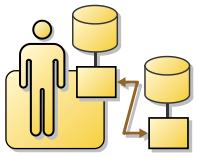
Unified Architecture Method

Logical Perspective Language UML Profile

Contents

Ottawa, Ontario, Canada

Introduction



The Logical Perspective defines a logical level view of the system. The various viewpoints defined show how actors interact with processes at various locations within the business, and the things they handle and use, the logical entities. The viewpoints also show how these different aspects relate to one another, both statically and dynamically, to produce the desired business results. This perspective places emphasis on the structure of entities, roles, the business locations and processes, and their active responsibilities and interactions. The models show all collaborations (relationships) needed to support business activities, and all classes (the static view) from which objects (operational instances) will be instantiated to obtain the desired business results for the various business roles. Computational aspects, such as *storage* and *server* and *database*, are added to this logical level model.

The Logical Perspective describes process structures and interactions including prescribing design choices for logical roles and logical actors in terms of bindings (to human workers or automated systems). There is still some flexibility in terms of the human interaction; that is, the logical actors and roles defined and their interactions with the system are likely executed by a set of (logical) users. However, this is more precisely specified at the next level in the Technical Perspective. The logical structure and deployment (i.e., Location Model) of the system is an important deliverable within the set of defined viewpoints.

The Logical Perspective models are used by stakeholders and system and process analysts to understand how the systems currently work (if in an *as-is* form), and to analyze the effect of changes to the business (if in a *to-be* form). The IT architect is responsible for the structure and integrity of the model, particularly as pertains to system components, while system analysts and other IT experts may assist detailing elements within the models. The Logical Perspective models are used by IT architects as the starting point to derive the Technical Perspective models. See the UAM Methodology for a complete description of the participants and activities involved in creating and using UAM architectures.

Various models are used in documenting the system at the Logical level; namely, the Logical Entity Model, the Logical Process Model, the Logical Locations Model and the Logical Roles Model, as shown in Figure 1. Several diagrams and models are used to document each of these aspects of the Logical Perspective. The viewpoints within Logical Perspective are:

- **Logical Entity Model** – a viewpoint that defines the entities involved in and important to the logical processes. The entities implemented by the architecture, including relationships between them required to support the processes, are incorporated into the model.
- **Logical Activity Model** – a viewpoint that defines the sequence of activities of the business, at the logical level, recognizing the use of generic technologies and addressing automation decisions. Activities, in the UAM context, are used to partition and understand the structure of a business, from a process perspective, resulting in manageable and functionally cohesive chunks that produce a tangible result. In addition to *events*, the model may also include relationships between *activities* and *entities*, and process elements and *rules*.
- **Logical Locations Model** – a viewpoint that defines the logical level business locations involved in the enterprise or system, as defined by the scope of the modelling effort. The notion of location is used to identify and describe in detail the classes of location at which the enterprise has a presence. Computational aspects are added, such as *server*, *storage*, and *network*.
- **Logical Roles Model** – a viewpoint that defines the roles and actors, along with their structure, involved in and important to the business. The purpose of modelling roles and actors is to describe how various actors, and most importantly customers and partners (external actors), interacted with activities. All interactions of *actors* (both internal and external) with *activities* and through *roles* are defined.

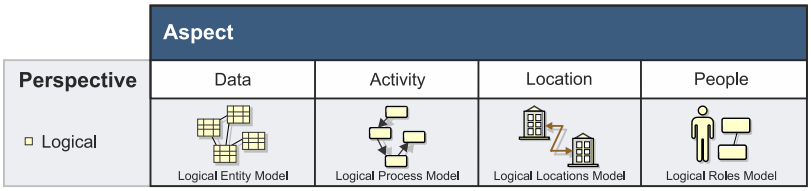
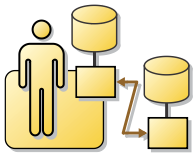


Figure 1 – Logical Perspective Viewpoints

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. 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.

Logical Perspective



The Logical Perspective Language is based upon the **BPMN Descriptive Conformance** sub-class elements and attributes as defined in Table 1, with the addition of un-typed Throwing Event, Link Events (Catch and Throw), Message (Send and Receive) and Send and Receive Task elements. This conformance profile is used for logical-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.

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

Element	Attributes
participant (pool)	id, name, processRef
laneSet	id, lane with name, childLaneSet, flowElementRef
sequenceFlow (unconditional)	id, name, sourceRef, targetRef
messageFlow	id, name, sourceRef, targetRef
exclusiveGateway	id, name
parallelGateway	id, name
task (None)	id, name
userTask	id, name
serviceTask	id, name
subProcess (expanded)	id, name, flowElement
subProcess (collapsed)	id, name, flowElement
CallActivity	id, name, calledElement
DataObject	id, name
TextAnnotation	id, text
association/dataAssociation[a]	id, name, sourceRef, targetRef, associationDirection[b]
dataStoreReference	id, name, dataStoreRef
startEvent (None)	id, name
endEvent (None)	id, name
messageStartEvent	id, name, messageEventDefinition
messageEndEvent	id, name, messageEventDefinition
timerStartEvent	id, name, timerEventDefinition
terminateEndEvent	id, name, terminateEventDefinition
documentation[c]	text
Group	id, categoryRef

a. **Data Association** is ABSTRACT: **Data Input Association** and **Data Output Association** will appear in the XML serialization. These both have REQUIRED attributes [`sourceRef` and `targetRef`] which refer to `itemAwareElements`. To be consistent with the metamodel, this will require the following additional elements: `ioSpecification`, `inputSet`, `outputSet`, **Data Input, Data Output**. When a **BPMN** editor draws a **Data Association** to an **Activity** or **Event** it should generate this supporting invisible substructure. Otherwise, the metamodel would have to be changed to make

`sourceRef` and `targetRef` optional or allow reference to non-`itemAwareElements`, e.g., **Activity** and **Event**.

b. `associationDirection` not specified for **Data Association**

c. `Documentation` is not a visible element. It is an attribute of most elements.

BPMN forms the heart of the Logical (and Technical) Perspective, with additional elements added in support of UAM essentially through the BPMN extension mechanisms to define the complete Logical 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 Logical 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 **Descriptive Conformance** profile for BPMN V2.0, UAM has added the following BPMN elements to the Logical Process Viewpoint language:

Activity – Ad Hoc Subprocess, Sequential, Parallel and Loop activities;
Conditional Events – Start, Catching, Intermediate Interrupting, & Intermediate Non-Interrupting;
Message Event – Message Catching;
Event Gateways – Event-Based, Event-Exclusive, and Event-Parallel gateways;
Link Events – Catching and Throwing;
Messages – Send and Receive;
Sequence Flow – conditional and default flows;
Signal Events – Start, Intermediate Interrupting, Intermediate Non-Interrupting, Throw and End;
Timer Event – Timer Catching;
Tasks – Send, Receive, Manual, Business Rule and Script tasks;
Untyped Event – Throwing.

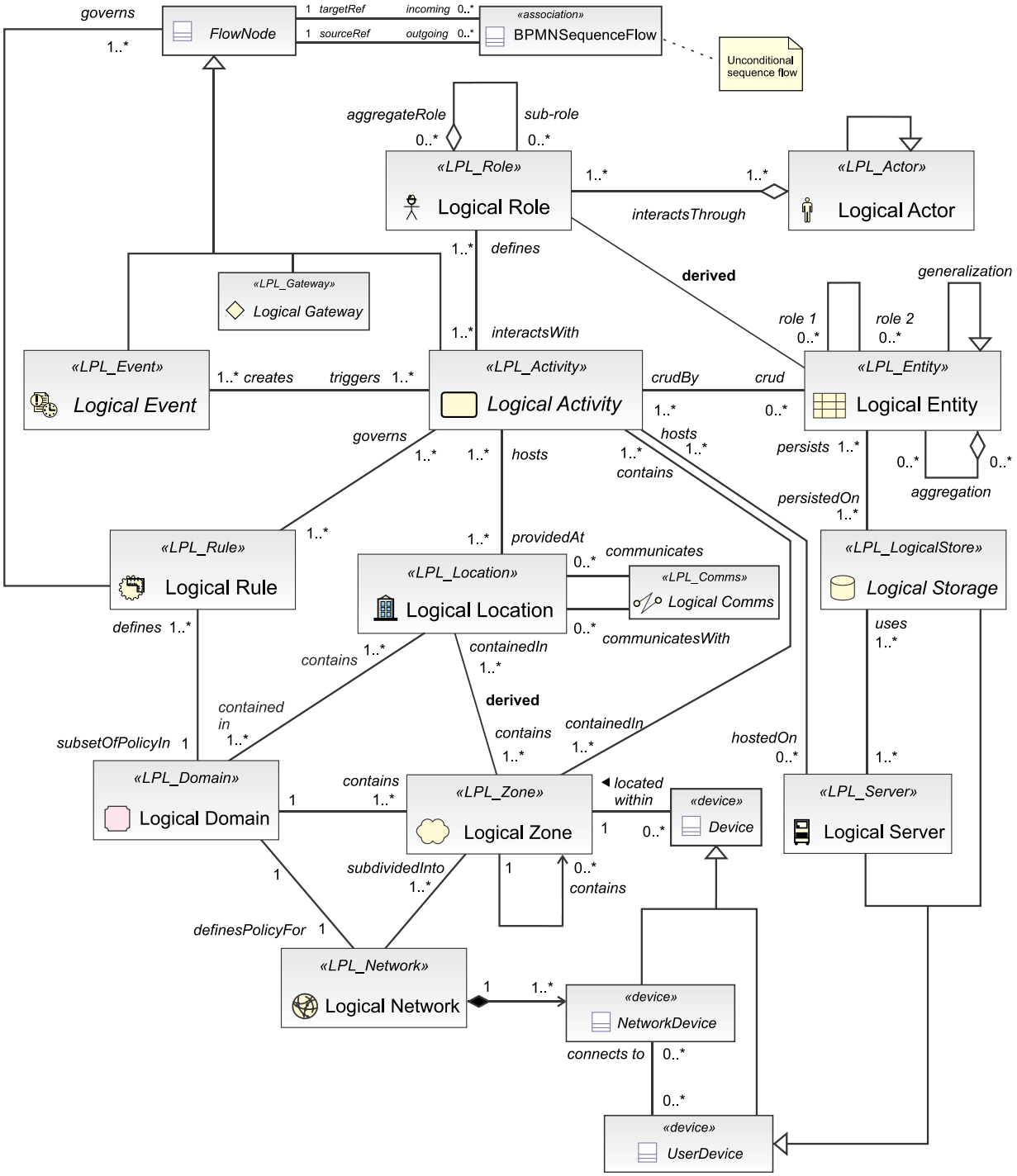


Figure 2 – Logical Perspective Language (metamodel) Overview in UML

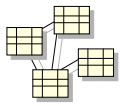
The metamodel in Figure 2 defines a high level overview of the Logical Perspective language. It is very much simplified in order to portray the main concepts and elements of the Logical Perspective. The metamodel in words says:

- A Logical Actor aggregates (has) one or more associated Logical Roles;
- A Logical Actor interacts with the system through one or more Logical Roles;
- A Logical Actor may be generalized (or specialized);

- A Logical Role interacts with one or more Logical Activities;
- A Logical Activity defines one or more Logical Roles;
- A Logical Role may be aggregated into another role;
- A Logical Event triggers one or more Logical Activities;
- A Logical Activity may create zero or more Logical Events;
- A Logical Activity may contain a (hierarchical) structure of (more detailed) activities (i.e. it may be a sub-process);
- Logical Activities may create, read, update, and delete (CRUD) Logical Entities;
- Logical Entities may have (named) associations between them (in support of the business) with roles defining the relationship;
- Logical Entities have an (important and useful) derived relationship with Logical Roles (and therefore also Actors);
- Logical Entities may have aggregation or generalization relationships between them;
- Logical Entities are persisted on Logical Storage, which is a user device;
- Logical Servers use Logical Storage;
- Logical Servers host Logical Activities;
- Both Logical Storage and Logical Servers (and other devices, see below) are located within a Logical Zone.
- Logical Entities have derived associations to Logical Roles (and in turn to Actors and specific Users) through Logical Activities;
- Logical Activities are provided at one or more Logical Locations;
- Logical Locales communicate with each other through Logical Comms (which may be anything from physical transport of goods to electronic communications);
- Logical Locations contain one or more Logical Zones;
- A Logical Zone may contain other Logical Zones;
- A Logical Zone is contain in one Logical Domain;
- Logical Rules govern one or more Logical Activities;
- Logical Rules govern one or more Logical Gateways;
- A Logical Domain defines: the policy and rules used within Logical Rules (and the Gateways or Activities and Processes they relate to), the policy that governs Logical Locales, the policy that governs Logical Networks and Logical Zones contained within these locales;
- A Logical Domain contains one or more Logical Zones;
- Logical Zones may be contained within other Logical Zones;
- A Logical Network is subdivided into Logical Zones, which are contained within Logical Locations;
- Logical Rules have derived associations to Logical Events through a Logical Activity.

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

Logical Entity Viewpoint



The Logical Entity Model defines a logical data model for the enterprise or system, as defined by the scope of the modelling effort. The objective is to clearly define the entities involved, and the relationships between them needed to support the defined logical processes. The definition must be valid for the defined scope, and must eliminate any ambiguity regarding what they represent.

Relationships between entities are derived directly from analysis of the business, its activities and processes. Two main aspects are important, the relationships supporting the business processes and functions, and the relationships supporting the monitoring and management of the business. Note that generalizations, aggregations and other types of analysis are added at this logical level. All attributes of the entities should be defined along with candidate keys and identities for each if desired.

The logical actors and logical roles that work with these entities are also identified if desired. The structure of defined actors and roles, along with their relationships to business entities, is used to identify the access controls aspect of IT security required. This part of the model is derived from and uses elements from other views, such as the Logical Process Model and the Logical Roles Model, which illustrates how intertwined and dependent the viewpoints are within UAM, or any architecture for that matter. See the Logical Process Model and Logical Roles Model definitions for more detail regarding these derived relationships to Logical Roles.

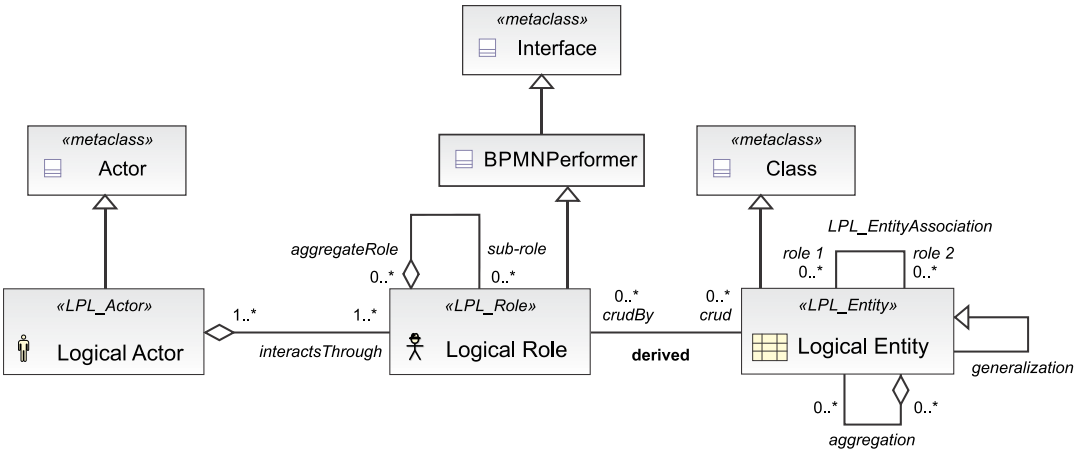


Figure 3 – Logical Entity Viewpoint Metamodel

The elements defined for the Logical Entity Viewpoint Language are:

«stereotype» LPL_Entity

Extends

«metaclass» Class

Semantics

A Logical Entity is a representation of the information and data used within the system being modelled. At the logical level a very complete view of the data entities are defined, including their attributes and relationships. Logical Entities are used to define a logical data model in support of the processes and activities of the system.

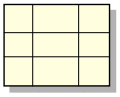
Properties

The properties of Logical Entities, including attributes and other characteristics, are defined as required by the context and subject that they represent. In the Logical Perspective logical entities are used

to develop a complete logical data model for the architectural context and supporting the system under analysis. Simple associations (directed or not) are supported between entities, however more complex relationships are also possible: *aggregation* and *generalization* (specialization).

Name	Description
id: string	This attribute is used to uniquely identify model elements.
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 Logical Roles or with other Logical Entities (all relationships types are allowed, such as aggregations, generalizations, specializations, etc.);
- Candidate keys may be identified in support of structured persistent storage.

«stereotype» LPL_EntityAssociation

Extends

«metaclass» Association

Semantics

Used to define Logical 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.

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

Simple associations:  or or

Aggregation association: 

Composition association: 

Generalization association: 

Constraints

➲ May define relationships between any two `LPL_Entity` elements.

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

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

Guidance > Guidelines > Logical Entity Model

Logical Process Viewpoint

The Logical 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 at this logical level may be quite extensive or be very broad and conceptual.

In Figure 4, Figure 5 and Figure 6 the elements of the Logical Process Viewpoint language are defined along with some elements from BPMN V2.0 to provide context. It should be noted that eventually this metamodel will conform to the BPMN UML Profile as sanctioned by OMG. The elements (stereotypes) forming the Logical Process Viewpoint Language are defined in detail, including any constraints.

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

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

Guidance > Guidelines > Logical Process Model

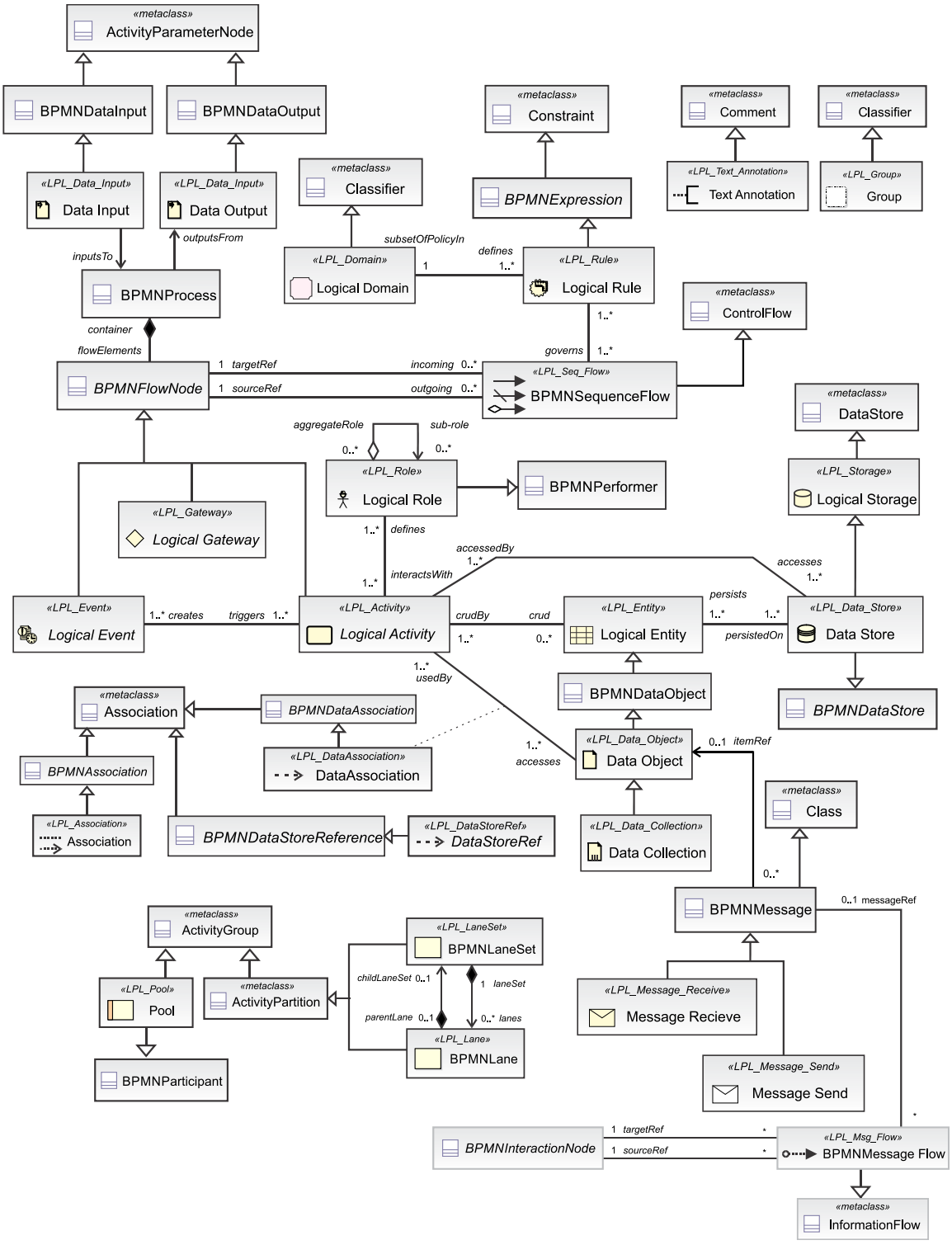


Figure 4 – Logical Process Viewpoint Metamodel (part 1)[1]

[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.

In BPMN there are five basic categories of elements defined that are used to specify business processes. These categories, also adopted by UAM for defining business 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 business 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 Business 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. Default and Conditional flows may be defined;
2. **Message Flows** – used to define interactions between Processes and Activities that cross Pool and Lane boundaries;
3. **Associations** – used to associate information, Text Annotations, and other Artefacts with BPMN elements;
4. **Data Associations** – used to associate Data (Objects, Inputs, Outputs, and Stores) with BPMN elements;

Note that a Data Store Reference, as shown in Figure 4, is in effect a Data Store, or a representation of the Data Store, that can be reused in multiple processes to refer to the same Data Store.

Business Processes are grouped and organized using two mechanisms through the notion of swim lanes, 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: Logical Rule and Logical 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;

Logical Rule – the business rules associated with an Event, Activity or Gateway;

Logical Domain – defines the complete set of policies and rules that apply to the system under study (and often to a wider corporate scope).

From Figure 4 it can be seen that *flow nodes*, namely Events, Gateways or Activities, are linked together through *Sequence Flow* associations to form business processes. The *Logical Domain* defines all policies and rules that control this workflow along with other required policies and rules for the system under study (or the Security Domain of which the system is part), with *Logical Rules* being associated with specific *flow node* elements. *Logical Roles* interact with *Logical Activities* to execute portions of business processes, which may result in the creation, reading, update or deletion of *Logical Entities*. A *Data Object* is a type of *Logical Entity* that is accessed or received by a *Logical Activity*. *Logical Entities* may be persisted on *Data Stores*, which are *Logical Stores* that support business processes and access to *Data Objects*.

The complete set of *Logical Gateways* and *Logical Activities* for the Logical Process Viewpoint Language is defined in Figure 5. Note that *Logical Gateway* and *Logical 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" gateways and activities (tasks) are defined that allow for the creation of high-level models if desired.

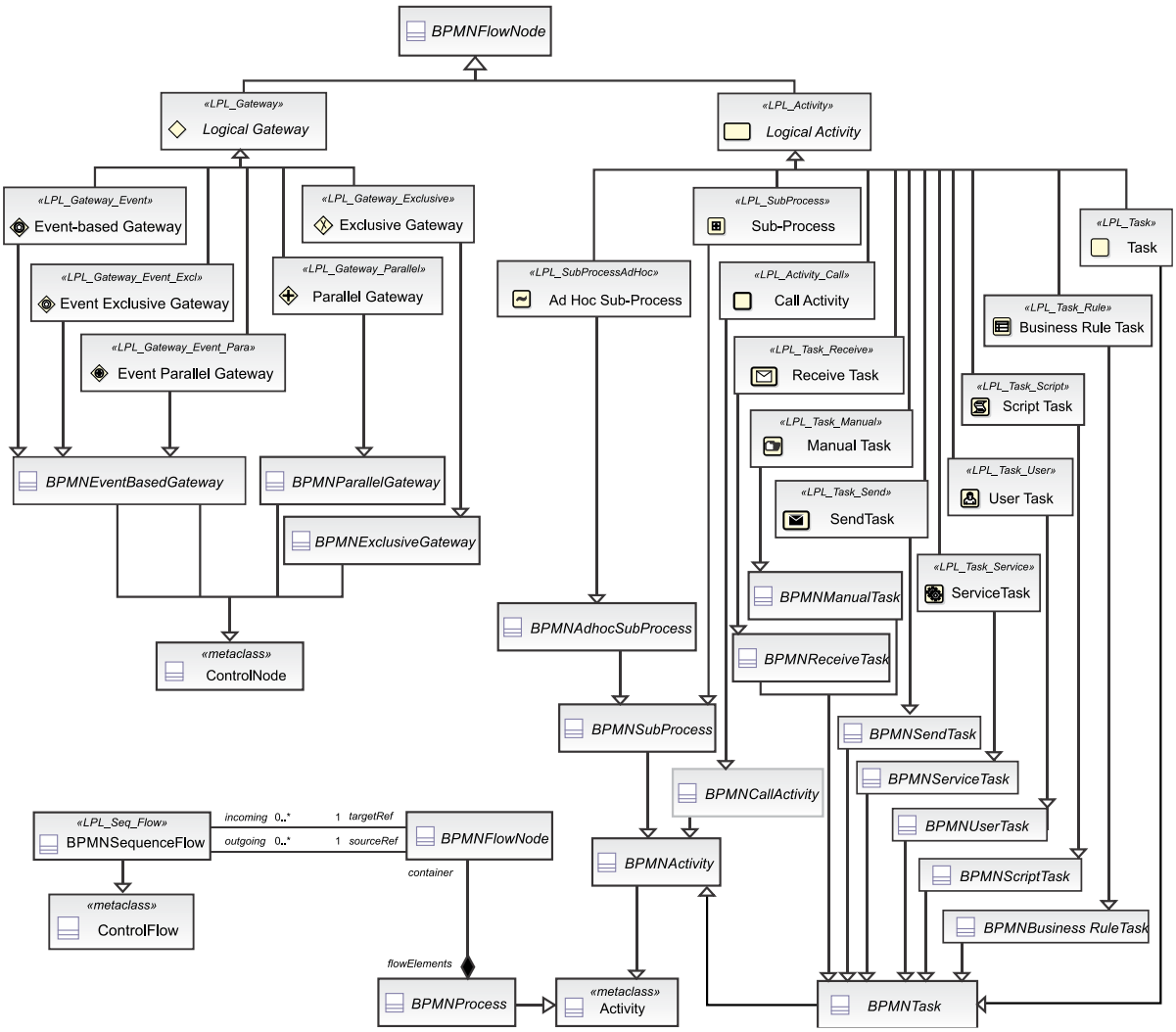


Figure 5 – Logical Process Viewpoint Metamodel (part 2)

A *Process* in BPMN is defined as a container for *Flow Nodes* such as *Sequence Flows*, *Events*, *Gateways*, and *Activities*. There is more to the definition of *Process*, but for UAM purposes this is sufficient. Thus, *Pools* (Participant) contain *LaneSets* and *Lanes* which may contain *Flow Nodes*. A *Pool* defines a Process and contains all the elements of the process. *Sequence Flow* cannot cross *Pool* boundaries; therefore the concept of a *Message* is defined, which allows communications between *Processes* (Pools) via *Send* and *Receive Messages* (and corresponding *Send* and *Receive Tasks*, see Figure 5). Given the *Logical Domain* a set of *Logical Rules* (and policies) are defined that govern *Processes*, *Activities*, *Gateways* and other aspects of the architecture as shown in Figure 4, Figure 5, and Figure 6.

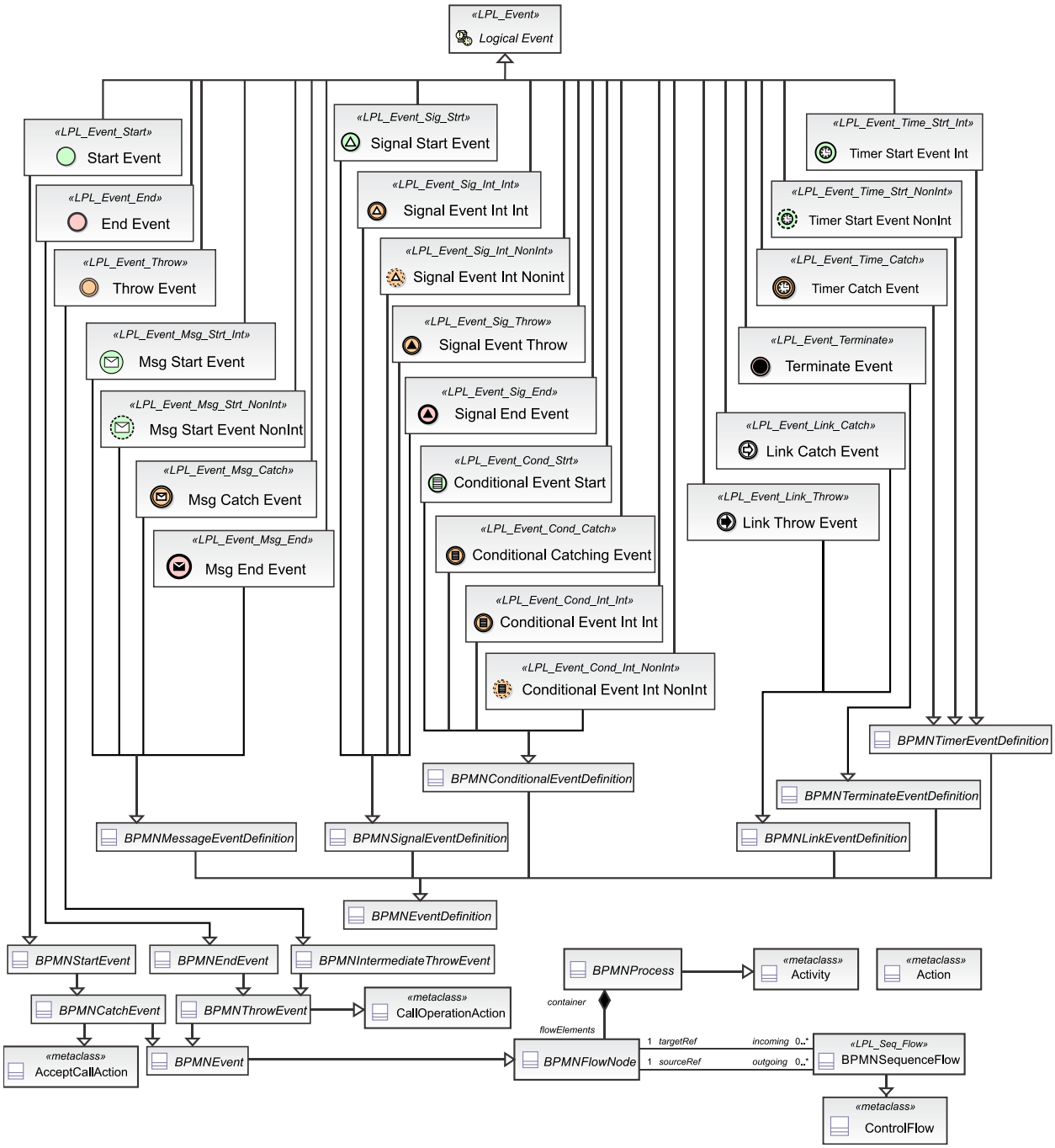


Figure 6 – Logical Process Viewpoint Metamodel (part 3)

This portion of the metamodel also 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*. *Send Message* and *Receive Message* are also shown on this model for context.

The elements and structure of the Logical Process language is completed in Figure 6. The complete set of *Logical Events* is defined in Figure 6. Note that *Logical Event* is 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 are defined that allow for the creation of high-level models if desired.

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

- **Activity** – the various types of activities and tasks defined in Figure 5 as types of Logical Activities, 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,
- **Data Object** – a Logical 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;
- **Event-Based Gateways** – always followed by catching events or receive tasks, the flow is routed to the event that happens first. For exclusive event-based gateways each occurrence of an event starts a new process instance, and for parallel event-based gateways all events start a new process instance;
- **Exclusive Gateway** – the flow goes to only one outgoing branch, or when merging any input branch can trigger the flow;
- **Parallel Gateway** – the flow goes to all outgoing branches, or when merging all input branches must complete before triggering the outgoing flow(s);
- **Group** – used to group together tasks or other process elements, for analysis or other purposes (has no effect on the process);
- **Lane** – used to define and structure Processes;
- **LaneSet** – used to define and structure Lanes;
- **Logical Domain** – the set of policy and rules governing the system under study;
- **Logical Role** – the role assumed by an actor when interacting with an activity;
- **Logical Rule** – rule governing the activities of a specific *FlowNode*;
- **Logical Storage** – persistence for any information within the system;
- **Message Flow** – connections and flows of messages between two *Pools / Lanes*;
- **Pool** – used to define 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:

- **Activity / Sub-Process** – high-level abstraction for an Activity or Process definition;
- **Communications** – high-level abstraction of complex communications between Locations;
- **Data Object** – high-level abstraction for a data definition;
- **Domain** – high-level abstraction for the system and applicable business rules and policies;
- **LAN** – high-level representation of a complete local area network;
- **Location** – high-level representation of a business "location";
- **Network** – high-level representation of a complete (IP) network;
- **Pool** – high-level abstraction for grouping of Activities and sub-processes;
- **Server / Mainframe** – high-level representation of computational resources;
- **Store** or **Data Store** – high-level abstraction for persistence;
- **TelecommNet** – high-level representation of a complete telecommunications network;
- **Untyped Events** – high-level abstraction for events;
- **WAN** – high-level representation of a complete wide area network link or links;
- **WiFi** – high-level representation of a complete (wireless IP) network;
- **Zone** – high-level for grouping of activities and sub-processes that have a common security posture (i.e. common set of policy and rules).

Definitions for all of these Logical Process Viewpoint Language elements follow.

«stereotype» LPL_Activity_Call

Extends

«metaclass» ActivityNode

«metaclass» StructuredActivityNode

Semantics

A (Logical) 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)
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 (see page 281 for more information on *scopes*).
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 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» LPL_Association

Extends

«metaclass» Association

Semantics

Used to define a (Logical) 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.
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.

Name	Description (OMG 2013)
sourceRef: BaseElement	The `BaseElement` that the Association is connecting from.
targetRef: BaseElement	The `BaseElement` that the Association is connecting to.

Notation

- - - - - - or - - - - → or ←- - -→

Constraints

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

«stereotype» LPL_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  (collection)

Constraints

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

«stereotype» LPL_Data_Object and LPL_Data_Collection

Extends

«metaclass» Class

Semantics

Defines a (Logical) 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» LPL_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 Outputs 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

 or  (collection)

Constraints

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

«stereotype» LPL_Data_Store

Extends

«metaclass» Classifier

«metaclass» DataStoreNode

«metaclass» Node

Semantics

Defines a (Logical) 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 (dataStoreReference), 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.

Notation

Constraints

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

«stereotype» LPL_Storage

Extends

«metaclass» Node

Semantics

A Logical Storage provides a mechanism to store, retrieve or update stored information that will persist beyond the scope of an Activity. Note that `Data Store` supports business processes and are a specialization of Logical Storage. Logical Storage is therefore used to support all other persistence requirements (i.e. Management processes and systems). This element provides the ability to manage complexity in architectures through the definition of hierarchical models.

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*.

Notation

Constraints

- May be connected to a Logical Network through a Logical Switch;
- May have relationships with Activity elements (i.e. a Data Store supports Activities and Processes);
- Relationships define the usage of stores by management processes and systems.

«stereotype» LPL_DataAssociation

Extends

«metaclass» Association

Semantics

The core concepts of a (Logical) 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.
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» LPL_DataStoreRef

Extends

«metaclass» Association

Semantics

Defines a Logical `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» LPL_Event_Cond_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Conditional Catch 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» LPL_Event_Cond_Int_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Conditional 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. 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» LPL_Event_Cond_Int_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Conditional 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 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» LPL_Event_Cond_Start (interrupting)

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Conditional Start Event that is interrupting. This type of event is triggered when the specified condition (expression) becomes *true*.

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.
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

Only the top-level and interrupting variants are supported.

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» LPL_Event_End

Extends

«metaclass» FinalNode

«metaclass» CallOperationAction

Semantics

Defines a (Logical) 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» LPL_Event_Link_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) 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» LPL_Event_Link_Throw

Extends

«metaclass» CallOperationAction

Semantics

Defines a (Logical) 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» LPL_Event_Msg_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Message Catch Event. An **Event** is something that happens during the course of a **Process**. These **Events** affect the flow of the **Process** and usually have a cause or an impact. A **Message Catch Event** is used to receive a **Message**. This causes the **Process** to continue if it was waiting for the **Message**, or changes the flow for *exception handling*. (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» LPL_Event_Msg_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Logical) 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» LPL_Event_Msg_Strt_Int & LPL_Event_Msg_Strt_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) 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).
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» LPL_Event_Signal_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Signal Catch 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	Description (OMG 2013)
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» LPL_Event_Signal_End

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Logical) Signal End Event. 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**.

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» LPL_Event_Signal_Int_Int

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Signal Boundary (Intermediate) Interrupting Event. 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**.

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» LPL_Event_Signal_Int_NonInt

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Signal Boundary (Intermediate) Non-interrupting Event. 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**.

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

Name	Description (OMG 2013)
	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» LPL_Event_Signal_Start (interrupting)

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Error Start Event that is interrupting, in the usage within UAM it is always an interrupting event. 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**.

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

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» LPL_Event_Signal_Throw

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Signal Throw Event. 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**.

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» LPL_Event_Start

Extends

«metaclass» AcceptCallAction

Semantics

Defines a (Logical) 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.

Name	Description (OMG 2013)
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» LPL_Event_Terminate

Extends

«metaclass» FinalNode

«metaclass» ActivityFinalNode

Semantics

Defines a (Logical) 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» LPL_Event_Throw

Extends

«metaclass» SendSignalAction

Semantics

Defines a (Logical) 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.
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» LPL_Event_Timer_Catch

Extends

«metaclass» AcceptEventAction

Semantics

Defines a (Logical) Timer Catch Event. An **Event** is something that happens during the course of a **Process**. In *normal flow* the **Timer Intermediate Event** acts as a delay mechanism based on a specific time-date or a specific cycle (e.g., every Monday at 9am) can be set that will trigger 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 **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» LPL_Event_Timer_Strt_Int & LPL_Event_Timer_Strt_NonInt

Extends

«metaclass» AcceptEventAction

«metaclass» TimeEvent

Semantics

Defines a (Logical) 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» LPL_Gateway_Event and LPL_Gateway_Event_Excl and LPL_Gateway_Event_Para

Extends

«metaclass» ControlNode

Semantics

Defines a (Logical) Event-Based Gateway used to control flows. 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» LPL_Gateway_Exclusive

Extends

«metaclass» ControlNode

Semantics

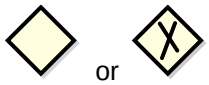
Defines a (Logical) Exclusive Gateway. A diverging Exclusive Gateway (Decision) is used to create alternative paths within a Process flow. This is basically the “diversion point in the road” for a Process. For a given *instance* of the Process, only one of the paths can be taken. A converging Exclusive 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



or

Constraints

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

«stereotype» LPL_Gateway_Parallel

Extends

«metaclass» ControlNode

Semantics

Defines a (Logical) 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	Description (OMG 2013)
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**.

Notation

Constraints

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

«stereotype» LPL_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» LPL_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» LPL_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» LPL_Message_Flow

Extends

«metaclass» Association

Semantics

Defines a (Logical) 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» LPL_Message_Receive

Extends

«metaclass» Class

Semantics

Defines a (Logical) 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» LPL_Message_Send

Extends

«metaclass» Class

Semantics

Defines a (Logical) 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» LPL_Pool

Extends

«metaclass» Activity Partition

Semantics

A (Logical) 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

Constraints

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

«stereotype» LPL_Rule

Extends

«metaclass» Constraint

Semantics

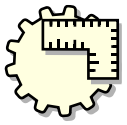
A Logical Rule is a declaration of policy or a condition that must be satisfied and are used (implemented) by Activities, Gateways, and Events within processes.

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

Name	Description (OMG 2013)
type: string	Defines the type of business rule (from Ronald Ross): ➢ Term ➢ Fact ➢ Constraint ➢ Derivation

Notation



Constraints

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

«stereotype» LPL_Seq_Flow

Extends

«metaclass» Association

Semantics

Defines a (Logical) 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**:

Name	Description (OMG 2013)
	▪ 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

Normal:  Default:  Conditional: 

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.

Table 2 – Sequence Flow Connection Rules

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

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*.

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*.

Table 3 – Message Flow Connection Rules

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

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.

«stereotype» LPL_SubProcess

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

Defines a (Logical) 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.

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`.

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**.

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» LPL_Subprocess_Adhoc

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Logical) 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. An Ad-Hoc Sub-Process is a specialized type of Sub-Process that is a group of Activities that have no required sequence relationships. A set of Activities can be defined for the Process, but the sequence and number of performances for the Activities is determined by the performers of the Activities (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.

Name	Description (OMG 2013)
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**.
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» LPL_Task

Extends

«metaclass» Action

Semantics

A (Logical) 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.

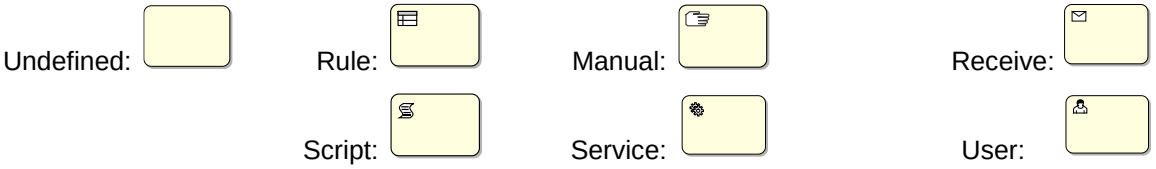
Name	Description (OMG 2013)
name: string	A descriptive name for the element.
isForCompensation: boolean = false	A flag that identifies whether this **Activity** is intended for the purposes of *compensation*. If *false*, then this **Activity** executes as a result of normal execution flow. If *true*, 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).

Notation

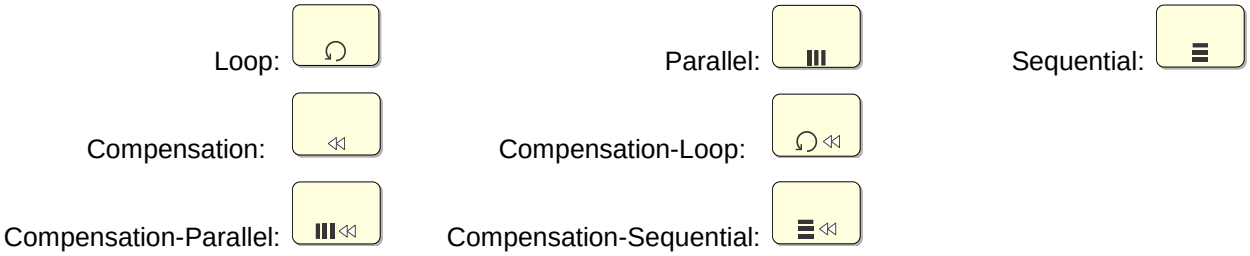
There are seven types of Task: *undefined*, *Business Rule*, *Manual*, *Receive*, *Script*, *Service* and *User* Tasks. Each of these types Task may also have additional "markers" signifying that they are *loop* (of three types) or *compensation* Tasks. These markers may also be used in combination, 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:



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



Constraints

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

«stereotype» LPL_Task_Manual

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

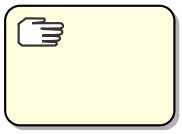
A (Logical) Manual Task is a type of activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process". A Manual Task is a Task that is expected to be performed without the aid of any business process execution engine or any application. An example of this could be a telephone 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

Name	Description (OMG 2013)
	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).

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» LPL_Task_Receive

Extends

«metaclass» Activity

«metaclass» AcceptEventAction

Semantics

A (Logical) Receive 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 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:

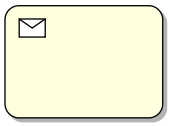
Name	Description (OMG 2013)
id: string	This attribute is used to uniquely identify BPMN elements. The `id` is REQUIRED if this element is referenced or intended to be referenced by something else. If the element is not currently referenced and is never intended to be referenced, the `id` MAY be omitted.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

Name	Description (OMG 2013)
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 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

Name	Description (OMG 2013)
	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» LPL_Task_Rules

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Logical) 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" (OMG 2013). 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).

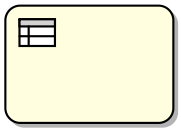
Properties

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.

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).
implementation: string = ##webService	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» LPL_Task_Script

Extends

«metaclass» Activity

«metaclass» CallOperationAction

Semantics

A (Logical) 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).

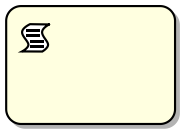
Properties

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`.

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).
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» LPL_Task_Send

Extends

«*metaclass*» Activity

«*metaclass*» CallOperationAction

Semantics

A (Logical) Send 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 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.

Name	Description
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 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

Name	Description (OMG 2013)
	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» LPL_Task_Service

Extends

«*metaclass*» Activity

«*metaclass*» StructuredActivityNode

Semantics

A (Logical) Service 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 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.

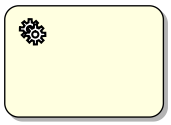
In addition to the above, **Service Tasks** have the following attributes:

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

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).
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.
operationRef: Operation	This attribute specifies the operation that is invoked by the **Send 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» LPL_Task_User

Extends

«metaclass» Activity

«metaclass» StructuredActivityNode

Semantics

A (Logical) User 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 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.

Name	Description
crud: BPL_Entity [0..*]	References to the Entities that are Created, Read, Updated or Deleted by this activity.

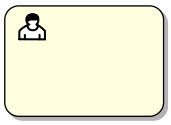
In addition to the above, **User Tasks** have the following attributes:

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.
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

Name	Description (OMG 2013)
	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 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» LPL_TextAnnotation

Extends

«metaclass» comment

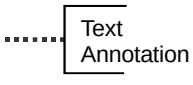
Semantics

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).

Logical Locations Viewpoint

The Logical Locations viewpoint describes the structure of locations (each of which contain activities, roles, and processes) that when structured into a model define the complete set of locations, and their interrelationships, at which the system under study operates. At the logical level locations are abstracted, likely into types of locations. For example a bank operates at many locations, but each one can be abstracted into Logical Locations such as Head Office, Main Branch, Branch, Full Service ATM, and ATM among others.

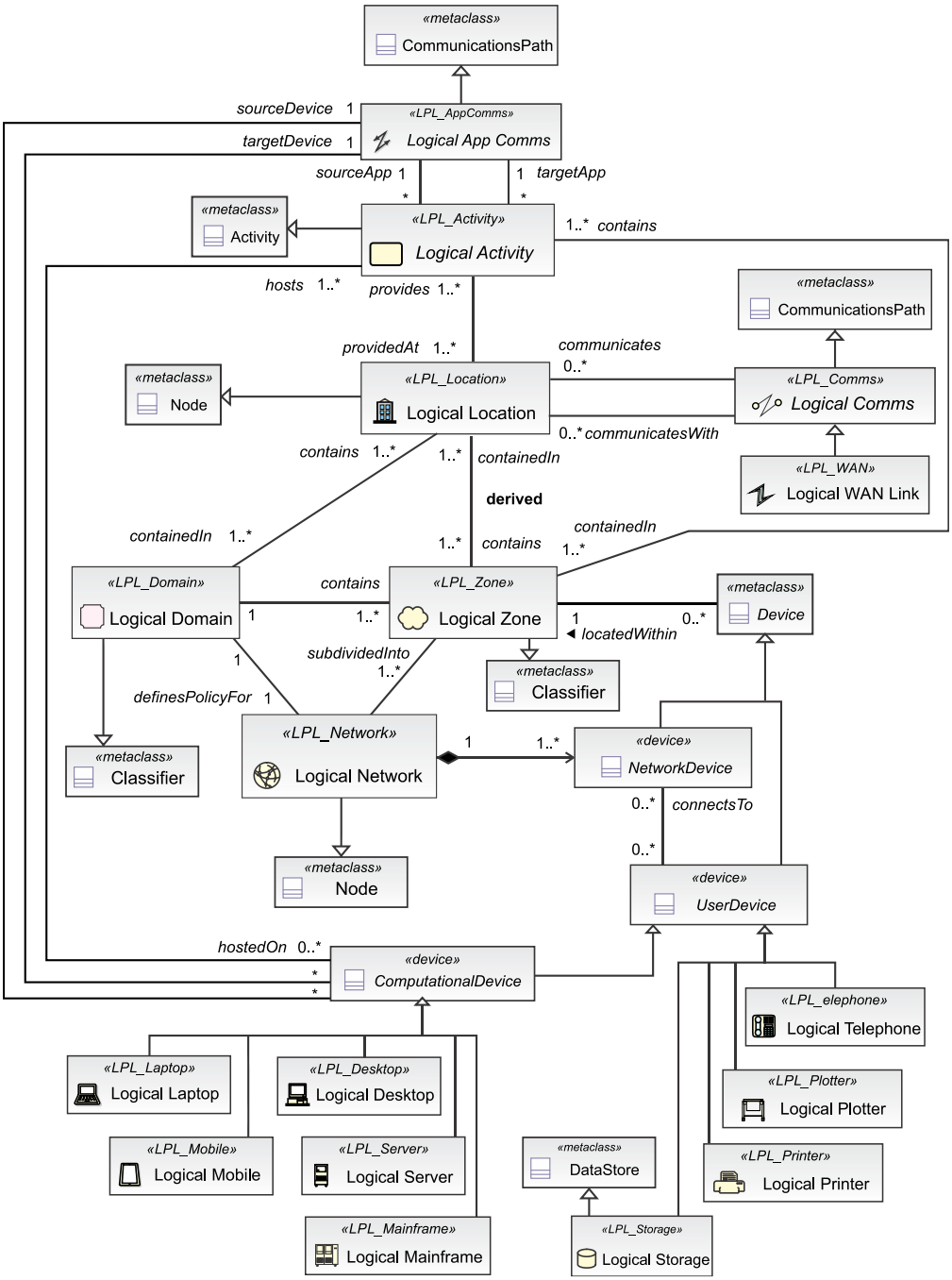


Figure 7 – Logical Locations Viewpoint Metamodel (part 1)

The Logical Locations Model defines the logical level business locations involved in the enterprise or system, as defined by the scope of the modeling effort. The Logical Locations viewpoint is very much concerned with physical locations but at the logical level these locations are abstracted, as noted. The network components, IT security components, computer systems and other physical elements may be identified at each location, but specific technologies such as make and model for these elements are not defined.

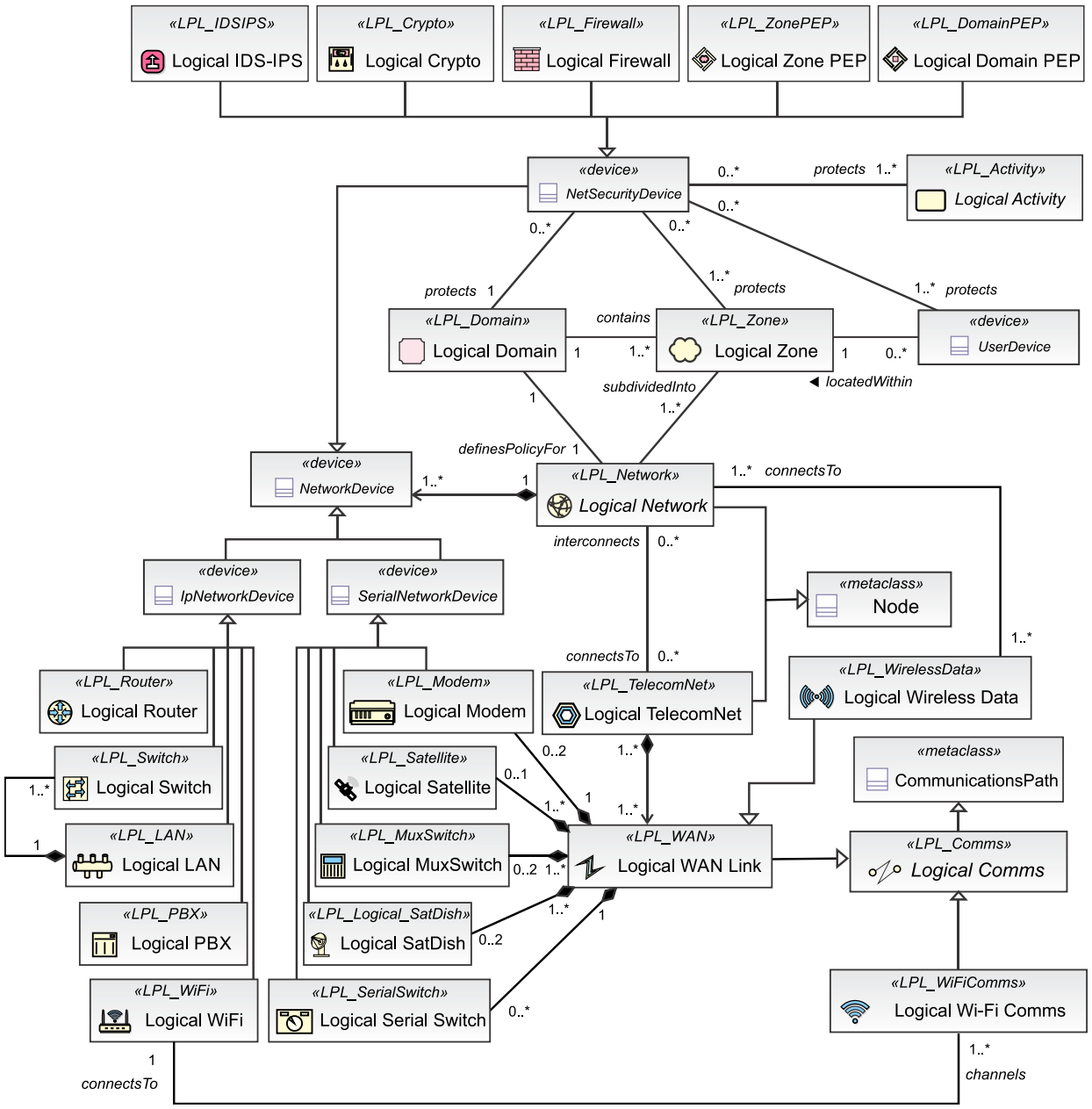


Figure 8 – Logical Locations Viewpoint Metamodel (part 2)

From Figure 7 we can see that:

- Logical Activities are provided at Logical Locations;
- Logical Activities are hosted on Computational Devices;
- Logical Activities are Contained In Logical Zones;

- Logical Locations communicate with each other through Logical Comms which are a type of Wide-Area Network link;
- Logical Locations have one or more Logical Domains;
- Logical Domains contain one or more Logical Zones;
- Logical Domains are classifiers that define the policies for Logical Networks;
- Logical Zones are classifiers that have Network and User devices are located within them;
- Logical Networks (and Domains as well) are sub-divided into one or more Logical Zones;
- The policies and rules governing all aspects of the Logical Location are defined by the Logical Domain;
- Policies and rules defined by the Logical Domain govern the usage and operation of the Logical Network that implements the Domain;

Figure 8 defines the set of network devices and network security devices used to model the system. This metamodel is described simply as follows:

- Network devices define (are composed into) a Logical Network for the system under study;
- A Logical LAN is composed of one or more Logical Switches;
- A Logical Wide Area Network (WAN) Link is composed of Modems, Satellites, Multiplexors, Satellite Dishes and Serial Switches;
- A Logical WAN Link in turn forms part of a Logical Telecommunications Link;
- Logical Telecommunications Links interconnects Logical Networks;
- Network Security Devices are Intrusion Detection and Prevention devices, Cryptos, Firewalls, Zone PEPs and Domain PEPs;
- Logical Activities are protected by zero or more Network Security Devices;
- Network Security Devices protect a Logical Domain, one or more Logical Zones, one or more User Devices and one or more Logical Activities;
- The perimeters of Logical Domains, Zones, and User Devices are protected by zero or more Network Security 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:

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

Definitions for all of these Logical Location Viewpoint Language elements follow.

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

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

Guidance > Guidelines > Logical Locations Model

«stereotype» LPL_Comms

Extends

«metaclass» CommunicationPath

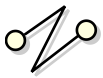
Semantics

Logical Communications represents the communications, of any type, between Logical 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 (i.e. higher level abstractions of communications).

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.

Notation



Constraints

➲ Defines communications path relationships between Logical Locations only.

«stereotype» LPL_Crypto

ExtendNode

«metaclass» Device

Semantics

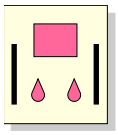
A Logical Crypto provides information encryption services on the network. These services include link and end-to-end encryption (i.e. Encryption of an IP stream or a WAN link) or offline encryption (i.e. the encryption of files, hard drives, or emails, etc.).

Properties

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

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

Notation



Constraints

- May connect (have associations) to Logical Telephones, Logical Storage, Logical Servers, Logical Desktops, Logical WAN, or Logical Routers;
- Forms part of a Logical WAN or Network.

«stereotype» LPL_Desktop

Extends

«metaclass» Device

Semantics

A Logical 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 Logical Comms.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

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

«stereotype» LPL_Domain

Extends

«metaclass» Node

«metaclass» Classifier

«metaclass» Package

Semantics

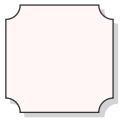
A Logical (Security) Domain is an environment or context that is defined by security policies, security models, and security architecture, including a set of resources and set of system entities that are authorized to access the resources. A Logical Domain is managed by a single *authority*, and may 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 logical 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 Logical 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

- Cannot contain other domains.

«stereotype» LPL_DomainPEP

Extends

«metaclass» Node

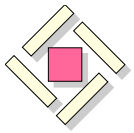
Semantics

A Logical 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 logical level (if desired) through the use of a Domain PEP in the architecture description.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical 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 Logical LANs, Logical Switches or Logical Routers.

«stereotype» LPL_Firewall

Extends

«metaclass» Device

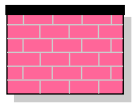
Semantics

A Logical 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 Logical Domain (i.e. a Domain PEP) or for a Logical Zone (i.e. a Zone PEP).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Firewall.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

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

«stereotype» LPL_IDSIPS

Extends

«metaclass» Device

Semantics

A Logical 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 Logical IDS/IPS.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

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

«stereotype» LPL_LAN

Extends

«metaclass» Node

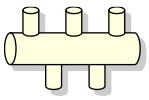
Semantics

A Logical LAN is a higher level of abstraction of a Logical 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 (i.e. higher level abstractions of LANs).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical LAN.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to `UserDevices` and to Logical Routers;
- Forms part of a Logical Network.

«stereotype» LPL_Laptop

Extends

«metaclass» Device

Semantics

A Logical Desktop defines the user device used to interact with the system. Users as Actors assume Roles when they 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 Logical Laptop.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

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

«stereotype» LPL_Location

Extends

«metaclass» Node

Semantics

A Logical Location defines the locations where the enterprise has a presence that it implements and manages. Logical level activities, devices, and processes are defined that are found at the locations. Locations are a categorization of locations, based upon the logical (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 (i.e. higher level abstractions of Locations).

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 Logical Process Model and how the 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 Logical Domains or with Logical Activities (provided at the location);
- May contain Logical Zones;
- May communicate with other Locations via Logical Communications;
- All owned properties shall be public.

«stereotype» LPL_Mainframe

Extends

«metaclass» Device

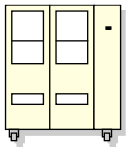
Semantics

A Logical Mainframe is a large usually central compute resource to which Activities and other functions may be deployed. Logical Servers are located within (Network Domains and) Zones and are deployed to Logical Locations. This element provides the ability to manage complexity in architectures through the definition of hierarchical models (i.e. higher level abstractions of computational resources).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Mainframe.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Is connected to a Logical Network through a Logical Switch,
- Wholly located within a Logical Zone.

«stereotype» LPL_Mobile

Extends

«metaclass» Device

Semantics

A Logical Mobile defines a generic mobile user device (e.g. smartphone, mobile phone, tablet, etc.) used to interact with the system through Wi-Fi or other network communications. 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 Logical Mobile device.

Notation

Constraints

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

«stereotype» LPL_Modem

Extends

«metaclass» Device

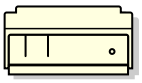
Semantics

A Logical Modem is used as a component part of a Logical 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 Logical Modem.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Forms part of a Logical WAN Link.

«stereotype» LPL_MuxSwitch

Extends

«metaclass» Device

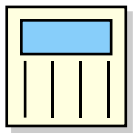
Semantics

A Logical Multiplexor/Switch is part of a Logical 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 Logical Mux.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Forms part of a Logical WAN Link.

«stereotype» LPL_Network

Extends

«metaclass» Node

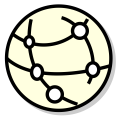
Semantics

A Logical Network is the complete set of devices and elements that comprise and define a computer network at the logical 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 (i.e. higher level abstractions of IP networks).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical 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 Logical Network will be drilled-down in a hierarchical fashion starting from this high-level element.

«stereotype» LPL_PBX

Extends

«metaclass» Device

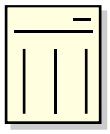
Semantics

A Logical PBX provides telephone network services such as the dialling and switching of voice calls between telephones. At the logical level this one "device" provides all of the required telephony services.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical PBX.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

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

«stereotype» LPL_Plotter

Extends

«metaclass» Device

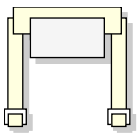
Semantics

A Logical 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 Logical Plotter.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

➲ Is connected to a Logical Network through a Logical Switch.

«stereotype» LPL_Printer

Extends

«*metaclass*» Device

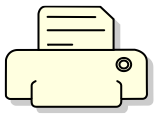
Semantics

A Logical Printer defines a standard 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 Logical Printer.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Is connected to a Logical Network through a Logical Switch.

«stereotype» LPL_Router

Extends

«*metaclass*» Device

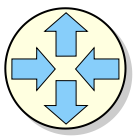
Semantics

A Logical 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 Logical Router.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to Logical Modems, Logical MuxSwitches, or Logical Serial Switches;
- May connect (have associations) to `UserDevices`, or to `IpNetworkDevices`;

➲ Forms part of a Logical Network.

«stereotype» LPL_SatDish

Extends

«metaclass» Device

Semantics

A Logical Satellite Dish is part of a Logical 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 Logical Satellite Dish.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

➲ Forms part of a Logical WAN Link.

«stereotype» LPL_Satellite

Extends

«metaclass» Device

Semantics

A Logical Satellite provides communications links between two points. At the logical 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.

Notation

Constraints

Forms part of a Logical WAN Link.

«stereotype» LPL_SerialSwitch

Extends

«metaclass» Device

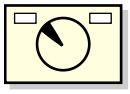
Semantics

A Logical Serial Switch is part of a Logical WAN Link, providing the ability to switch a serial communications link between equipment. At the logical 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 Logical Serial Switch.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Forms part of a Logical WAN Link.

«stereotype» LPL_Server

Extends

«metaclass» Device

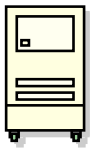
Semantics

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

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Server.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- Is connected to a Logical Network through a Logical Switch;
- Wholly located within a Logical Zone.

«stereotype» LPL_Storage

Extends

«metaclass» Node

Semantics

Logical 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 Logical Storage. Logical 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.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- May connect (have associations) to a Logical Network through a Logical 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 logical usage of stores by processes and systems.

«stereotype» LPL_Switch

Extends

«metaclass» Device

Semantics

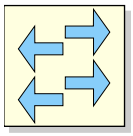
A Logical Switch is used to implement a Logical Area Network (LAN) by switching network traffic

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 Logical Switch.
location: string	The location of the device within the system or enterprise.

Notation



Constraints

- May connect (have associations) to `UserDevices` and to Logical Routers;
- Forms part of a Logical Network.

«stereotype» LPL_TelecomNet

Extends

«metaclass» Node

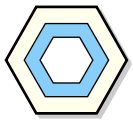
Semantics

A Logical 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 (i.e. higher level abstractions of telecommunications networks).

Properties

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

Notation



Constraints

- Interconnects Logical Networks or Logical Locations.

«stereotype» LPL_Telephone

Extends

«metaclass» Device

Semantics

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

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical Telephone.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- Is connected to a Logical Network through a Logical Switch.

«stereotype» LPL_WAN

Extends

«metaclass» node

Semantics

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

Properties

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

Notation

Constraints

- Forms part of a Logical Telecommunications Network.

«stereotype» LPL_WiFi

Extends

«metaclass» Node

Semantics

A Logical Wi-Fi IP network (hot spot) provides communications between a Logical 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 Logical Wi-Fi network.
location: string	The location of the device within the system or enterprise.

Notation

Constraints

- Forms part of a Logical (IP) Network.

«stereotype» LPL_WiFiComms

Extends

«metaclass» communicationsPath

Semantics

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

Properties

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

Notation

Constraints

- Forms part of a Logical (IP) Network.

«stereotype» LPL_WirelessData

Extends

«metaclass» communicationsPath

Semantics

A Logical Wireless Data communications provides data communications between a mobile device with data capability and a Logical 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 Logical wireless data link.

Notation

Constraints

- Forms part of a Logical Telecommunications Network.

«stereotype» LPL_Zone

Extends

«metaclass» Node

«metaclass» Classifier

Semantics

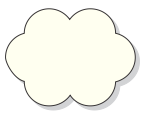
A Logical Security Zone is an environment or context that is defined by security policies, security models, and security architecture, including a set of resources and set of system entities that are authorized to access the resources. A Logical Zone may contain one or more sub-zones. Different sub-zones are created when security models or policies (and possibly architecture) are significantly different from one zone to the other, or are conflicting. Separate Logical Zones provide clearer separation of concerns and ease policy enforcement and system management. Synonyms: security zone or policy zone. This element provides the ability to manage complexity in architectures through the definition of hierarchical models (i.e. higher level abstractions of a collection of Activities within Locations and Domains).

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical 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 Logical 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 Logical Domains;
- Zones are defined only when needed.

«stereotype» LPL_ZonePEP

Extends

«metaclass» Node

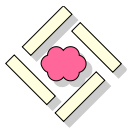
Semantics

A Logical 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 logical level (if desired) through the use of a Zone PEP in the architecture description.

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the Logical 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 within the system or enterprise

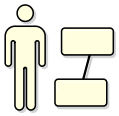
Notation



Constraints

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

Logical Roles Viewpoint



The Logical Roles viewpoint defines the roles, at the logical level, involved in and important to the defined business processes. The roles implemented by the logical systems are incorporated into the model, optionally including relationships to business functions (activities) involved with the role. Roles have a direct relationship to user interactions with the system. That is, for every role users have within a system, there is a set of activities (defining a business process) that include the user interfaces (show as User Tasks). Security is thus an important aspect of this model as well.

The focus of the roles model is on all interactions with the business processes and activities, with the Logical 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 Logical 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 Logical Roles Viewpoint, the Activities within this model do not represent the complete business Activity, just the presentation layer or interface. These Logical Activities therefore represent the boundary portion of the Logical Process Model—the Logical Process Model captures all Processes, Activities and Tasks, but with the interface/GUI portion also being represented in the Logical Roles Model through a User Task (or Service Task for system interactions). The Logical Roles Model is incomplete from a process perspective; therefore the Logical Process Model defines the complete business solution for Processes and Activities.

The Logical Roles viewpoint is a model of the structure of logical roles (each of which contains a (human) interface design) that when formed into a model define the set of logical roles, and their interrelationships, defining the system's interactions with defined logical roles.

This simple metamodel shown in Figure 9 is described as follows:

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

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

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

Guidance > Guidelines > Logical Roles Model

Note that the "User Task" and "Service Task" are defined within the Logical 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 Logical Roles Model.

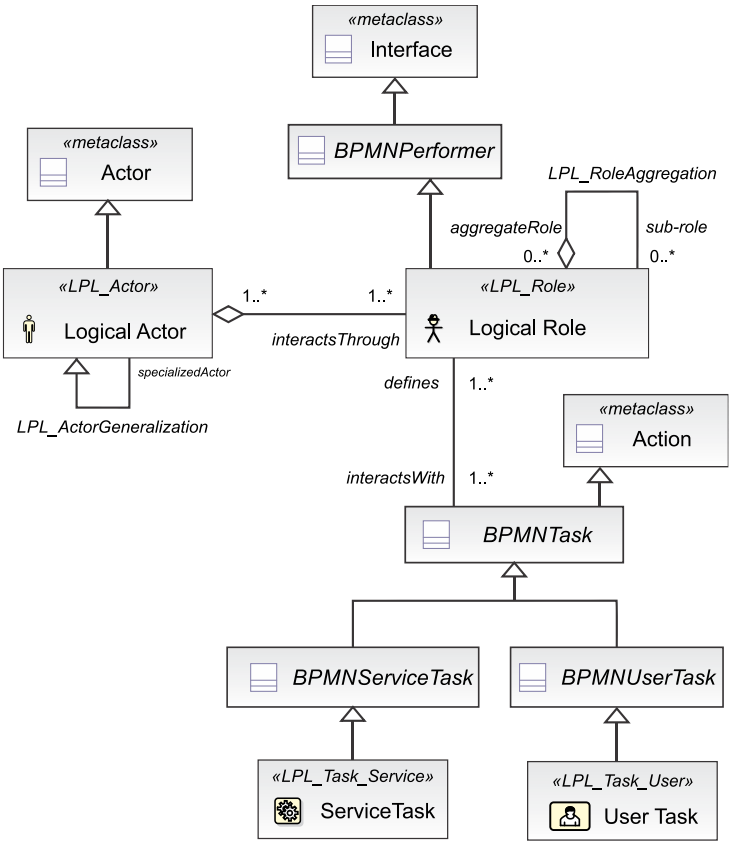


Figure 9 – Logical Roles Viewpoint Metamodel

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

«stereotype» LPL_Actor

Extends

«metaclass» Actor

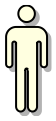
Semantics

A Logical Actor can be 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. Logical Actors are high-level conceptualizations of the Actors with the system; further analysis is required to fully understand and define 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".

Properties

Name	Description
id: string	This attribute is used to uniquely identify elements.
name: string	A descriptive name for the role.
characteristics: string	A logical 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 logical.

Notation



Constraints

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

«stereotype» LPL_Role

Extends

«metaclass» Interface

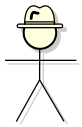
Semantics

Defines a Logical Role that 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. 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

- Has relationships only with Activities, Actors, or with other Roles;
- A derived relationship with Logical Entities is used to identify what Roles (Actors) interact with these Entities.

«stereotype» LPL_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
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).

Notation

Generalization association: Specialized Actor  Generalized Actor

Constraints

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

«stereotype» LPL_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 `LPL_Role` element and a `LPL_Actor`;

«stereotype» LPL_Task_User / LPL_Task_Service (from LP Viewpoint)

Extends

«*metaclass*» Activity

«*metaclass*» StructuredActivityNode

«*metaclass*» OpaqueAction

Semantics

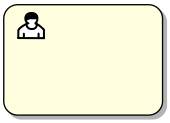
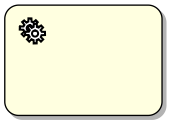
A (Logical) User Task is an atomic activity, as defined in BPML V2.0, "an Activity is work performed as part of a business process" (OMG 2013). A User Task "is a typical 'workflow' Task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort" (OMG 2013). These Tasks in the Logical 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 (for User Tasks) which allows **BPMN** adopters to specify task rendering attributes by using the **BPMN** Extension mechanism.

Notation


or


Constraints

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

Logical Perspective Language Summary

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

	Aspect			
Perspective	**Data**	**Activity**	**Location**	**People**
▪ Logical	**Logical Entity Model** ➢ Logical Entity ➢ Entity Associations ▪ Simple ▪ Aggregation ▪ Generalization / Specialization	**Logical Process Model** ➢ Activity Call ➢ Association ➢ Data Collection ➢ Data Input ➢ Data Object ➢ Data Output ➢ Data Store ➢ DataAssociation ➢ DataStoreRef ➢ Event Cond Catch ➢ Event Cond Int Int ➢ Event Cond Int Nonint ➢ Event Cond Start ➢ Event End ➢ Event Link Catch ➢ Event Link Throw ➢ Event Message End ➢ Event Message Start ➢ Event Msg Start Nonint ➢ 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 Start ➢ Event Timer Strt Nonint ➢ Gateway Event-based ➢ Gateway Event Excl ➢ Gateway Event Parallel ➢ Gateway Exclusive ➢ Gateway Parallel ➢ Group ➢ Lane / LaneSet ➢ Message Flow ➢ Message Receive ➢ Message Send ➢ Pool ➢ Rule ➢ Sequence Flow ➢ Sub-process ▪ Loop ▪ Parallel ▪ Sequential ➢ Sub-process Ad Hoc ▪ Loop ▪ Parallel ▪ Sequential ➢ Task ▪ Loop ▪ Parallel	**Logical Locations Model** ➢ Logical Comms ➢ Logical Crypto ➢ Logical Storage ➢ Logical Desktop ➢ Logical Domain ➢ Logical Domain PEP ➢ Logical Firewall ➢ Logical IDSIPS ➢ Logical LAN ➢ Logical Laptop ➢ Logical Location ➢ Logical Mainframe ➢ Logical Mobile ➢ Logical Modem ➢ Logical MuxSwitch ➢ Logical Network ➢ Logical PBX ➢ Logical Plotter ➢ Logical Printer ➢ Logical Router ➢ Logical SatDish ➢ Logical Satellite ➢ Logical SerialSwitch ➢ Logical Server ➢ Logical Switch ➢ Logical TelecomNet ➢ Logical Telephone ➢ Logical WAN ➢ Logical WiFi ➢ Logical WiFi Comms ➢ Logical Wireless Data ➢ Logical Zone ➢ Logical Zone PEP	**Logical Roles Model** ➢ Logical Actor ➢ Logical Role ➢ User Task (from LPV) ➢ Relationships: ▪ Actor Generalization ▪ Role Aggregation

	Aspect			
Perspective	**Data**	**Activity**	**Location**	**People**
		▪ 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.