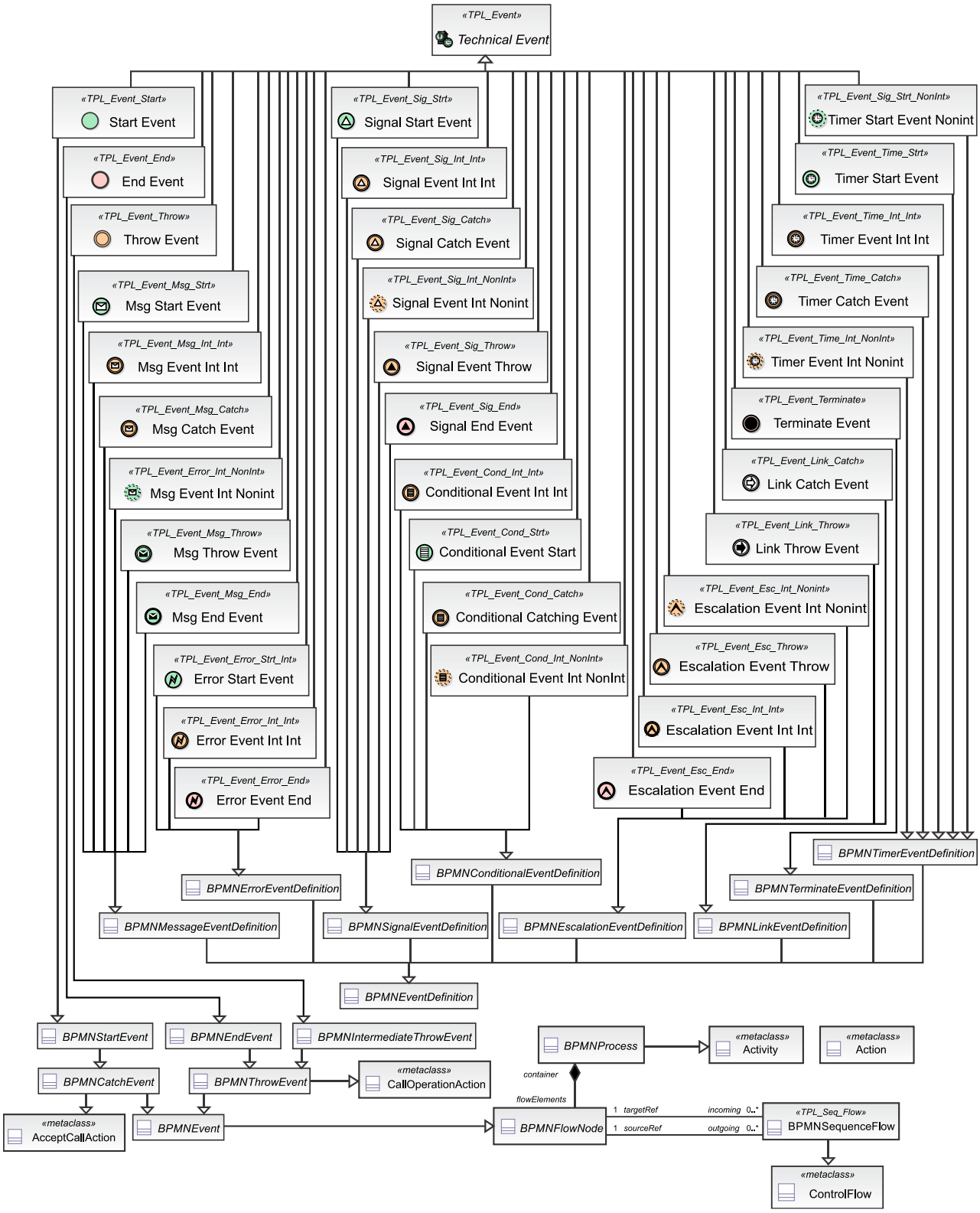


Figure 6 – Technical Process Viewpoint Language (part 3)

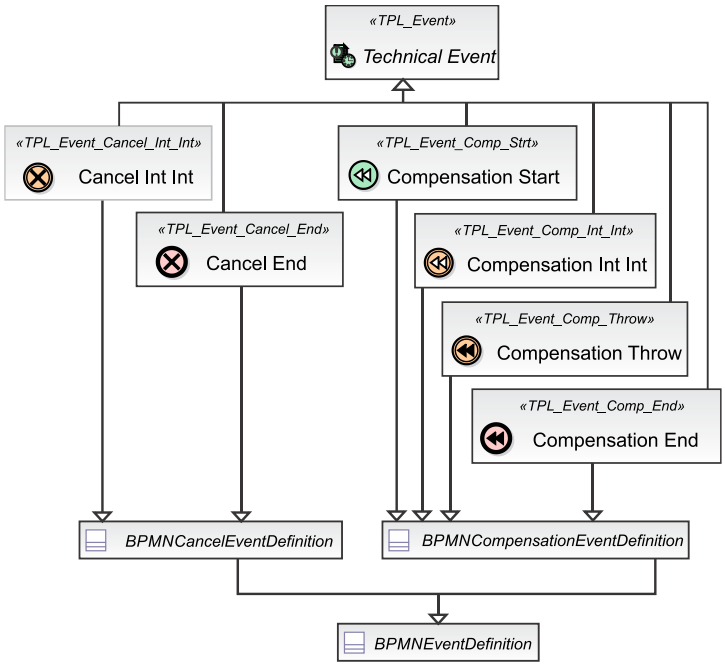


Figure 7 – Technical Process Viewpoint Language (part 4)

Definitions for all of these Technical Perspective language elements follow.

«stereotype» TPL_Activity_Call

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Technical) Call Activity is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process"(OMG 2013). A Call Activity identifies a point in the Process where a global Process or a Global Task is used. The Call Activity acts as a 'wrapper' for the invocation of a global Process or Global Task within the execution. The activation of a Call Activity results in the transfer of control to the called global Process or Global Task (OMG 2013). It is essentially a reusable Activity within the context of the architecture.

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Call Activities** have the following attributes:

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope.
loopCharacteristics: LoopCharac-teristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The `InputOutputSpecification` defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**. See page 211 for more information on the `InputOutputSpecification`.
properties: Property [0..*]	Modeler-defined properties MAY be added to an **Activity**. These properties are contained within the **Activity**.
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataIn-putAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used

Name	Description (OMG 2013)
	with caution.
calledElement: CallableElement [0..1]	The element to be called, which will be either a **Process** or a `GlobalTask`. Other `CallableElements`, such as **Choreography**, `GlobalChoreographyTask`, **Conversation**, and `GlobalCommunication` MUST NOT be called by the **Call Conversation** element.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).

Notation

Call Activity Types:

Undefined:  Collapsed: 

Rule:  Manual:  User:  Script: 

Call Activities with markers (NOTE: the other Call Activity Types defined above may also use these markers, however only the undefined type collapsed variant is shown):

Loop:  Parallel:  Sequential: 

Compensation:  Compensation-Loop: 

Compensation-Parallel:  Compensation-Sequential: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Association

Extends

«metaclass» Association

Semantics

Used to define a (Technical) Association between two elements in the model, for example associating a text annotation with a Task or other element.

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Name	Description (OMG 2013)
associationDirection: AssociationDirection = None {None | One | Both}	`associationDirection` is an attribute that defines whether or not the Association shows any directionality with an arrowhead. The default is None (no arrowhead). A value of `One` means that the arrowhead SHALL be at the Target Object. A value of `Both` means that there SHALL be an arrowhead at both ends of the Association line.
sourceRef: BaseElement	The `BaseElement` that the Association is connecting from.
targetRef: BaseElement	The `BaseElement` that the Association is connecting to.

Notation

Constraints

- May define relationships between any two `baseElements`;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Data_Input

Extends

«metaclass» ParameterSet

«metaclass» ActivityParameterNode

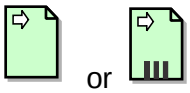
Semantics

A Data Input is a declaration that a particular kind of data will be used as input of the `InputOutputSpecification`. There may be multiple Data Inputs associated with an `InputOutputSpecification` for an Activity. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
inputSetRefs: InputSet [1..*]	A `DataInput` is used in one or more `InputSets`. This attribute is derived from the `InputSets`.
inputSetwithOptional: InputSet [0..*]	Each `InputSet` that uses this `DataInput` can determine if the Activity can start executing with this `DataInput` state in "unavailable." This attribute lists those `InputSets`.
inputSetWithWhileExecuting: Inputset [0..*]	Each `InputSet` that uses this `DataInput` can determine if the Activity can evaluate this `DataInput` while executing. This attribute lists those `InputSets`.
isCollection: boolean = false	Defines if the DataInput represents a collection of elements. It is needed when no `itemDefinition` is referenced. If an `itemDefinition` is referenced, then this attribute *must* have the same value as the `isCollection` attribute of the referenced `itemDefinition`. The default value for this attribute is *false*.

Notation

 or

Constraints

- Has a relationship with an `inputSet` (`inputSetRefs`);
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Data_Object and TPL_Data_Collection

Extends

«metaclass» Class

Semantics

Defines a (Technical) Data Object (`DataObject`) is the primary construct for modeling data within a Process flow. It has a well-defined lifecycle, with resulting access constraints (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isCollection: boolean = false	Defines if the Data Object represents a collection of elements. It is needed when no `itemDefinition` is referenced. If an `itemDefinition` is referenced, then this attribute MUST have the same value as the `isCollection` attribute of the referenced `itemDefinition`.

Notation

 or (collection)

Constraints

- A kind of `FlowElement` and `ItemAwareElement` that has relationships defined through a `DatObjectReference`;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Data_Output

Extends

«metaclass» ParameterSet

«metaclass» ActivityParameterNode

Semantics

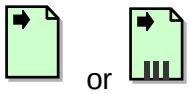
A Data Output is a declaration that a particular kind of data will be used as input of the

InputOutputSpecification. There may be multiple Data Inputs associated with an InputOutputSpecification for an Activity. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The id is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the id MAY be omitted.
name: string	A descriptive name for the element.
outputSetRefs: OutputSet [1..*]	A DataOutput is used in one or more OutputSets. This attribute is derived from the OutputSets.
outputSetwithOptional: OutputSet [0..*]	Each OutputSet that uses this DataOutput can determine if the Activity can start executing with this DataOutput state in "unavailable." This attribute lists those OutputSets.
outputSetWithWhileExecuting: OutputSet [0..*]	Each OutputSet that uses this DataOutput can determine if the Activity can evaluate this DataOutput while executing. This attribute lists those OutputSets.
isCollection: boolean = false	Defines if the DataOutput represents a collection of elements. It is needed when no itemDefinition is referenced. If an itemDefinition is referenced, then this attribute *must* have the same value as the isCollection attribute of the referenced itemDefinition. The default value for this attribute is *false*.

Notation



Constraints

- Has a relationship with an outputSet (outputSetRefs);
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Data_Store

Extends

«metaclass» Class

«metaclass» DataStoreNode

«metaclass» Node

Semantics

Defines a (Technical) DataStore that provides a mechanism for Activities to retrieve or update stored information that will persist beyond the scope of the Process. The same DataStore can be visualized, through a Data Store Reference, in one or more places in the Process (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
capacity: integer [0..1]	Defines the `capacity` of the **Data Store**. This is not needed if the `isUnlimited` attribute is set to *true*.
isUnlimited: boolean = false	If `isUnlimited` is set to *true*, then the capacity of a **Data Store** is set as unlimited and will override any value of the `capacity` attribute.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- May be referenced through a `dataStoreReference` in one or more places in a Process;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_DataAssociation

Extends

«metaclass» Association

Semantics

The core concepts of a (Technical) DataAssociation are that they have sources, a target, and an optional transformation. When a data association is "executed," data is copied to the target. What is copied depends if there is a transformation defined or not. If there is no transformation defined or referenced, then only one source *must* be defined, and the contents of this source will be copied into the target (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
transformation: Expression [0..1]	Specifies an optional transformation `Expression`. The actual scope of accessible data for that `Expression` is defined by the source and target of the specific **Data Association** types.
assignment: Assignment [0..*]	Specifies one or more data elements `Assignments`. By using an `Assignment`, single data structure elements can be assigned from the source structure to the target structure.

Name	Description (OMG 2013)
sourceRef: ItemAwareElement [0..*]	Identifies the source of the **Data Association**. The source *must* be an `ItemAwareElement`.
targetRef: ItemAwareElement	Identifies the target of the **Data Association**. The target *must* be an `ItemAwareElement`.

Notation

Constraints

- May define relationships between `ItemAwareElements`;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_DataStoreRef

Extends

«metaclass» Association

Semantics

Defines a Techncial `DataStoreReference` that points to a `DataStore`.

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
dataStoreRef: DataStore	Provides the reference to a global `DataStore`.

Notation

Constraints

- References only a Data Store;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cancel_End

Extends

«metaclass» FinalNode

«metaclass» CallOperationAction

Semantics

Defines a (Technical) Cancel End Event. As the name implies, the End Event indicates where a Process will end. **Cancel Events** are only used in the context of modeling **Transaction Sub-Processes** (see *Transaction*). There are two variations: a *catch* **Intermediate Event** and an **End Event.** The **Cancel End**

Event MUST only be used within a **Transaction Sub-Process** and, thus, MAY NOT be used in any other type of **Sub-Process** or **Process** (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cancel_Int_Int

Extends

«*metaclass*» FinalNode

«*metaclass*» CallOperationAction

Semantics

Defines a (Technical) Boundary (Intermediate) Interrupting Cancel Event. As the name implies, the Cancel Event indicates where a Process will end. **Cancel Events** are only used in the context of modeling **Transaction Sub-Processes** (see *Transaction*). There are two variations: a *catch* **Intermediate Event** and an **End Event**. The **Catch Cancel Intermediate Event** MUST only be attached to the boundary of a **Transaction Sub-Process** and, thus, MAY NOT be used in *normal flow*. (OMG 2013)

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Comp_End

Extends

«metaclass» FlowFinalNode

«metaclass» CallOperationAction

Semantics

Defines a (Technical) Compensation End Event. **Compensation Events** are used in the context of triggering or handling *compensation* (see page 301 for more details on *compensation*). There are four variations: a **Start Event**, both a *catch* and *throw* **Intermediate Event**, and an **End Event** (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
activityRef: Activity [0..1]	For a **Start Event**: This **Event** "catches" the *compensation* for an **Event Sub-Process**. No further information is REQUIRED. The **Event Sub-Process** will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast. For an **End Event**: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). For an **Intermediate Event** within *normal flow*: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). This "throws" the *compensation*. For an **Intermediate Event** attached to the boundary of an **Activity**: This Event "catches" the *compensation*. No further information is REQUIRED. The **Activity** the **Event** is attached to will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast.
waitForCompletion: boolean = true	For a *throw* **Compensation Event**, this flag determines whether the *throw* **Intermediate Event** waits for the triggered *compensation* to complete (the default), or just triggers the *compensation* and immediately continues.

Notation

Constraints

- The **Compensation End Event** MAY be used within any **Sub-Process** or **Process;**
- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Comp_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Compensation Boundary Intermediate Event. **Compensation Events** are used in the context of triggering or handling *compensation* (see page 301 for more details on *compensation*). There are four variations: a **Start Event**, both a *catch* and *throw* **Intermediate Event**, and an **End Event** (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
activityRef: Activity [0..1]	For a **Start Event**: This **Event** "catches" the *compensation* for an **Event Sub-Process**. No further information is REQUIRED. The **Event Sub-Process** will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast. For an **End Event**: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). For an **Intermediate Event** within *normal flow*: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). This "throws" the *compensation*. For an **Intermediate Event** attached to the boundary of an **Activity**: This Event "catches" the *compensation*. No further information is REQUIRED. The **Activity** the **Event** is attached to will provide the Id necessary to match the **Compensation Event** with the

Name	Description (OMG 2013)
	Event that *threw* the *compensation*, or the *compensation* will have been a broadcast.
waitForCompletion: boolean = true	For a *throw* **Compensation Event**, this flag determines whether the *throw* **Intermediate Event** waits for the triggered *compensation* to complete (the default), or just triggers the *compensation* and immediately continues.

Notation

Constraints

- The *catch* **Compensation Intermediate Event** MUST only be attached to the boundary of an **Activity** and, thus, MAY NOT be used in *normal flow*;
- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Comp_Strt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Compensation Start Event. **Compensation Events** are used in the context of triggering or handling *compensation* (see page 301 for more details on *compensation*). There are four variations: a **Start Event**, both a *catch* and *throw* **Intermediate Event**, and an **End Event** (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
activityRef: Activity [0..1]	For a **Start Event**: This **Event** "catches" the *compensation* for an **Event Sub-Process**. No further information is REQUIRED. The **Event Sub-Process** will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast. For an **End Event**: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). For an **Intermediate Event** within *normal flow*: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). This "throws" the *compensation*. For an **Intermediate Event** attached to the boundary of an **Activity**: This Event "catches" the *compensation*. No further information is REQUIRED. The **Activity** the **Event** is attached to

Name	Description (OMG 2013)
	will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast.
waitForCompletion: boolean = true	For a *throw* **Compensation Event**, this flag determines whether the *throw* **Intermediate Event** waits for the triggered *compensation* to complete (the default), or just triggers the *compensation* and immediately continues.

Notation

Constraints

- The **Compensation Start Event** MAY NOT be used for a *top-level* **Process;**
- The **Compensation Start Event** MAY be used for an **Event Sub-Process;**
- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Comp_Throw

Extends

«metaclass» FlowFinalNode

«metaclass» CallOperationAction

Semantics

Defines a (Technical) Compensation Throw Event. **Compensation Events** are used in the context of triggering or handling *compensation* (see page 301 for more details on *compensation*). There are four variations: a **Start Event**, both a *catch* and *throw* **Intermediate Event**, and an **End Event** (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
activityRef: Activity [0..1]	For a **Start Event**: This **Event** "catches" the *compensation* for an **Event Sub-Process**. No further information is REQUIRED. The **Event Sub-Process** will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast. For an **End Event**: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level). For an **Intermediate Event** within *normal flow*: The **Activity** to be compensated MAY be supplied. If an **Activity** is not supplied, then the *compensation* is broadcast to all completed **Activities** in the current **Sub-Process** (if present), or the entire **Process** *instance* (if at the global level).

Name	Description (OMG 2013)
	This "throws" the *compensation*. For an **Intermediate Event** attached to the boundary of an **Activity**: This Event "catches" the *compensation*. No further information is REQUIRED. The **Activity** the **Event** is attached to will provide the Id necessary to match the **Compensation Event** with the **Event** that *threw* the *compensation*, or the *compensation* will have been a broadcast.
waitForCompletion: boolean = true	For a *throw* **Compensation Event**, this flag determines whether the *throw* **Intermediate Event** waits for the triggered *compensation* to complete (the default), or just triggers the *compensation* and immediately continues.

Notation

Constraints

- The *throw* **Compensation Intermediate Event** MAY be used in *normal flow;*
- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cond_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Catch (Intermediate) Conditional Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Message arrives from a participant and triggers the Event. This type of Event is triggered when a condition becomes *true*. A condition is a type of `Expression` (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
condition: Expression	The `Expression` might be underspecified and provided in the form of natural language. For executable Processes (`isExecutable` = true), if the *trigger* is `Conditional`, then a `FormalExpression` must be entered.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cond_Int_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Interrupting Conditional Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. This type of Event is triggered when a condition becomes *true*. A condition is a type of `Expression` (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
condition: Expression	The `Expression` might be underspecified and provided in the form of natural language. For executable Processes (`isExecutable` = true), if the *trigger* is `Conditional`, then a `FormalExpression` must be entered.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cond_Int_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Non-interrupting Conditional Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. This type of Event is triggered when a condition becomes *true*. A condition is a type of `Expression` (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
condition: Expression	The `Expression` might be underspecified and provided in the form of natural language. For executable Processes (`isExecutable` = true), if the *trigger* is `Conditional`, then a `FormalExpression` must be entered.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Cond_Strt (interrupting)

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Conditional Start Event that is interrupting. This type of Event is triggered when a condition becomes *true*. A condition is a type of `Expression`. (OMG 2013)**.**

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isInterrupting: boolean = true	This attribute only applies to **Start Events** of **Event Sub-Processes**; it is ignored for other **Start Events**. This attribute denotes whether the **Sub-Process** encompassing the **Event Sub-Process** should be canceled or not, If the encompassing **Sub- Process** is not canceled, multiple *instances* of the **Event Sub-Process** can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
condition: Expression	The `Expression` might be underspecified and provided in the form of natural language. For executable Processes (`isExecutable` = true), if the *trigger* is `Conditional`, then a `FormalExpression` must be entered.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_End

Extends

«metaclass» FlowFinalNode

«metaclass» CallOperationAction

Semantics

Defines a (Technical) un-typed End Event. As the name implies, the End Event indicates where a Process will end. In terms of Sequence Flows, the End Event ends the flow of the Process, and thus, will not have any *outgoing* Sequence Flows—no Sequence Flow can connect from an End Event (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Error_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Error End Event that is interrupting (all Error Events are interrupting).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
error: Error [0..1]	If the *trigger* is an Error, then an Error payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Error_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Error Intermediate Event that is interrupting (all Error Events are interrupting).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.

Name	Description (OMG 2013)
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
error: Error [0..1]	If the *trigger* is an Error, then an Error payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Error_Strt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Error Start Event that is interrupting (all Error Events are interrupting).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
error: Error [0..1]	If the *trigger* is an Error, then an Error payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Escalation_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Escalation End Event. As the name implies, the Escalation End Event indicates where a Process will end. In terms of Sequence Flows, the End Event ends the flow of the Process, and thus, will not have any *outgoing* Sequence Flows—no Sequence Flow can connect from an End Event. This type of Event is used for handling a named Escalation. If attached to the boundary of an Activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the Activity to which the *boundary* Event is attached. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
escalationRef: Escalation [0..1]	If the *trigger* is an `Escalation`, then an `Escalation` payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Escalation_Int_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Interrupting Escalation Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. This type of Event is used for handling a named Escalation. If attached to the boundary of an Activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the Activity to which the *boundary* Event is attached. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be

Name	Description (OMG 2013)
	referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: Boolean	Denotes whether the Activity should be cancelled or not, i.e., whether the *boundary catch* Event acts as an Error or an Escalation. If the Activity is not cancelled, multiple *instances* of that handler can run concurrently.
escalationRef: Escalation [0..1]	If the *trigger* is an `Escalation`, then an `Escalation` payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Escalation_Int_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Non-interrupting Escalation Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. This type of Event is used for handling a named Escalation. If attached to the boundary of an Activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the Activity to which the *boundary* Event is attached. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: Boolean	Denotes whether the Activity should be cancelled or not, i.e., whether the *boundary catch* Event acts as an Error or an

Name	Description (OMG 2013)
	Escalation. If the Activity is not cancelled, multiple *instances* of that handler can run concurrently.
escalationRef: Escalation [0..1]	If the *trigger* is an `Escalation`, then an `Escalation` payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Escalation_Throw

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Throw (Intermediate) Event, the creation of an Escalation Event (in this case) which is passed on to an Escalation Catching Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. This type of Event is used for handling a named Escalation. If attached to the boundary of an Activity, the Intermediate Event catches an Escalation. In contrast to an Error, an Escalation by default is assumed to not abort the Activity to which the *boundary* Event is attached. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
escalationRef: Escalation [0..1]	If the *trigger* is an `Escalation`, then an `Escalation` payload MAY be provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Link_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Link Catch Event. A Link Event is a mechanism for connecting two sections of a Process. Link Events can be used to create looping situations or to avoid long Sequence Flow lines. The use of Link Events is limited to a single Process level (i.e., they cannot link a *parent* Process with a Sub-Process). Paired Link Events can also be used as "Off-Page Connectors" for printing a Process across multiple pages. (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	If the *trigger* is a *Link*, then the name MUST be entered.
sources: LinkEventDefinition [1..*]	Used to reference the corresponding 'catch' or 'target' `LinkEventDefinition`, when this `LinkEventDefinition` represents a 'throw' or 'source' `LinkEventDefinition`.
target: LinkEventDefinition [1]	Used to reference the corresponding 'throw' or 'source' `LinkEventDefinition`, when this `LinkEventDefinition` represents a 'catch' or 'target' `LinkEventDefinition`.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Link_Throw

Extends

«metaclass» CallOperationAction

Semantics

Defines a (Technical) Link Throw Event. A Link Event is a mechanism for connecting two sections of a Process. Link Events can be used to create looping situations or to avoid long Sequence Flow lines. The use of Link Events is limited to a single Process level (i.e., they cannot link a *parent* Process with a Sub-Process). Paired Link Events can also be used as "Off-Page Connectors" for printing a Process across multiple pages. (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	If the *trigger* is a *Link*, then the name MUST be entered.
sources: LinkEventDefinition [1..*]	Used to reference the corresponding 'catch' or 'target' `LinkEventDefinition`, when this `LinkEventDefinition` represents a 'throw' or 'source' `LinkEventDefinition`.
target: LinkEventDefinition [1]	Used to reference the corresponding 'throw' or 'source' `LinkEventDefinition`, when this `LinkEventDefinition` represents a 'catch' or 'target' `LinkEventDefinition`.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Catch (Intermediate) Message Event, the handling of a Message Event (in this case) which is created by a Message Throw Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Message arrives from a participant and triggers the Event. When the *trigger* of the Event occurs (for example, the Message is received), the data is assigned automatically to the Data Output that corresponds to the `EventDefinition` that described that trigger. (OMG 2013)

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Message End Event. As the name implies, the End Event indicates where a Process will end. In terms of Sequence Flows, the End Event ends the flow of the Process, and thus, will not have any *outgoing* Sequence Flows—no Sequence Flow can connect from an End Event (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_Int_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Interrupting Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A

Message arrives from a participant and triggers the Event. If a Message Event is attached to the boundary of an Activity, it will change the *normal flow* into an *exception flow* upon being triggered. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_Int_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Boundary (Intermediate) Non-interrupting Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Message arrives from a participant and triggers the Event. If a Message Event is attached to the boundary of an Activity, it will change the *normal flow* into an *exception flow* upon being triggered. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by

Name	Description (OMG 2013)
	something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_Strt and TPL_Event_Msg_Strt_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Message Start Event that is interrupting. If there is only one `EventDefinition` associated with the Start Event and that `EventDefinition` is of the subclass `MessageEventDefinition`, then the Event is a Message Start Event (OMG 2013)**.**

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isInterrupting: boolean = true	This attribute only applies to **Start Events** of **Event Sub-Processes**; it is ignored for other **Start Events**. This attribute denotes whether the **Sub-Process** encompassing the **Event Sub-Process** should be canceled or not, If the encompassing **Sub- Process** is not canceled, multiple *instances* of the **Event Sub-Process** can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).

Name	Description (OMG 2013)
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Interrupting:  Non-interrupting: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Msg_Throw

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Throw (Intermediate) Event, the creation of a Message Event (in this case) which is passed on to a Message Catching Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Message arrives from a participant and triggers the Event. When the Event is activated, the data in the Data Input is assigned automatically to the Message, Escalation, Error, or Signal referenced by the corresponding `EventDefinition`. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
messageRef: Message [0..1]	The **Message** MUST be supplied (if the isExecutable attribute of the **Process** is set to *true*).
operationRef: Operation [0..1]	This attribute specifies the Operation that is used by the **Message Event**. It MUST be specified for executable **Processes**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Technical) Catch (Intermediate) Signal Event, the handling of a Signal Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams. A Signal arrives that has been broadcast from another **Process** and triggers the start of the **Process**. Note that the Signal is not a **Message**, which has a specific target for the **Message** (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Signal End Event. As the name implies, the End Event indicates where a Process will end. In terms of Sequence Flows, the End Event ends the flow of the Process, and thus, will not have any *outgoing* Sequence Flows—no Sequence Flow can connect from an End Event. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams. A Signal arrives that has been broadcast from another **Process** and triggers the end of the **Process**. Note that the Signal is not a **Message**, which has a specific target for the **Message** (OMG 2013).

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_Int_Int

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Boundary (Intermediate) Interrupting Signal Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams. (OMG 2013)

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_Int_NonInt

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Boundary (Intermediate) Non-interrupting Signal Event. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_Strt (interrupting)

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Signal Start Event that is interrupting; in the usage within UAM it is always an interrupting event. This type of Event is used for sending or receiving Signals. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams (OMG 2013)**.**

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isInterrupting: boolean = true	This attribute only applies to **Start Events** of **Event Sub-Processes**; it is ignored for other **Start Events**. This attribute denotes whether the **Sub-Process** encompassing the **Event Sub-Process** should be canceled or not, If the encompassing **Sub- Process** is not canceled, multiple *instances* of the **Event Sub-Process** can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Signal_Throw

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) Throw (Intermediate) Event, the creation of a Signal Event (in this case) which is passed on to a Signal Catching Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Signal is for general communication within and across Process levels, across Pools, and between Business Process Diagrams. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
signalRef: Signal [0..1]	If the *trigger* is a `Signal`, then a `Signal` is provided.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Start

Extends

«metaclass» AcceptCallAction

Semantics

Defines a (Technical) un-typed Start Event. As the name implies, the Start Event indicates where a particular Process will start. In terms of Sequence Flows, the Start Event starts the flow of the Process, and thus, will not have any *incoming* Sequence Flows—no Sequence Flow can connect to a Start Event (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isInterrupting: boolean = true	This attribute only applies to **Start Events** of **Event Sub-Processes**; it is ignored for other **Start Events**. This attribute denotes whether the **Sub-Process** encompassing the **Event Sub-Process** should be canceled or not, If the encompassing **Sub- Process** is not canceled, multiple *instances* of the **Event Sub-Process** can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Terminate

Extends

«metaclass» FinalNode

«metaclass» ActivityFinalNode

Semantics

Defines a (Technical) Terminate Event. This event terminates the Process.

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Throw

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Technical) un-typed (None) Throw Event. The None Intermediate Event is only valid in *normal flow*, i.e., it *may not* be used on the boundary of an Activity. Although there is no specific *trigger* for this Event, it is defined as throw Event. It is used for modeling methodologies that use Events to indicate some change of state in the Process (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Name	Description (OMG 2013)
eventDefinitionRefs: EventDefinition [0..*]	References the reusable `EventDefinitions` that are *results* expected for a *throw* **Event**. Reusable `EventDefinitions` are defined as top-level elements. These `EventDefinitions` can be shared by different *catch* and *throw* **Events**. ▪ If there is no `EventDefinition` defined, then this is considered a *throw* **None Event** and the **Event** will not have an internal marker. ▪ If there is more than one `EventDefinition` defined, this is considered a *throw* **Multiple Event** and the **Event** will have the pentagon internal marker. This is an ordered set.
dataInputAssociations: DataInput Association [0..*]	The Data Associations of the *throw* Event. The `dataInputAssociation` of a *throw* Event is responsible for the assignment of a data element that is in scope of the Event to the Event data. For a *throw* **Multiple Event**, multiple **Data Associations** might be REQUIRED, depending on the individual *results* of the **Event**.
dataInputs: DataInput [0..*]	The Data Inputs for the *throw* Event. This is an ordered set.
inputSet: InputSet [0..1]	The `InputSet` for the *throw* Event.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Timer_Catch

Extends

«metaclass» AcceptEventAction

«metaclass» TimeEvent

Semantics

Defines a (Technical) Catch (Intermediate) Timer Event, the handling of a Timer Event (in this case) which is created by a Timer Throw Event. As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A Message arrives from a participant and triggers the Event. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
timeDate: Expression [0..1]	If the *trigger* is a Timer, then a timeDate MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDate MUST NOT be set (if the isExecutable attribute of the

Name	Description (OMG 2013)
	Process is set to *true*). The return type of the attribute timeDate MUST conform to the ISO-8601 format for date and time representations.
timeCycle: Expression [0..1]	If the *trigger* is a Timer, then a timeCycle MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeCycle MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeCycle MUST conform to the ISO-8601 format for recurring time interval representations.
timeDuration: Expression [0..1]	If the *trigger* is a Timer, then a timeDuration MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDuration MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeDuration MUST conform to the ISO-8601 format for time interval representations.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**, and other message flows as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Timer_Int_Int

Extends

«metaclass» AcceptEventAction

«metaclass» TimeEvent

Semantics

Defines a (Technical) Timer Boundary (Intermediate) Event that is interrupting. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process. If there is only one `EventDefinition` associated with the Start Event and that `EventDefinition` is of the subclass `TimerEventDefinition`, then the Event is a Timer Start Event (OMG 2013)**.**

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Name	Description (OMG 2013)
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
timeDate: Expression [0..1]	If the *trigger* is a Timer, then a timeDate MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDate MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeDate MUST conform to the ISO-8601 format for date and time representations.
timeCycle: Expression [0..1]	If the *trigger* is a Timer, then a timeCycle MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeCycle MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeCycle MUST conform to the ISO-8601 format for recurring time interval representations.
timeDuration: Expression [0..1]	If the *trigger* is a Timer, then a timeDuration MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDuration MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeDuration MUST conform to the ISO-8601 format for time interval representations.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Timer_Int_NonInt

Extends

«metaclass» AcceptEventAction

«metaclass» TimeEvent

Semantics

Defines a (Technical) Timer Boundary (Intermediate) Event that is non-interrupting. Boundary Events are intermediate events that can be attached to the boundary of an Activity**.** As the name implies, the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process. A specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger the start of the Process. If there is only one `EventDefinition` associated with the Start Event and that `EventDefinition` is of the subclass `TimerEventDefinition`, then the Event is a Timer Start Event (OMG 2013)**.**

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
attachedTo: Activity	If the Event is attached to the boundary of an Activity, this reference points to that the Activity.
cancelActivity: boolean	Denotes whether the **Activity** should be canceled or not, i.e., whether the *boundary catch* **Event** acts as an `Error` or an `Escalation`. If the **Activity** is not canceled, multiple *instances* of that handler can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
timeDate: Expression [0..1]	If the *trigger* is a Timer, then a timeDate MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDate MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeDate MUST conform to the ISO-8601 format for date and time representations.
timeCycle: Expression [0..1]	If the *trigger* is a Timer, then a timeCycle MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeCycle MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeCycle MUST conform to the ISO-8601 format for recurring time interval representations.
timeDuration: Expression [0..1]	If the *trigger* is a Timer, then a timeDuration MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, timeDuration MUST NOT be set (if the isExecutable attribute of the **Process** is set to *true*). The return type of the attribute timeDuration MUST conform to the ISO-8601 format for time interval representations.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Event_Timer_Strt (interrupting and non–interrupting)

Extends

«metaclass» AcceptEventAction

«metaclass» TimeEvent

Semantics

Defines a (Technical) Timer Start Event that is interrupting. A specific time-date or a specific cycle

(e.g., every Monday at 9am) can be set that will trigger the start of the Process. If there is only one `EventDefinition` associated with the Start Event and that `EventDefinition` is of the subclass `TimerEventDefinition`, then the Event is a Timer Start Event (OMG 2013)**.**

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isInterrupting: boolean = true	This attribute only applies to **Start Events** of **Event Sub-Processes**; it is ignored for other **Start Events**. This attribute denotes whether the **Sub-Process** encompassing the **Event Sub-Process** should be canceled or not, If the encompassing **Sub- Process** is not canceled, multiple *instances* of the **Event Sub-Process** can run concurrently. This attribute cannot be applied to **Error Events** (where it's always *true*), or **Compensation Events** (where it doesn't apply).
timeDate: Expression [0..1]	If the *trigger* is a Timer, then a `timeDate` MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, `timeDate` MUST NOT be set (if the `isExecutable` attribute of the **Process** is set to *true*). The return type of the attribute `timeDate` MUST conform to the ISO-8601 format for date and time representations.
timeCycle: Expression [0..1]	If the *trigger* is a Timer, then a `timeCycle` MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, `timeCycle` MUST NOT be set (if the `isExecutable` attribute of the **Process** is set to *true*). The return type of the attribute `timeCycle` MUST conform to the ISO-8601 format for recurring time interval representations.
timeDuration: Expression [0..1]	If the *trigger* is a Timer, then a `timeDuration` MAY be entered. Timer attributes are mutually exclusive and if any of the other Timer attributes is set, `timeDuration` MUST NOT be set (if the `isExecutable` attribute of the **Process** is set to *true*). The return type of the attribute `timeDuration` MUST conform to the ISO-8601 format for time interval representations.

Notation

Top-Level:  Sub-Process Interrupting:  Sub-Process Non-interrupting: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Gateway_Event and TPL_Gateway_Event_Excl and TPL_Gateway_Event_Para

Extends

«metaclass» ControlNode

Semantics

Defines a (Technical) Event-based Gateway. The Event-Based Gateway represents a branching point in the Process where the alternative paths that follow the Gateway are based on Events that occur, rather than the evaluation of Expressions using Process data (as with an Exclusive or Inclusive Gateway). A specific Event, usually the receipt of a Message, determines the path that will be taken. Basically, the *decision* is made by another *Participant*, based on data that is not visible to Process, thus, requiring the use of the Event-Based Gateway. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
gatewayDirection: GatewayDirection = Unspecified { Unspecified | Converging | Diverging | Mixed }	An attribute that adds constraints on how the **Gateway** MAY be used. ▪ **Unspecified**: There are no constraints. The **Gateway** MAY have any number of *incoming* and *outgoing* **Sequence Flows**. ▪ **Converging**: This **Gateway** MAY have multiple *incoming* **Sequence Flows** but MUST have no more than one (1) *outgoing* **Sequence Flow**. ▪ **Diverging**: This **Gateway** MAY have multiple *outgoing* **Sequence Flows** but MUST have no more than one (1) *incoming* **Sequence Flow**. ▪ **Mixed**: This **Gateway** contains multiple *outgoing* and multiple *incoming* **Sequence Flows**.
instantiate: boolean = false	When *true*, receipt of one of the **Events** will instantiate the **Process** *instance*.
eventGatewayType: EventGatewayType = Exclusive { Exclusive | Parallel }	The `eventGatewayType` determines the behavior of the **Gateway** when used to instantiate a **Process** (as described above). The attribute can only be set to parallel when the instantiate attribute is set to *true*.

Notation

Event:  Exclusive Event:  Parallel Event: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Gateway_Exclusive

Extends

«metaclass» ControlNode

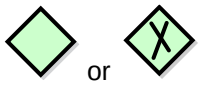
Semantics

Defines a (Technical) Exclusive Gateway. A diverging Exclusive Gateway (Decision) is used to create alternative paths within a Process flow. This is basically the "diversion point in the road" for a Process. For a given *instance* of the Process, only one of the paths can be taken. A converging Exclusive Gateway is used to merge alternative paths. Each *incoming* Sequence Flow *token* is routed to the *outgoing* Sequence Flow without synchronization. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
gatewayDirection: GatewayDirection = Unspecified { Unspecified | Converging | Diverging | Mixed }	An attribute that adds constraints on how the **Gateway** MAY be used. ▪ **Unspecified**: There are no constraints. The **Gateway** MAY have any number of *incoming* and *outgoing* **Sequence Flows**. ▪ **Converging**: This **Gateway** MAY have multiple *incoming* **Sequence Flows** but MUST have no more than one (1) *outgoing* **Sequence Flow**. ▪ **Diverging**: This **Gateway** MAY have multiple *outgoing* **Sequence Flows** but MUST have no more than one (1) *incoming* **Sequence Flow**. ▪ **Mixed**: This **Gateway** contains multiple *outgoing* and multiple *incoming* **Sequence Flows**.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such `Expression` SHALL be ignored.

Notation



Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Gateway_Inclusive

Extends

«metaclass» ControlNode

Semantics

Defines a (Technical) Inclusive Gateway. A diverging Inclusive Gateway (Inclusive Decision) can be used to create alternative but also parallel paths within a Process flow. Unlike the Exclusive Gateway, all condition Expressions are evaluated. The *true* evaluation of one condition Expression does not exclude the evaluation of other condition Expressions. All Sequence Flows with a *true* evaluation will be traversed by a *token*. Since each path is considered to be independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
gatewayDirection: GatewayDirection = Unspecified { Unspecified | Converging | Diverging | Mixed }	An attribute that adds constraints on how the **Gateway** MAY be used. ▪ **Unspecified**: There are no constraints. The **Gateway** MAY have any number of *incoming* and *outgoing* **Sequence Flows**. ▪ **Converging**: This **Gateway** MAY have multiple *incoming* **Sequence Flows** but MUST have no more than one (1) *outgoing* **Sequence Flow**. ▪ **Diverging**: This **Gateway** MAY have multiple *outgoing* **Sequence Flows** but MUST have no more than one (1) *incoming* **Sequence Flow**. ▪ **Mixed**: This **Gateway** contains multiple *outgoing* and multiple *incoming* **Sequence Flows**.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such `Expression` SHALL be ignored.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Gateway_Parallel

Extends

«metaclass» ControlNode

Semantics

Defines a (Technical) Parallel Gateway. A Parallel Gateway is used to synchronize (combine) parallel flows and to create parallel flows. A Parallel Gateway creates parallel paths without checking any conditions; each *outgoing* Sequence Flow receives a *token* upon execution of this Gateway. For *incoming* flows, the Parallel Gateway will wait for all *incoming* flows before triggering the flow through its *outgoing* Sequence Flows. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Name	Description (OMG 2013)
gatewayDirection: GatewayDirection = Unspecified { Unspecified | Converging | Diverging | Mixed }	An attribute that adds constraints on how the **Gateway** MAY be used. ▪ **Unspecified**: There are no constraints. The **Gateway** MAY have any number of *incoming* and *outgoing* **Sequence Flows**. ▪ **Converging**: This **Gateway** MAY have multiple *incoming* **Sequence Flows** but MUST have no more than one (1) *outgoing* **Sequence Flow**. ▪ **Diverging**: This **Gateway** MAY have multiple *outgoing* **Sequence Flows** but MUST have no more than one (1) *incoming* **Sequence Flow**. ▪ **Mixed**: This **Gateway** contains multiple *outgoing* and multiple *incoming* **Sequence Flows**.

Notation

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Group

Extends

«metaclass» Classifier

Semantics

The Group object is an `Artifact` (i.e. additional information about a Process that is not directly related to the Sequence Flows or Message Flows) that provides a visual mechanism to group elements of a diagram informally. The grouping is tied to the `CategoryValue` supporting element. That is, a Group is a visual depiction of a single `CategoryValue`. The graphical elements within the Group will be assigned the `CategoryValue` of the Group. `Categories`, which have user-defined semantics, can be used for documentation or analysis purposes. For example, `FlowElements` can be categorized as being customer oriented vs. support oriented. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
categoryValueRef: CategoryValue [0..1]	The `categoryValueRef` attribute specifies the `CategoryValue` that the **Group** represents. The name of the `Category` and the value of the `CategoryValue` separated by delineator "." provides the label for the **Group**. The graphical elements within the boundaries of the **Group** will be assigned the `CategoryValue`.

Notation

Constraints

- May group (categorize) any elements, even across Pools and Lanes;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Lane

Extends

«metaclass» Activity Partition

Semantics

A Lane is a sub-partition within a Process (often within a Pool) and will extend the entire length of the Process level, either vertically or horizontally. Lanes are used to organize and categorize Activities within a Pool. The meaning of the Lanes is up to the modeller. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the Lane.
partitionElement: BaseElement [0..1]	A reference to a `BaseElement` that specifies the partition value and partition type. Using this partition element a **BPMN** compliant tool can determine the `FlowElements` that have to be partitioned in this **Lane**.
partitionElementRef: BaseElement [0..1]	A reference to a `BaseElement` that specifies the partition value and partition type. Using this partition element a **BPMN** compliant tool can determine the `FlowElements` that have to be partitioned in this **Lane**.
childLaneSet: LaneSet [0..1]	A reference to a `LaneSet` element for embedded Lanes**.**
flowNodeRefs: FlowNode [0..*]	The list of `FlowNodes` partitioned into this Lane according to the `partitionElement` defined as part of the Lane element.

Notation

Constraints

- May have relationships (message flows) with other message flow elements as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_LaneSet

Extends

«metaclass» Activity Partition

Semantics

The `LaneSet` element defines the container for one or more `Lanes`. A Process can contain one

or more `LaneSets`. Each `LaneSet` and its Lanes can partition the *Flow Nodes* in a different way. (OMG 2013)

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	The name of the `LaneSet`. A `LaneSet` is not visually displayed on a **BPMN** diagram. Consequently, the name of the `LaneSet` is not displayed as well.
process: Process	The **Process** owning the `LaneSet`
lanes: Lane [0..*]	One or more **Lane** elements, which define a specific partition in the `LaneSet`.
parentLane: Lane [0..1]	The reference to a **Lane** element which is the parent of this `LaneSet`.

Notation

Constraints

- May have relationships (message flows) with other message flow elements as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Message_Flow

Extends

«metaclass» Association

Semantics

Defines a (Technical) Message Flow which is used to show the flow of Messages between two Participants that are prepared to send and receive them. A Message Flow *must* connect two separate Pools. They connect either to the Pool boundary or to Flow Objects within the Pool boundary. They *must not* connect two objects within the same Pool. (OMG 2013)

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	Name is a text description of the **Message Flow**.
sourceRef: InteractionNode	The `InteractionNode` that the **Message Flow** is connecting from. Of the types of `InteractionNode`, only **Pools**/*Participants*, **Activities**, and **Events** can be the *source* of a **Message Flow**.

Name	Description (OMG 2013)
targetRef: InteractionNode	The `InteractionNode` that the **Message Flow** is connecting to. Of the types of `InteractionNode`, only **Pools**/*Participants*, **Activities**, and **Events** can be the *target* of a **Message Flow**.
messageRef: Message [0..1]	The `messageRef` model association defines the **Message** that is passed via the **Message Flow**.

Notation

Constraints

- Specifies relationships (message flows) between message flow elements as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Message_Receive

Extends

«metaclass» Class

Semantics

Defines a (Technical) Receive Message. A Message represents the content of a communication between two *Participants*. In BPMN 2.0, a Message is a graphical decorator (it was a supporting element in BPMN 1.2). An `ItemDefinition` is used to specify the Message structure. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	Name is a text description of the **Message**
itemRef: ItemDefinition [0..1]	An `ItemDefinition` is used to define the “payload” of the **Message**.

Notation

Constraints

- Part of the message flow definitions with other message flow elements; constraints are defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Message_Send

Extends

«metaclass» Class

Semantics

Defines a (Technical) Send Message. A Message represents the content of a communication between two *Participants*. In BPMN 2.0, a Message is a graphical decorator (it was a supporting element in BPMN 1.2). An `ItemDefinition` is used to specify the Message structure. (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	Name is a text description of the **Message**
itemRef: ItemDefinition [0..1]	An `ItemDefinition` is used to define the "payload" of the **Message**.

Notation

Constraints

- Part of the message flow definitions with other message flow elements; constraints are defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Pool

Extends

«metaclass» Activity Partition

Semantics

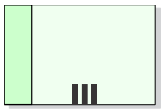
A (Technical) Pool is the graphical representation of a `Participant` in a Collaboration. A *Participant* can be a specific `PartnerEntity` (e.g., a company) or can be a more general `PartnerRole` (e.g., a buyer, seller, or manufacturer). A Pool *may* or *may not* reference a Process. A Pool is *not required* to contain a Process, i.e., it can be a "black box." (OMG 2013)

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string [0..1]	Name is a text description of the *Participant*. The name of the *Participant* can be displayed directly or it can be substituted by the associated `PartnerRole` or `PartnerEntity`. Potentially, both the `PartnerEntity` name and `PartnerRole` name can be displayed for the *Participant*.
processRef: Process [0..1]	The `processRef` attribute identifies the **Process** that the `Participant` uses in the *Collaboration*. The **Process** will be displayed within the *Participant's* Pool.

Name	Description (OMG 2013)
partnerRoleRef: PartnerRole [0..*]	The `partnerRoleRef` attribute identifies a `PartnerRole` that the *Participant* plays in the Collaboration. Both a `PartnerRole` and a `PartnerEntity` may be defined for the *Participant*. This attribute is derived from the `participantRefs` of `PartnerRole`.
partnerEntityRef: PartnerEntity [0..*]	The partnerEntityRef attribute identifies a PartnerEntity that the *Participant* plays in the *Collaboration*. Both a PartnerRole and a PartnerEntity MAY be defined for the *Participant*.This attribute is derived from the participantRefs of PartnerEntity.
interfaceRef: Interface [0..*]	This association defines `Interfaces` that a *Participant* supports. An `Interface` defines a set of operations that are implemented by `Services`.
participantMultiplicity: participantMultiplicity [0..1]	The `participantMultiplicityRef` model association is used to define *Participants* that represent more than one (1) instance of the *Participant* for a given interaction. See the next section for more details on `ParticipantMultiplicity`.
endPointRefs: EndPoint [0..*]	This attribute is used to specify the address (or endpoint reference) of concrete services realizing the *Participant*.

Notation


Multi-Instance:


Constraints

- May have relationships (message flows) with other message flow elements as defined in **Table 3**.
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Rule

Extends

«metaclass» Constraint

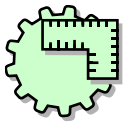
Semantics

Defines a Technical Rule which is a declaration of policy or a condition that must be satisfied. Techncial Rules are used (implemented) by Activities, Gateways and Events within processes.

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	Name is a short name / description of the **Business Rule.**
rule: string	The **Business Rule** definition

Notation



Constraints

- Technical Rules are sub-sets of the Technical Domain rules and policies;
- Technical Rules are used and implemented by Activities, Gateways and Events but are separate from these elements.

«stereotype» TPL_Seq_Flow

Extends

«metaclass» Association

Semantics

Defines a (Technical) Sequence Flow which is used to show the order of Flow Elements in a Process or a Choreography. Each Sequence Flow has only one *source* and only one *target*. The *source* and *target* must be from the set of the following Flow Elements: Events (Start, Intermediate, and End), Activities (Task and Sub-Process; for Processes), Choreography Activities (Choreography Task and Sub-Choreography; for Choreographies), and Gateways. (OMG 2013)

Properties

Name	Description (OMG 2013)
sourceRef: FlowNode	The `FlowNode` that the Sequence Flow is connecting from. For a Process: Of the types of `FlowNode`, only Activities, Gateways, and Events can be the *source*. However, Activities that are Event Sub-Processes are not allowed to be a *source*.
targetRef : FlowNode	The `FlowNode` that the Sequence Flow is connecting to. For a Process: Of the types of `FlowNode`, only Activities, Gateways, and Events can be the *target*. However, Activities that are Event Sub-Processes are not allowed to be a *target*.
conditionExpression: Expression [0..1]	An optional boolean `Expression` that acts as a gating condition. A *token* will only be placed on this Sequence Flow if this `conditionExpression` evaluates to true.
isImmediate: boolean [0..1]	An optional boolean value specifying whether Activities or Choreography Activities not in the model containing the Sequence Flow can occur between the elements connected by the Sequence Flow. If the value is true, they may not occur. If the value is false, they may occur. Also see the isClosed attribute on Process, Choreography, and Collaboration. When the attribute has no value, the default semantics depends on the kind of model containing **Sequence Flows**: ▪ For non-executable **Processes** (public **Processes** and non-executable private **Processes**) and **Choreographies** no value has the same semantics as if the value were *false*. ▪ For an executable **Process** no value has the same semantics as if the value were *true*. ▪ For executable **Processes**, the attribute MUST NOT be *false*.

Notation

Constraints

- Used to define / diagram the sequence flows between flow elements; constraints are defined in Table 2;
- See BPMN v2.0 (OMG 2013).

Sequence Flow Connections Rules (OMG 2013)

Table 2 displays the BPMN *Flow Objects* and shows how these objects can connect to one another through *Sequence Flows*. These rules apply to the connections within a *Process Diagram* and within a *Choreography Diagram*. The ⇑ symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies are not specified here. Note that if a *Sub-Process* has been expanded within a diagram, the objects within the *Sub-Process* cannot be connected to objects outside of the *Sub-Process*. Nor can *Sequence Flows* cross a *Pool* boundary.

Only those objects that can have incoming and outgoing *Sequence Flows* are shown in the table. Thus, *Pool*, *Lane*, *Data Object*, *Group*, and *Text Annotation* are not listed in the table. Also, the *Activity* shapes in the table represent *Activities* and *Sub-Processes* for *Processes*, and *Choreography Activities* and *Sub-Choreographies* for *Choreography*.

Table 2 – Sequence Flow Connection Rules

From\To						
		⇑	⇑	⇑	⇑	⇑
		⇑	⇑	⇑	⇑	⇑
		⇑	⇑	⇑	⇑	⇑
		⇑	⇑	⇑	⇑	⇑
		⇑	⇑	⇑	⇑	⇑

Message Flow Connection Rules (OMG 2013)

Table 3 displays the *BPMN* modeling objects and shows how these objects can connect to one another through *Message Flows*. These rules apply to the connections within a *Collaboration* diagram. The ⇑ symbol indicates that the object listed in the row can connect to the object listed in the column. The quantity of connections into and out of an object is subject to various configuration dependencies that are not specified here. *Note that Message Flows cannot connect to objects that are within the same Pool*.

Only those objects that can have incoming and outgoing *Message Flows* are shown in the table. Thus, *Lane, Gateway, Data Object, Group,* and *Text Annotation* are not listed in the table.

Table 3 – Message Flow Connection Rules

From\To						
						
	⇑	⇑	⇑	⇑	⇑	
	⇑	⇑	⇑	⇑	⇑	
	⇑	⇑	⇑	⇑	⇑	
	⇑	⇑	⇑	⇑	⇑	
	⇑	⇑	⇑	⇑	⇑	

«stereotype» TPL_Subprocess

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Technical) Sub-process Activity is a type of activity, as defined in BPML V2.0, “an Activity is work performed as part of a business process”, that acts as a wrapper for an inner Activity that can be executed multiple times in sequence. (OMG 2013)

A Sub-Process may also Loop, or Multi-instance Parallel or Sequential Activity. These are activities that loop or that have multiple *instances* spawned in parallel or multiple *instances* spawned sequentially. The *instances* MAY execute in parallel or MAY be sequential. Either an Expression is used to specify or calculate the desired number of *instances* or a data driven setup can be used. In that case a data input can be specified, which is able to handle a collection of data. The number of items in the collection determines the number of **Activity** *instances*. This data input can be produced by an input **Data Association**. (OMG 2013)

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Subprocesses** have the following attributes:

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
triggeredByEvent: boolean = false	A flag that identifies whether this **Sub-Process** is an **Event Sub-Process**.

Name	Description (OMG 2013)
	➢ If *false*, then this **Sub-Process** is a normal **Sub-Process**. ➢ If *true*, then this **Sub-Process** is an **Event Sub-Process** and is subject to additional constraints In UAM only *false* is supported, Event Sub-Processes are not used.
artifacts: Artifact [0..*]	This attribute provides the list of **Artifacts** that are contained within the **Sub- Process**.

Notation

There are two views of sub-processes, either *expanded* or *collapsed*. Each of these types may also have additional "markers" signifying that they are *loop* (of three types) or *compensation* sub-processes. These markers may also be used in combination, as illustrated in the figures below that show how they are graphically represented. Note that the *loop*, *parallel* and *sequential* markers are however mutually exclusive.

Sub-process:

Expanded:  Collapsed: 

Sub-processes with markers (only collapsed variant shown):

Loop:  Parallel:  Sequential: 

Compensation:  Compensation-Loop: 

Compensation-Parallel:  Compensation-Sequential: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Subprocess_Adhoc

Extends

«*metaclass*» Activity

«*metaclass*» StructuredActivityNode

Semantics

A (Technical) Ad-Hoc Sub-process is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process", that acts as a wrapper for an inner Activity that can be executed multiple times in sequence. (OMG 2013)

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Adhoc Subprocesses** have the following attributes:

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater

Name	Description (OMG 2013)
	than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
triggeredByEvent: boolean = false	A flag that identifies whether this **Sub-Process** is an **Event Sub-Process**. ➢ If *false*, then this **Sub-Process** is a normal **Sub-Process**. ➢ If *true*, then this **Sub-Process** is an **Event Sub-Process** and is subject to additional constraints In UAM only *false* is supported, Event Sub-Processes are not used.
artifacts: Artifact [0..*]	This attribute provides the list of **Artifacts** that are contained within the **Sub- Process**.
completionCondition: Expression	This `Expression` defines the conditions when the **Process** will end. When the `Expression` is evaluated to *true*, the **Process** will be terminated.
ordering: AdHocOrdering = Parallel { Parallel | Sequential }	This attribute defines if the **Activities** within the **Process** can be performed in parallel or MUST be performed sequentially. The default setting is `parallel` and the setting of `sequential` is a restriction on the performance that can be needed due to shared resources. When the setting is `sequential`, then only one Activity can be performed at a time. When the setting is `parallel`, then `zero (0)` to all the **Activities** of the **Sub-Process** can be performed in parallel.
cancelRemaining-Instances: boolean = true	This attribute is used only if ordering is parallel. It determines whether running *instances* are canceled when the `completionCondition` becomes *true*.

Notation

There are two views of ad-hoc sub-processes, either *expanded* or *collapsed*. Each of these types may also have additional "markers" signifying that they are *loop* (of three types) or *compensation* sub-processes. These markers may also be used in combination, as illustrated in the figures below that show how they are graphically represented. Note that the *loop*, *parallel* and *sequential* markers are however mutually exclusive.

Ad-hoc Sub-process:

Expanded:  Collapsed: 

Ad-hoc Sub-processes with Markers (only collapsed variant shown)

Loop:  Parallel:  Sequential: 

Compensation:  Compensation-Loop: 

Compensation-Parallel:  Compensation-Sequential: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task

Extends

«metaclass» Activity

Semantics

A (Technical) Task is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process" (OMG 2013). A Task is atomic in that it cannot be broken down into smaller component activities.

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Tasks** have the following attributes:

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.

Name	Description (OMG 2013)
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
triggeredByEvent: boolean = false	A flag that identifies whether this **Sub-Process** is an **Event Sub-Process**. ➢ If *false*, then this **Sub-Process** is a normal **Sub-Process**. ➢ If *true*, then this **Sub-Process** is an **Event Sub-Process** and is subject to additional constraints In UAM only *false* is supported, Event Sub-Processes are not used.
artifacts: Artifact [0..*]	This attribute provides the list of **Artifacts** that are contained within the **Sub- Process**.

Notation

There are seven types of Task: *undefined*, *Business Rule*, *Manual*, *Receive*, *Script*, *Service* and *User* Tasks. Each of these types Task may also have additional "markers" signifying that they are *loop* (of three types) or *compensation* Tasks. These markers may also be used in combination, as illustrated in the figures below that show how they are graphically represented. Note that the *loop*, *parallel* and *sequential* markers are however mutually exclusive.

Task Types:

Undefined:  Rule:  Manual:  Receive: 

Script:  Service:  User: 

Task with Markers (only the *undefined* Task Type shown):

Loop:  Parallel:  Sequential: 

Compensation:  Compensation-Loop: 

Compensation-Parallel:  Compensation-Sequential: 

Constraints

➲ May have relationships (sequence flows) with other flow elements as defined in Table 2; See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Manual

Extends

«metaclass» Activity

«metaclass» OpaqueAction

Semantics

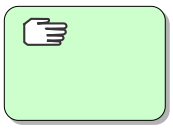
A (Technical) Manual Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process" (OMG 2013). A Manual Task is a Task that is expected to be performed without the aid of any business process execution engine or any application. An example of this could be a telephone technician installing a telephone at a customer location (OMG 2013).

Properties

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.

Name	Description (OMG 2013)
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).

Notation



Manual Task with markers (NOTE: the other Task Types (i.e. Receive, Business Rules, etc.) defined below may also use these markers, however only the basic notation is shown in their definitions):

Loop: 

Parallel: 

Sequential: 

Compensation: 

Compensation-Loop: 

Compensation-Parallel: 

Compensation-Sequential: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Receive

Extends

«metaclass» Activity

«metaclass» AcceptEventAction

Semantics

A (Technical) Receive Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process" (OMG 2013). A Receive Task is a simple Task that is designed to wait for a Message to arrive from an external Participant (relative to the Process). Once the Message has been received, the Task is completed (OMG 2013).

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Receive Tasks** have the following attributes:

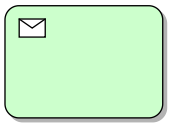
Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows**

Name	Description (OMG 2013)
	evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
messageRef: Message [0..1]	A **Message** for the `messageRef` attribute MAY be entered. This indicates that the **Message** will be received by the **Task**. The **Message** in this context is equivalent to an *in-only* message pattern (Web service). One (1) or more corresponding *incoming* **Message Flows** MAY be shown on the diagram. However, the display of the **Message Flows** is NOT REQUIRED. The **Message** is applied to all *incoming* **Message Flows**, but can arrive for only one (1) of the *incoming* **Message Flows** for a single *instance* of the **Task**.
instantiate: boolean = false	**Receive Tasks** can be defined as the instantiation mechanism for the Process with the instantiate attribute. This attribute MAY be set to *true* if the **Task** is the first **Activity** (i.e., there are no *incoming* **Sequence Flows**). Multiple **Tasks** MAY have this attribute set to *true*.
operationRef: Operation	This attribute specifies the operation through which the **Receive Task** receives the **Message**.
implementation: string = ##webService	This attribute specifies the technology that will be used to send and receive the **Messages**. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol A Web service is the default technology.

Notation

NOTE: this Task Type may also use the *loop, parallel, sequential, compensation, compensation loop, compensation parallel* and *compensation sequential* markers, however only the basic notation is

shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Rules

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Techncial) Business Rules Task is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process". A Business Rules Task is a Task that uses some sort of service, which could be a Web service or an automated application (OMG 2013).

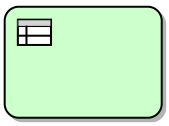
Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**

Name	Description (OMG 2013)
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
implementation: string = ##unspecified	This attribute specifies the technology that will be used to implement the **Business Rule Task**. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol. The default technology for this task is unspecified.

Notation

NOTE: this Task Type may also use the *loop*, *parallel*, *sequential*, *compensation*, *compensation loop*, *compensation parallel* and *compensation sequential* markers, however only the basic notation is shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Script

Extends

«metaclass» Activity

«metaclass» CallOperationAction

Semantics

A (Techncial) Script Task is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process". A Script Task is executed by a business process engine. The

modeller or implementer defines a script in a language that the engine can interpret. When the Task is ready to start, the engine will execute the script. When the script is completed, the Task will also be completed (OMG 2013).

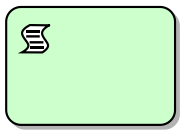
Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.

Name	Description (OMG 2013)
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
scriptFormat: string [0..1]	Defines the format of the script. This attribute value MUST be specified with a mime-type format. And it MUST be specified if a script is provided.
script: string [0..1]	The modeler MAY include a script that can be run when the **Task** is performed. If a script is not included, then the **Task** will act as the equivalent of an **Abstract Task**.

Notation

NOTE: this Task Type may also use the *loop*, *parallel*, *sequential*, *compensation*, *compensation loop*, *compensation parallel* and *compensation sequential* markers, however only the basic notation is shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Send

Extends

«metaclass» Activity

«metaclass» CallOperationAction

Semantics

A (Technical) Send Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process"(OMG 2013). A Send Task is a simple Task that is designed to send a Message to an external Participant (relative to the Process). Once the Message has been sent, the Task is completed (OMG 2013).

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Send Tasks** have the following attributes:

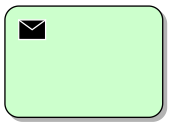
Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by

Name	Description (OMG 2013)
	something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
messageRef: Message [0..1]	A **Message** for the `messageRef` attribute MAY be entered. This indicates that the **Message** will be received by the **Task**. The **Message** in this context is equivalent to an *in-only* message pattern (Web service). One (1) or more corresponding *incoming* **Message Flows** MAY be shown on the diagram. However, the display of the **Message Flows** is NOT REQUIRED. The **Message** is applied to all *incoming* **Message**

Name	Description (OMG 2013)
	Flows, but can arrive for only one (1) of the *incoming* **Message Flows** for a single *instance* of the **Task**.
operationRef: Operation	This attribute specifies the operation through which the **Send Task** receives the **Message**.
implementation: string = ##webService	This attribute specifies the technology that will be used to send and receive the **Messages**. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol A Web service is the default technology.

Notation

NOTE: this Task Type may also use the *loop*, *parallel*, *sequential*, *compensation*, *compensation loop*, *compensation parallel* and *compensation sequential* markers, however only the basic notation is shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_Service

Extends

«*metaclass*» Activity

«*metaclass*» StructuredActivityNode

Semantics

A (Technical) Service Task is an atomic activity, as defined in BPML V2.0, “an Activity is work performed as part of a business process”(OMG 2013). A Service Task is a Task that uses some sort of service, which could be a Web service or an automated application (OMG 2013).

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

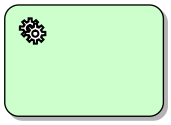
In addition to the above, **Service Tasks** have the following attributes:

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
implementation: string = ##webService	This attribute specifies the technology that will be used to send and receive the **Messages**. Valid values are "##unspec-ified" for leaving the implementation technology open, "##WebService" for the Web service

Name	Description (OMG 2013)
	technology or a URI identifying any other technology or coordination protocol. A Web service is the default technology.
operationRef: Operation [0..1]	This attribute specifies the operation that is invoked by the Service Task.

Notation

NOTE: this Task Type may also use the *loop*, *parallel*, *sequential*, *compensation*, *compensation loop*, *compensation parallel* and *compensation sequential* markers, however only the basic notation is shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in Table 2;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Task_User

Extends

«metaclass» Activity

«metaclass» OpaqueAction

Semantics

A (Technical) User Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process"(OMG 2013). A User Task "is a typical 'workflow' Task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort"(OMG 2013).

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

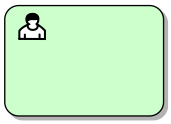
In addition to the above, **User Tasks** have the following attributes:

Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.

Name	Description (OMG 2013)
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
implementation: string = ##unspecified	This attribute specifies the technology that will be used to implement the **User Task**. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol. The default technology for this task is unspecified.
renderings: Rendering [0..*]	This attributes acts as a hook which allows **BPMN** adopters to specify task rendering attributes by using the **BPMN** Extension mechanism.

Notation

NOTE: this Task Type may also use the *loop*, *parallel*, *sequential*, *compensation*, *compensation loop*, *compensation parallel* and *compensation sequential* markers, however only the basic notation is shown here—see these notions in the Manual Task definition.



Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_TextAnnotation

Extends

«metaclass» comment

Semantics

The (Technical) **Text Annotations** are a mechanism for a modeller to provide additional information for the reader of a **BPMN** Diagram (OMG 2013).

Properties

Name	Description (OMG 2013)
text: string	Text is an attribute that is text that the modeller wishes to communicate to the reader of the Diagram.
textFormat: string	This attribute identifies the format of the text. It MUST follow the mime-type format. The default is "text/plain."

Notation

Constraints

- The **Text Annotation** object can be connected to a specific object on the Diagram with an **Association**, but does not affect the flow of the **Process**. Text associated with the **Annotation** can be placed within the bounds of the open rectangle;
- See BPMN v2.0 (OMG 2013).

«stereotype» TPL_Transaction

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

Defines a (Technical) Sub-Process, either expanded or collapsed, and is a type of activity. As defined in BPML V2.0 "an Activity is work performed as part of a business process" (OMG 2013); a sub-

process contains a set of activities, gateways, events, and sequence flows. It can be viewed as a subroutine that captures a possibly reusable set of activities and flows.

Properties

Name	Description
description: string	A description of the business activities, task, and functions captured by the activity.
owner: string	Defines the owner of the business activity or process—the organizational element that makes decisions about the process.
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

In addition to the above, **Transactions** have the following attributes:

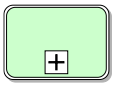
Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.

Name	Description (OMG 2013)
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
triggeredByEvent: boolean = false	A flag that identifies whether this **Sub-Process** is an **Event Sub-Process**. ➢ If *false*, then this **Sub-Process** is a normal **Sub-Process**. ➢ If *true*, then this **Sub-Process** is an **Event Sub-Process** and is subject to additional constraints In UAM only *false* is supported, Event Sub-Processes are not used.
artifacts: Artifact [0..*]	This attribute provides the list of **Artifacts** that are contained within the **Sub- Process**.
method: TransactionMethod	The method is an attribute that defines the **Transaction** method used to commit or cancel a **Transaction**. For executable **Processes**, it SHOULD be set to a technology specific URI, e.g., `http://schemas.xmlsoap.org/ws/2004/10/wsat` for WS-AtomicTransaction, **or** `http://docs.oasis-open.org/ws-tx/wsba/2006/06/AtomicOutcome` for WS-BusinessActivity.

Notation

There are two views of Transactions, either *expanded* or *collapsed*. Each of these types may also have additional "markers" signifying that they are *loop* (of three types) or *compensation* Transactions. These markers may also be used in combination, as illustrated in the figures below that show how they are graphically represented. Note that the *loop*, *parallel* and *sequential* markers are however mutually exclusive.

Transaction:

Expanded:  Collapsed: 

Transactions with Markers (only collapsed variant shown):

Loop: 

Parallel: 

Sequential: 

Compensation: 

Compensation-Loop: 

Compensation-Parallel: 

Compensation-Sequential: 

Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

Loop Characteristics Summary

Tasks or Sub-processes or Call Activities may have loop characteristics, which are summarised below. This summary is divided into two parts: `StandardLoopCharacteristics` and `MultiInstanceLoopCharacteristics`.

Standard characteristics apply to Activities and Tasks that have a simple looping structure; the attributes associated with this behaviour are the `StandardLoopCharacteristics`:

Name	Description (OMG 2013)
testBefore: boolean = false	Flag that controls whether the loop condition is evaluated at the beginning (`testBefore` = *true*) or at the end (`testBefore` = *false*) of the loop iteration.
loopMaximum: integer [0..1]	Serves as a cap on the number of iterations.
loopCondition: Expression [0..1]	A boolean `Expression` that controls the loop. The **Activity** will only loop as long as this condition is *true*. The looping behavior MAY be underspecified, meaning that the modeler can simply document the condition, in which case the loop cannot be formally executed.

Multi-instance characteristics apply to Activities and Tasks that are instantiated (run) either in parallel or sequentially; the attributes associated with this behaviour are the `MultiInstanceLoopCharacteristics`:

Name	Description (OMG 2013)
isSequential: boolean = false	This attribute is a flag that controls whether the **Activity** *instances* will execute sequentially or in parallel.
loopCardinality: Expression [0..1]	A numeric Expression that controls the number of **Activity** *instances* that will be created. This Expression MUST evaluate to an *integer*. This MAY be underspecified, meaning that the modeler MAY simply document the condition. In such a case the *loop* cannot be formally executed. In order to initialize a valid *multi-instance,* either the `loopCardinality` Expression or the `loopDataInput` MUST be specified.
loopDataInputRef: ItemAwareElement [0..1]	This `ItemAwareElement` is used to determine the number of **Activity** *instances*, one **Activity** *instance* per item in the collection of data stored in that `ItemAwareElement` element. For **Tasks** it is a reference to a **Data Input** which is part of the **Activity**'s `InputOutputSpecification`. For **Sub-Processes** it is a reference to a collection-valued **Data Object** in the context that is visible to the **Sub-Processes**. In order to initialize a valid *multi-instance*, either the `loopCardinality` Expression or the `loopDataInput` MUST be specified.
loopDataOutputRef: ItemAwareElement [0..1]	This `ItemAwareElement` specifies the collection of data, which will be produced by the *multi-instance*. For **Tasks** it is a reference to a **Data Output** which is part of the **Activity**'s `InputOutputSpecification`.

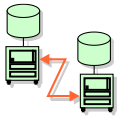
Name	Description (OMG 2013)
	For **Sub-Processes** it is a reference to a collection-valued **Data Object** in the context that is visible to the **Sub-Processes**.
inputDataItem: DataInput [0..1]	A **Data Input**, representing for every **Activity** *instance* the single item of the collection stored in the `loopDataInput`. This **Data Input** can be the source of `DataInputAssociation` to a data input of the **Activity**'s `InputOutputSpecification`. The type of this **Data Input** MUST the scalar of the type defined for the `loopDataInput`.
outputDataItem: DataOutput [0..1]	A **Data Output**, representing for every **Activity** *instance* the single item of the collection stored in the `loopDataOutput`. This **Data Output** can be the target of `DataOutputAssociation` to a data output of the **Activity**'s `InputOutputSpecification`. The type of this **Data Output** MUST the scalar of the type defined for the `loopDataOutput`.
behavior: MultiInstanceBehavior = all { None | One | All | Complex }	The attribute behavior acts as a shortcut for specifying when events SHALL be thrown from an **Activity** *instance* that is about to complete. It can assume values of None, One, All, and Complex, resulting in the following behavior: ➢ **None**: the `EventDefinition` which is associated through the `noneEvent` association will be thrown for each *instance* completing. ➢ **One**: the `EventDefinition` referenced through the `oneEvent` association will be thrown upon the first *instance* completing. ➢ **All**: no **Event** is ever thrown; a *token* is produced after completion of all *instances*. ➢ **Complex**: the `complexBehaviorDefinitions` are consulted to determine if and which **Events** to throw. For the behaviors of none and one, a default `SignalEventDefinition` will be thrown which automatically carries the current runtime attributes of the **MI Activity**. Any thrown **Events** can be caught by *boundary* **Events** on the **Multi- Instance Activity**.
complexBehaviorDefinition: ComplexBehaviorDefinition [0..*]	Controls when and which **Events** are thrown in case behavior is set to complex.
completionCondition: Expression [0..1]	This attribute defines a boolean Expression that when evaluated to *true*, cancels the remaining **Activity** *instances* and produces a *token*.
oneBehaviorEventRef: EventDefinition [0..1]	The `EventDefinition` which is thrown when behavior is set to one and the first internal **Activity** *instance* has completed.
noneBehaviorEventRef: EventDefinition [0..1]	The `EventDefinition` which is thrown when the behavior is set to none and an internal **Activity** *instance* has completed.

The table below lists all *instance* attributes available at runtime for loop activities. For each *instance* of the **Multi-Instance Activity** (outer *instance*), there exists a number of generated (inner) *instances* of the **Activity** at runtime.

Name	Description (OMG 2013)
loopCounter: integer	This attribute is provided for each generated (inner) *instance* of the **Activity**. It contains the sequence number of the generated *instance*, i.e., if this value of some *instance* in n, the *instance* is the n-th *instance* that was generated.
numberOfInstances: integer	This attribute is provided for the outer *instance* of the **Multi-Instance Activity** only. This attribute contains the total number of inner *instances* created for the **Multi-Instance Activity**.

Name	Description (OMG 2013)
numberOfActiveInstances: integer	This attribute is provided for the outer *instance* of the **Multi-Instance Activity** only. This attribute contains the number of currently active inner *instances* for the **Multi-Instance Activity**. In case of a sequential **Multi-Instance Activity**, this value can't be greater than 1. For parallel **Multi-Instance Activities**, this value can't be greater than the value contained in numberOfInstances.
numberOfCompletedInstances: integer	This attribute is provided for the outer *instance* of the **Multi-Instance Activity** only. This attribute contains the number of already com-pleted inner *instances* for the **Multi-Instance Activity**.
numberOfTerminatedInstances: integer	This attribute is provided for the outer *instance* of the **Multi-Instance Activity** only. This attribute contains the number of terminated inner *instances* for the **Multi-Instance Activity**. The sum of `numberOfTerminatedInstances`, `numberOfCompletedInstances`, and `numberOfActiveInstances` always sums up to `numberOfInstances`.

Technical Locations Viewpoint



The Technical Locations Viewpoint is used, within the enterprise architecture context, to either specify technical patterns for the enterprise, or if the scope, scale and resources are appropriate to document a physical (near-implementation) level view of enterprise locations. Both may be described, if desired and appropriate, as well. UAM processes and templates support either or both of these approaches and objectives.

The locations viewpoint (Figure 8) is a key element of the Technical Perspective. It may be used to show how entities (data), processes and the network aspects (and the required supporting infrastructure) are organized and deployed throughout the system (or enterprise). In other words, some entity model, process model and roles model aspects may be shown to best advantage in the locations model. Partitioning and layering is used to separate these aspects in the locations model.

The locations model works best if layering is used to deal with complexity and to provide views appropriate for the audience and objectives; and comprehension is greatly improved. Detail is hidden, while allowing for its definition in lower layers, as required in meeting the objectives of the architecture effort.

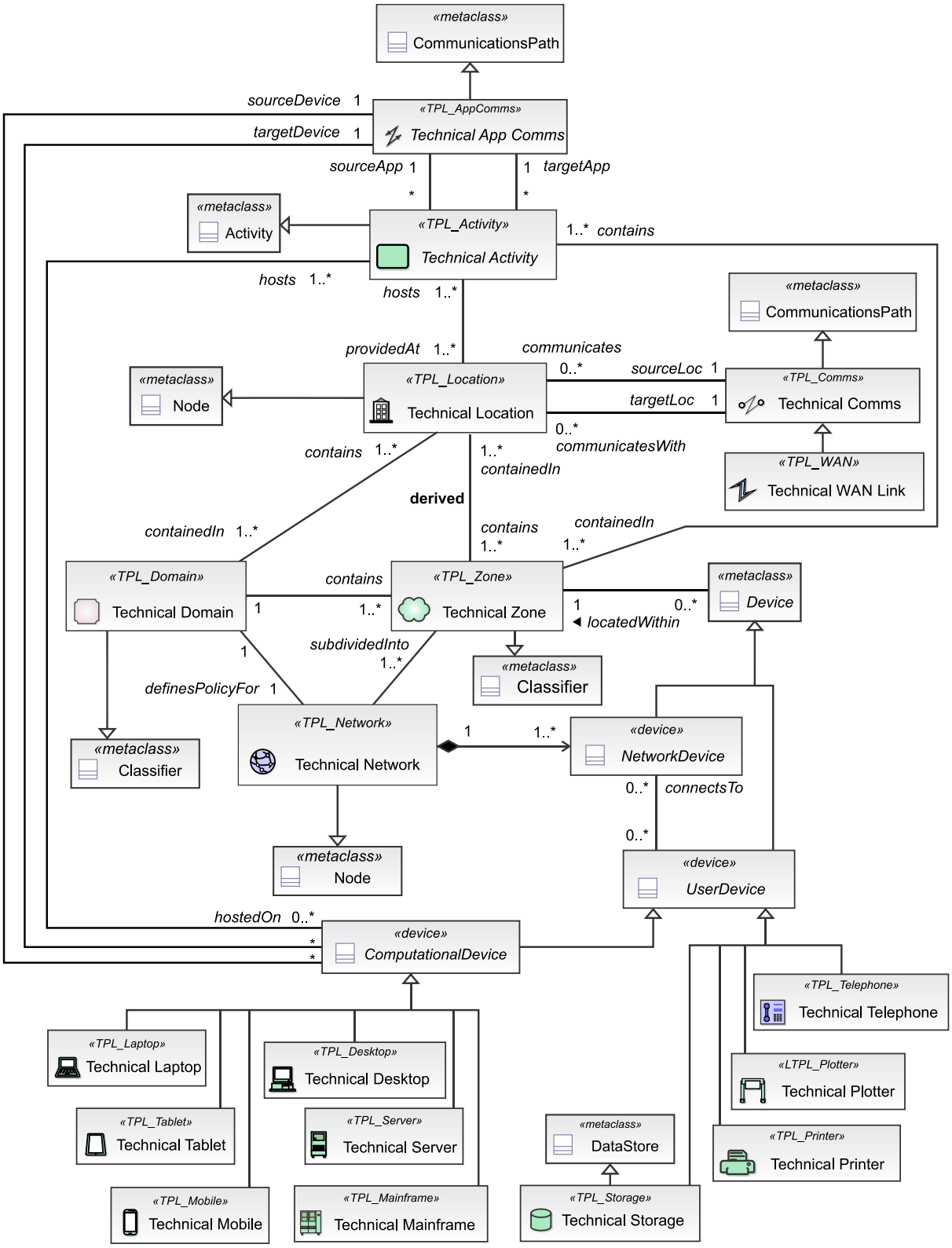


Figure 8 – Technical Locations Viewpoint Language (part 1)

For example, at each *location* layers at different levels of complexity may be used to show the boundary network technologies, with more detail for hidden components being provided at lower layers.

These layers are also partitioned in order to separate out the different aspects (i.e. entity, process, network and role). The main focus of the technical locations model is the 'network' aspect. The network topology and technologies are defined, as generic patterns at each location, or they may be specific in terms of technologies and where and how to use them, such as 'CISCO Router/Firewall XYZ' with configuration 'abc' (a very detailed specification, which may be appropriate for some architecture contexts). These standards and patterns may be shown in this model (providing an overall picture including usage); however they *must* be documented in Architectural Decisions.

Complexity may also be dealt with through partitioning of each 'location' into the four different aspects: the Data, Activity, Network, and People Viewpoints with each showing the deployment to locations of data, processes, technologies and services. Layering these aspects (partitions) is also used manage the complexity of technologies at each of these locations. These deployments may be shown as generic patterns (system or enterprise level) or as the rough blueprint for implementation.

Figure 8 shows part one of the Technical Location Viewpoint language. Technical Communications between locations is abstract but is defined by Technical WAN Links. Added to the vocabulary are user devices connected to network devices that are part of a Technical Network.

Figure 9 shows part two of the Technical Locations Viewpoint language that adds network security devices along with IP network devices and serial network devices. The relationships between Technical Networks, Technical TelecomNets and IP network and serial network devices illustrates how these devices are used to define these different but interdependent networks. The network security devices are used to provide the required protections for the activities, information and services of Domains, Zones, and User Devices.

Note that the following high-level elements are provided in the viewpoint language as a way for dealing with complexity through the use of hierarchically structured models—these elements define something at a high-level, with the detail provided in lower level detailed models:

- **Technical Location** – the highest level of abstraction for a location definition;
- **Technical WAN Link** – the highest level of abstraction for a telecommunications network link definition;
- **Technical Network** – the highest level of abstraction for an IP (or other) network;
- **Technical Zone** – the highest level of abstraction for a portion of an IP (or other) network;
- **Technical Domain or Zone PEP** – the highest level of abstraction for the boundary protection for a Domain or Zone;
- **Technical TelecomNet** – the highest level of abstraction for a WAN definition;
- **Technical LAN** – the highest level of abstraction for a LAN definition.

The UAM methodology provides guidance and advice on defining this viewpoint.

More information on the Technical Locations Model, its recommended structure and content along with how-to advice:

Guidance > Guidelines > Technical Locations Model

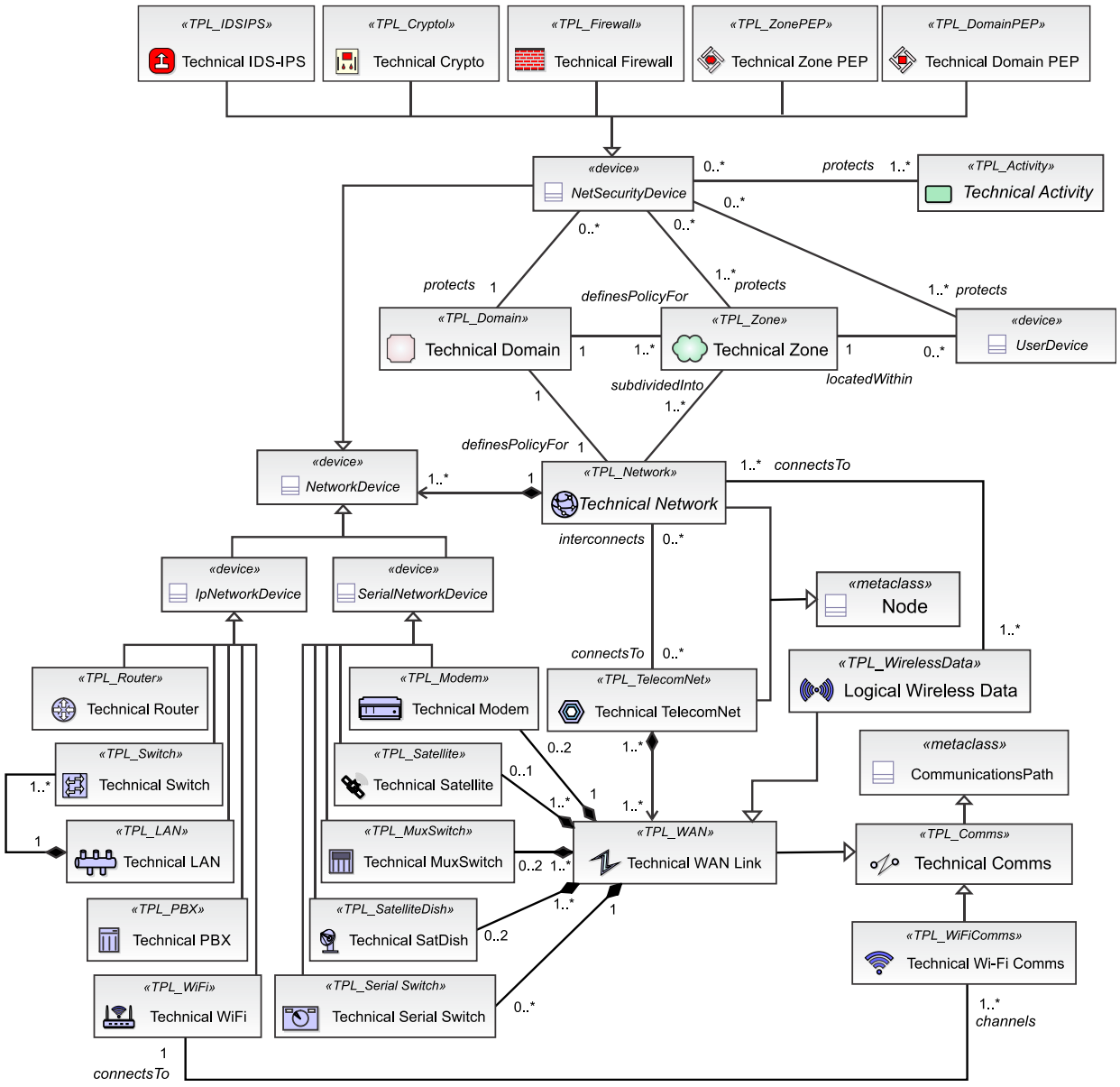


Figure 9 – Technical Locations Viewpoint Language (part 2)

«stereotype» TPL_Comms

Extends

«metaclass» CommunicationPath

«metaclass» Node

«metaclass» Property

Semantics

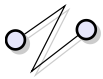
Technical Communications represents the communications, of any type, between Technical Locations. This communications may be the physical transport of goods or telecommunications and network links, all in support of the business and the activities and processes that are found at these locations. This element provides the ability to manage complexity in architectures through the definition of

hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Communications.
communicates: string	
communicatesWith: string	The *desticnation* end of the communications.
nonFunctional: string	Defines things such as business importance, survivability, and availability requirements of the communications path.
protocol: string	The network protocol(s) carried over the communications link.

Notation



Constraints

- Defines communications path relationships between Technical Locations only.

«stereotype» TPL_Crypto

ExtendNode

«metaclass» Device

«metaclass» Node

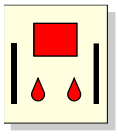
Semantics

A Technical Crypto provides information encryption services on the network. These services include link and end-to-end encryption (i.e. Encryption of an IP stream or a WAN link) or offline encryption (i.e. the encryption of files, hard drives, or emails, etc.).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Comms.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to Technical Telephones, Technical Storage, Technical Servers, Technical Desktops or Technical Routers.

«stereotype» TPL_Desktop

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Desktop defines the user device used to interact with the system. Users as Actors assume Roles when the use Desktops to interact with defined Activities and Processes.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the desktop.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- May be connected to a Technical Network through a Technical Switch;
- Desktops support and are part of the interfaces defined and used in support of Technical Roles;
- Various types of Desktops may be defined that support different defined interfaces or sets of users.

«stereotype» TPL_Domain

Extends

«metaclass» Node

«metaclass» Classifier

«metaclass» Package

Semantics

A Technical Security Domain is an environment or context that is defined by security policies, security models, and security architecture, including a set of resources and set of system entities that are authorized to access the resources. A Technical Domain is managed by a single *authority*, and may contain one or more sub-domains. Different sub-domains are created when security models or policies (and possibly architecture) are significantly different from one domain to the other, or are conflicting. Separate Technical

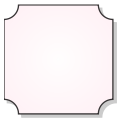
Domains provide clearer separation of concerns and ease policy enforcement and system management. Synonyms: security domain or policy domain.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Domain.
authority: string	The authority for the Domain, normally defined as a specific organizational position within the enterprise or business line (e.g. COO).

Notation

Normally an architecture deals with a single Technical Domain and therefore it may be left off (but documented in the preamble to the architecture description), however if it is required in a viewpoint then a simple rectangular background geometric shape (of an appropriate colour if desired to illustrate the fundamental nature of the Domain) may be used to depict the Security Domain, with the *name* applied to one corner or on the boundary (e.g. “COMPANY-CONFIDENTIAL” or “TOP SECRET”).



Constraints

- Must not contain other domains.

«stereotype» TPL_DomainPEP

Extends

«metaclass» Node

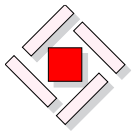
Semantics

A Technical Domain Policy Enforcement Point (PEP) is an aggregation of security services that provide network security services at the perimeter of a network Domain. A number of services are included within a PEP, the specifics of which are hidden at the technical level (if desired) through the use of a Zone PEP in the architecture description. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Domain PEP.
devices: Device[0..*]	The list of device(s) that are used to implement the Policy Enforcement Point.
location: string	The location of the device(s) within the system or enterprise.

Notation



Constraints

- May connect (have associations) to Technical LANs, Technical Switches or Technical Routers.

«stereotype» TPL_Firewall

Extends

«metaclass» Device

«metaclass» Node

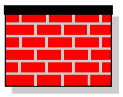
Semantics

A Technical Firewall is used to protect network Domains and Zones and the systems and services contained within them. It does this by controlling the types of access allowed through the firewall and the services accessed. A firewall may be part of the perimeter defences for a Technical Domain (i.e. a Domain PEP) or for a Technical Zone (i.e. a Zone PEP). This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the firewall.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to Technical LANs, Technical Switches or Technical Routers.

«stereotype» TPL_IDSIPS

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Intrusion Detection System / Intrusion Prevention System are IT network security

devices that provide intrusion detection and prevention services for the network.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the IDS or IPS.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- May connect (have associations) to Technical LANs, Technical Switches or Technical Routers.

«stereotype» TPL_LAN

Extends

«metaclass» Node

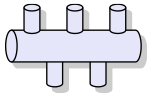
Semantics

A Technical LAN is a higher level of abstraction of a Technical Switch. Depending upon the context and complexity of the architecture being defined, this element may be useful for managing this complexity. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical LAN.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to `UserDevices` and to Technical Routers.

«stereotype» TPL_Laptop

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Laptop defines the user device used to interact with the system. Users as Actors assume Roles when they use Laptops to interact with defined Activities and Processes.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Laptop.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- Is connected to a Technical Network through a Logical Switch;
- Desktops support and are part of the interfaces defined and used in support of Technical Roles;
- Various types of Laptops may be defined that support different defined interfaces.

«stereotype» TPL_Location

Extends

«metaclass» Node

Semantics

A Technical Location defines the locations where the enterprise has a presence that it implements and manages. Technical level activities, devices, and processes are defined that are found at the locations. Locations are a categorization of locations, based upon the technical (business) functions supported, with having processes and services that are under the authority of the enterprise (or system) in question. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the location.
activities: Activities[0..*]	A list of the business activities provides at the location, which relates directly to the Technical Process Model and how the

Name	Description
	business (or system under study) wants to deliver its products and services.
nonFunctional: string	Defines things such as business importance, survivability, and availability requirements of the location.
description: string	A description of the location in terms of building, address or other distinguishing characteristics.

Notation

Constraints

- Shall not have any owned operations;
- Shall not have any owned behaviours;
- May have relationships with only Technical Domains or with Technical Activities (provided at the location);
- May contain Technical Zones;
- May communicate with other Locations via Technical Communications;
- All owned properties shall be public.

«stereotype» TPL_Mainframe

Extends

«metaclass» Device

«metaclass» Node

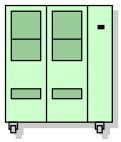
Semantics

A Technical Mainframe is a large usually central compute resource to which Activities and other functions may be deployed. Technical Servers are located within Network Domains and Zones and are deployed to Technical Locations. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the mainframe.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May be connected to a Technical Network through a Technical Switch;
- Wholly located within a Technical Zone.

«stereotype» TPL_Mobile

Extends

«metaclass» Device

«metaclass» Node

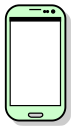
Semantics

A Technical Mobile defines the user device used to interact with the system through Wi-Fi or other network communications. At the technical level this is a mobile phone or similar device. Users as Actors assume Roles when they use a Mobile device to interact with defined Activities and Processes.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the mobile device.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.

Notation



Constraints

- Is connected to a Technical Network through a Technical Wi-Fi Comms or Technical Wireless Data Comms;
- Mobiles support and are part of the interfaces defined and used in support of Technical Roles;
- Various types of Mobiles may be defined that support different defined interfaces.

«stereotype» TPL_Modem

Extends

«metaclass» Device

«metaclass» Node

Semantics

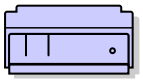
A Technical Modem is used as a component part of a Technical WAN Link. Two identical types

of modems are typically used in a given WAN link.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the modem.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Forms part of a Technical WAN Link.

«stereotype» TPL_MuxSwitch

Extends

«metaclass» Device

«metaclass» Node

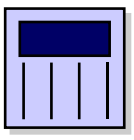
Semantics

A Technical Multiplexor-Switch is part of a Technical WAN Link, allowing several serial links, including IP network links, to be combined into a single larger stream. The same type of multiplexor is used at either end of the WAN link.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the multiplexer.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Forms part of a Technical WAN Link.

«stereotype» TPL_Network

Extends

«metaclass» Node

Semantics

A Technical Network is the complete set of devices and elements that comprise and define a computer network at the technical level. It may be used as a black box definition, with more detail defined in later modelling iterations or as a white box definition, as required by the objectives of the architecture effort. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Network.

Notation

Constraints

- An architecture may or may not define a detailed representation of the network, but if the detail is provided then the detail of the Technical Network will be drilled-down in a hierarchical fashion starting from this element.

«stereotype» TPL_PBX

Extends

«metaclass» Device

«metaclass» Node

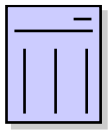
Semantics

A Technical PBX provides telephone network services such as the dialling and switching of voice calls between telephones.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the PBX.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to Technical Telephones, Technical Desktops and to Technical Routers.

«stereotype» TPL_Plotter

Extends

«metaclass» Device

«metaclass» Node

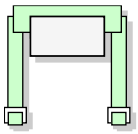
Semantics

A Technical Plotter defines a large format printing device, typically network connected.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the plotter.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May be connected to a Technical Network through a Technical Switch.

«stereotype» TPL_Printer

Extends

«metaclass» Device

«metaclass» Node

Semantics

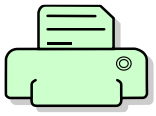
A Technical Printer defines a standard format printing device, typically network connected.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.

Name	Description
name: string	A descriptive name for the printer.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May be connected to a Technical Network through a Technical Switch.

«stereotype» TPL_Router

Extends

«metaclass» Device

«metaclass» Node

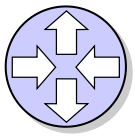
Semantics

A Technical Router manages the routing of network traffic between (TCP/IP) networked devices.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the router.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

May connect (have associations) to Technical Modems, Technical MuxSwitches, or Technical Serial Switches;

May connect (have associations) to `UserDevices`, or to `IpNetworkDevices`.

«stereotype» TPL_Satellite

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Satellite provides communications links between two points. At the technical level it may represent a set of satellite, depending upon the context a level of detail defined in the architecture.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the satellite.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
carrier: string	The carrier managing and providing the satellite communications.

Notation

Constraints

- Forms part of a Technical WAN Link.

«stereotype» TPL_SatelliteDish

Extends

«metaclass» Device

«metaclass» Node

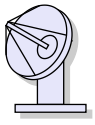
Semantics

A Technical Satellite Dish is part of a Technical WAN Link, providing one or more communications circuits via a satellite to another Satellite Dish located elsewhere within the satellites terrestrial footprint.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the dish.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

➲ Forms part of a Technical WAN Link.

«stereotype» TPL_SerialSwitch

Extends

«metaclass» Device

«metaclass» Node

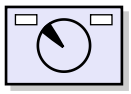
Semantics

A Technical Serial Switch is part of a Technical WAN Link, providing the ability to switch a serial communications link between equipment. At the technical level this may not be necessarily shown, however this level of detail may be required in some architectural contexts.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the switch.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

➲ Forms part of a Technical WAN Link.

«stereotype» TPL_Server

Extends

«metaclass» Device

«metaclass» Node

Semantics

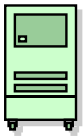
A Technical Server is a compute resource to which Activities and other functions may be deployed. Technical Servers are located within (Network Domains and) Zones and are deployed to Technical Locations. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the server.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.

Name	Description
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May be connected to a Technical Network through a Technical Switch;
- Wholly located within a Technical Zone.

«stereotype» TPL_Storage

Extends

«metaclass» Node

Semantics

Technical Storage provides a mechanism to store, retrieve or update stored information that will persist beyond the scope of an **Activity**. Note that a **Data Store** supports business processes and are a specialization of Technical Storage. Technical Storage is therefore used to support all persistence requirements (i.e. Management processes and systems) with **Data Stores** being specializations that support automated and semi-automated business processes. This element provides the ability to manage complexity in architectures through the definition of hierarchical models (i.e. higher level abstractions of stores).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the element.
capacity: integer [0..1]	Defines the `capacity` of the storage.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- May connect (have associations) to a Technical Network through a Technical Switch;
- May connect (have associations) directly to a Desktop, Server or Mainframe;
- May have relationships with Activity elements (i.e. it may support Activities and Processes at a high-level abstract level but should be defined as Data Store in future iterations);
- Relationships define the technical usage of stores by processes and systems.

«stereotype» TPL_Switch

Extends

«metaclass» Device

«metaclass» Node

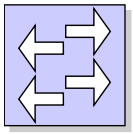
Semantics

A Technical Switch is used to implement a Technical Area Network (LAN) by switching network traffic between the devices connected to the switch.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the network switch.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

➲ May connect (have associations) to `UserDevices` and to Technical Routers.

«stereotype» TPL_Tablet

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Tablet defines a portable user device used to interact with the system through Wi-Fi or other network communications. At the technical level this is a tablet or similar device. Users as Actors assume Roles when they use a tablet device to interact with defined Activities and Processes.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the tablet device.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.

Notation

Constraints

- Is connected to a Technical Network through a Technical Wi-Fi Comms or Technical Wireless Data Comms;
- Tablets support and are part of the interfaces defined and used in support of Technical Roles;
- Various types of Tablets may be defined that support different defined interfaces.

«stereotype» TPL_TelecomNet

Extends

«metaclass» Node

Semantics

A Technical Telecommunications Network is used to define a telecommunications network which is a collection of Wide Area Network (WAN) links. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Telecommunications Network.

Notation

Constraints

- Interconnects Technical Networks or Technical Locations.

«stereotype» TPL_Telephone

Extends

«metaclass» Device

«metaclass» Node

Semantics

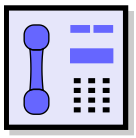
A Technical Telephone supports voice, and possibly video, interactions between users.

Properties

Name	Description
id	This attribute is used to uniquely identify elements.
name	A descriptive name for the telephone.

Name	Description
make	The manufacturer of the device(s).
model	The model of the device or family of devices.
location	The location of the device within the system or enterprise.

Notation



Constraints

- May be connected to a Technical Network through a Technical Switch.

«stereotype» TPL_WAN

Extends

«metaclass» Node

Semantics

A Technical Wide Area Network (WAN) link is composed of a number of components that together provide communications between Technical Networks or between Technical Locations. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical WAN Link.

Notation

Constraints

- Forms part of a Technical Telecommunications Network.

«stereotype» TPL_WiFi

Extends

«metaclass» Device

«metaclass» Node

Semantics

A Technical Wi-Fi IP network (hot spot) provides communications between a Technical Network and devices such as Desktops, Mobile, Printers and other Wi-Fi enabled devices. This element provides the ability to manage complexity in architectures through the definition of hierarchical models (i.e. higher level abstractions of Wi-Fi networks).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Wi-Fi network and device.
make: string	The manufacturer of the device(s).
model: string	The model of the device or family of devices.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- Forms part of a Technical (IP) Network.

«stereotype» TPL_WiFiComms

Extends

«*metaclass*» communicationsPath

«*metaclass*» Node

«*metaclass*» Property

Semantics

Technical Wi-Fi Communications provides IP (network) communications between a Wi-Fi enabled device and a Technical Network via a Technical Wi-Fi network device.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Wi-Fi network that this communications link connects to.

Notation

Constraints

- Forms part of a Technical (IP) Network.

«stereotype» LPL_WirelessData

Extends

«*metaclass*» communicationsPath

Semantics

A Technical Wireless Data communications provides data communications between a mobile

device with data capability and a Technical Network via a telecommunication network.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical wireless data link.

Notation

Constraints

- Forms part of a Technical Telecommunications Network.

«stereotype» TPL_Zone

Extends

«metaclass» Node

«metaclass» Classifier

Semantics

A Technical Security Zone is an environment that is defined by security policies, security models, and security architecture, including a set of resources and set of system entities that are authorized to access the resources. A Technical Zone may contain one or more sub-zones. Different sub-zones are created when security models or policies (and possibly architecture) are significantly different from one zone to the other, or are conflicting. Separate Technical Zones provide clearer separation of concerns and ease policy enforcement and system management. Synonyms: security zone or policy zone.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Zone.
owner: string	The owner of the Zone normally defined as a specific organizational position within the enterprise or business line (e.g. HR) which owns the information within the Zone. Sub-zones inherit ownership from the parent Zone. The (parent Domain) **Authority** may delegate responsibilities to Zone owners.
location: string	The location of the zone within the system or enterprise

Notation

A Technical Security Zone is represented as a simple cloud shaped background geometric shape (of an appropriate colour if desired to illustrate the fundamental nature of the Zone) with the *name* applied to one corner or outside the perimeter (e.g. “HR Services”).

Constraints

- Zones must be wholly contained within other Zones;
- Zones must be wholly contained within Technical Domains;
- Zones are defined only when needed.

«stereotype» TPL_ZonePEP

Extends

«metaclass» Node

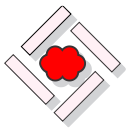
Semantics

A Technical Zone Policy Enforcement Point (PEP) is an aggregation of security services that provide network security services at the perimeter of a network Zone. A number of services are included within a PEP, the specifics of which are hidden at the technical level (if desired) through the use of a Zone PEP in the architecture description. This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Technical Zone PEP.
devices: Device[0..*]	The list of device(s) that are used to implement the Policy Enforcement Point.
location: string	The location of the Zone PEP within the system or enterprise

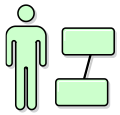
Notation



Constraints

- May connect (have associations) to Technical LANs, Technical Switches or Technical Routers.

Technical Roles Viewpoint



In UAM the primary purpose of modelling Roles and Actors is to describe how the activities are interacted with by various internal and external users, and most importantly by its customers and partners. All interactions of Roles and Actors (both internal and external) with activities are defined; therefore employees that interact with activities and process are also modelled.

The focus of the roles model is on all interactions with the business processes and activities, with the Technical Roles representing the “interface protocol” (in the case of system interactions: Service Tasks) or “graphical user interface” (GUI, in the case of human interactions: User Tasks) to the Technical Activities and Processes. At the Technical level these definitions are not implementable but will clearly specify the required structure and content for these interfaces.

In the context of IT architecture and the Technical Roles Viewpoint, the Activities within this model do not represent the complete business Activity, just the presentation layer or interface. These

Technical Activities therefore represent the boundary portion of the Technical Process Model—the Technical Process Model captures all Processes, Activities and Tasks, but with the interface/GUI portion also being represented in the Technical Roles Model through a User Task (or Service Task for system interactions). The Technical Roles Model is incomplete from a process perspective; therefore the Technical Process Model defines the complete business solution for Processes and Activities.

This simple metamodel shown in Figure 10 is described as follows:

- A Technical Actor interacts with the system through one or more Technical Roles,
- A Technical Role is a type of Performer from BPMN, and is modelled as an Interface in UAM,
- Technical Actors may be structured into generalizations or specializations,
- A Technical Role interacts with one or more Tasks in the system (either Service Tasks or User Tasks),
- The Technical Tasks that the Role interacts with define the Role,
- Technical Roles may be structured through aggregation relationships (to simplify the model),
- A Technical Role is the Interface to the Technical Task.

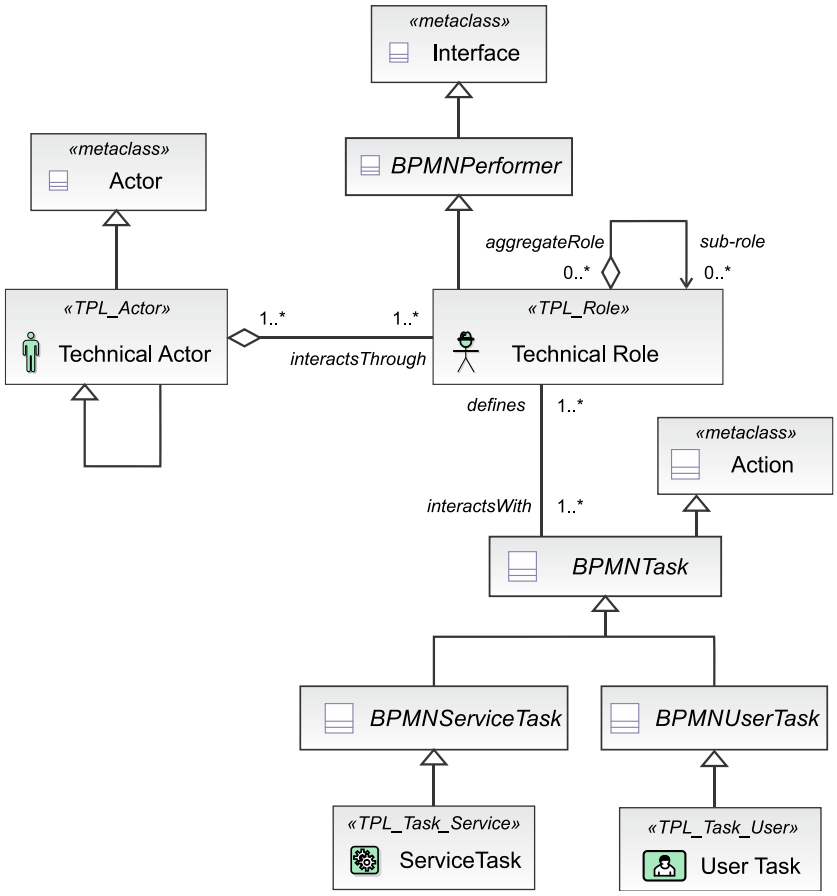


Figure 10 – Technical Roles Viewpoint Language

The UAM methodology provides guidance and advice on defining this viewpoint.

More information on the Technical Roles Model, its recommended structure and content along with how-to advice:

Guidance > Guidelines > Technical Roles Model

Note that the "User Task" and "Service Task" are defined within the Technical Process Model and are reused in this viewpoint. It represents the user interface (UI) portion of the User Task and may have further detail provided within the Technical Roles Model.

Definitions for all of these Technical Roles Viewpoint Language elements follow.

«stereotype» TPL_Actor

Extends

«metaclass» Actor

Semantics

A Technical Actor is equated to the external people or systems that interact with the system under study. Actors are not necessarily external to the business; they may also represent internal "user" interactions with the system. Technical Actors are derived from Technical Actors through further analysis in order to fully understand, structure, and define the required actors. Note that actors are ***not*** individuals nor are they necessarily equivalent to job titles**;** instead, they describe the behaviour in the enterprise and the responsibilities of the associated "user". At this technical level this final analysis may be done so that Actors (or collections of Actors) may be associated with organizational positions and users.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the role.
characteristics: string	A technical level description of the actor and what it represents. Specifics in terms of people, jobs or systems should be avoided at this point—keep it technical

Notation

Constraints

- Must be named;
- Can only have associations to Technical Roles, furthermore these associations must be binary;
- Shall aggregate at least one associated Technical Role.

«stereotype» TPL_Role

Extends

«metaclass» Interface

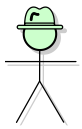
Semantics

Defines a Technical Role which is equated to or defined by the behaviour (and by extension its responsibilities) that it interacts with in the system—the activities that it interacts with. A role is in essence the "interface" to the activities with which it interacts. Technical Roles represent the "interface protocol" (in the case of system interactions) or "graphical user interface" (GUI, in the case of human interactions) to the Technical Activities and Processes. A Role can be assumed (aggregated) by any kind of actor, either human or a system.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the role.
description: string	Description of the Role, its interactions and other relevant information such as security aspects.

Notation



Constraints

- Shall not have any owned operations;
- Shall not have any owned behaviours;
- May have relationships only with other Technical Activities, Roles or with Technical Entities;
- A derived relationship with Technical Entities is used to identify what Roles (Actors) interact with these Entities;
- All owned properties shall be public.

«stereotype» TPL_ActorGeneralization

Extends

«metaclass» Association

Semantics

This is used to define generalization associations between two Actors in the model that permit better understanding of the Actors, their relationships and capabilities. Analysis may also result in the simplification of the model.

Properties

Name	Description
id: string	This attribute is used to uniquely identify model elements.
name: string	A descriptive name for the association.
sourceRef: entity	The `Entity` that the Association is connecting from
targetRef: entity	The `Entity` that the Association is connecting to

Name	Description
type: AssociationType = Simple {Simple | Aggregation | Generalization}	**Simple:** a simple association (with or without Roles and Multiplicities) with perhaps a defined meaning; **Aggregation**: represents a part-whole or part-of relationship; **Generalization**: a specialized form of the other (the *super type*) and super-class is considered as '***Generalization'*** of subclass (without Roles and without Multiplicities).
associationDirection: AssociationDirection = None {None | One | Both}	Defines whether or not the Association shows any directionality with an arrowhead. The default is `None` (no arrowhead). A value of `One` means that the arrowhead shall be at the Target Entity. A value of `Both` means that there shall be an arrowhead at both ends of the association line.

Notation

Generalization association: Specialized Actor  Generalized Actor

Constraints

- May define relationships between any two `TPL_Actor` elements;

«stereotype» TPL_RoleAggregation

Extends

«metaclass» Association

Semantics

This is used to define aggregation associations between a Role and an Actor in the model that permit better understanding of the Actors and Roles, their relationships and capabilities. Analysis may also result in the simplification of the model.

Properties

Name	Description
id: string	This attribute is used to uniquely identify model elements.
name: string	A descriptive name for the association.
sourceRef: entity	The `Entity` that the Association is connecting from
targetRef: entity	The `Entity` that the Association is connecting to
sourceRole: string	The role that the source entity has with the target entity for simple and aggregation/composition associations.
targetRole: string	The role that the target entity has with the source entity for simple and aggregation/composition associations.
type: AssociationType = Simple {Simple | Aggregation | Generalization}	**Simple:** a simple association (with or without Roles and Multiplicities) with perhaps a defined meaning; **Aggregation**: represents a part-whole or part-of relationship; **Generalization**: a specialized form of the other (the *super type*) and super-class is considered as '***Generalization'*** of subclass (without Roles and without Multiplicities).

Name	Description
associationDirection: AssociationDirection = None {None | One | Both}	Defines whether or not the Association shows any directionality with an arrowhead. The default is `None` (no arrowhead). A value of `One` means that the arrowhead shall be at the Target Entity. A value of `Both` means that there shall be an arrowhead at both ends of the association line.

Notation

Aggregation association: Actor  Role

Constraints

- May define relationships between a `TPL_Role` element and a `TPL_Actor`;

«stereotype» TPL_Task_User TPL_Task_Service (from TP Viewpoint)

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

«metaclass» OpaqueAction

Semantics

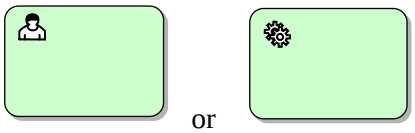
A (Technical) User Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process"(OMG 2013). A User Task "is a typical 'workflow' Task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort"(OMG 2013). These Tasks in the Technical Roles Model represent the support for the interactions with the system by Actors through Roles, either people through User Tasks or automated systems through Service Tasks.

Properties

Name	**Description** (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, this **Activity** is only activated when a **Compensation Event** is detected and initiated under **Compensation Event** visibility scope
loopCharacteristics: LoopCharacteristics [0..1]	An **Activity** MAY be performed once or MAY be repeated. If repeated, the **Activity** MUST have `loopCharacteristics` that define the repetition criteria (if the `isExecutable` attribute of the **Process** is set to *true*).
resources: ResourceRole [0..*]	Defines the resource that will perform or will be responsible for the **Activity**. The resource, e.g., a performer, can be specified in the form of

Name	Description (OMG 2013)
	a specific individual, a group, an organization role or position, or an organization.
default: SequenceFlow [0..1]	The **Sequence Flow** that will receive a *token* when none of the `conditionExpressions` on other *outgoing* **Sequence Flows** evaluate to *true*. The *default* **Sequence Flow** should not have a `conditionExpression`. Any such Expression SHALL be ignored.
ioSpecification: Input OutputSpecification [0..1]	The InputOutputSpecification defines the *inputs* and *outputs* and the `InputSets` and `OutputSets` for the **Activity**.
properties: Property [0..*]	Modeler-defined `properties` MAY be added to an **Activity**. These `properties` are contained within the **Activity.**
boundaryEventRefs: BoundaryEvent [0..*]	This references the **Intermediate Events** that are attached to the boundary of the **Activity**.
dataInputAssociations: DataInputAssociation [0..*]	An optional reference to the `DataInputAssociations`. A `DataInputAssociation` defines how the `DataInput` of the **Activity's** `InputOutputSpecification` will be populated.
dataOutputAssociations: DataOutputAssociation [0..*]	An optional reference to the `DataOutputAssociations`.
startQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST arrive before the **Activity** can begin. Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
completionQuantity: integer = 1	The default value is 1. The value MUST NOT be less than 1. This attribute defines the number of *tokens* that MUST be generated from the **Activity**. This number of tokens will be sent done any *outgoing* **Sequence Flow** (assuming any **Sequence Flow** conditions are satisfied). Note that any value for the attribute that is greater than 1 is an advanced type of modeling and should be used with caution.
state: string = None	The lifecycle of an **Activity** is described; see (OMG 2013).
implementation: string = ##unspecified	This attribute specifies the technology that will be used to implement the **User Task**. Valid values are "##unspecified" for leaving the implementation technology open, "##WebService" for the Web service technology or a URI identifying any other technology or coordination protocol. The default technology for this task is unspecified.
renderings: Rendering [0..*]	This attributes acts as a hook which allows **BPMN** adopters to specify task rendering attributes by using the **BPMN** Extension mechanism.

Notation



Constraints

- May have relationships (sequence flows) with other flow elements as defined in **Table 2**;
- See BPMN v2.0 (OMG 2013).

Technical Perspective Language Summary

The table lists the modelling elements for the technical level viewpoints. It is based upon the BPMN Analytic Conformance Sub-Class, with a number of elements added.

	Aspect			
Perspective	**Data**	**Activity**	**Location**	**People**
▪ Technical	**Technical Entity Model** ➢ Technical Entity ➢ Entity Associations ▪ Simple ▪ Aggregation ▪ Generalization / Specialization	**Technical Process Model** ➢ Activity Call ➢ Activity Transaction ➢ Association ➢ Data Collection ➢ Data Input ➢ Data Object ➢ Data Output ➢ Data Store ➢ DataAssociation ➢ DataStoreRef ➢ Event Cancel Int Int ➢ Event Cancel End ➢ Event Comp Start Int ➢ Event Comp Int Int ➢ Event Comp Throw ➢ Event Comp End ➢ Event Cond Catch ➢ Event Cond Int Int ➢ Event Cond Int Nonint ➢ Event Cond Start ➢ Event End ➢ Event Error End ➢ Event Error Int ➢ Event Error Start ➢ Event Escal Int Int ➢ Event Escal Int Nonint ➢ Event Escalation End ➢ Event Escalation Throw ➢ Event Link Catch ➢ Event Link Throw ➢ Event Message End ➢ Event Message Start ➢ Event Msg Catch ➢ Event Msg Int Int ➢ Event Msg Int Nonint ➢ Event Msg Start Nonint ➢ Event Msg Throw ➢ Event Signal Catch ➢ Event Signal End ➢ Event Signal Int Int ➢ Event Signal Int Nonnt ➢ Event Signal Start ➢ Event Signal Throw ➢ Event Start ➢ Event Terminate ➢ Event Throw ➢ Event Timer Catch ➢ Event Timer Int Int ➢ Event Timer Int Nonnt ➢ Event Timer Start ➢ Event Timer Strt Nonint ➢ Gateway Event-based ➢ Gateway Event Excl ➢ Gateway Event Parallel ➢ Gateway Exclusive	**Technical Locations Model** ➢ Technical Comms ➢ Technical Crypto ➢ Technical Storage ➢ Technical Desktop ➢ Technical Domain ➢ Technical Domain PEP ➢ Technical Firewall ➢ Technical IDS-IPS ➢ Technical LAN ➢ Technical Laptop ➢ Technical Location ➢ Technical Mainframe ➢ Technical Mobile ➢ Technical Modem ➢ Technical MuxSwitch ➢ Technical Network ➢ Technical PBX ➢ Technical Plotter ➢ Technical Printer ➢ Technical Router ➢ Technical Satellite ➢ Technical Satellite Dish ➢ Technical SerialSwitch ➢ Technical Server ➢ Technical Switch ➢ Technical Tablet ➢ Technical TelecomNet ➢ Technical Telephone ➢ Technical WAN Link ➢ Technical WiFi ➢ Technical WiFi Comm ➢ Technical Wireless Data ➢ Technical Zone ➢ Technical Zone PEP	**Technical Roles Model** ➢ Technical Actor ➢ Technical Role ➢ Service / User Task (from TPV) ➢ Relationships: ▪ Actor Generalization ▪ Role Aggregation

	Aspect			
Perspective	**Data**	**Activity**	**Location**	**People**
		➢ Gateway Inclusive ➢ Gateway Parallel ➢ Group ➢ Lane / LaneSet ➢ Message Flow ➢ Message Receive ➢ Message Send ➢ Pool ➢ Rule ➢ Seq Flow Conditional ➢ Seq Flow Default ➢ Sequence Flow ➢ Sub-process ▪ Compensation ▪ Loop ▪ Parallel ▪ Sequential ➢ Sub-process Ad-hoc ▪ Compensation ▪ Loop ▪ Parallel ▪ Sequential ➢ Task ▪ Compensation ▪ Loop ▪ Parallel ▪ Sequential ➢ Task Manual ➢ Task Receive ➢ Task Rules ➢ Task Script ➢ Task Send ➢ Task Service ➢ Task User ➢ Text Annotation		

Bibliography

OMG. "Business Process Model and Notation (BPMN)." Version 2.0.1 (September 2013): 532.