UML Activity Diagrams

Introduction

- The UML activity diagram presents the system's execution flow.
- The activity diagram presents activities composed of one or more small actions.

Activity

- The notation used for activity is a rectangle with rounded corners.
- In the upper left corner we specify the activity name.
- Below the name it is possible to list the involved parameters. It is also possible to use parameter nodes instead (explained later).

Actions

- Within the activity notation it is possible to draw the actions that belong to this activity.
- The notation used for action is the same notation we use for activity.

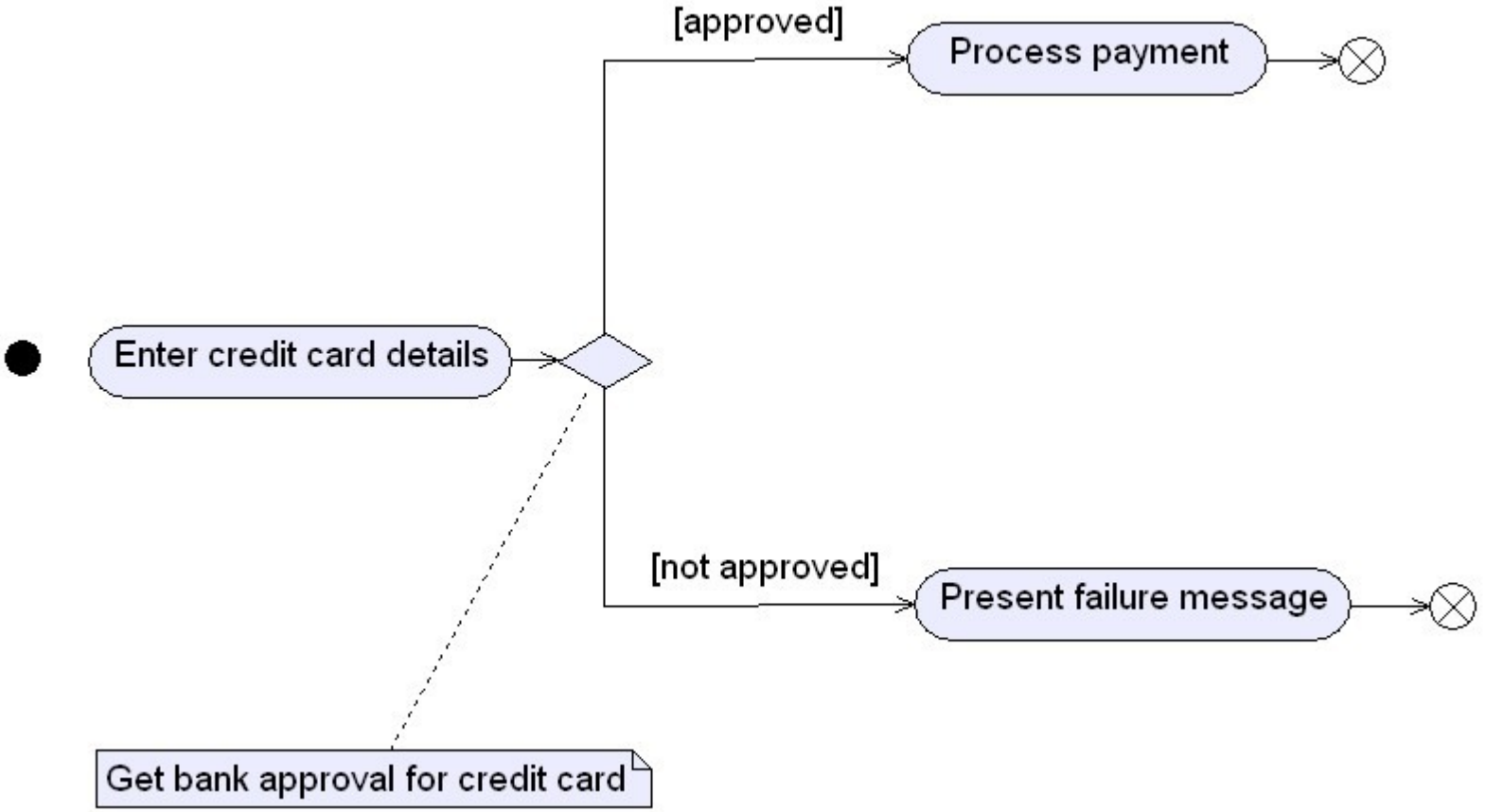
Initial Node

- Each activity starts with the initial node. The initial node notation is a small black circle.

Decision Node

- A decision node chooses an output flow from different available output flows based on its boolean expression (guard condition) value.
- Each decision node has one input edge and multiple output ones.
- The notation is an empty diamond. We put the boolean expression (guard condition) in brackets.
- A note can be added in order to present more info about the decision input.

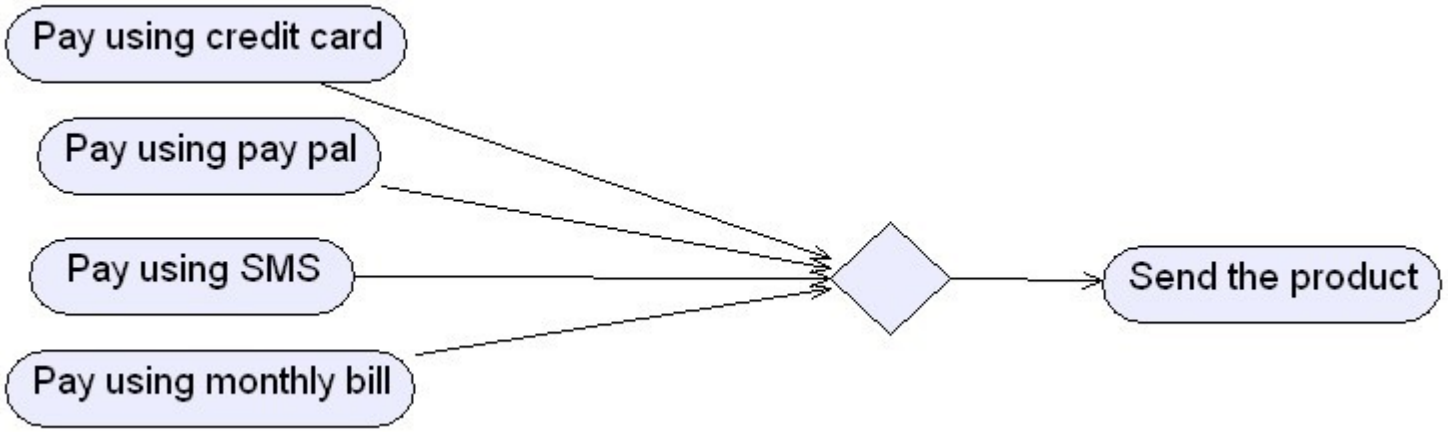
Decision Node



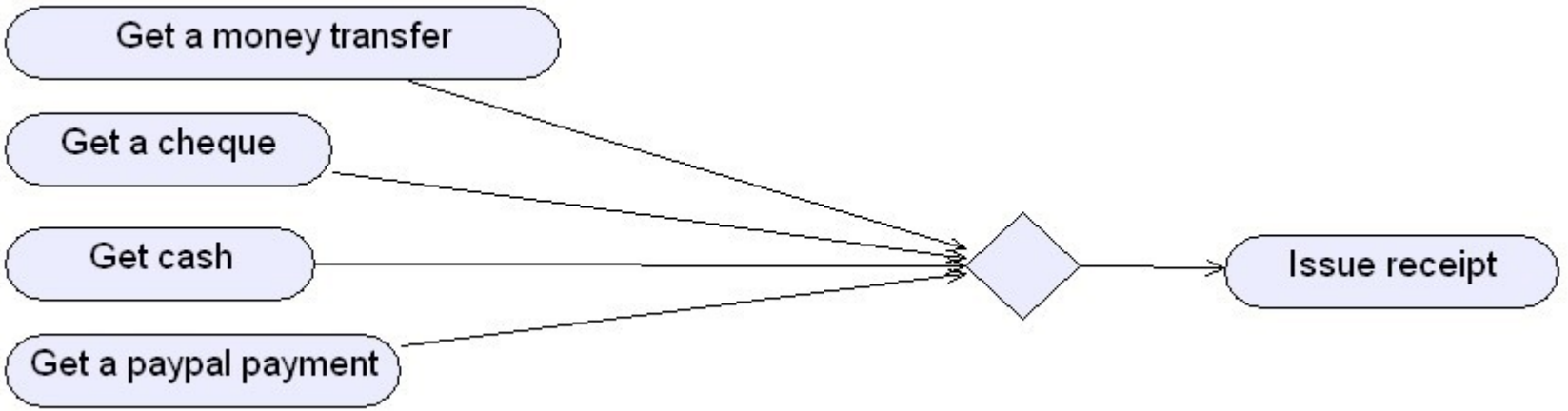
Merge Node

- A merge node unite the flows it receives into one single flow. It has multiple incoming edges and one output edge only.
- The notation for showing a merge node is an empty diamond.

Merge Node



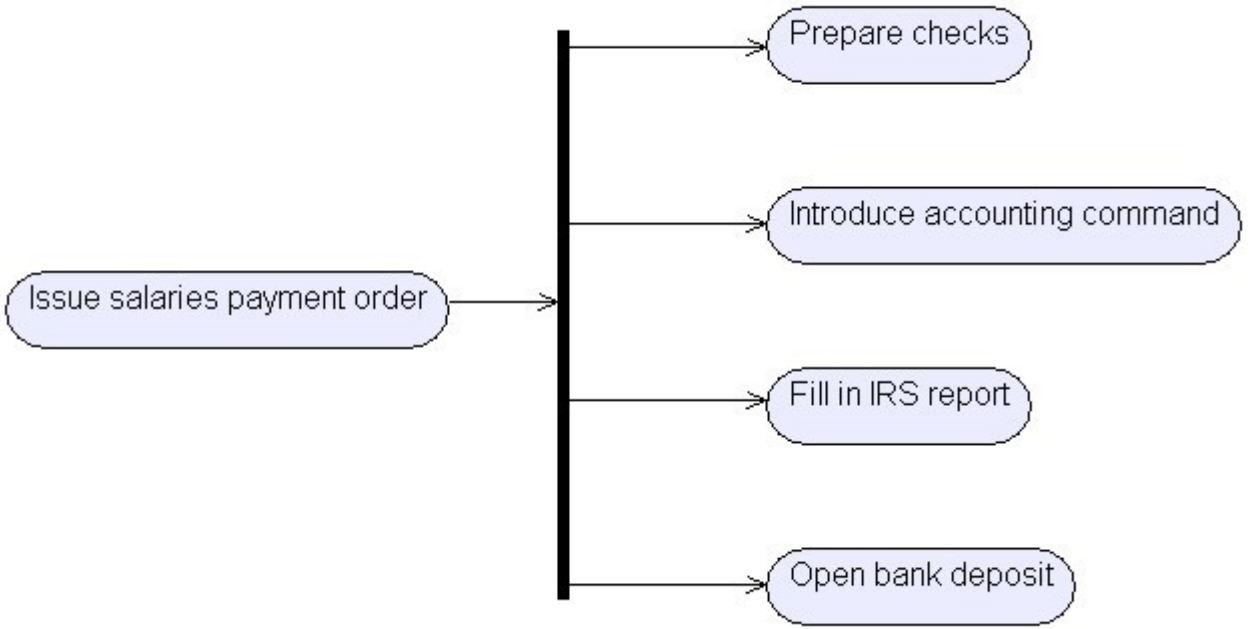
Merge Node



Fork Node

- A fork node split a flow into multiple concurrent flows.
- Unlike the merge node, the fork node duplicates each data item it receives and forward each copy through each one of the outgoing flows.

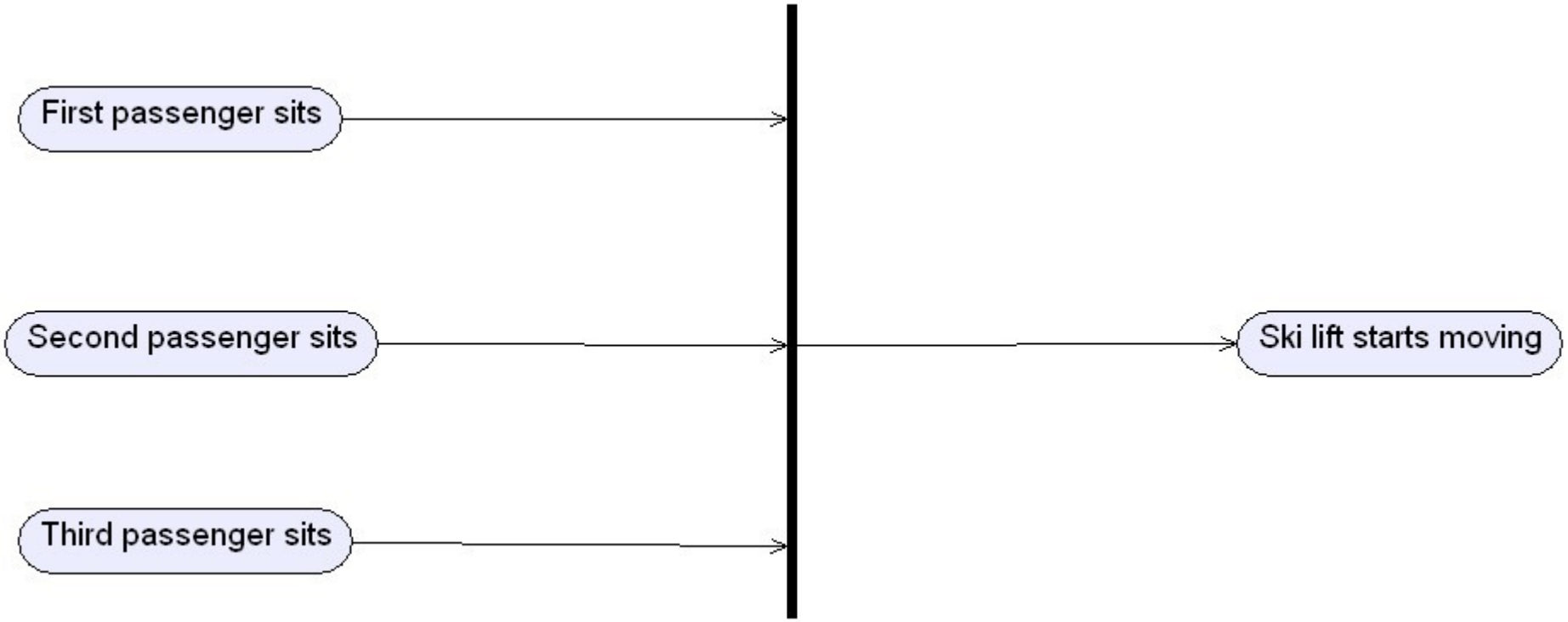
Fork Node



Join Node

- A join node synchronizes multiple flows (edges) back to a single flow (edge).
- The notation used for Join node is the same notation used for the fork node.

Join Node



Join Node

- Near the join node it is possible to write a condition within braces { } in order to inform of a condition that must be true before the join node forward the flow.

Join Node

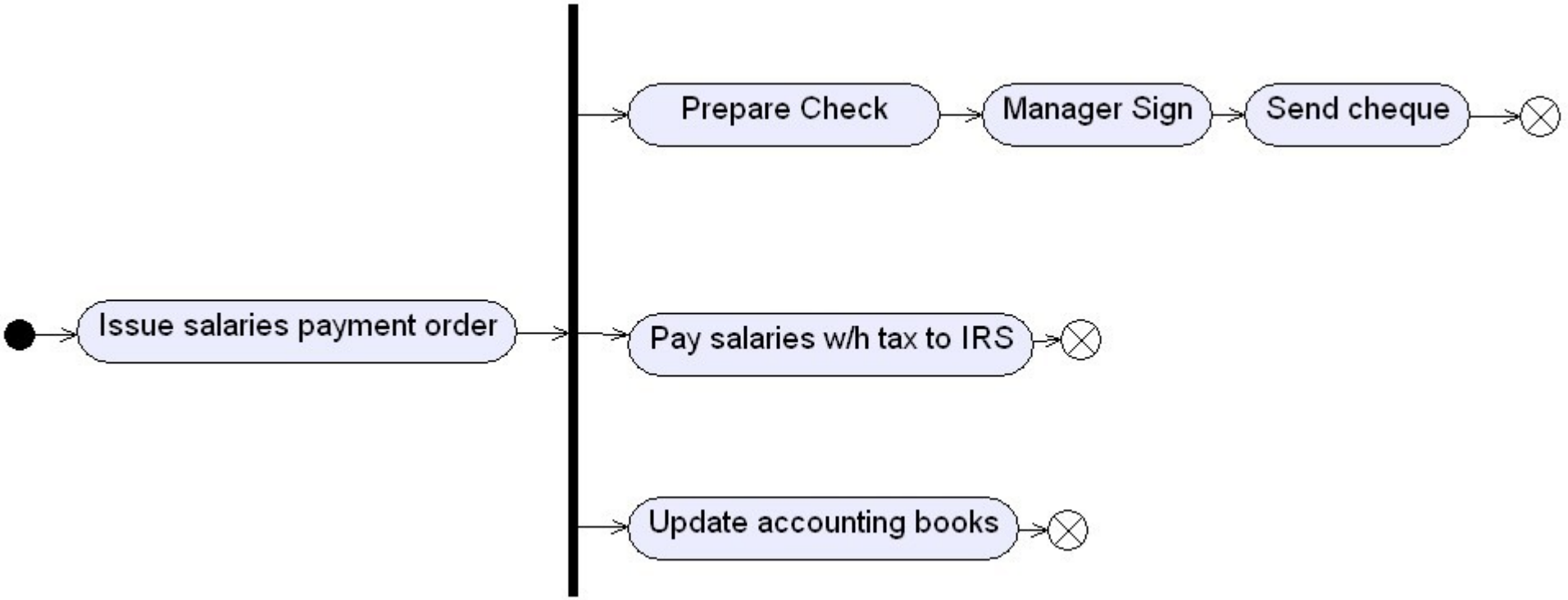
Activity Final Node

- The activity final node is a node that any flow of data that reaches it will cause the termination of the entire application.
- The notation used to draw an activity final node is a black circle with a black ring around it.

Flow Final Node

- The flow final node is a node that terminates a flow only. Data that reaches it won't terminate the application.
- The notation used to draw a flow final node is a black ring with X inside it.

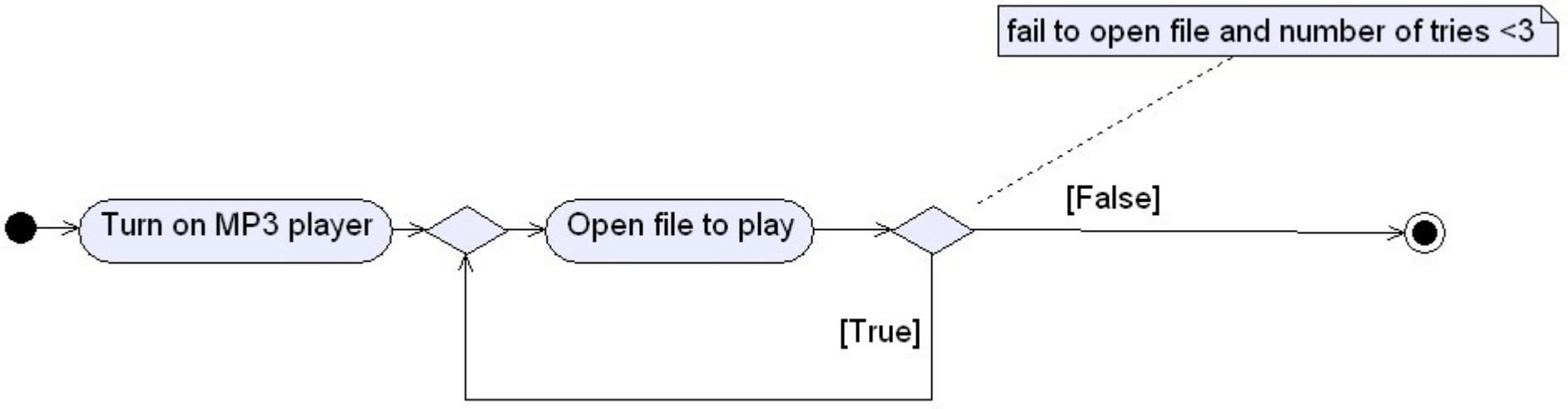
Flow Final Node



Loops

- The loop node usually includes three parts: setup, body & test. The setup part executes only once (when the loop starts) and it usually initializes the loop. The test part is evaluated before the body part of after it.

Loops



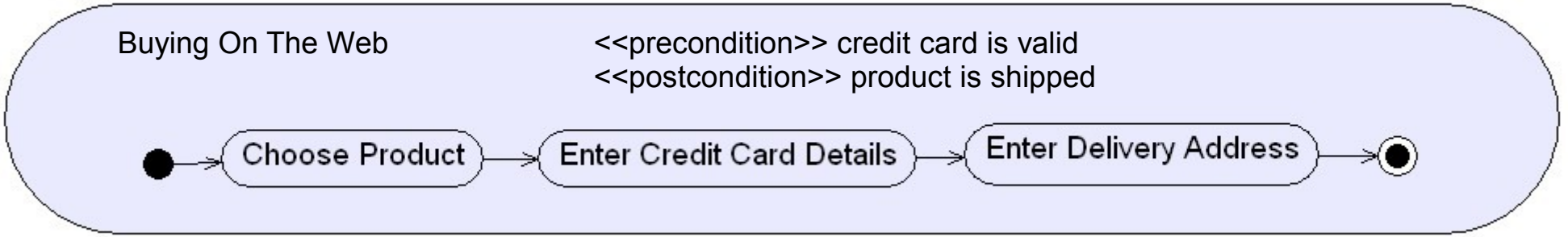
Preconditions & Postconditions

- Each activity can be specified together with its preconditions and postconditions.
- We write the precondition on top center of the diagram using the following format:

  <<precondition>> _____
- We write the postcondition on top center of the diagram (below the precondition) using the following format:

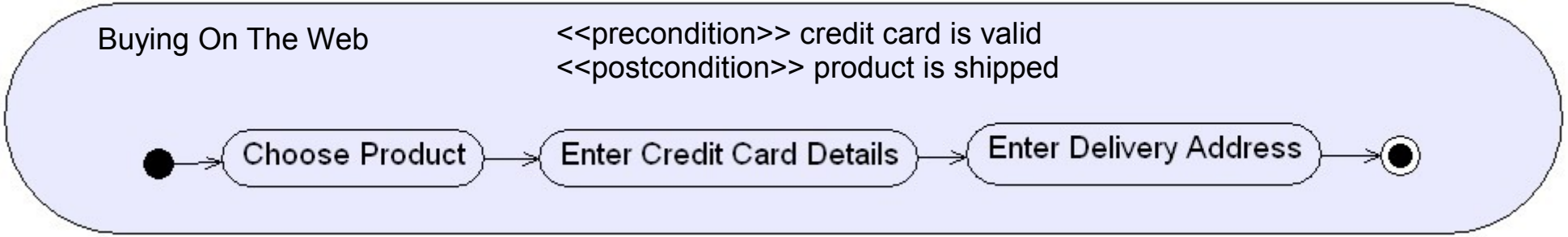
  <<postcondition>> _____

Preconditions & Postconditions



Activities Edges

- The activities edges are lines with open arrows that connect the activities with each other and represent the control and data flows from one activity to the other.

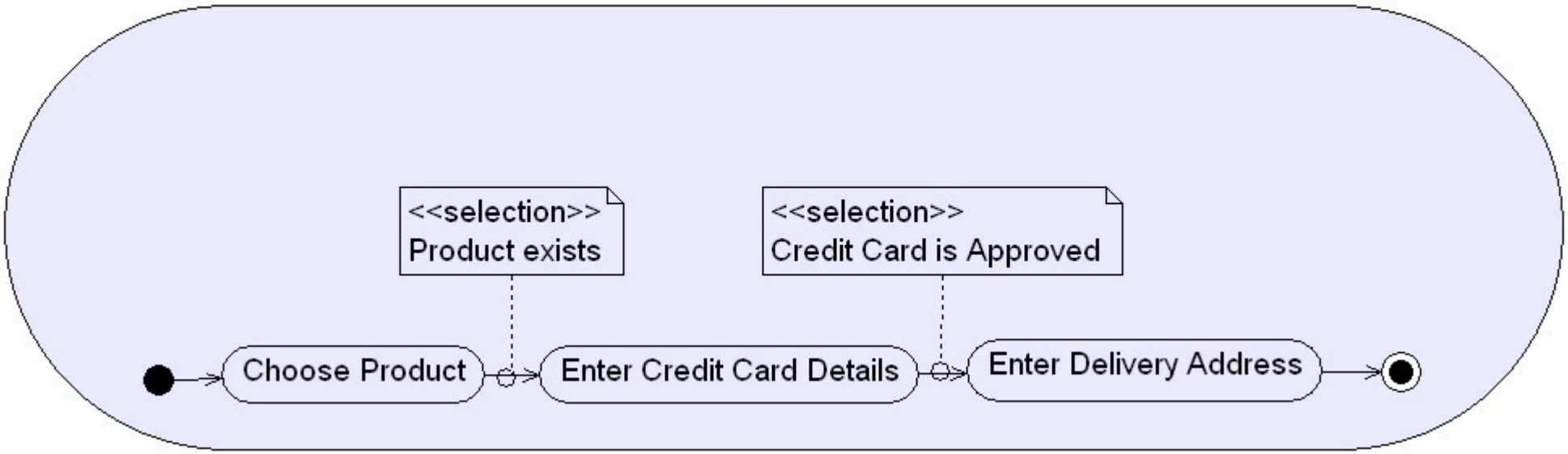


Object Flow Elements

- The object flow element presents a special flow of one of the following possibilities:
  1. A flow through which only objects that meet a selection condition can move forward.
  2. A flow that is multi casted to a multiple instances.
  3. A flow that is multi received from multiple senders.
  4. A flow that transforms the data flow and add specific data to it.
- We use a simple small empty circle placed above the activity edge together with a note in which we write `<<selection>>` or `<<multi cast>>` or `<<multi receive>>` or `<<transforms>>`.

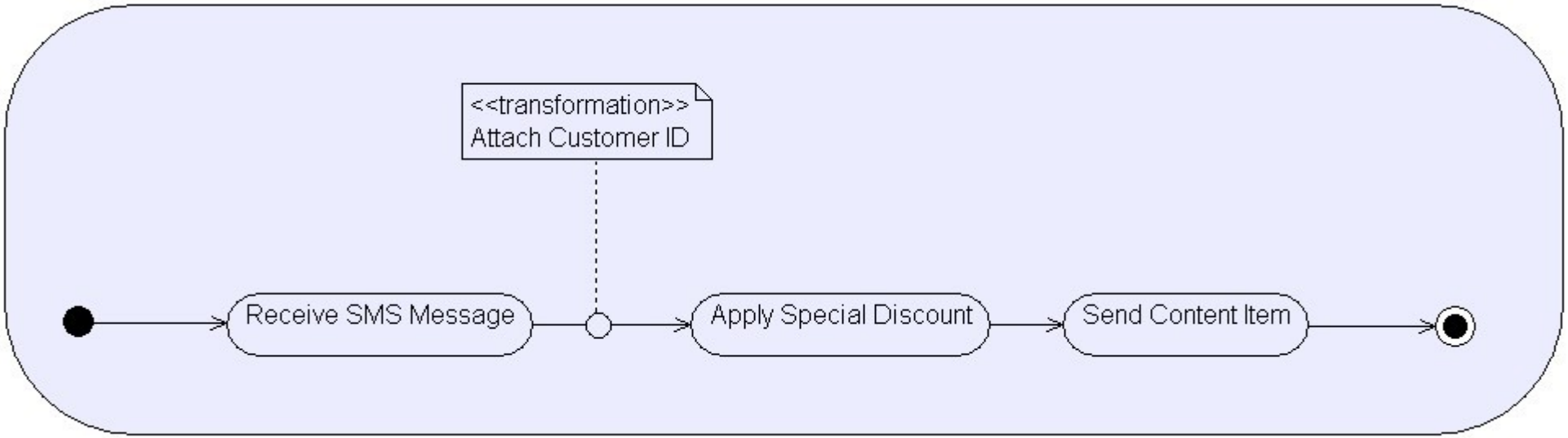
Object Flow Elements

<<selection>>



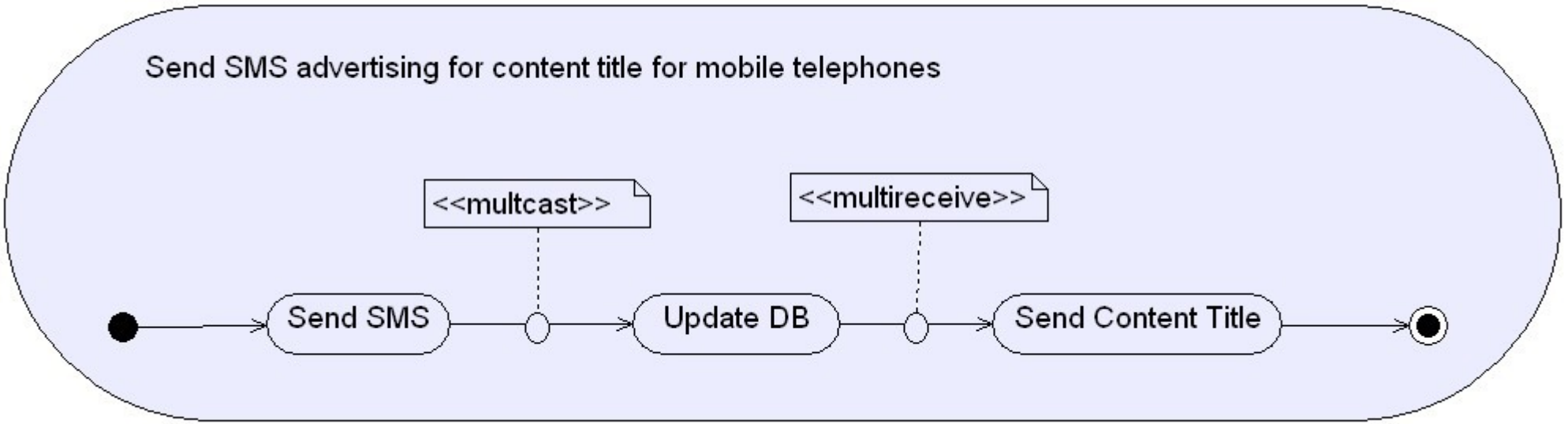
Object Flow Elements

<<transformation>>



Object Flow Elements

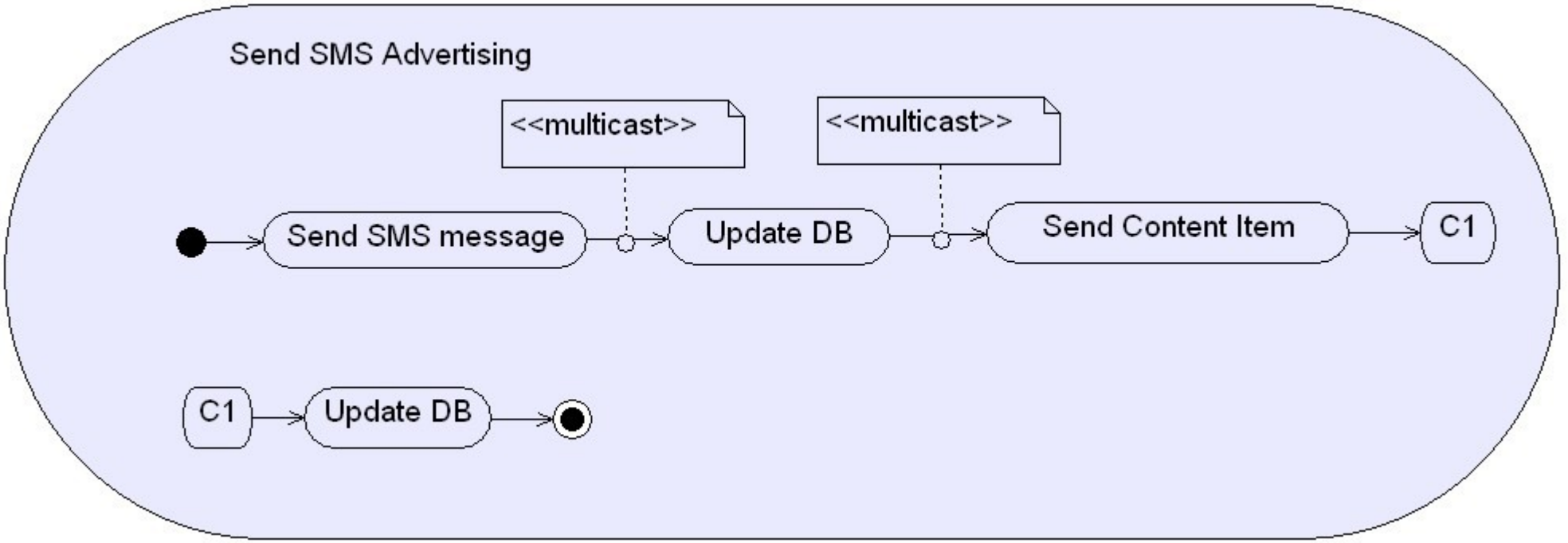
<<multicast>> <<multireceive>>



Connectors

- When dealing with big activity diagrams that can't be captured in one page or when drawing an activity edge is too complex due to a big number of other elements our diagram includes, it is possible to split the activity edge using a connector.
- The connector notation is a simple circle within we write its name.

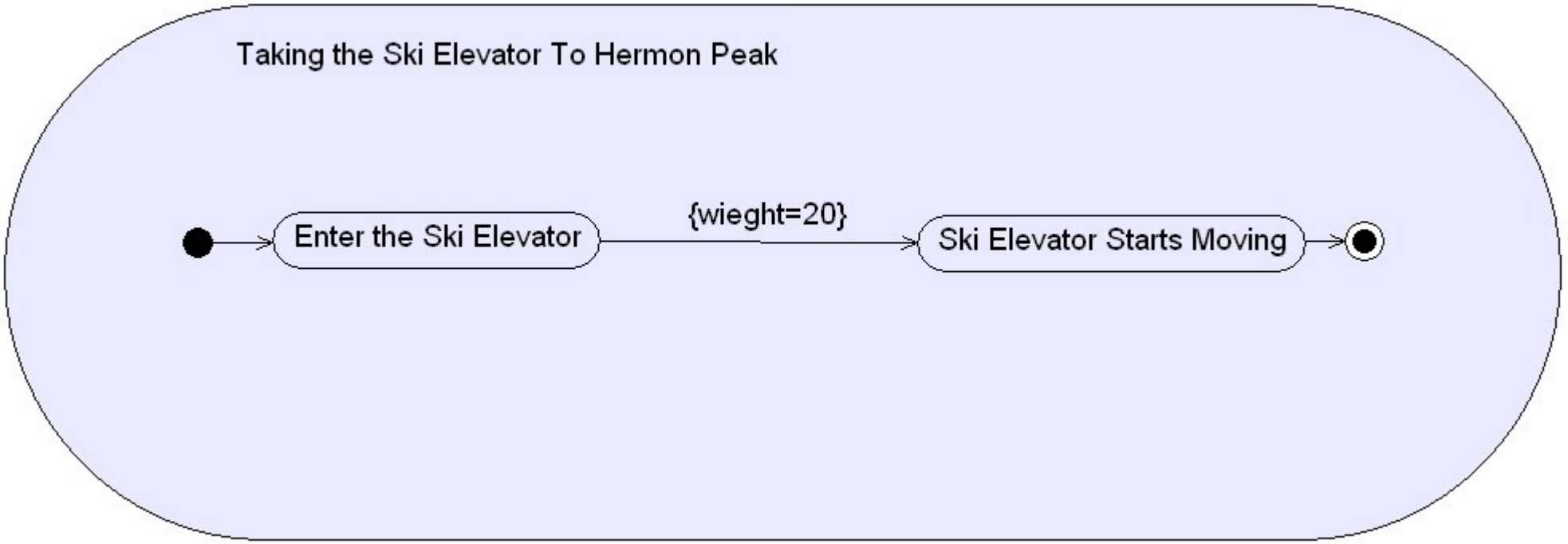
Connectors



Tokens & Weight

- Conceptually along the activities edges the flowing data can be perceived as tokens.
- Conceptually, each action that is performed has data input and data output that can be represented as tokens.
- We can represent the number of tokens that should be made available for a given action before it can be executed by writing `{weight=n}` above the relevant activity edge, while `n` is the number of the required tokens.

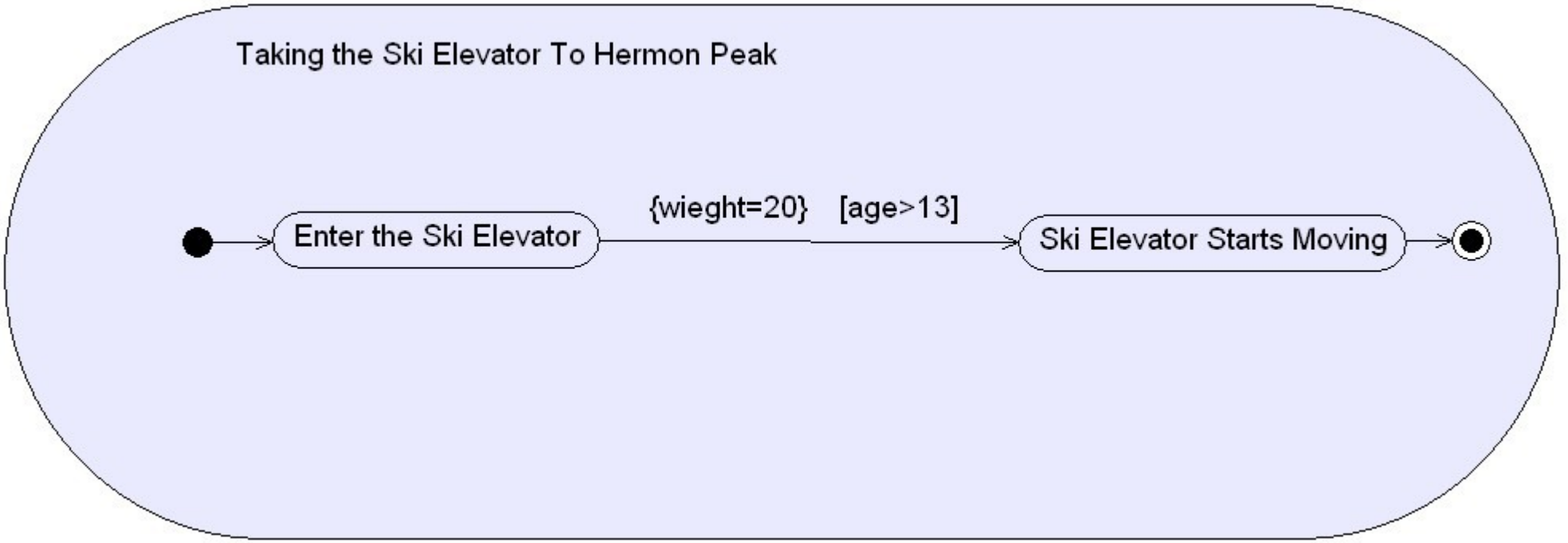
Tokens & Weight



Activity Edge Guard Condition

- On top of each activity edge we can add a guard condition that will be checked the moment the required weight is met.
- If one (or more) of the tokens fail(s) to pass the guard condition and the total number of tokens that pass doesn't meet the required weight then all tokens that passed will be held until the weight requirement is met.
- The notation for a guard condition includes a boolean expression within square brackets we draw near the relevant activity edge.

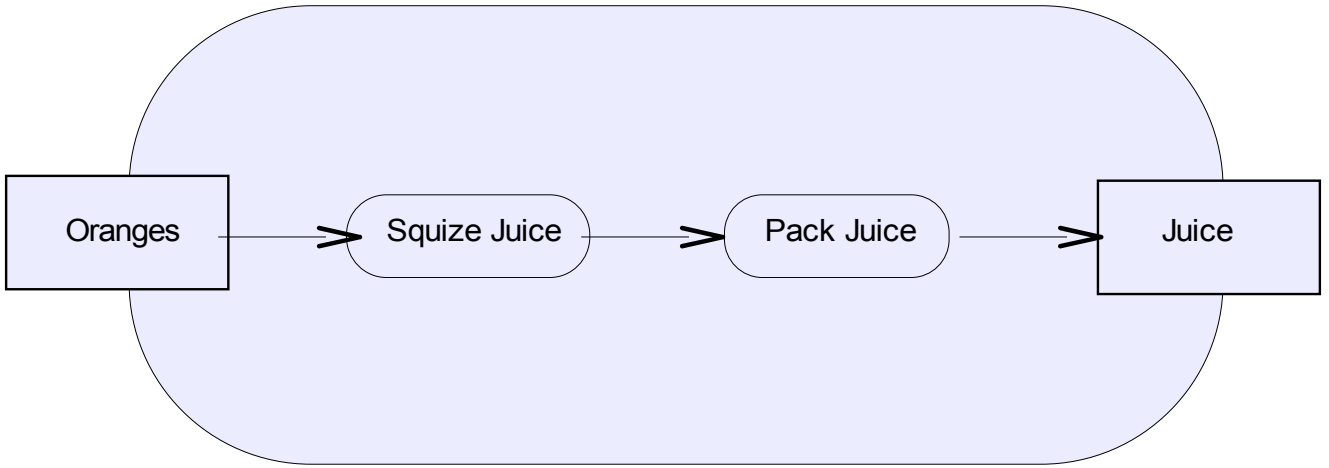
Activity Edge Guard Condition



Parameter Node

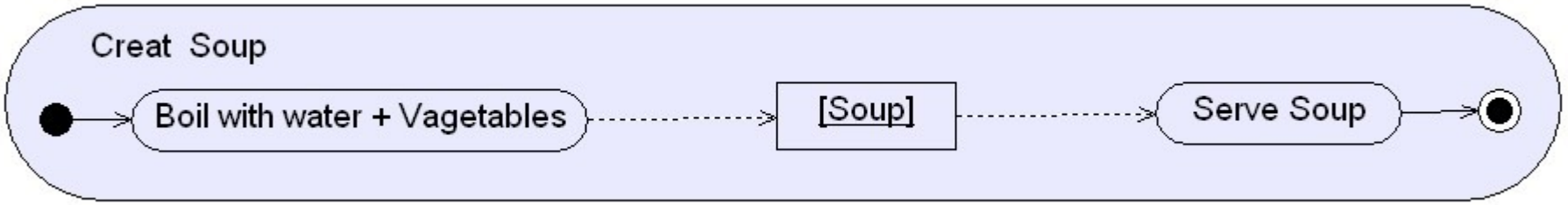
- We can depict a parameter sent/received to/from another activity by drawing a parameter node.
- A parameter node notation includes a simple rectangle within we write the parameter name (or description).
- Given an activity with input & output parameters, the input parameter is connected (edged) with the first action. The output parameter is connected (edged) with the last action.

Parameter Node



Object Node

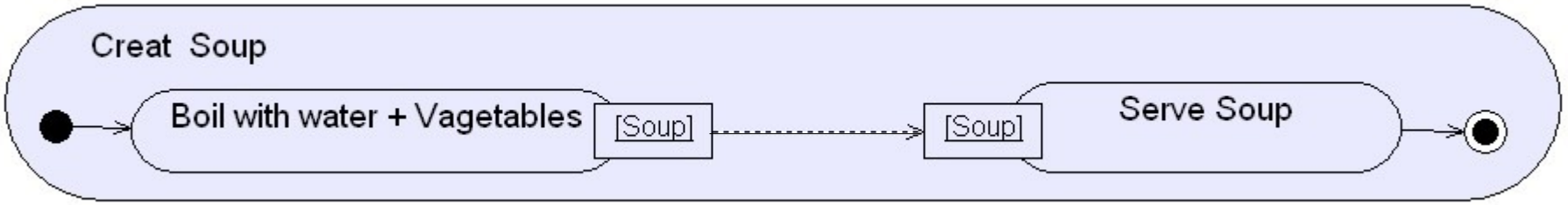
- We can depict an object instantiated from one of our designed classes with the same notation we use to represent a parameter.
- It is common to name the object node with the same name as its type.



Pins

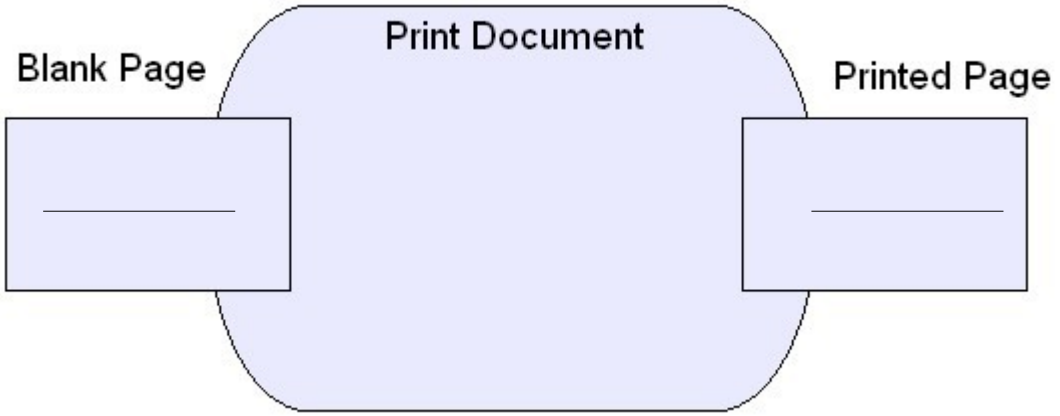
- Pin is a special object node that represents an object that is used as input/output to/from an action.
- The notation for pin is the same notation used for Object, just in a smaller dimension. In addition, the pin is drawn as part of an action and any edge that connects an action via a pin should connect the pin (instead of the action).

Pins



Input \ Output Pin

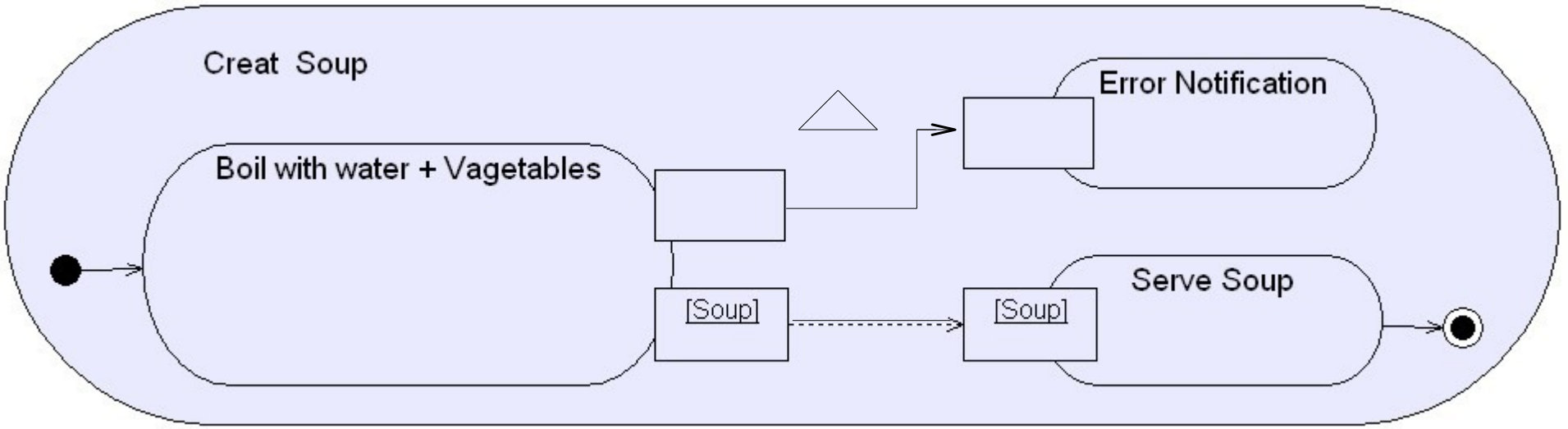
- Within the pin you can draw a small arrow to display whether it is an input (or output) pin. Excellent solution for those cases in which there is no leading line.



Exception Handling

- Representing an error output (exception) can be done using an exception pin.
- The notation for exception pin is the same notation we use for a simple pin, just that we add a small triangle (in a shape of arrow pointing up) near it.

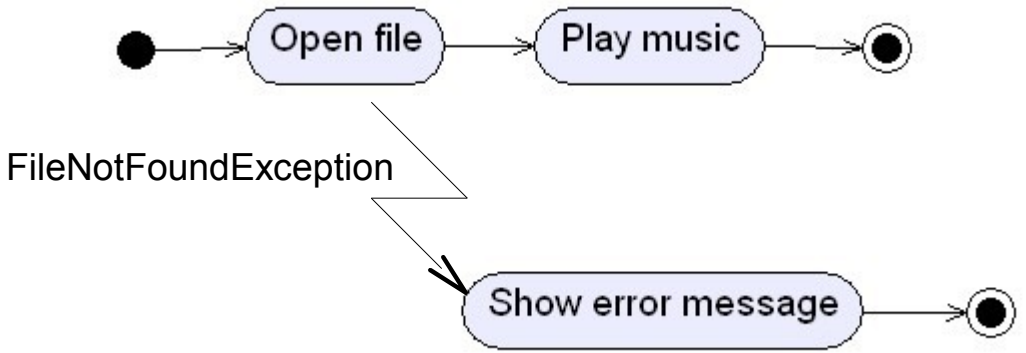
Exception Handling



Exception Handling

- An alternate possibility to depict exceptions and exceptions handling is by drawing a line shaped as a strike coming out of the activity where the exception can happen and lead to another activity that will be executed when the exception happens.

Exception Handling



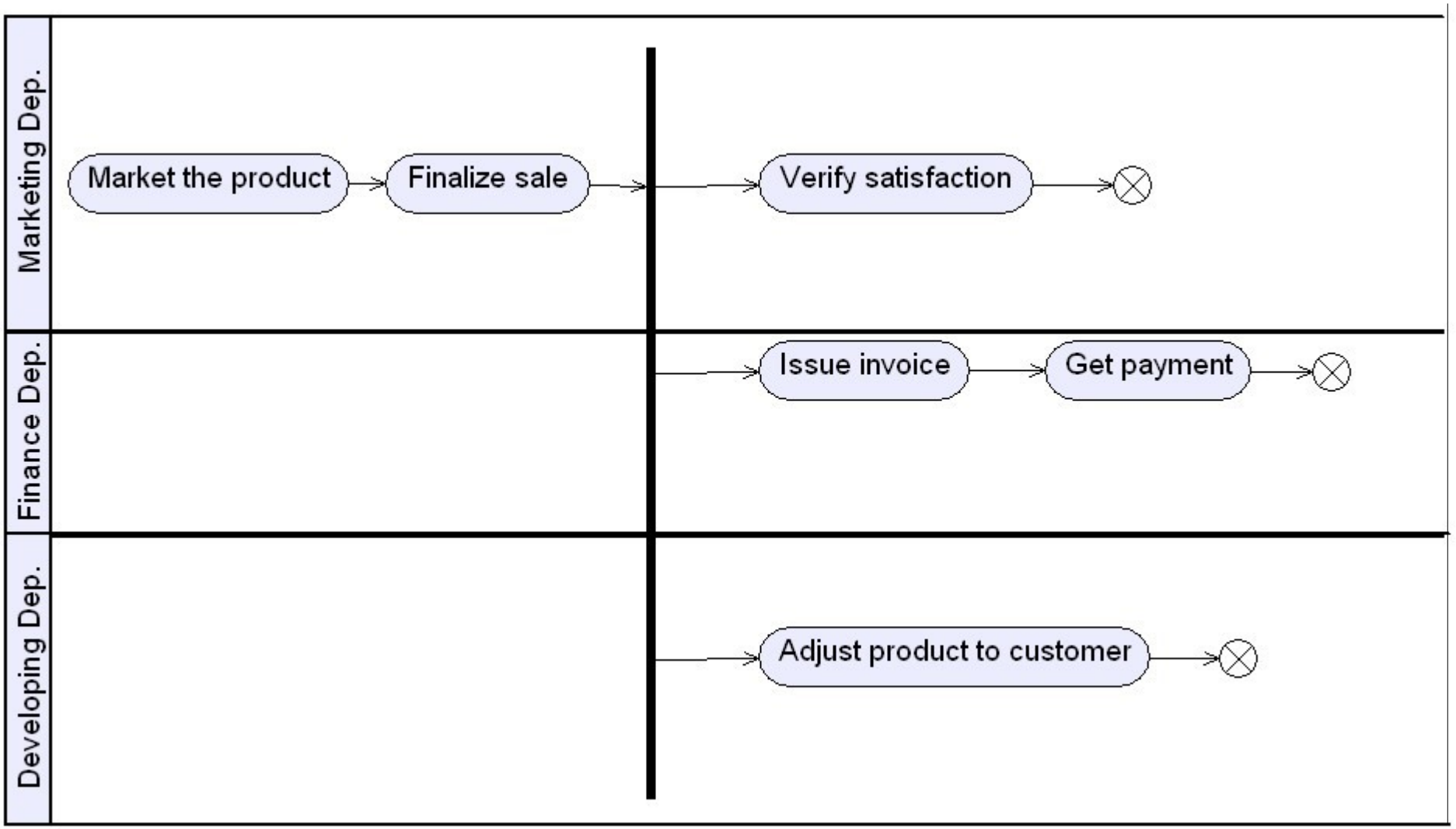
Activity Partitions

- The activity diagram can be divided into partitions (swim lanes).
- Separating the activity diagram into separated partitions can be done based on the business unit responsible for each activity, based on the manager responsible for each activity and/or base on any other criteria we set.
- Each partition can be labeled to indicate on what we have based the partition separation (swim lanes).

Activity Partitions

- If an activity shown in one of the partitions is beyond the scope of our work we can add the <<external>> stereotype to that activity.

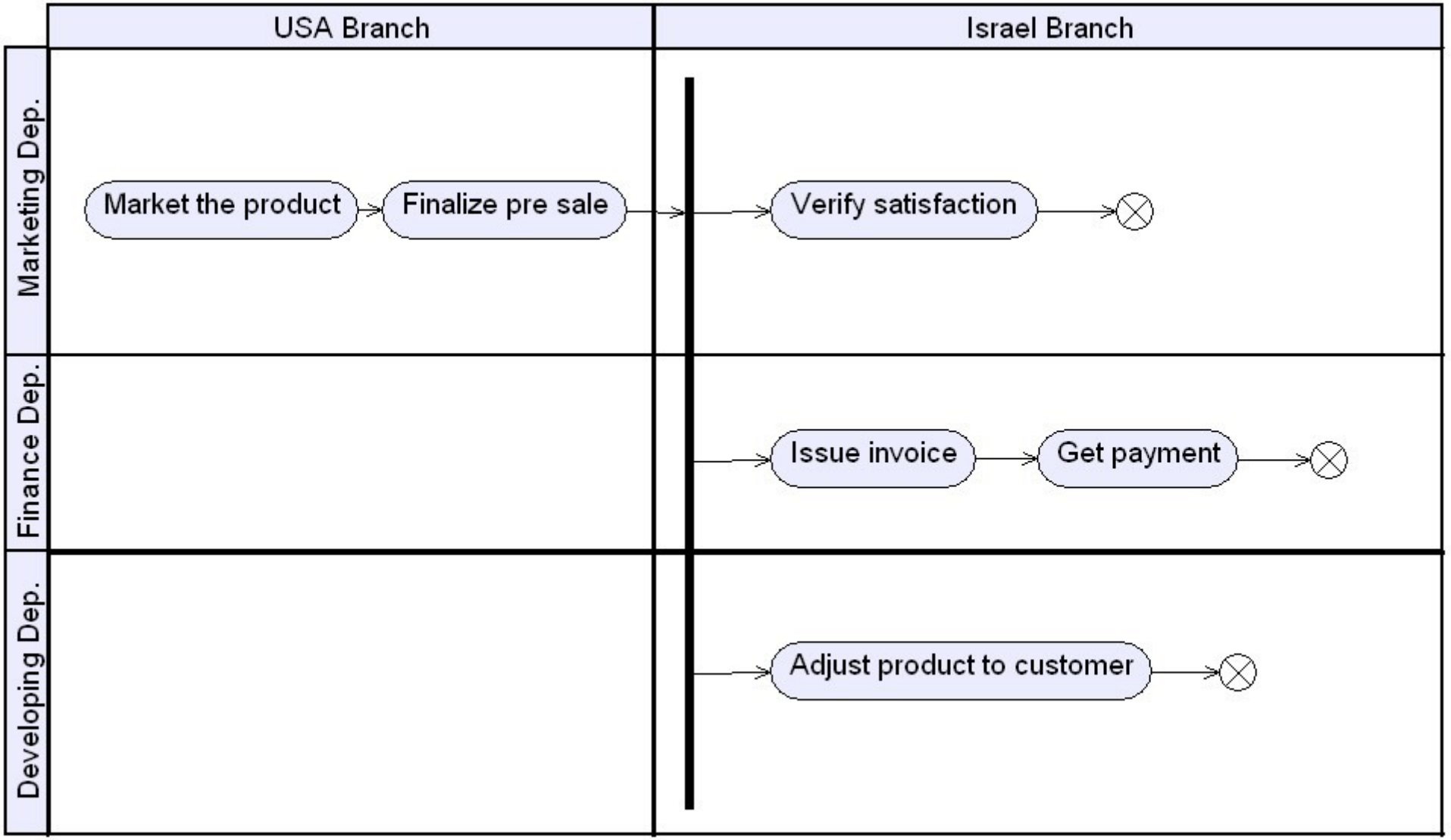
Activity Partitions



Activity Partitions

- Activity partitions can be implemented in two dimensions by dividing the activity diagram both horizontally and vertically.

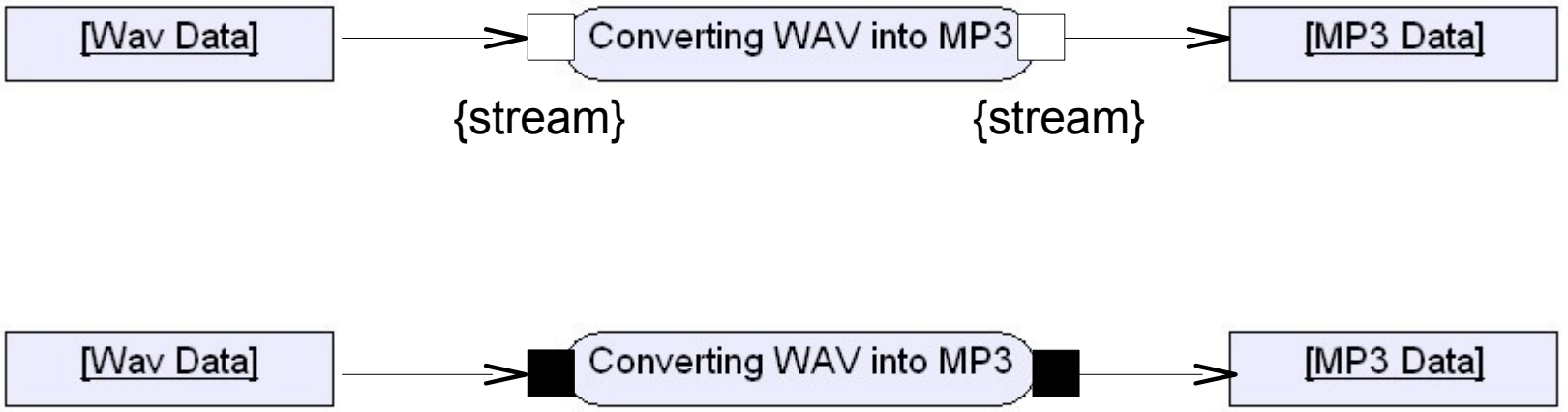
Activity Partitions



Streaming Action

- An action / activity is count to be a streaming one if it can produce the output while processing the input at the same time.
- We depict a streaming action / activity by adding `{stream}` both to its input and its output.
- Alternatively, we can also depict a streaming action / activity by drawing the pins as black rectangles instead of empty ones.

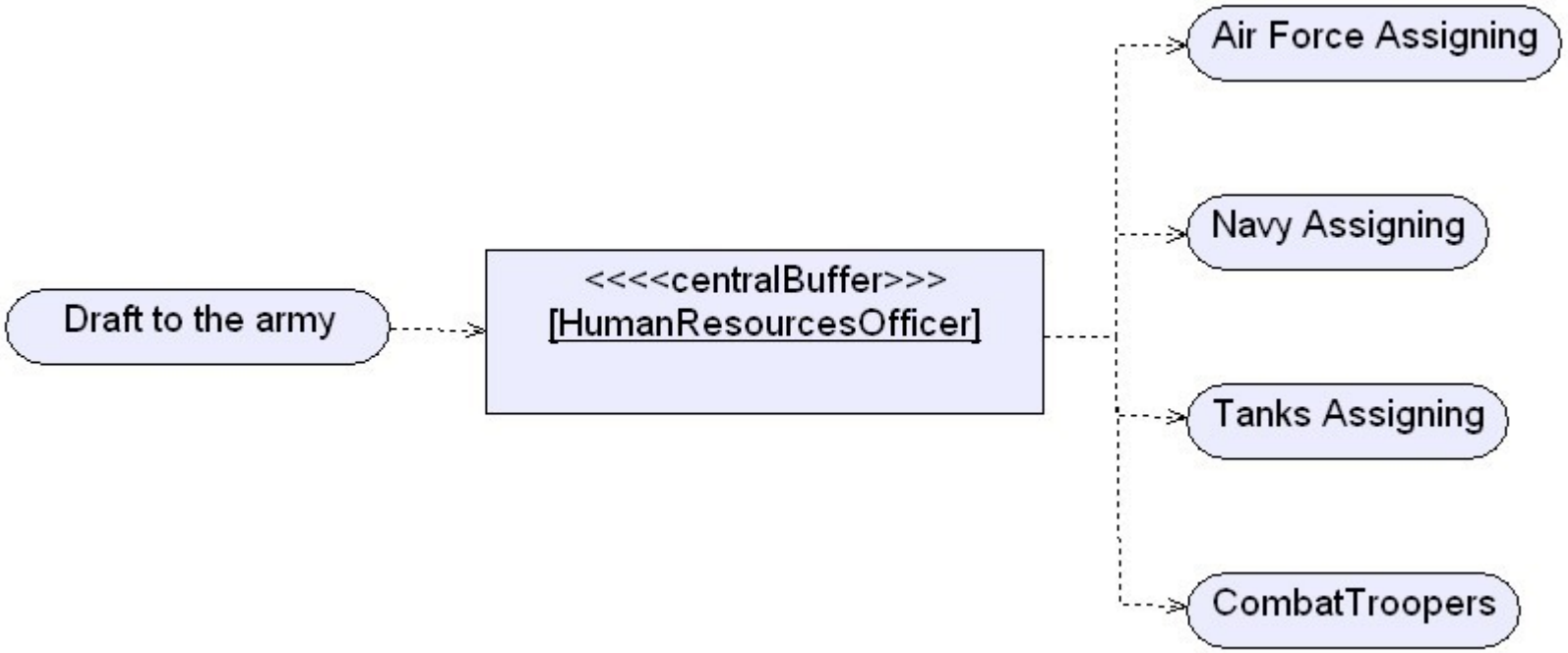
Streaming Action



Central Buffer Node

- A central buffer node, depicted using a simple rectangle with the `<<centralBuffer>>` stereotype on top and the object type below, represents an object that receives input data (mostly from more than one source) and buffers it forward (mostly to more than one destination) while either sorting the data or prioritizing it between its output edges.

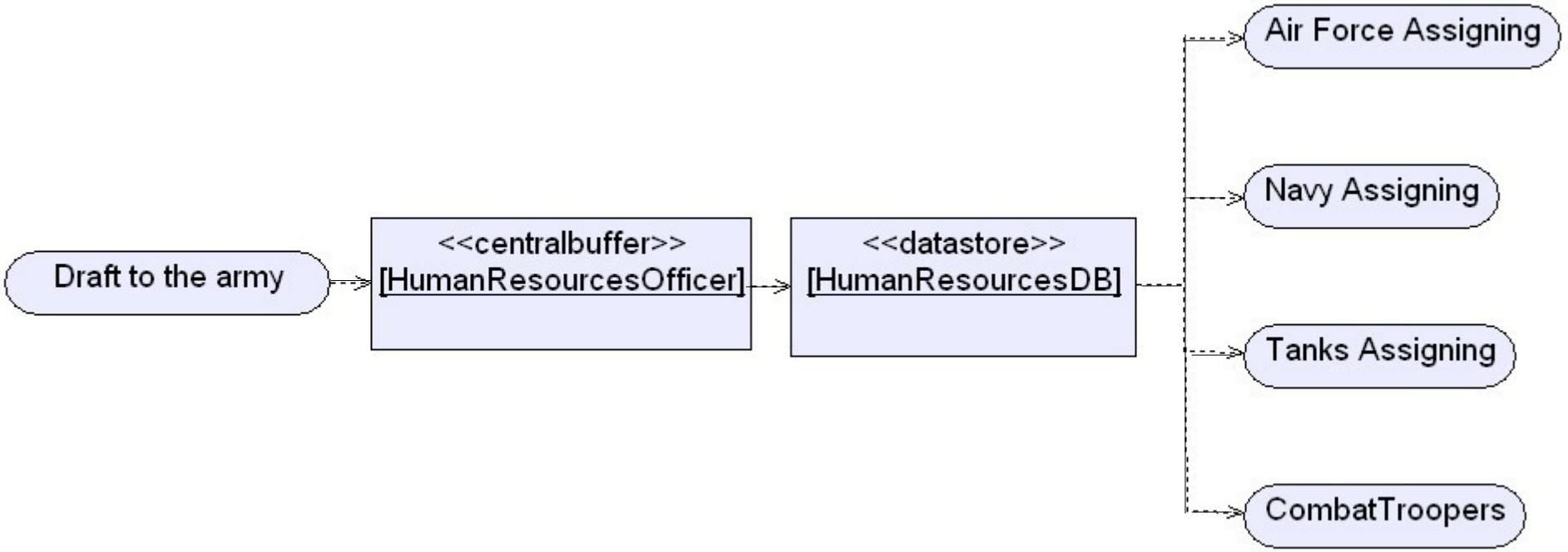
Central Buffer Node



Data Store Node

- A data store node is a variation of a central buffer node. A data store node copies all data that goes through it (e.g. Logs DB).
- The notation used for data store node is the same notation used for object... only with the “<<datastore>>” stereotype.
- According to UML 2.0 specification, if the same data passes again it will overwrite the previous log.

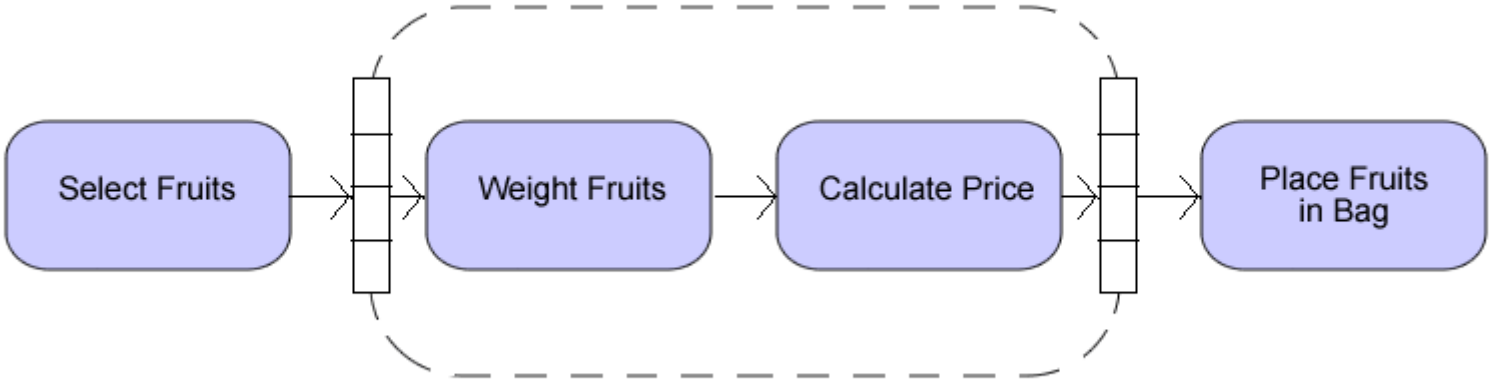
Data Store Node



Expansion Regions

- When having an action (or set of actions) that execute over a collection of input data we can show that by placing the action (or the set of actions) within an expansion region.
- An expansion region is showed using a dashed rectangle with rounded corners surrounding the action (or the set of actions). In addition, we should place a row of four input pins on the dashed boundary and connect it with a simple line to the first internal action, and do the same connecting the last action to a row of four output pins on the other side's dashed boundary.

Expansion Regions



UML Activity Diagrams

Introduction

- The UML activity diagram presents the system's execution flow.
- The activity diagram presents activities composed of one or more small actions.

Activity

- The notation used for activity is a rectangle with rounded corners.
- In the upper left corner we specify the activity name.
- Below the name it is possible to list the involved parameters. It is also possible to use parameter nodes instead (explained later).

Actions

- Within the activity notation it is possible to draw the actions that belong to this activity.
- The notation used for action is the same notation we use for activity.

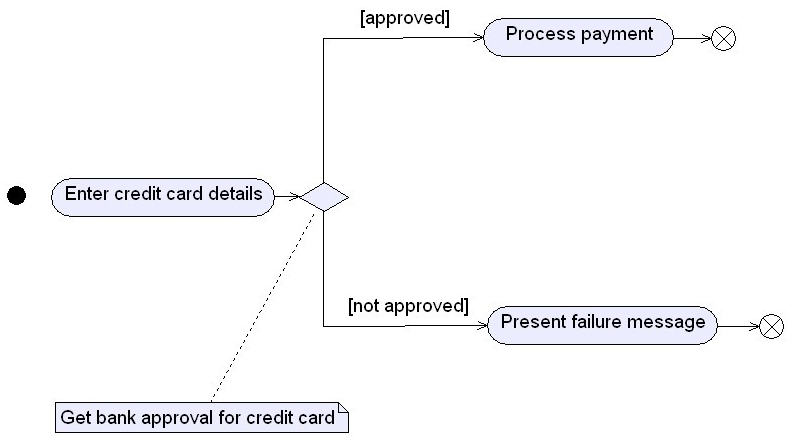
Initial Node

- Each activity starts with the initial node. The initial node notation is a small black circle.

Decision Node

- A decision node chooses an output flow from different available output flows based on its boolean expression (guard condition) value.
- Each decision node has one input edge and multiple output ones.
- The notation is an empty diamond. We put the boolean expression (guard condition) in brackets.
- A note can be added in order to present more info about the decision input.

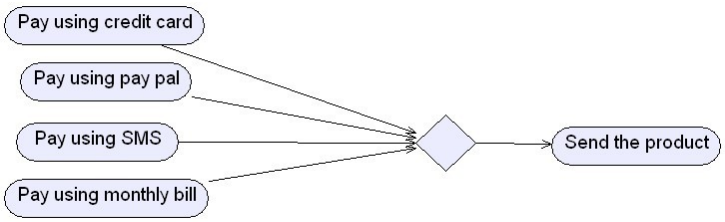
Decision Node



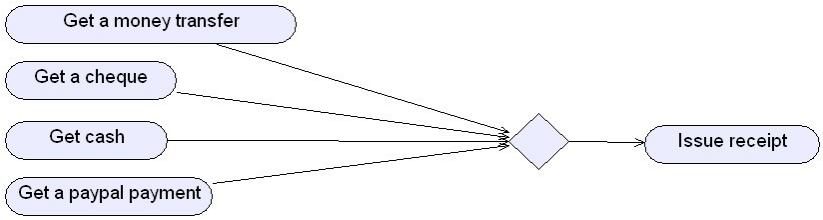
Merge Node

- A merge node unite the flows it receives into one single flow. It has multiple incoming edges and one output edge only.
- The notation for showing a merge node is an empty diamond.

Merge Node



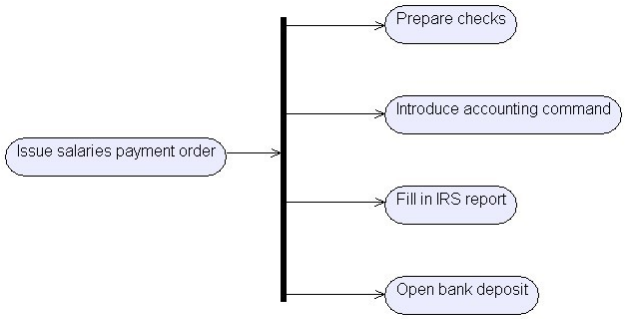
Merge Node



Fork Node

- A fork node split a flow into multiple concurrent flows.
- Unlike the merge node, the fork node duplicates each data item it receives and forward each copy through each one of the outgoing flows.

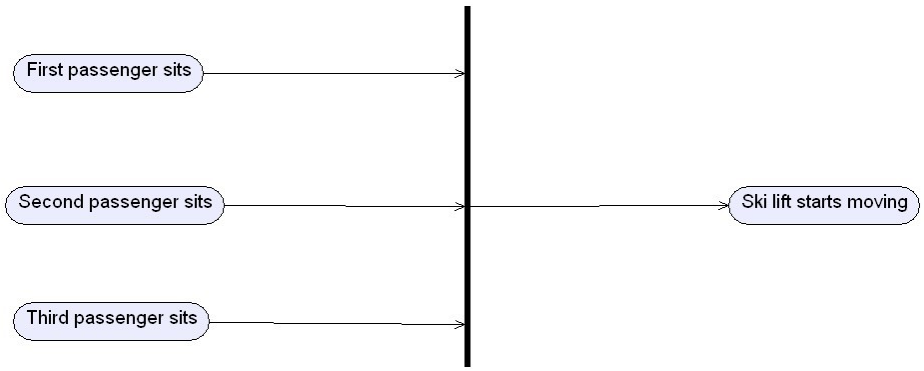
Fork Node



Join Node

- A join node synchronizes multiple flows (edges) back to a single flow (edge).
- The notation used for Join node is the same notation used for the fork node.

Join Node



This diagram describes a ski left for three people that doesn't move till all three people sit.

Join Node

- Near the join node it is possible to write a condition within braces { } in order to inform of a condition that must be true before the join node forward the flow.

Join Node

This diagram describes a ski lift for three people that doesn't move till all three people sit and the security belt of each one of them is active.

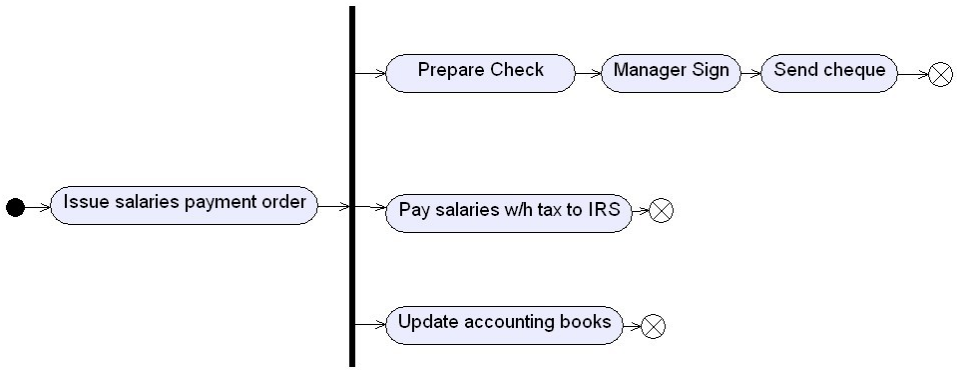
Activity Final Node

- The activity final node is a node that any flow of data that reaches it will cause the termination of the entire application.
- The notation used to draw an activity final node is a black circle with a black ring around it.

Flow Final Node

- The flow final node is a node that terminates a flow only. Data that reaches it won't terminate the application.
- The notation used to draw a flow final node is a black ring with X inside it.

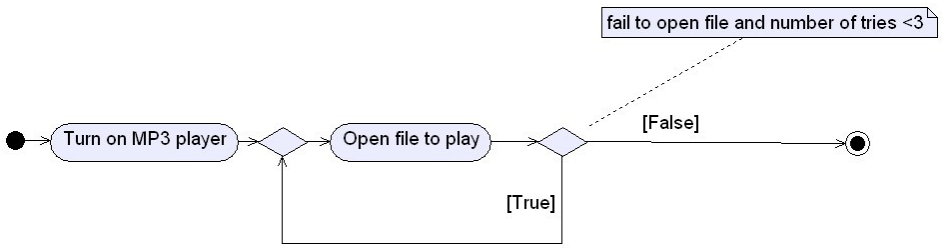
Flow Final Node



Loops

- The loop node usually includes three parts: setup, body & test. The setup part executes only once (when the loop starts) and it usually initializes the loop. The test part is evaluated before the body part of after it.

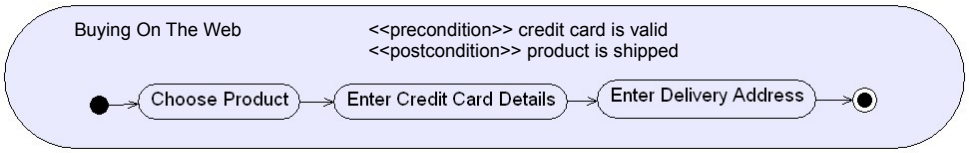
Loops



Preconditions & Postconditions

- Each activity can be specified together with its preconditions and postconditions.
- We write the precondition on top center of the diagram using the following format:
  <<precondition>> _____
- We write the postcondition on top center of the diagram (below the precondition) using the following format:
  <<postcondition>> _____

Preconditions & Postconditions

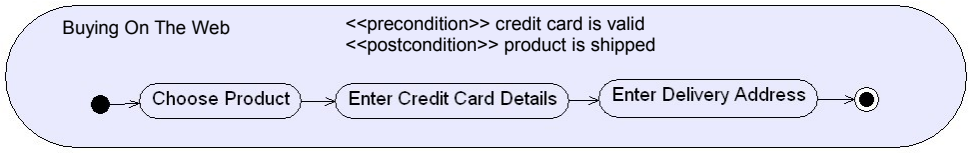


As with activities, we can add the preconditions and the postconditions to actions as well.

Specifying the preconditions and the postconditions – both for activities and for actions - using notes (labeled with <<precondition>> and <<postcondition>> accordingly) is feasible as well.

Activities Edges

- The activities edges are lines with open arrows that connect the activities with each other and represent the control and data flows from one activity to the other.



As with activities, we can add the preconditions and the postconditions to actions as well.

Specifying the preconditions and the postconditions – both for activities and for actions - using notes (labeled with <<precondition>> and <<postcondition>> accordingly) is feasible as well.

Object Flow Elements

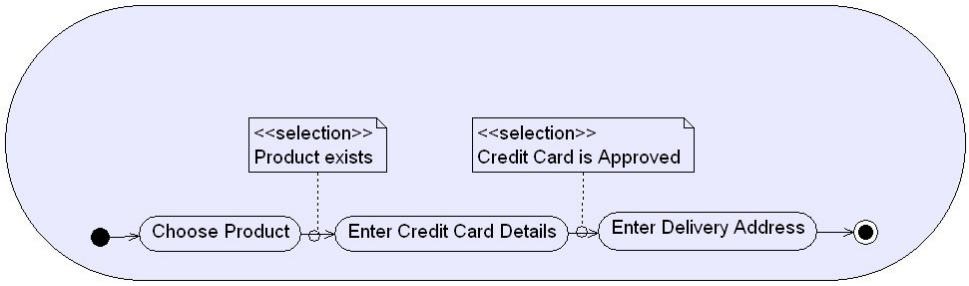
- The object flow element presents a special flow of one of the following possibilities:
  1. A flow through which only objects that meet a selection condition can move forward.
  2. A flow that is multi casted to a multiple instances.
  3. A flow that is multi received from multiple senders.
  4. A flow that transforms the data flow and add specific data to it.
- We use a simple small empty circle placed above the activity edge together with a note in which we write `<<selection>>` or `<<multi cast>>` or `<<multi receive>>` or `<<transforms>>`.

When it is a <<selection>> object flow we will write the condition below the <<selection>> title.

When it is a <<transformation>> object flow we will add the additional data below the <<transformation>> title.

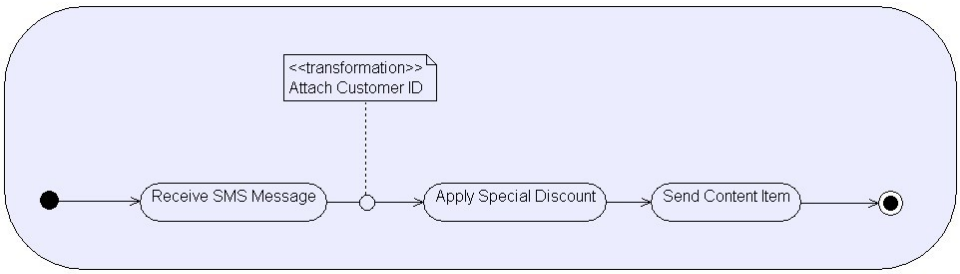
Object Flow Elements

<<selection>>



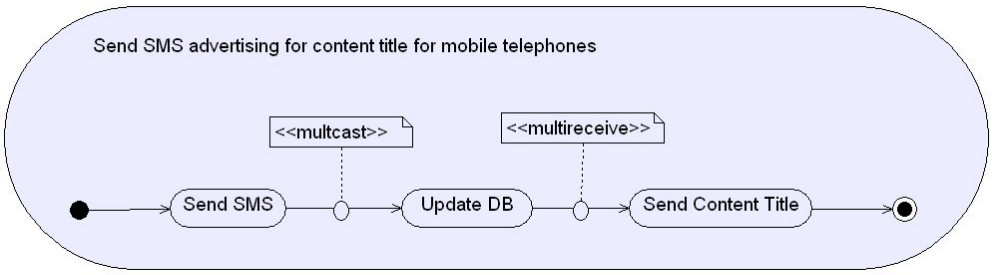
Object Flow Elements

<<transformation>>



Object Flow Elements

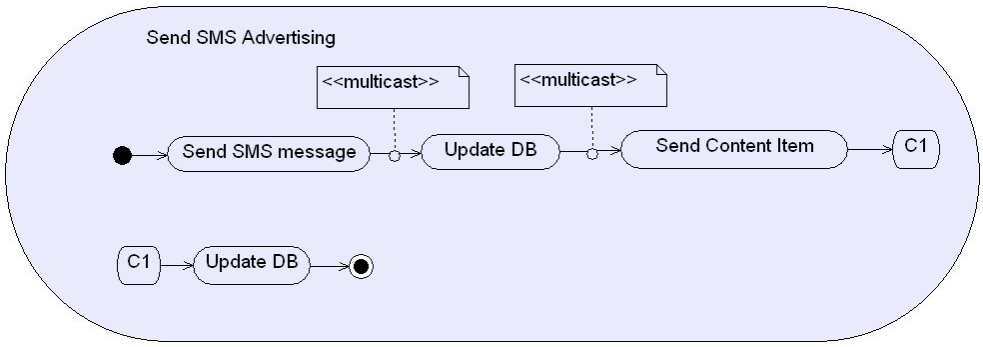
<<multicast>> <<multireceive>>



Connectors

- When dealing with big activity diagrams that can't be captured in one page or when drawing an activity edge is too complex due to a big number of other elements our diagram includes, it is possible to split the activity edge using a connector.
- The connector notation is a simple circle within we write its name.

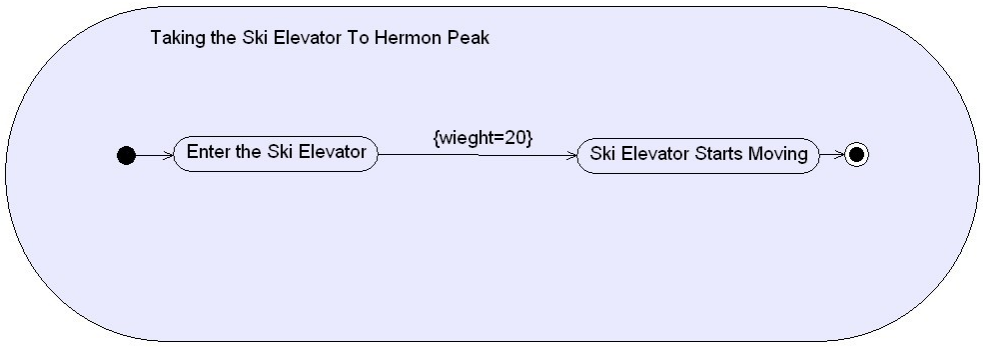
Connectors



Tokens & Weight

- Conceptually along the activities edges the flowing data can be perceived as tokens.
- Conceptually, each action that is performed has data input and data output that can be represented as tokens.
- We can represent the number of tokens that should be made available for a given action before it can be executed by writing `{weight=n}` above the relevant activity edge, while `n` is the number of the required tokens.

Tokens & Weight

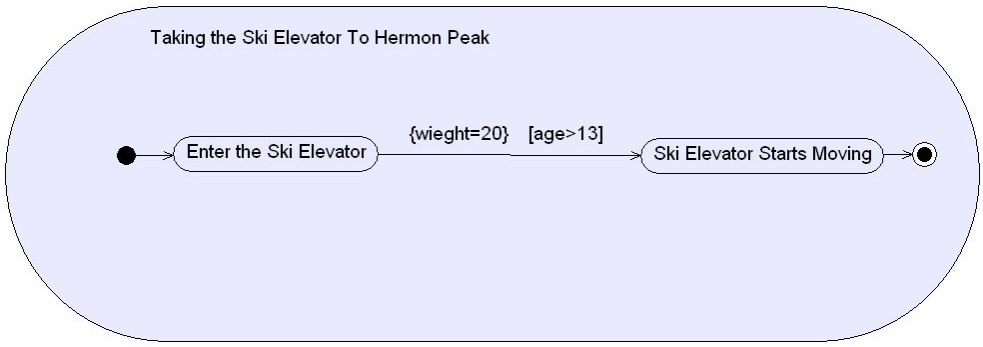


The ski elevator to Hermon peak has 20 seats. Only when 20 people enter that elevator it starts moving.

Activity Edge Guard Condition

- On top of each activity edge we can add a guard condition that will be checked the moment the required weight is met.
- If one (or more) of the tokens fail(s) to pass the guard condition and the total number of tokens that pass doesn't meet the required weight then all tokens that passed will be held until the weight requirement is met.
- The notation for a guard condition includes a boolean expression within square brackets we draw near the relevant activity edge.

Activity Edge Guard Condition

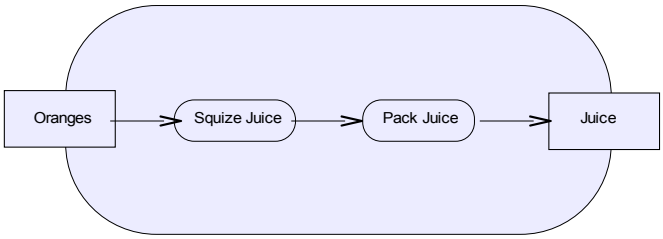


Assuming that our ski elevator is limited for people above 13 we can depict this fact by adding a guard condition associated with the edge that connects the two activities.

Parameter Node

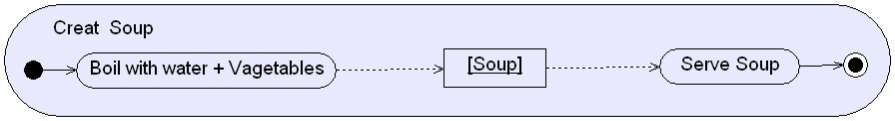
- We can depict a parameter sent/received to/from another activity by drawing a parameter node.
- A parameter node notation includes a simple rectangle within we write the parameter name (or description).
- Given an activity with input & output parameters, the input parameter is connected (edged) with the first action. The output parameter is connected (edged) with the last action.

Parameter Node



Object Node

- We can depict an object instantiated from one of our designed classes with the same notation we use to represent a parameter.
- It is common to name the object node with the same name as its type.

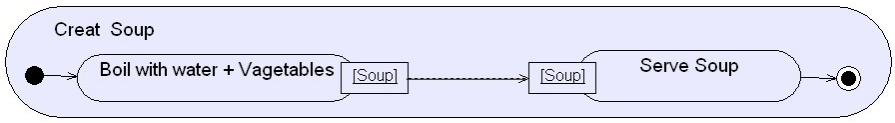


In this diagram we can see the object soup. Please note the lines connected the activities with the object should be plain ones. Yet, the tool I used (StarUML) nor others I have (e.g. Netbeans) support that feature.

Pins

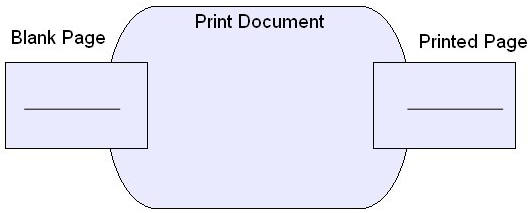
- Pin is a special object node that represents an object that is used as input/output to/from an action.
- The notation for pin is the same notation used for Object, just in a smaller dimension. In addition, the pin is drawn as part of an action and any edge that connects an action via a pin should connect the pin (instead of the action).

Pins



Input \ Output Pin

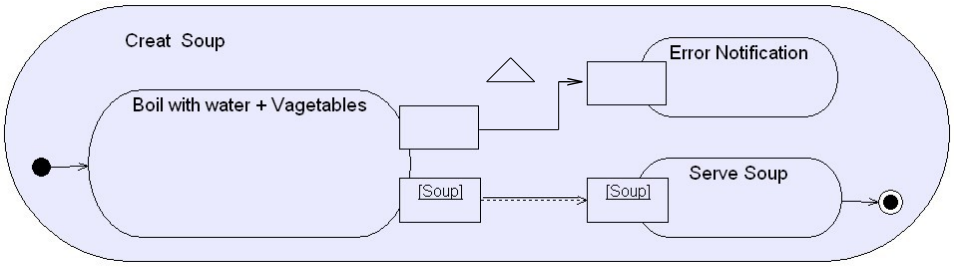
- Within the pin you can draw a small arrow to display whether it is an input (or output) pin. Excellent solution for those cases in which there is no leading line.



Exception Handling

- Representing an error output (exception) can be done using an exception pin.
- The notation for exception pin is the same notation we use for a simple pin, just that we add a small triangle (in a shape of arrow pointing up) near it.

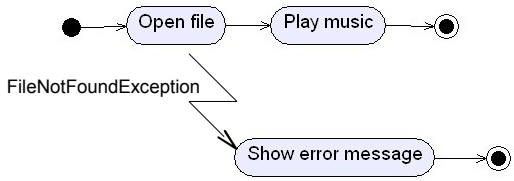
Exception Handling



Exception Handling

- An alternate possibility to depict exceptions and exceptions handling is by drawing a line shaped as a strike coming out of the activity where the exception can happen and lead to another activity that will be executed when the exception happens.

Exception Handling



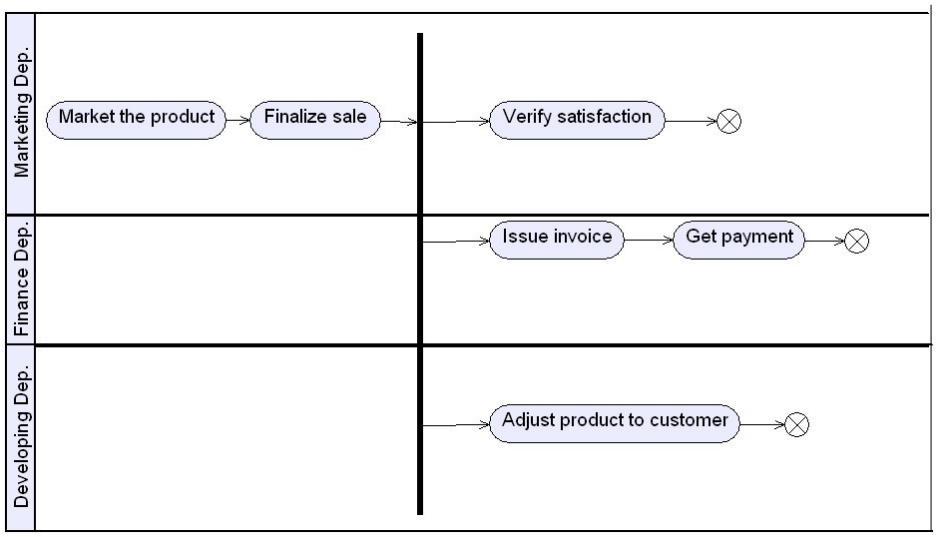
Activity Partitions

- The activity diagram can be divided into partitions (swim lanes).
- Separating the activity diagram into separated partitions can be done based on the business unit responsible for each activity, based on the manager responsible for each activity and/or base on any other criteria we set.
- Each partition can be labeled to indicate on what we have based the partition separation (swim lanes).

Activity Partitions

- If an activity shown in one of the partitions is beyond the scope of our work we can add the <<external>> stereotype to that activity.

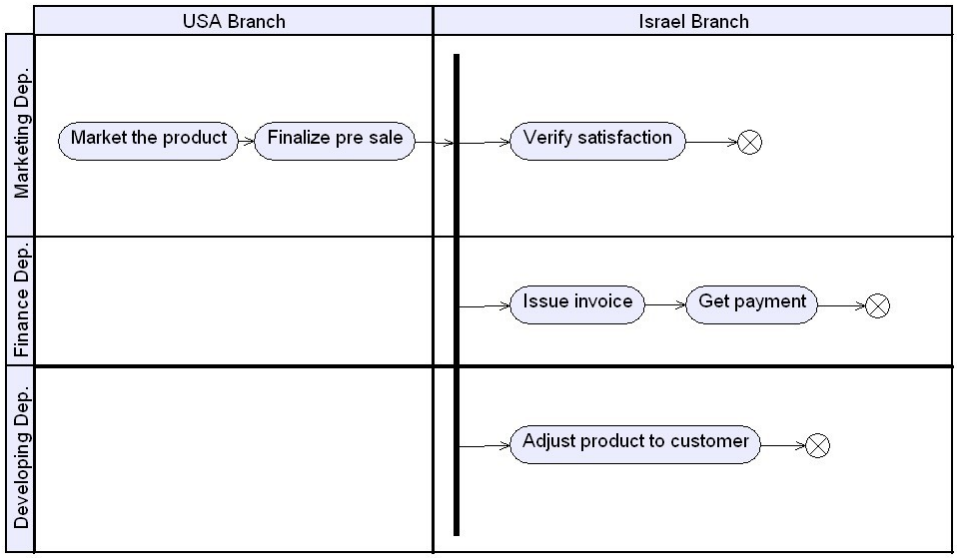
Activity Partitions



Activity Partitions

- Activity partitions can be implemented in two dimensions by dividing the activity diagram both horizontally and vertically.

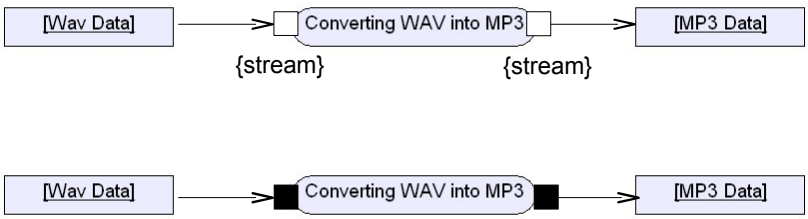
Activity Partitions



Streaming Action

- An action / activity is count to be a streaming one if it can produce the output while processing the input at the same time.
- We depict a streaming action / activity by adding `{stream}` both to its input and its output.
- Alternatively, we can also depict a streaming action / activity by drawing the pins as black rectangles instead of empty ones.

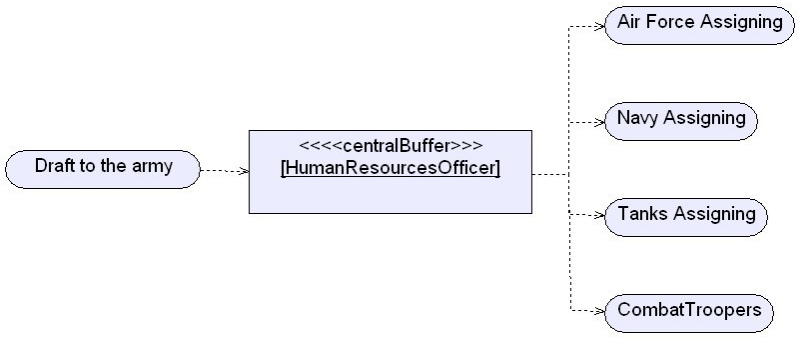
Streaming Action



Central Buffer Node

- A central buffer node, depicted using a simple rectangle with the `<<centralBuffer>>` stereotype on top and the object type below, represents an object that receives input data (mostly from more than one source) and buffers it forward (mostly to more than one destination) while either sorting the data or prioritizing it between its output edges.

Central Buffer Node

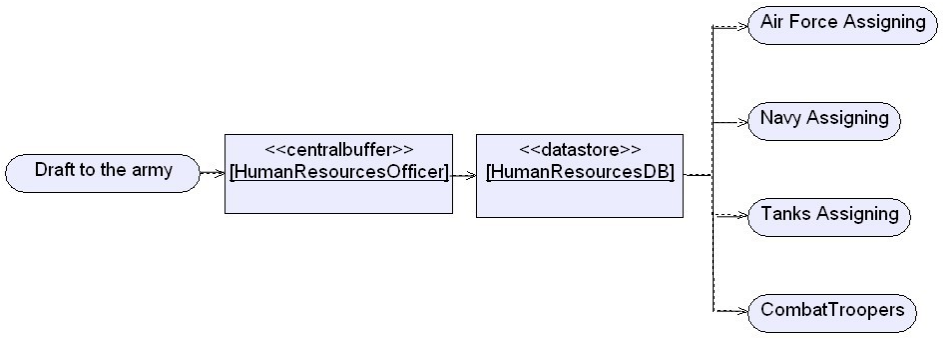


Though according the spec the lines in this diagram should be plain simple lines the available UML tools at this time don't allow to use. As a result of that the lines depicted in this diagram are dashed ones.

Data Store Node

- A data store node is a variation of a central buffer node. A data store node copies all data that goes through it (e.g. Logs DB).
- The notation used for data store node is the same notation used for object... only with the “<<datastore>>” stereotype.
- According to UML 2.0 specification, if the same data passes again it will overwrite the previous log.

Data Store Node



Expansion Regions

- When having an action (or set of actions) that execute over a collection of input data we can show that by placing the action (or the set of actions) within an expansion region.
- An expansion region is showed using a dashed rectangle with rounded corners surrounding the action (or the set of actions). In addition, we should place a row of four input pins on the dashed boundary and connect it with a simple line to the first internal action, and do the same connecting the last action to a row of four output pins on the other side's dashed boundary.

Expansion Regions