Activities [Functional View]

Docupedia Export

Author:Mukkamala Kiran (XC-AS/EDA2)
Date:29-Jan-2024 11:21

Table of Contents

Error!
Page named 'https://inside-docupedia.bosch.com/confluence/pages/resumedraft.action?draftId=2983378539&draftShareId=618f5950-3c28-4257-b928-eb5e26e9ce25&' is not accessible.

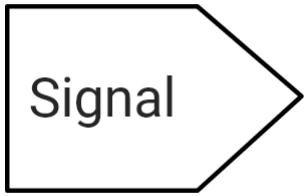
1 Overview

In this guideline, we will cover activities in functional view.
We will take a closer look at below modeling elements:

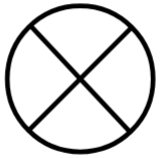
- Inputs and outputs of actions and activities
- Brief description of various activities related model elements
-
    - Actions
    - Nodes
    - Flows
- Examples for
    - Basic activity with activity diagram
    - Various activity related nodes and connections
    - Activity partitions and allocating to other subsystems
    - Assigning call behavior to an activity
    - Creating action pins and connecting to
    - Extending a simple model with Control nodes etc...

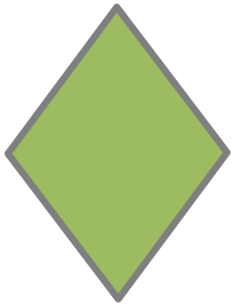
2 Activity Model Elements in Enterprise Architect

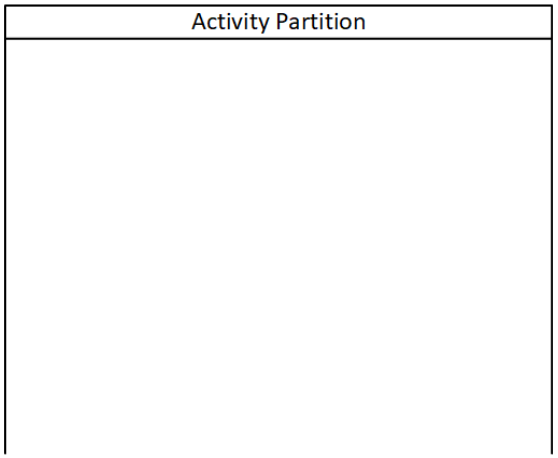
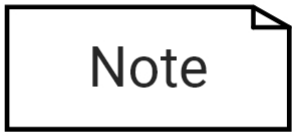
Table 1: List of Diagram elements allowed in Activity Diagram

Element	Graphic/ Notation	Section
Action		Action
Send Signal Action		Send Signal Actions
Accept Event Action		Accept Event Actions

Element	Graphic/ Notation	Section
Time Event		Time Actions
Action Input/Output Pin		Pins
Activity Diagram Frame		Activity Diagram Frame

Element	Graphic/ Notation	Section
Activity Initial node		Initial Nodes
Activity Final node		Flow Final Nodes and Activity Final Nodes
Flow Final node		Flow Final Nodes and Activity Final Nodes

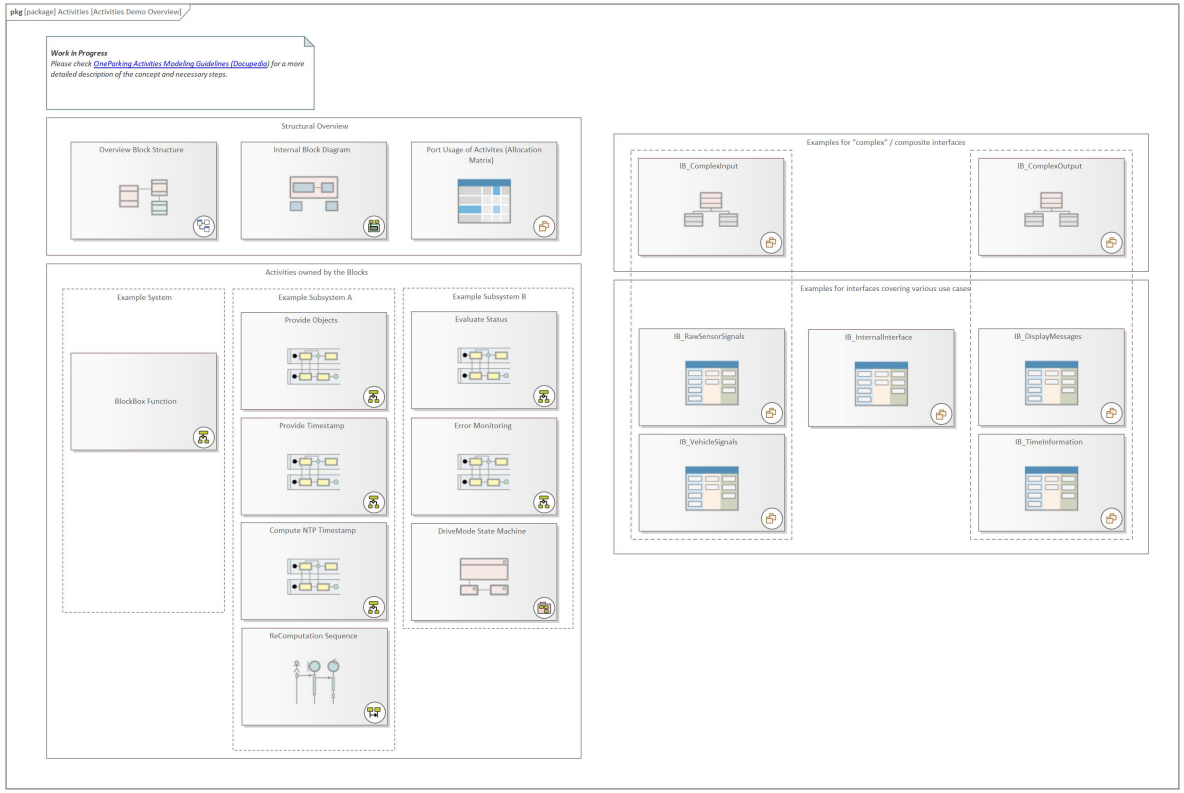
Element	Graphic/ Notation	Section
Join / Fork node		Join node / Fork node
Decision / Merge node		Decision node / Merge node

Element	Graphic/ Notation	Section
Activity Partition		Activity Partitions
edge / connection		Edges
Note		NA, General Notes

Element	Graphic/ Notation	Section
Pin		Pins
Activity Parameter		Activity Parameter

3 Training Material

https://xc-ea.de.bosch.com/WebEA?m=55&o=276E4394-207D-48fa-8851-A62044085523



Slides:
OneParking_Activiity_Modeling_02_Activity_Training_Essentials_V3.pptx
Enterprise Architect Files: 02_Activities_Training.zip

4 Activity Diagram

- An activity diagram is a kind of behavior diagram, it's a dynamic view of the system that expresses sequences of behaviors and event occurrences over time.
- An activity diagram can display various kinds of actions, enabling you to convey even the most complex behavioral narratives.
- Object nodes let you model the flow of matter, energy, and data through an activity, and you can use control nodes to steer the execution of an activity.
- Activity partitions enable you to allocate system behaviors to system structures.

4.1 Advantage

One of its key advantages is its readability; activity diagrams can express complex control logic better than sequence diagrams and state machine diagrams. And activity diagrams are uniquely capable of expressing continuous system behaviors.

4.2 Disadvantage

Activity diagrams do have a disadvantage: moderate ambiguity. Activity diagrams express the order in which actions are performed, and they can optionally express which structure performs each action. They do not, however, offer any mechanism to express which structure invokes each action. (Sequence diagrams, in contrast, can express all three pieces of information.)

4.3 Typical Usage

Modelers typically use activity diagrams as analysis tools when working with stakeholders to define the problem space and specify the required behavior of the system. Also use activity diagrams to communicate with other members of your team and capture the expected behaviors of the system's internal parts. Activity diagrams are not particularly useful tools for detailed design - that is, unambiguous specifications of behavior suitable for system implementation. Hence Activity diagrams are not tied to any particular stage of the system life cycle.

NOTE: Activity diagrams, sequence diagrams, and state machine diagrams are the three options that SysML offers you to specify system behavior.

4.4 Activity Diagram Frame

The diagram kind abbreviation for an activity diagram is act. The only allowable model element type for an activity diagram is activity.

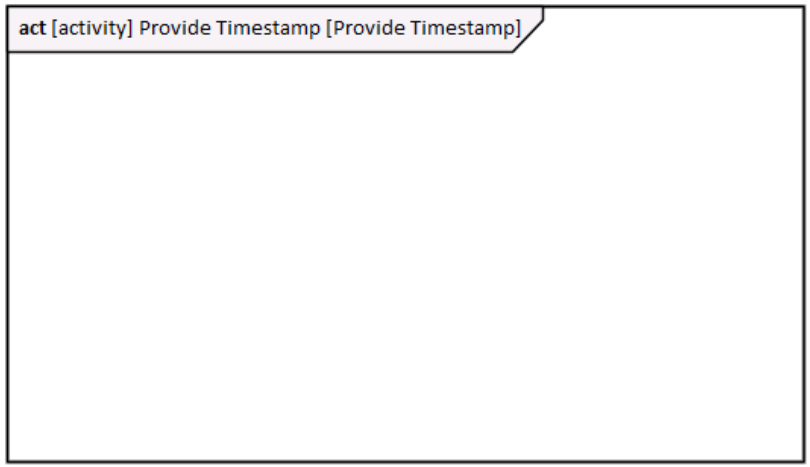


Figure 1: Activity Diagram Frame

5 Activity

An activity is itself a model element; it's a kind of behavior. It contain a set of named elements-nodes and edges/connectors-within the model hierarchy.

NOTE: An activity and activity diagram are not synonyms. When we use the term activity, we are referring to a model element and not its associated diagram. A diagram of the model is never the model itself; it is merely one view of the model.

5.1 Actions

An action is one kind of node that can exist within an activity; it's a node that models a basic unit of functionality within the activity. You connect the actions in an activity by using edges that define ordered (and sometimes concurrent) sequences.

5.1.1 Purpose

An activity diagram convey more than sequences of actions; it can also convey the flow of objects—the inputs and outputs of those actions and of the activity as a whole.

5.1.2 Notation

The notation for a basic action is a round-cornered rectangle (colloquially called a round-angle).

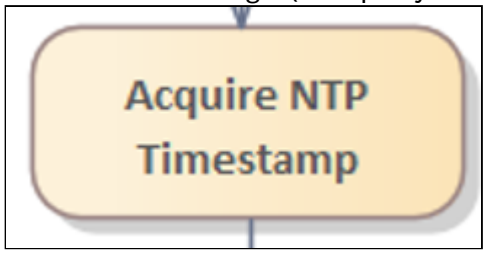


Figure 2: Action(atomic)

5.1.3 Rule

Always write an action as a phrase that begins with a strong, unambiguous verb.

5.2 Object Nodes

An object node, another kind of node that can exist within an activity, models the flow of object tokens through an activity (where each object token, again, represents an instance of matter, energy, or data). An object node most often appears between two actions to convey that the first action produces object tokens as outputs, and the second action consumes those object tokens as inputs.

5.2.1 Notation

The notation for an object node is a rectangle.

<object node name> : <type> [<multiplicity>]

- *object node name* --> modeler defined

- *type* --> match the name of a block, value type, or signal that defined somewhere in model hierarchy; it specifies the nature of the object tokens that the object node can hold
- *multiplicity* --> how many object tokens the object node can hold at any given moment during an execution of the activity. The default multiplicity for an object node is 1..1

5.2.2 Object Node Types

There are 2 types of Object Nodes: Pins and Activity parameters

5.2.2.1 Pins

A pin is a specialized kind of object node. We often refer as Action Pin. You attach a pin to an action to represent an input or output of the action.

5.2.2.1.1 Notation

The notation for a pin is a small square attached to the boundary on the outside of an action.

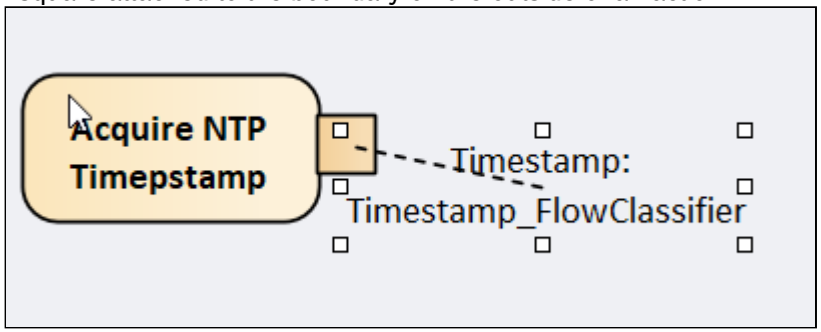


Figure 3: Action Pin

5.2.2.1.2 Optional Input and Output Pins

Below Figure shows an example of an output pin, each having a lower multiplicity of zero. This is how you would model an action that has optional inputs or outputs.

5.2.2.1.3 Notation

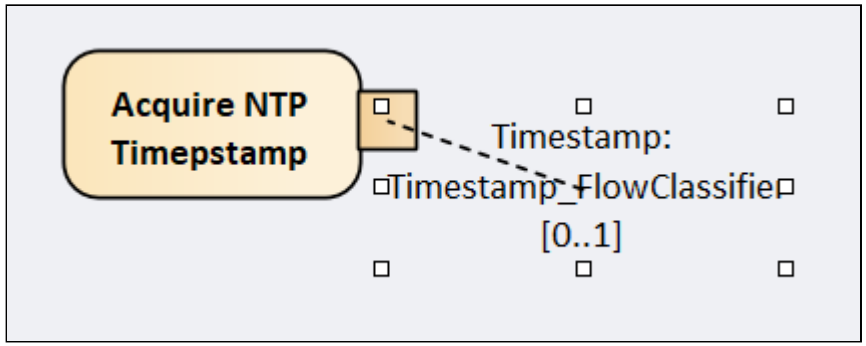


Figure 4: Action Pin(optional)

NOTE:

- An action with an optional input pin can start even with no object tokens at that pin.
- An action with an optional output pin can execute and possibly produce no object tokens at that pin.

5.2.2.2 Activity Parameters

An activity parameter is another specialized kind of object node. It has to be attached to the frame of an activity diagram to represent an input or an output of the activity as a whole.

5.2.2.2.1 Notation

The notation for an activity parameter is a rectangle straddling the frame of an activity diagram. The format of the name string is the same for an activity parameter as for an object node (and a pin).

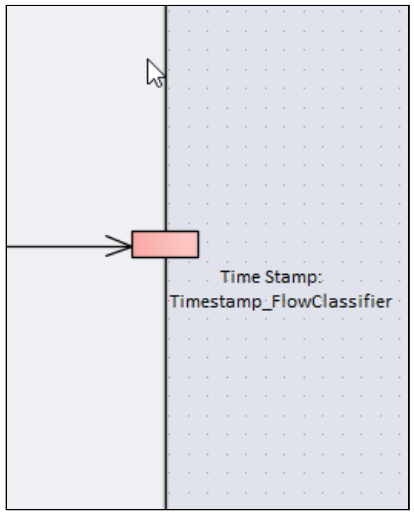


Figure 5: Activity parameter

5.2.2.2.2 Rule

A common modeling practice is to place input activity parameters either on the top or left side of the frame and output activity parameters either on the bottom or right side of the frame. In fact, the only definitive way to tell the difference is the direction of the edge that's attached to an activity parameter.

5.2.2.3 Optional parameter

Like a pin, an activity parameter can have a lower multiplicity of zero. This is how you would model an optional parameter for the activity as a whole.

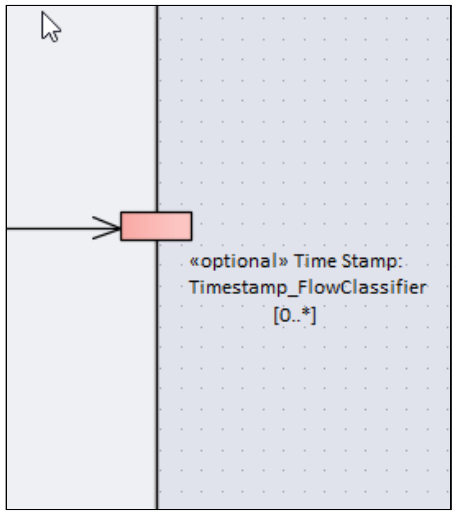


Figure 6: Activity parameter(optional)

5.2.2.3.1 Rule

SysML requires you to apply the «*optional*» stereotype to an activity parameter (preceding the name) when its lower multiplicity is zero.

5.3 Edges

There are two kinds of edges that can be used to connect nodes to form ordered sequences in an activity: object flows and control flows.

5.3.1 Object Flows

Object flows Are used to convey that instances of matter, energy, or data flow through an activity from one node to another when the activity executes during system operation.

5.3.1.1 Notation

The notation for an object flow is a solid line with an open arrowhead.

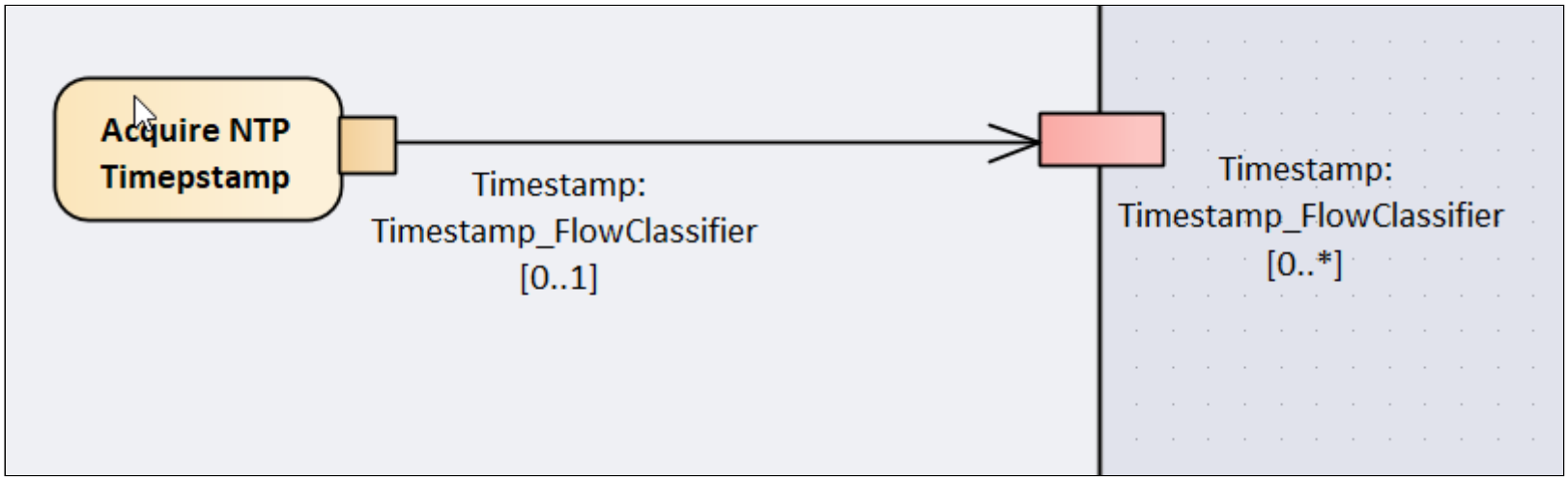


Figure 7: Edges

5.3.1.2 Rule

We must ensure that the object nodes at the ends of an object flow have compatible types; the object token produced as an output on the tail end must be acceptable as an input on the arrowhead end.
We can satisfy this constraint in one of two ways:

- The types can be identical(as shown in Figure)
- The upstream type can be a subtype of the downstream type(as shown in below Figure)

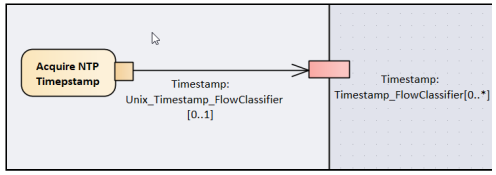


Figure 8.1: Edges (Object Flow from subtype to supertype)

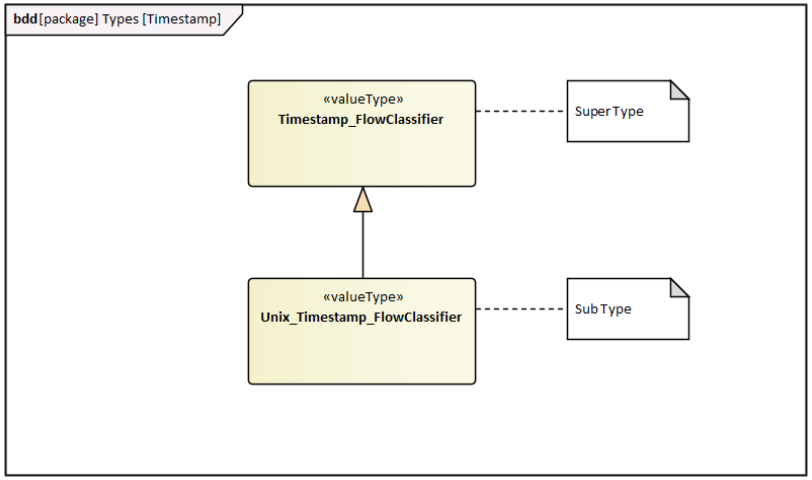


Figure 8.2: Type Definition(subtype, supertype)

NOTE:

- Object Flow from super type to sub type is **prohibited**.
- In addition to an object node, though, you can have a decision node, merge node, fork node, or join node at one end of an object flow to direct the flow of the object tokens.

5.3.2 Control Flows

A control flow is the kind of edge that transports control tokens. And the arrival of a control token enables an action that's waiting for one.
When one action completes, it offers a control token on its outgoing control flow, and that enables the next action in the sequence to begin.

5.3.2.1 Purpose

Use control flows to convey sequencing constraints among a set of actions when the object flows in your activity do not by themselves convey the intended sequence.

5.3.2.2 Notation

A solid line with an open arrowhead.

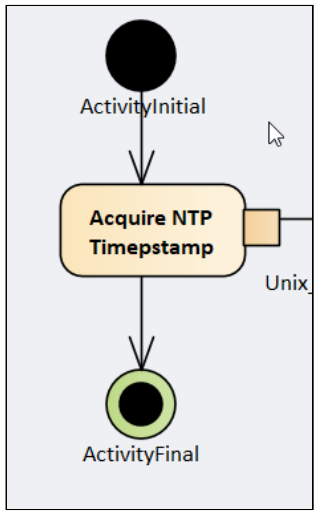


Figure 8: Control Flow

5.4 Action Execution or Start

Three conditions must be satisfied for an action to start:

- The activity that owns the action is currently executing.
- A control token arrives on each of the incoming control flows.
- A sufficient number of object tokens arrive on each of the incoming object flows to satisfy the lower multiplicity of the respective input pin.

5.5 Specialized Actions

There are four specialized kinds of actions: call behavior actions, send signal actions, accept event actions, and wait time actions.

5.5.1 Call Behavior Actions

A call behavior action is a specialized action that invokes another behavior when it becomes enabled. Call behavior actions let you decompose a higher-level behavior into a set of lower-level behaviors.
The behavior that it calls can be any one of the three kinds: an interaction, a state machine, or another activity.

5.5.1.1 Notation

The notation for a call behavior action is the same as the one for an action—a rectangle with rounded corners—except that the name string inside has a particular format:

<action name> : <Behavior Name>

- *action name* --> modeler defined

- *Behavior name* --> must match the name of an interaction, a state machine, or an activity that you've defined somewhere in your model hierarchy.

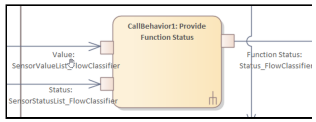


Figure 9.1: Call Behavior Action(activity)

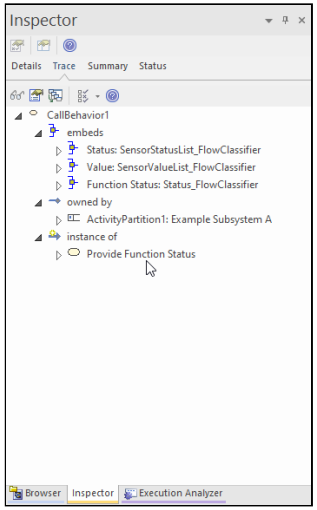


Figure 9.2: Inspector Trace view of Call Behavior Action(activity)

A rake symbol appears in the lower right corner of a call behavior action when it has call behavior as activity or interaction or state machine.
As shown in Figure a call behavior action such as *CallBehavior1*, it conveys that the behavior getting called *Provide Function Status* is an activity.

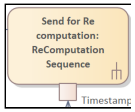


Figure 10.1: Call Behavior Action(Sequence Diagram)

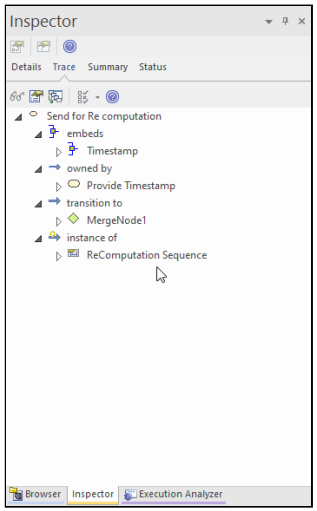


Figure 10.2: Inspector Trace view of Call Behavior Action(Sequence Diagram)

As shown in Figure a call behavior action such as *Send for Re computation*, it conveys that the behavior getting called *ReComputation Sequence* is a sequence diagram(interaction).

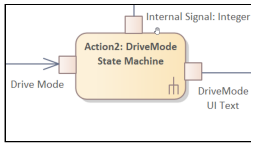


Figure 11.1: Call Behavior Action(State Machine Diagram)

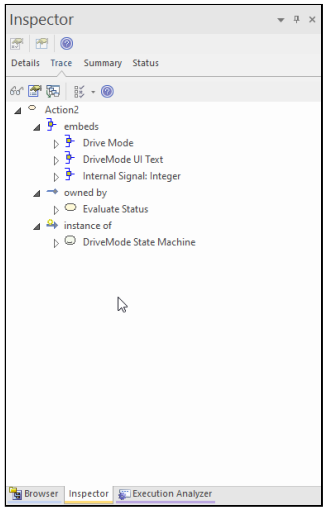


Figure 11.2: Inspector Trace view of Call Behavior Action(State Machine Diagram)

As shown in Figure a call behavior action such as *Send for Re computation*, it conveys that the behavior getting called *ReComputation Sequence* is a sequence diagram.

5.5.1.2 Rule

When a call behavior action invokes another activity, the pins of the call behavior action must match the activity parameters of the called activity.

5.5.2 Send Signal Actions

This is a special kind of action that asynchronously generates and sends a signal instance to a target when it becomes enabled as mentioned in Action Execution or Start

5.5.2.1 Notation

The notation for a send signal action is a convex pentagon shaped like a signpost.

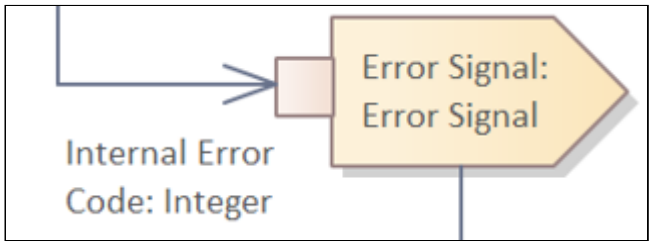


Figure 12: Send Signal Action

5.5.2.2 Rule

- The string displayed inside a send signal action (e.g., *Error Signal*) should match the name of a signal that had been defined somewhere in the model hierarchy.
- The send signal action doesn't wait for a response from the target, instead it completes immediately and offers control token on its outgoing edge as it is asynchronous.

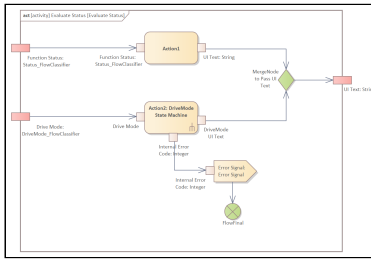


Figure 13.1: Send Signal Action Usage

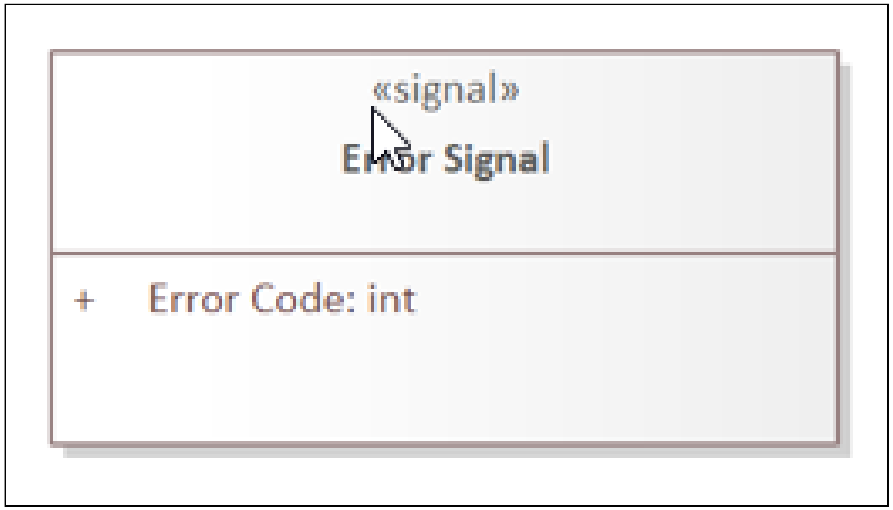


Figure 13.2: Signal Definition in Model(*Error Signal*)

5.5.2.3 Mechanism

The Signal(*Error signal)* owns a property called *Error Code* of type *integer.* An instance of *Error Code* will be transferred from sender to target as soon as Input pin (*Internal Error Code*) arrives.

5.5.3 Accept Event Actions

An accept event action is the element that is used in an activity to convey that the activity must wait for an asynchronous event occurrence before it can continue its execution.
Typically this action is on the receiving side of asynchronous signal instance sent by send signal action which is considered as event occurrence.

NOTE: An accept event action is not limited to receiving signal instances (from send signal actions); it can also receive asynchronous time event occurrences (for details, see Wait Time Actions).

5.5.3.1 Notation

The notation for an accept event action is a concave pentagon that looks like a rectangle with a triangular notch cut out on one side.

Figure 14: Accept Event Action

5.5.3.2 Mechanism

An accept event action with no incoming edges becomes enabled and begins listening for a signal instance as soon as the activity begins executing. When it does, the accept event action completes, and the control token will be given to the next node in the activity.

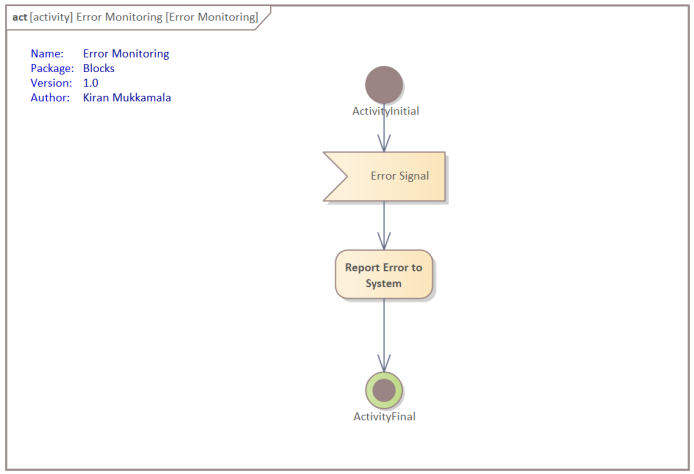


Figure 15: Accept Event Action Usage

5.5.3.3 Rule

- The string displayed inside an accept event action (e.g., *Error Signal*) often matches the name of a signal that had been defined somewhere in your model hierarchy.
- An accept event action with no incoming edges remains enabled even after the first signal instance arrives, and the accept event action continues to listen for additional instances. This is one way to model a system behavior that continually responds to asynchronous event occurrences.
- The accept event action that receives a signal instance may appear in the same activity as the send signal action that generates it. Or it may instead appear in a separate activity; in this way, you can model asynchronous communication between two distinct system behaviors.

5.5.4 Wait Time Actions

An accept event action that waits for a **time** event occurrence is called a wait time action.

5.5.4.1 Notation

The notation for a wait time action is a stylized hourglass symbol with a time expression string beneath it

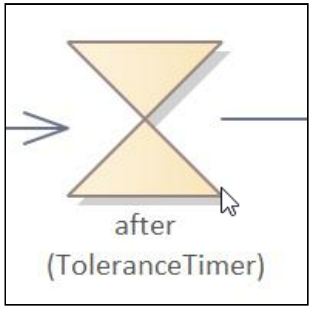


Figure 16: Wait Time Action

The time expression beneath the hourglass can specify either an **absolute time event** or a **relative time event**.

- An absolute time event expression begins with the keyword at—for example, at (1430 GMT) or at (14 NOV 2106, 1200 CST) or at (*Tolerance Time*).
- A relative time event expression begins with the keyword after, as in after (30 days) or after (50 ms) or after (*timerCount*).

5.5.4.2 Mechanism

The wait time action becomes enabled when a control token arrives on its incoming control flow upon completion of the upstream action.
If the relative time event specified by *ToleranceTimer,* the clock for the time event begins, the relative time event occurs after the timer elapsed, and the wait time action offers a control token on its outgoing control flow; then execution proceeds to the next node. If *ToleranceTimer* has not elapsed, then the wait time action simply waits for that time event to occur.

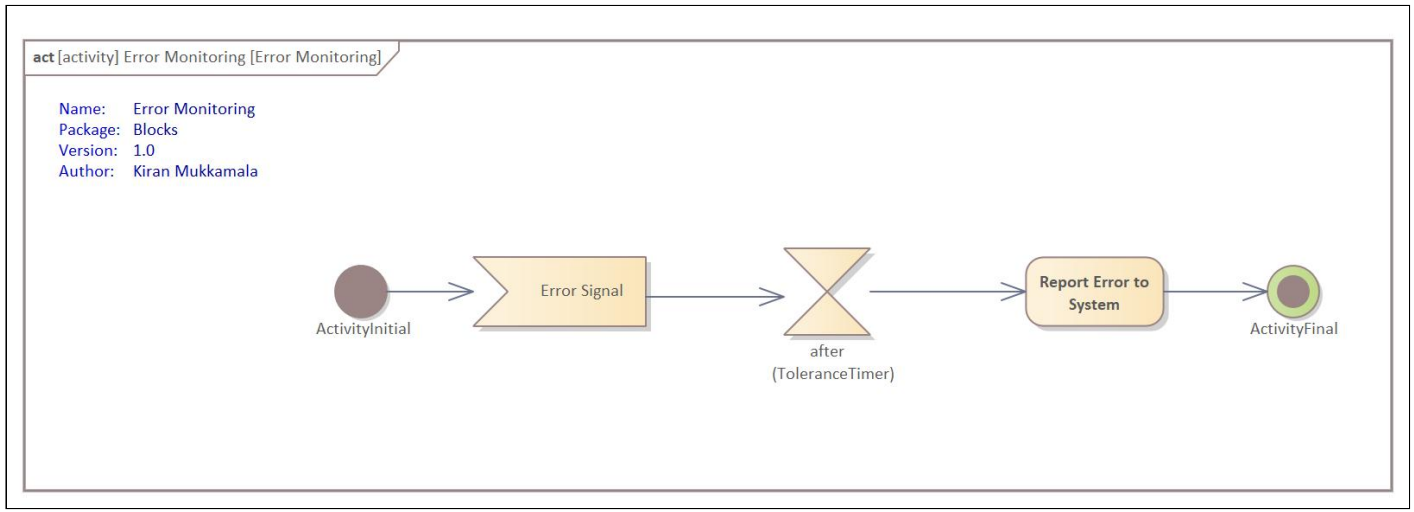


Figure 17: Wait Time Action Usage

5.6 Control Nodes

Control nodes can direct the flow of control tokens as well as object tokens within an activity.
There are seven kinds of control nodes: initial nodes, activity final nodes, flow final nodes, decision nodes, merge nodes, fork nodes, and join nodes.

5.6.1 Initial Nodes

An initial node marks a place in the activity where the flow of a control token begins.

5.6.1.1 Notation

The notation for an initial node is a small, filled-in circle.
An initial node generally has exactly one outgoing control flow.

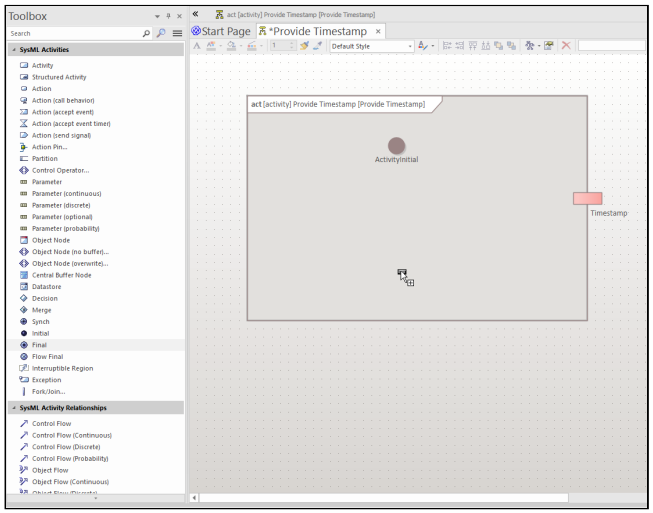


Figure 18: Activity Initial Node

5.6.1.2 Rule

An activity **need not** have an **initial node**. The flow of a control token can begin instead at an action with no incoming edges.
Recall that such actions start as soon as the activity begins executing. It's also possible that the flow of object tokens alone will be sufficient to define the correct sequences of actions within an activity. Object tokens generally begin at input activity parameters (on the frame of an activity diagram). In such cases, you **don't need an initial node** in your activity.

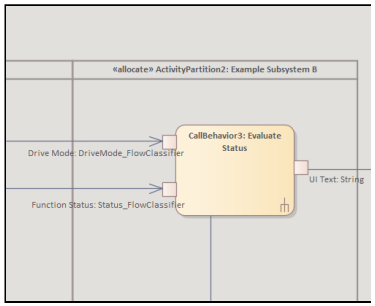


Figure 19.1: Evaluate Status is being called with activity parameters

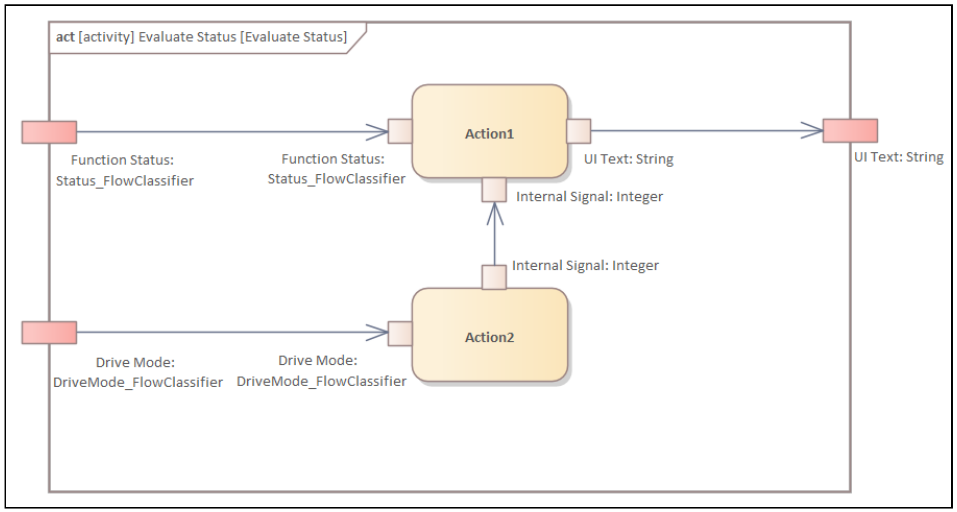


Figure 19.2: Activity without Initial Node

5.6.2 Flow Final Nodes and Activity Final Nodes

Flow final nodes and activity final nodes are control nodes that mark the end of the flow of a control token.
When a control token arrives at a flow final node, that token is destroyed, marking the end of a single flow of control.
When a control token arrives at an activity final node, the entire activity terminates, marking the end of all flows of control (no matter where they are currently in their execution).

5.6.2.1 Notation

The notation for a flow final node is a circle containing an X.
The notation for an activity final node is a circle that contains a smaller, filled-in circle.

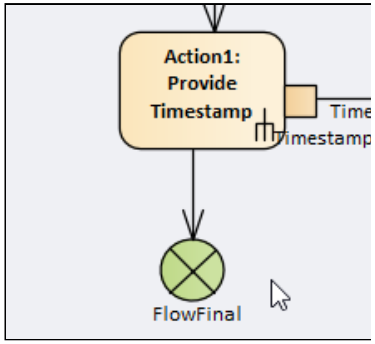


Figure 20.1: Flow Final Node

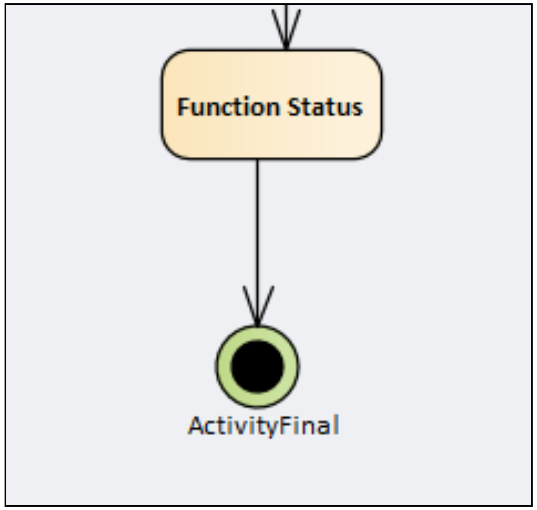


Figure 20.2: Activity Final Node

NOTE:

You're allowed to add multiple activity final nodes to an activity. However, you should be careful to avoid inadvertently modeling a race condition between concurrent flows of control; the first control token that arrives at any activity final node will terminate the entire activity. You should **use flow final nodes** instead of **activity final nodes** to avoid modeling a **race condition**.

5.6.3 Decision node

A decision node marks the start of alternative sequences in an activity.

5.6.3.1 Notation

The notation is a hollow diamond.

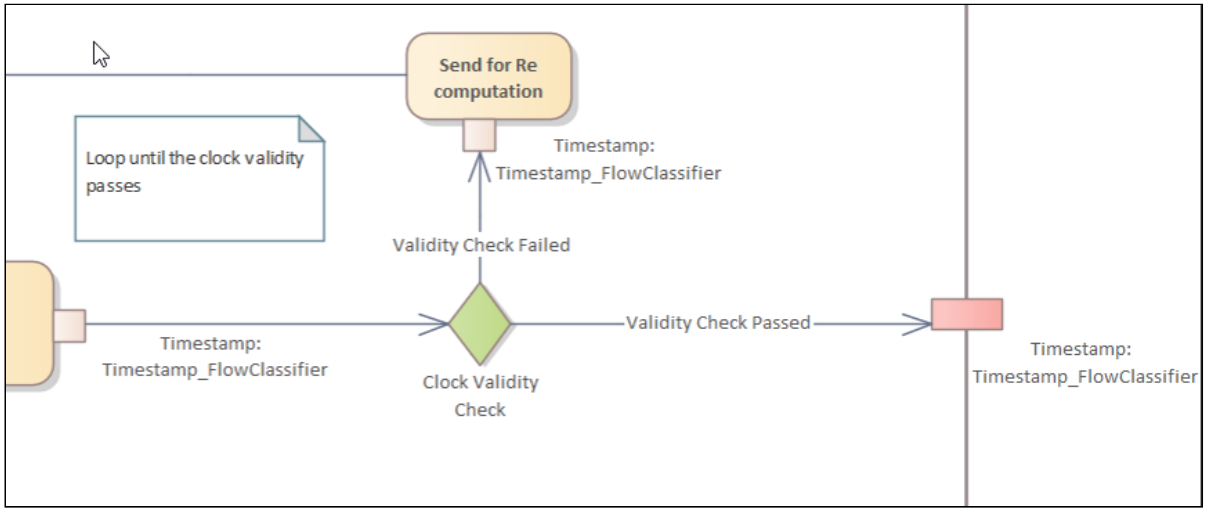


Figure 21: Decision Node

5.6.3.2 Rules

- A decision node must have a **single incoming** edge and generally has two or more outgoing edges. Each outgoing edge is labeled with a Boolean expression called a **guard**, which is displayed as a string between square brackets.
- If the Decision node has only one incoming edge, then it is the primary incoming edge. If the **primary incoming edge** of a Decision node is a **Control Flow**, then all outgoing edges shall be **Control Flows** and, if the **primary incoming edge** is an **Object Flow**, then all outgoing edges shall be **Object Flows**.

5.6.3.3 Mechanism

When a token either an object token or a control token arrives at a decision node, the guards on the outgoing edges are evaluated. The token is offered to the outgoing edge whose guard evaluates to true at that moment.

5.6.4 Merge node

A merge node marks the end of alternative sequences in an activity.

5.6.4.1 Notation

The notation is a hollow diamond, which is same as decision node.

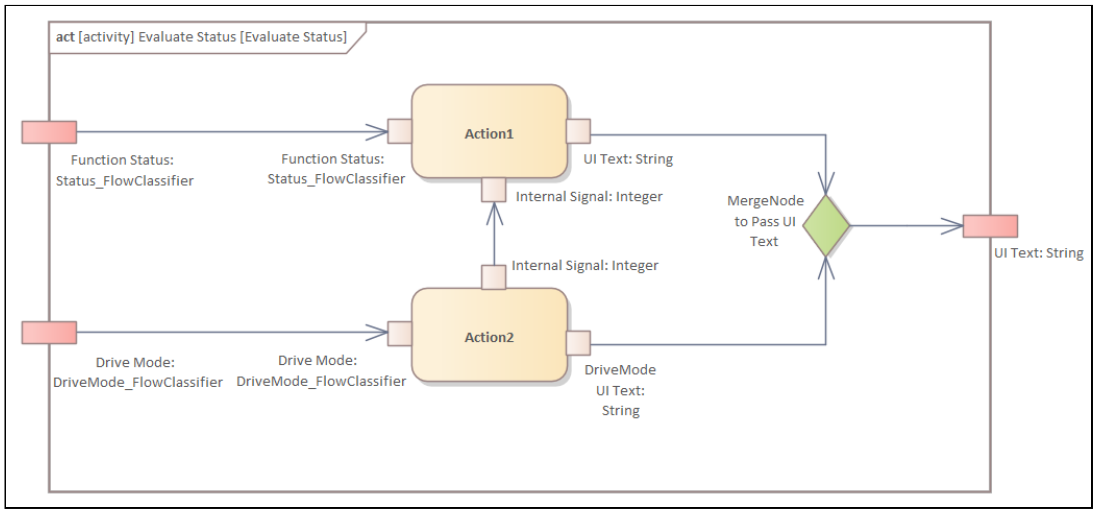


Figure 22: Merge Node

5.6.4.2 Rule

- A merge node has two or more incoming edges and a **single outgoing** edge.
- There is no joining of control tokens. Hence Merge **should not** be used to **synchronize concurrent flows**.

5.6.4.3 Mechanism

When a token either an object token or a control token arrives at a merge node via any of its incoming edges, the token is immediately offered to the outgoing edge.

5.6.4.4 Additional Usage

Often when we need to model a loop or repetitive action/behavior, it is recommended to use merge node in conjunction with a decision node along with it. A merge node is, in fact, essential for modeling a loop.

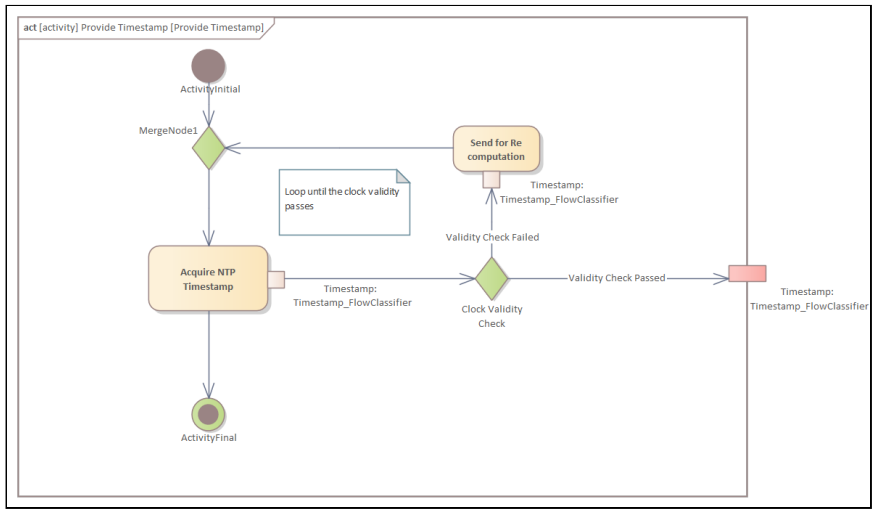


Figure 23: Merge Node in loop

5.6.5 Fork node

A fork node marks the start of concurrent sequences in an activity.

5.6.5.1 Notation

The notation for a fork node is a line segment (oriented in any direction you like).

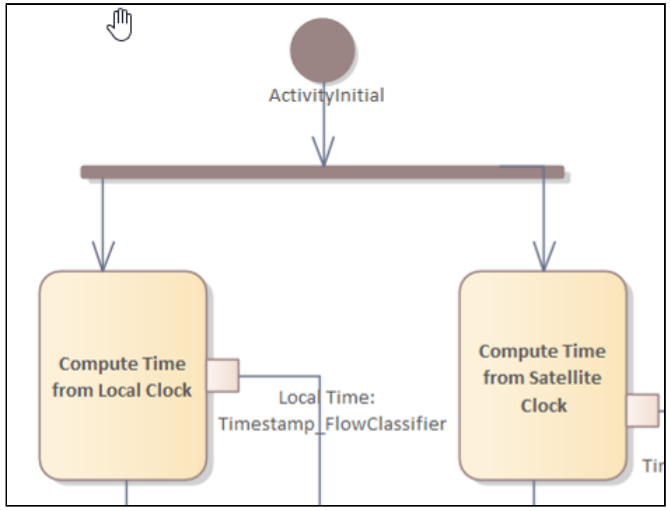


Figure 24: Fork Node

5.6.5.2 Rule

A fork node must have a single incoming edge and two or more outgoing edges.

5.6.5.3 Mechanism

When a token either an object token or a control token arrives at a fork node, it is duplicated on all of the outgoing edges.

5.6.5.4 Usage

Each of those copies of the original token represents an independent, concurrent flow of control along its respective path thus creating a concurrent flow in the model.

5.6.6 Join node

A join node marks the end of concurrent sequences in an activity. Join nodes are introduced to **support parallelism in activities**.

5.6.6.1 Notation

The notation for a Join node is a line segment (oriented in any direction you like).

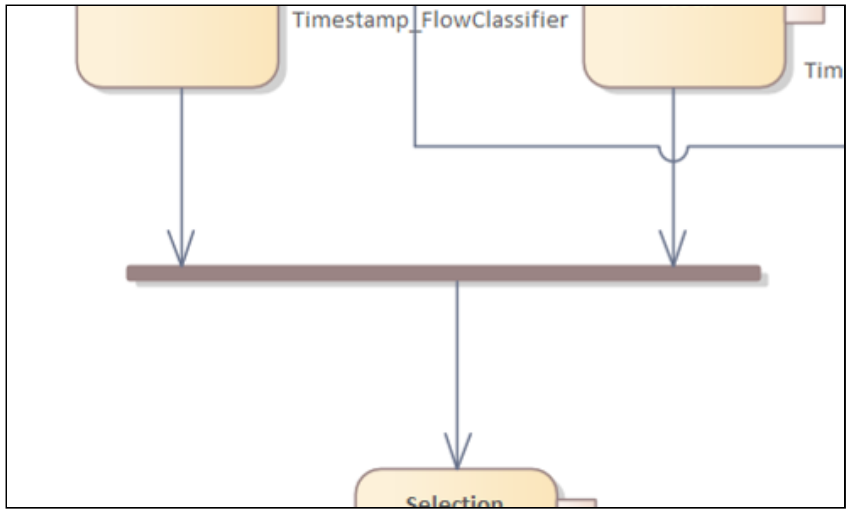


Figure 25: Join Node

5.6.6.2 Rule

A join node generally has two or more incoming edges and a single outgoing edge.

5.6.6.3 Usage

You use a join node to model a synchronization point for concurrent sequences of actions in an activity.

5.6.6.4 Mechanism

When a token arrives on each of the incoming edges, a single token is offered on the outgoing edge. The concurrent sequences end, and a single flow of control proceeds past the point marked by the join node.

NOTE: The presence of the join node at the bottom of the diagram conveys that all the sequences of actions must complete before the activity can come to an end.

5.7 Activity Partitions

5.7.1 Allocating Behaviors to Structures

An Activity Partition is used to group execution elements according to the node responsible for their execution. Most often, an activity partition represents either a block or a part property that exists somewhere in the system model. Simply an Activity partition often referred as behavior.

5.7.2 Notation

The notation for an activity partition is a large rectangle (which contains one or more nodes) with a header at one end, other end is left open. It can be oriented either vertically or horizontally.

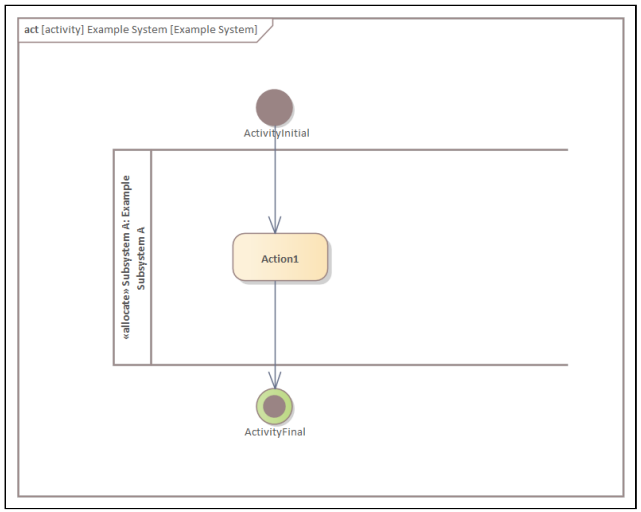


Figure 26: Activity Partition

5.8 KEY POINTS

1. Always think of modularity at each and every level, remember create activities that can be reused in many other places.

2. Drawing multiple incoming edges to an action to convey alternative paths to that action. **Multiple incoming edges** does **not** represents an **or** condition rather it represents an **and** condition.
3. If you need to model alternative paths to a given action, you must insert a merge node preceding the action(refer Merge node section)
4. Read all Notes and Rules in this page carefully to proceed with modelling of activities.

Sources:
(1): OMG Systems Modeling Language, v1.5, Chapter 11 Activities
(2): Sparx Systems Enterprise Architect User Guide: Action Pin
(3): Sparx Systems Enterprise Architect User Guide: Activity Parameter Node
(4): Book: SysML Distilled_ A Brief Guide - Lenny Delligatti