



IBM Software Group

IBM Rational Rhapsody Advanced Systems Training v7.5

Advanced Activity Modeling



Rational software



IBM Software Group

IBM Rational Rhapsody Advanced Systems Training v7.5

SysML™ 1.0 for Systems Engineers



Rational® software



IBM Software Group

IBM Rational Rhapsody Advanced Systems Training v7.5

Advanced Activity Modeling – Token Flow



Rational software

Objectives

- After completing this module you should be able to:
 - ▶ Identify how frames are used in Rational Rhapsody Activity diagrams
 - ▶ Identify the different types of tokens and how they are used in Activity diagrams
 - ▶ Describe how nodes, edges, parameters, streams, forks, joins, and swimlanes are used on activity diagrams
 - ▶ Define the two Action types that allow invocation of other behaviors:
 - Call Behavior
 - Call Operation
 - ▶ Confirm that a Control value is a special kind of token
 - ▶ Confirm that a Control operator is a special kind of behavior

A note on frames

- The SysML frame header for Activity Diagrams takes the form:
 - ▶ act [Activity] activity name [diagram name]
- You will find in Rhapsody the header takes the form:
 - ▶ act [type of owner] name of owner [activity name]
 - ▶ This was found to be more informative
 - ▶ Examples:



act [Package] ActivityExample [Brewing Coffee]

The diagram shows a SysML frame header for an activity diagram. It consists of a rectangular box with a blue border. The text 'act [Package] ActivityExample [Brewing Coffee]' is written inside the box in blue. The box is slightly tilted to the right.



act [block] Filling Station [Pumping]

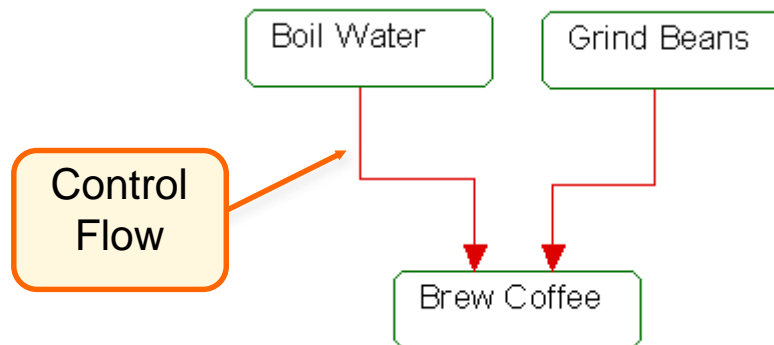
The diagram shows a SysML frame header for an activity diagram. It consists of a rectangular box with a blue border. The text 'act [block] Filling Station [Pumping]' is written inside the box in blue. The box is slightly tilted to the right.

Token semantics

- SysML activity diagrams are based on token flow semantics:
 - ▶ Similar to petri-nets (if you are familiar with them – if not don't worry)
 - ▶ Expands the possibilities for flow modeling especially for parallel behavior.
- Actions produce and consume tokens.
 - ▶ Tokens may be either control or data ('Object').
 - ▶ Actions begin when tokens are available on all control inputs and required data inputs.
 - ▶ Tokens move along edges.
- For a token to traverse an edge:
 - ▶ The edge must be ready to allow the token to pass.
 - Edges have rules regarding when tokens can flow.
 - ▶ The destination node must be ready to accept the token.
 - Nodes have rules regarding when they will accept tokens.

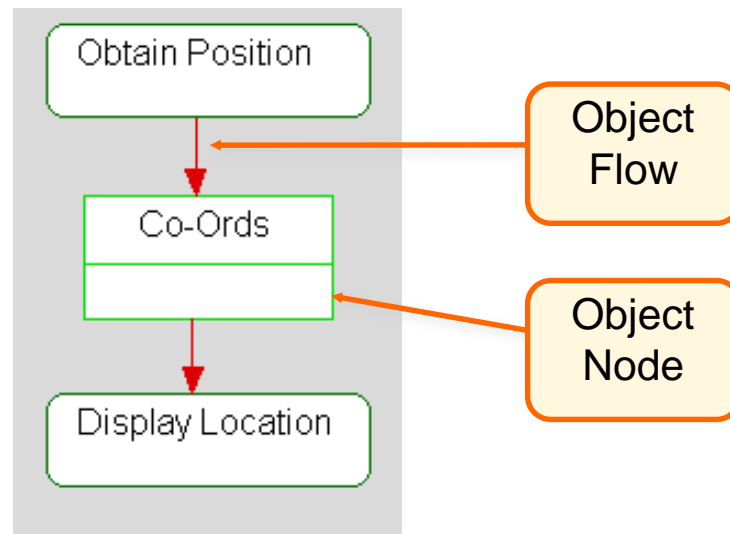
Control token example

- Control tokens traverse control flows.
- In this example:
 - ▶ When Boil Water finishes it presents a token to its outgoing edge.
 - The 'edge' rule is met.
 - However this token cannot traverse the edge until both edge conditions and node conditions have been met.
 - Brew Coffee is not yet ready to accept the token.
 - ▶ Once Grind Beans finishes – it presents a token to its outgoing edge
 - Now all conditions (in this case they are all edge conditions) have been met – both tokens now traverse and Brew Coffee begins (consuming both tokens in the process).



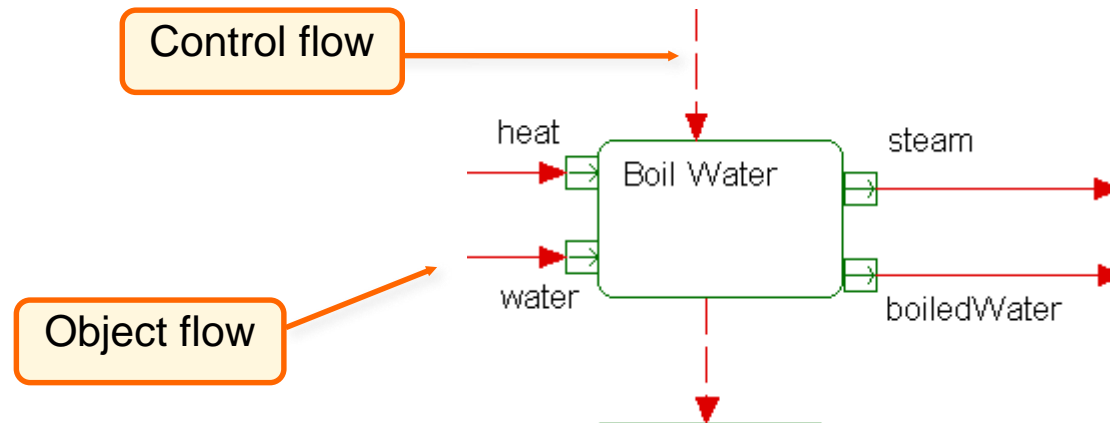
Actions and data

- Actions consume, transform and produce data.
 - ▶ The data gets into / out of the Action via Object Tokens
 - ▶ Object Nodes act as containers for Object Tokens
 - ▶ Object flows go to / from Object Nodes
 - Note that SysML allows Control Flows to be depicted as dashed lines to distinguish them from Object Flows.

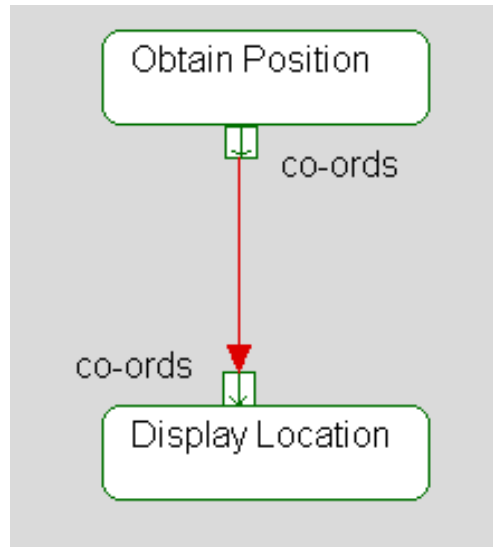


Action execution with pins – the simple version

- Actions may have *pins*.
- Pins are a type of Object Node.
 - ▶ An action begins execution when:
 - There are object tokens present on all its input pins.
 - There are control tokens present on all its input edges
 - ▶ Once execution is finished:
 - Object tokens are placed on all its output pins
 - Control tokens are placed on all its output edges

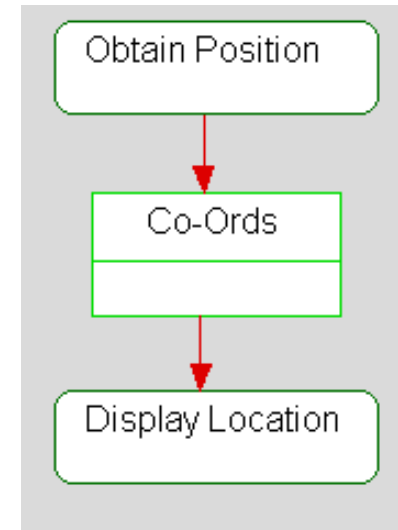


Object Nodes



An object flow linking two object node pins.

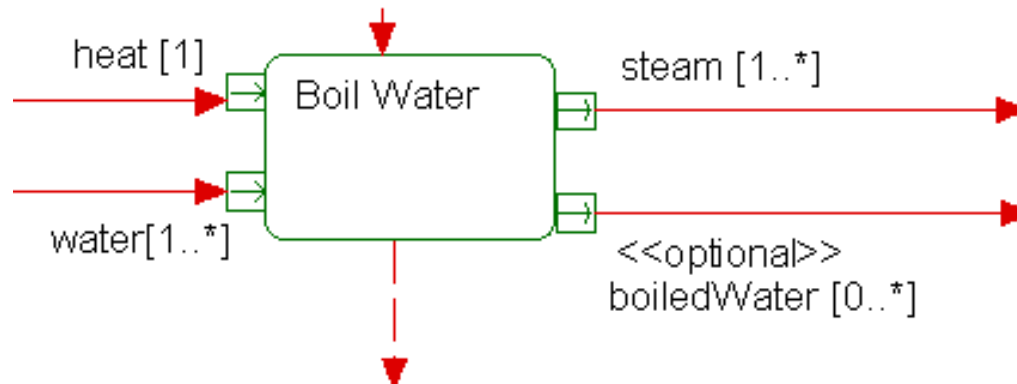
Equivalent



Two object flow arrows linking object nodes and actions.

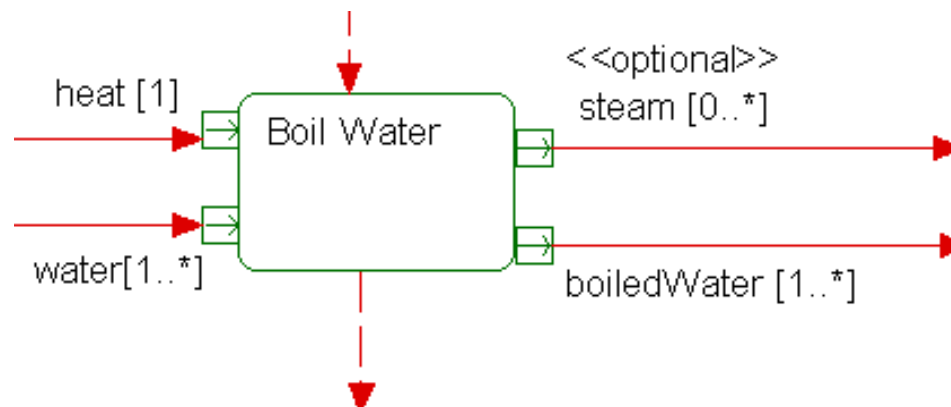
Multiplicity of pins

- Pins are types of Object Node.
 - ▶ They act as containers for tokens.
 - ▶ Tokens collect in pins waiting to be processed by an Action.
- Pins have multiplicity:
 - ▶ The minimum and maximum (actual) number of tokens consumed or produced in any one execution of that action.
 - ▶ A lower multiplicity of zero indicates an optional input.
 - ▶ A stereotype of `<<optional>>` may also be applied.
 - The lower multiplicity must be zero in order to apply this stereotype.



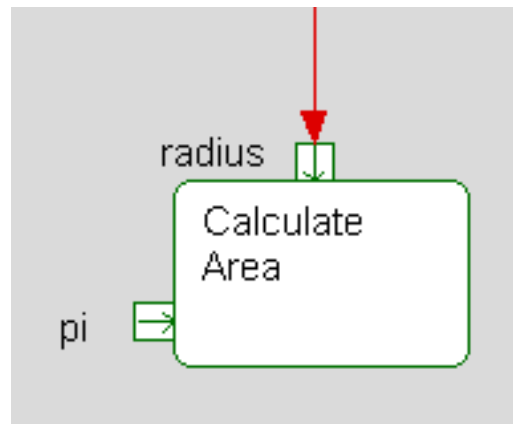
Action execution with pin multiplicity

- Actions may have *pins*.
 - ▶ An action executes when:
 - There are data tokens present on all its (required) input pins.
 - There are control tokens present on all its input edges.
 - ▶ For an action to terminate:
 - It must have placed enough tokens on its output pins to satisfy minimum multiplicity requirements.
 - ▶ Once execution is finished:
 - Data tokens are placed on all its (required) output pins.
 - Control tokens are placed on all its output edges.



Value pins

- A value pin is an input pin that provides a value to an action but has no incoming edge.
 - ▶ Used primarily to reduce activity size for constant values.

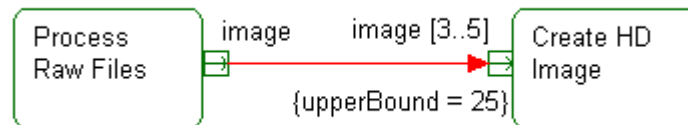


Object nodes

- Object nodes are an abstract container for tokens.
 - ▶ Object nodes can contain multiple tokens of the same type.
 - ▶ Unless specified they are assumed to be capable of holding an infinite number of tokens.
 - ▶ Object nodes may restrict the number of tokens they can hold with an {upperBound}.
 - This merely constrains the number of tokens the node can hold. Further tokens will not be accepted by the node.
 - ▶ Object nodes can be ordered such that incoming tokens:
 - Do not overtake each other (the default).
 - {ordering=FIFO}
 - Overtake previous tokens.
 - {ordering=LIFO}

Multiplicity versus upper bound

- The multiplicity of a pin controls the Action execution.
 - ▶ If the pin has fewer tokens than the lower multiplicity the action cannot start.
 - ▶ The upper multiplicity determines how many tokens are consumed by a single execution of the action.
- The upper bound dictates the maximum number of tokens that an object node may hold.



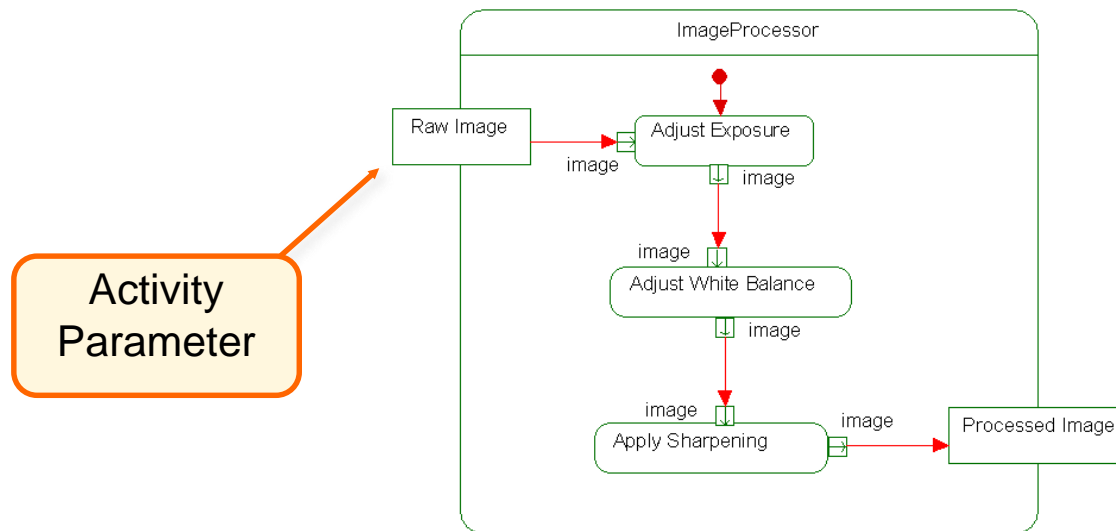
Create HD Image can take up to 5 images and make a composite HD image from them.

It requires at least 3 images to perform the job, but can take 4 or 5.

The upperBound says that the object node can hold up to 25 images – once this limit is reached any further images produced by **Process Raw Files** will be rejected.

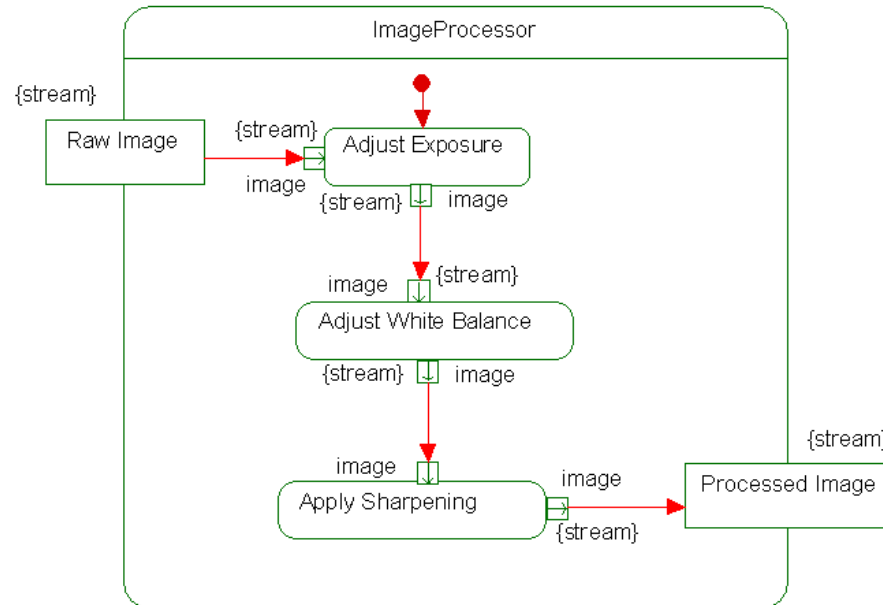
Activity parameters

- Activities can have parameters.
 - ▶ When an Activity begins:
 - A control token is placed on all its initial nodes.
 - If any data is available to the Activity it appears as Object Tokens on its input parameters.
 - ▶ When an Activity ends:
 - It outputs values to its invoker via its output parameters.



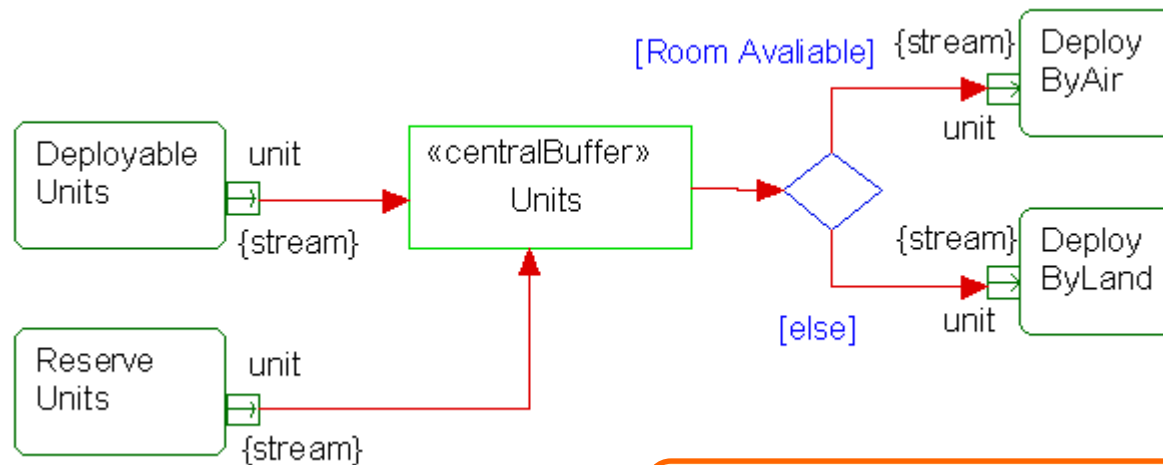
Streaming

- Pins/Parameters may have the constraint {stream} applied:
 - ▶ With {stream}
 - Actions / Activities continue to accept new tokens even though the Action / Activity is executing.
 - ▶ Without {stream}
 - Once the Action / Activity has begun executing any further inputs are ignored.



Central Buffer Nodes

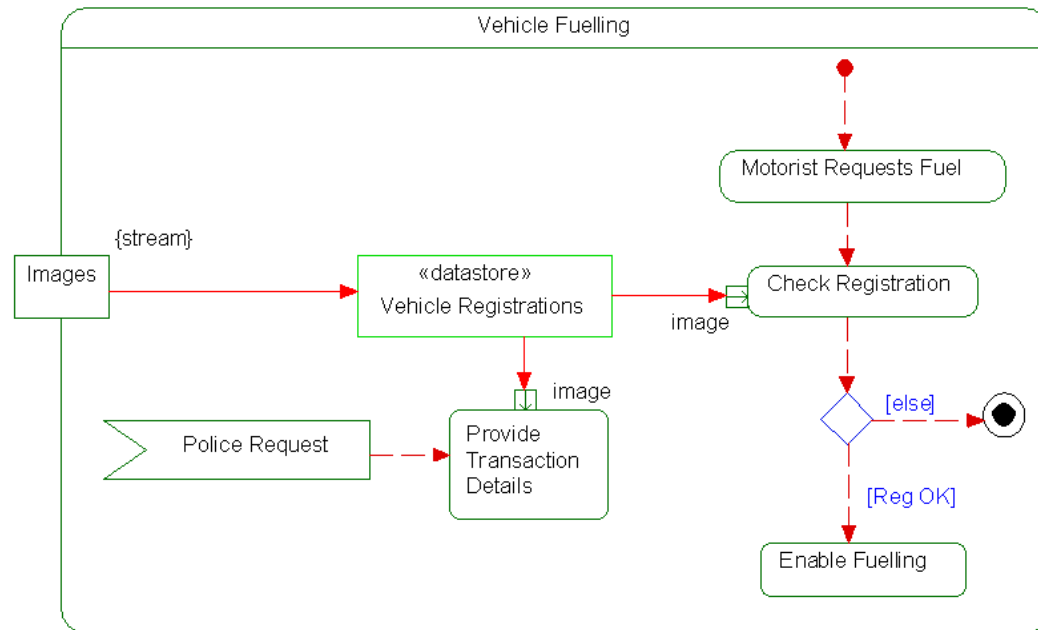
- Central Buffer Nodes are types of Object Node.
 - ▶ All Object Nodes have buffer capability, however:
 - Pins and Activity Parameters have a single source / destination
 - ▶ Central Buffer Nodes may have multiple sources / destinations.
 - ▶ Central Buffer Nodes act as a temporary storage area for tokens.



When there is no more room left on the air transport, units are deployed by land.

Data Store Nodes

- Data Stores are types of Object Node:
 - ▶ When a token enters a Data Store, it is kept there.
 - ▶ When an outgoing edge requests a token, the Data Store provides a copy of it – so the token is always available.
 - ▶ Data is persistent and used when needed rather than transient and used when available.
 - ▶ Implements the pull semantics of earlier forms of data flow modeling.
 - ▶ Note that this is not permanent storage – tokens only exist for as long as the Activity is executing.

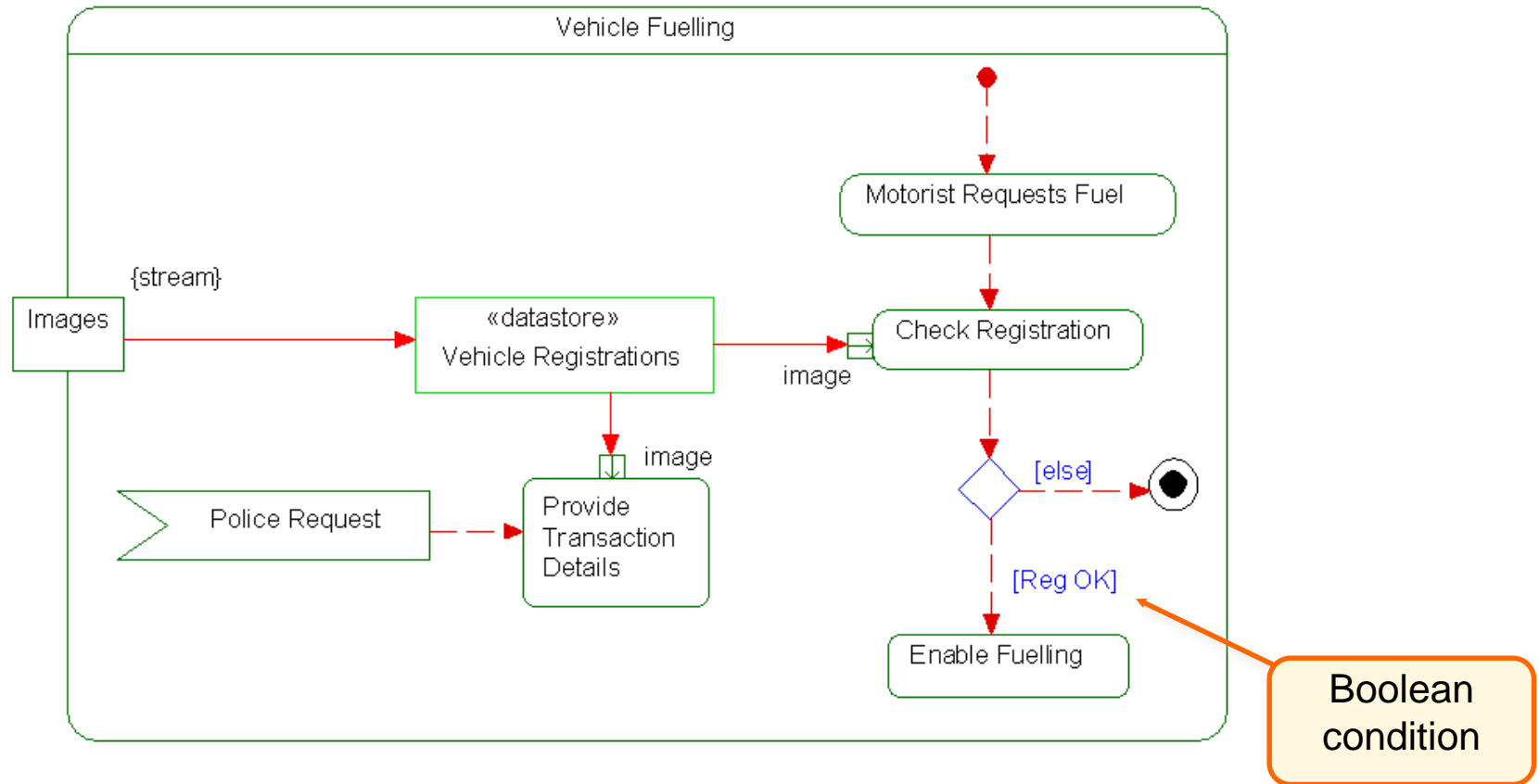


Edges

- Activity Edges are either:
 - ▶ Control Flow
 - ▶ Object Flow
- Edges may have rules that control token flow:
 - ▶ Guards
 - ▶ Weight
 - ▶ Transformation

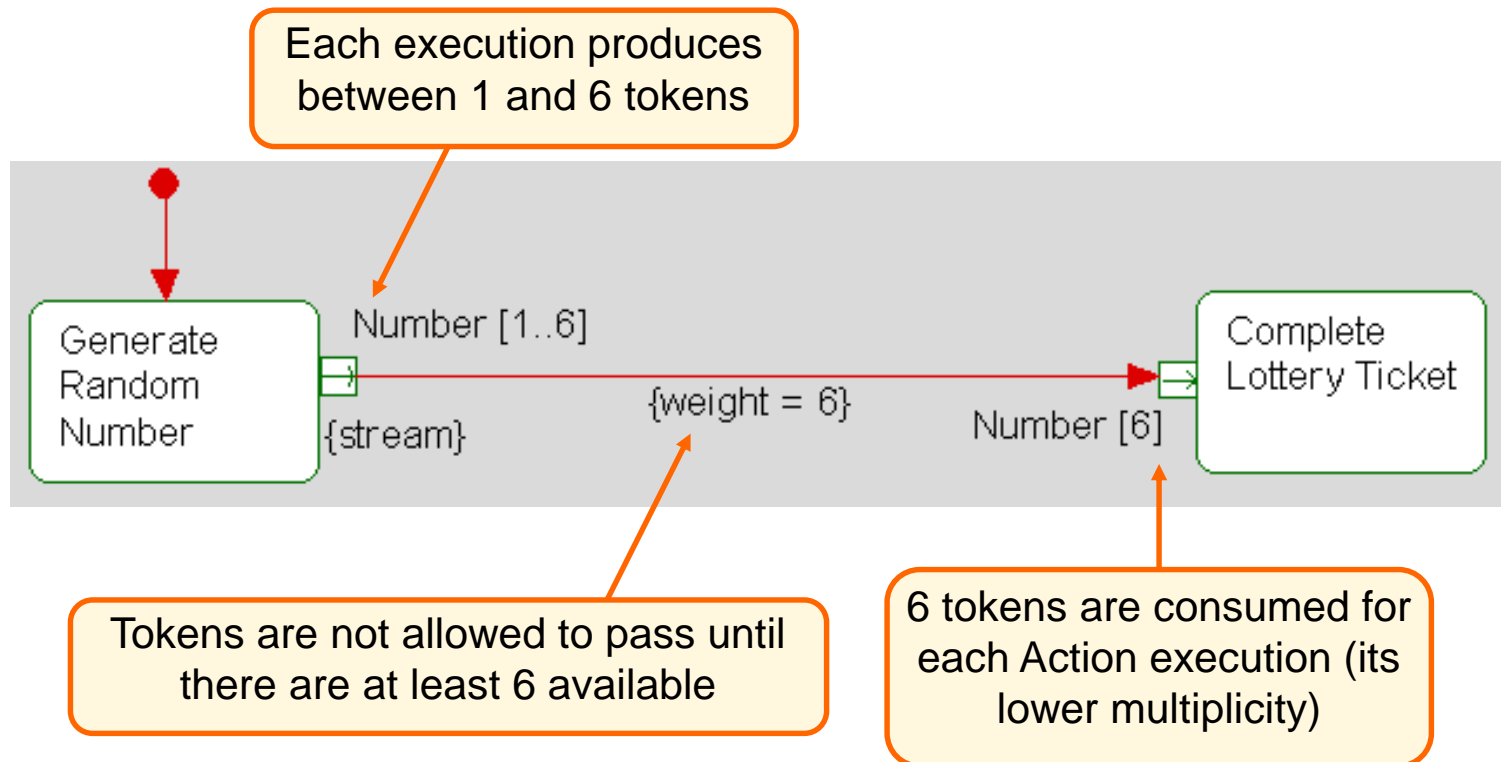
Boolean conditions

- An edge may have a Boolean condition that only allows token flow if the condition evaluates to true.



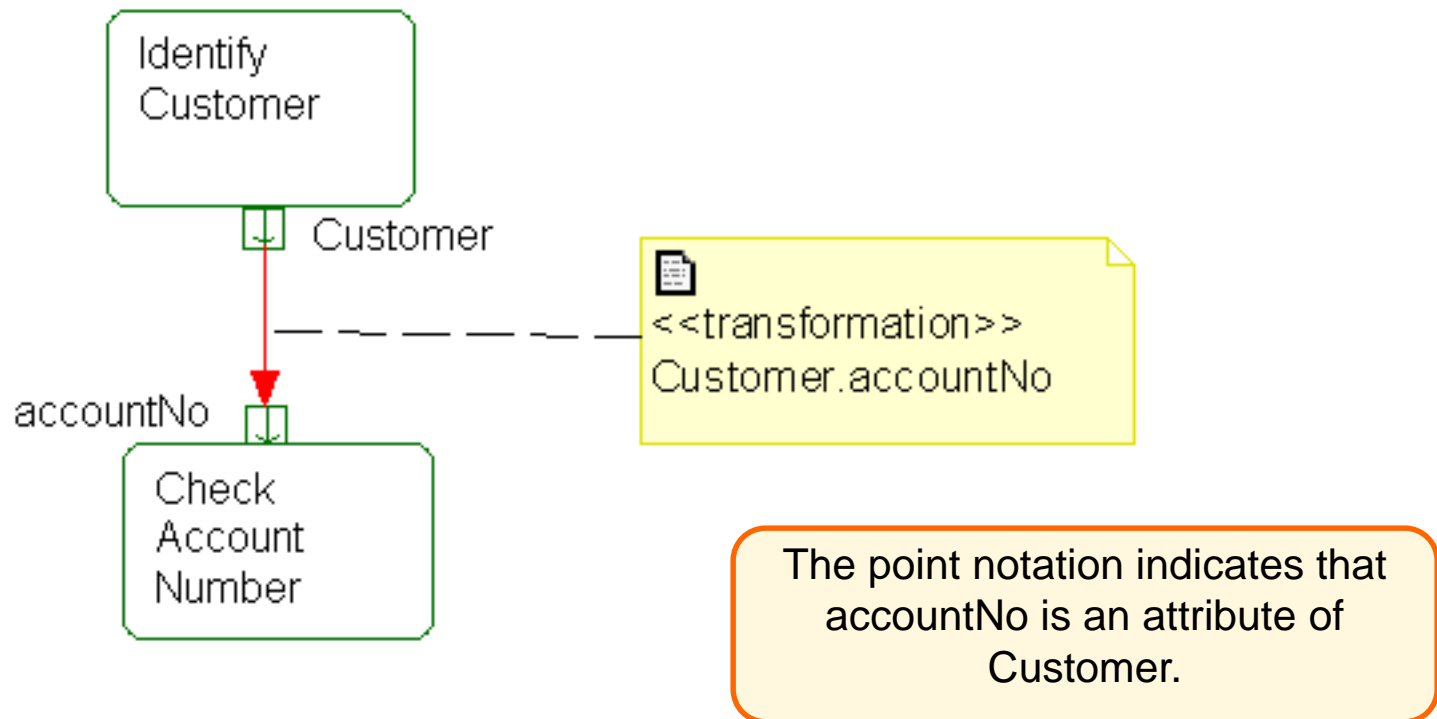
Weight on edges

- Edges may have a {weight} specification.
- The weight dictates the minimum number of tokens that must pass along an edge at one time.
 - ▶ It is evaluated whenever a new token is offered to an edge.
 - ▶ When the minimum number is reached all tokens are offered at the same time.



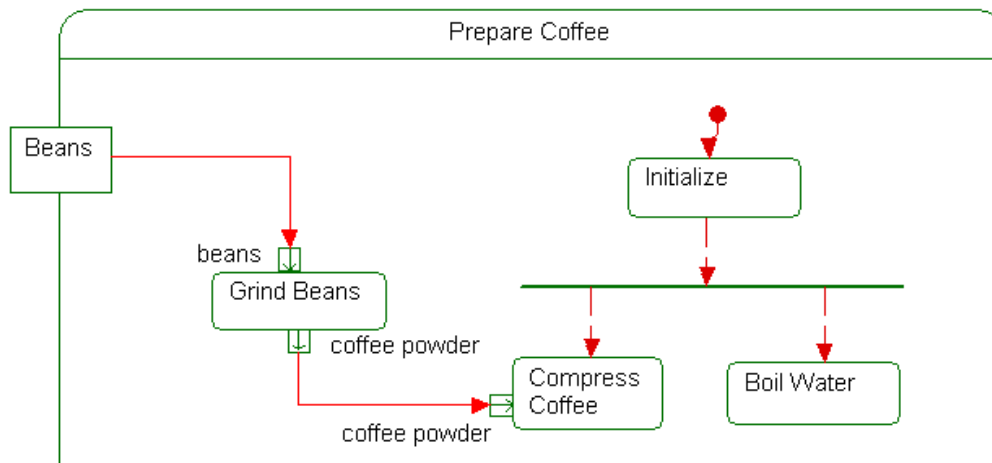
Transformation

- A Transformation transforms an object into another object.
 - ▶ That object may be an attribute / relation of the first



Forks

- Forks are subject to extra rules in the token flow model
 - ▶ When a token hits a Fork, it is copied to all outgoing edges
 - ▶ Forks are the only node where control tokens can wait.

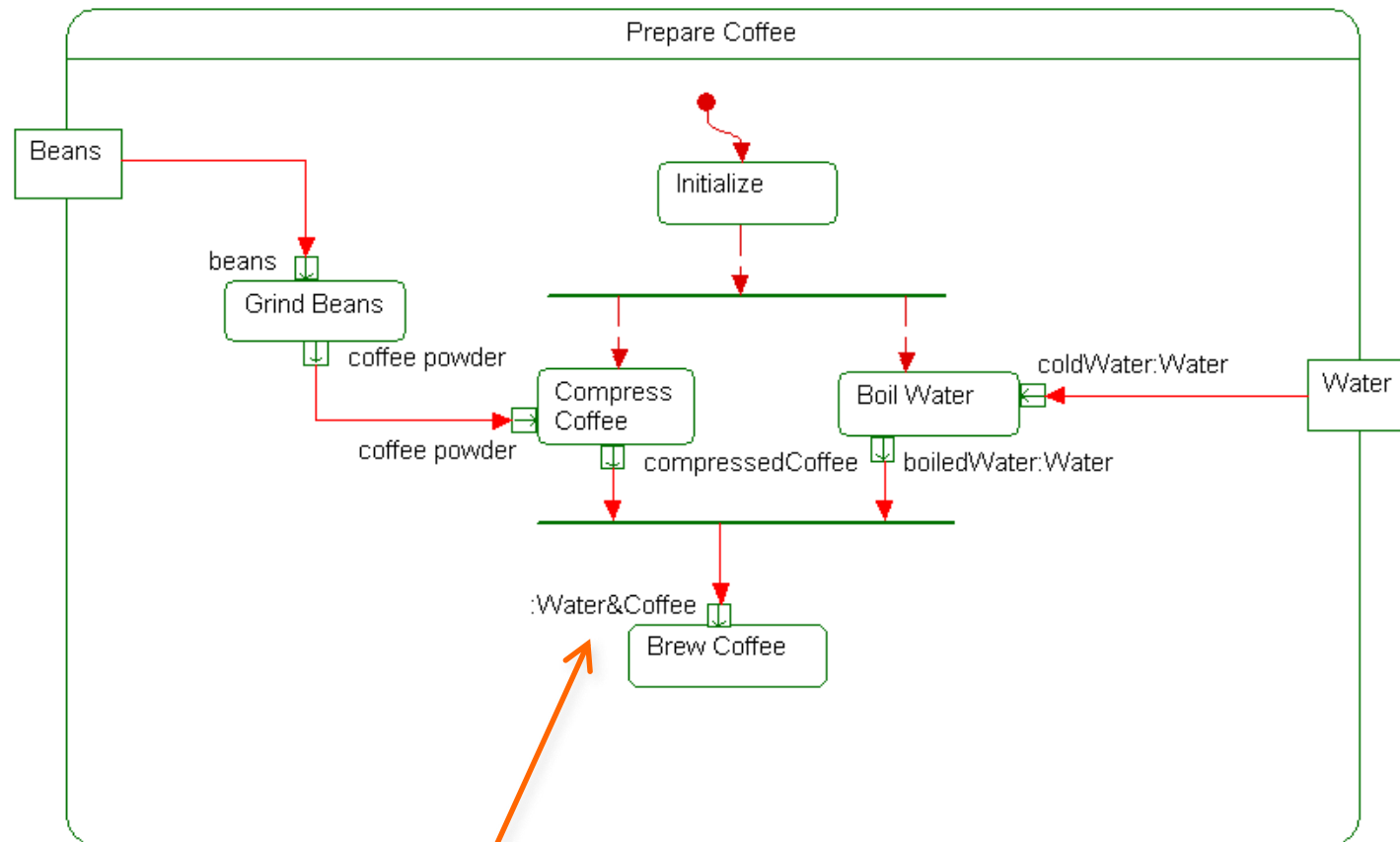


Compress Coffee must wait until coffee powder is available.
However, Boil Water may start immediately.

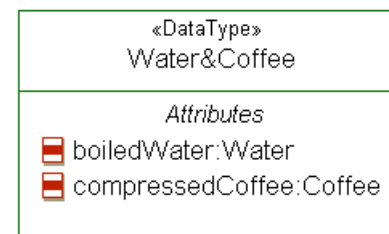
Joins

- Joins still synchronize flows, however they are also subject to extra rules in the token flow model.
- When tokens arrive at a Join:
 - ▶ If they are all control tokens:
 - A single control token is presented at the outgoing edge.
 - ▶ If they are all object tokens:
 - All tokens are presented at the outgoing edge.
 - Exception: If multiple tokens refer to the same object then only one is presented.
 - ▶ If they are mixed:
 - All object tokens are presented.
 - Control tokens are deleted.

Join Example

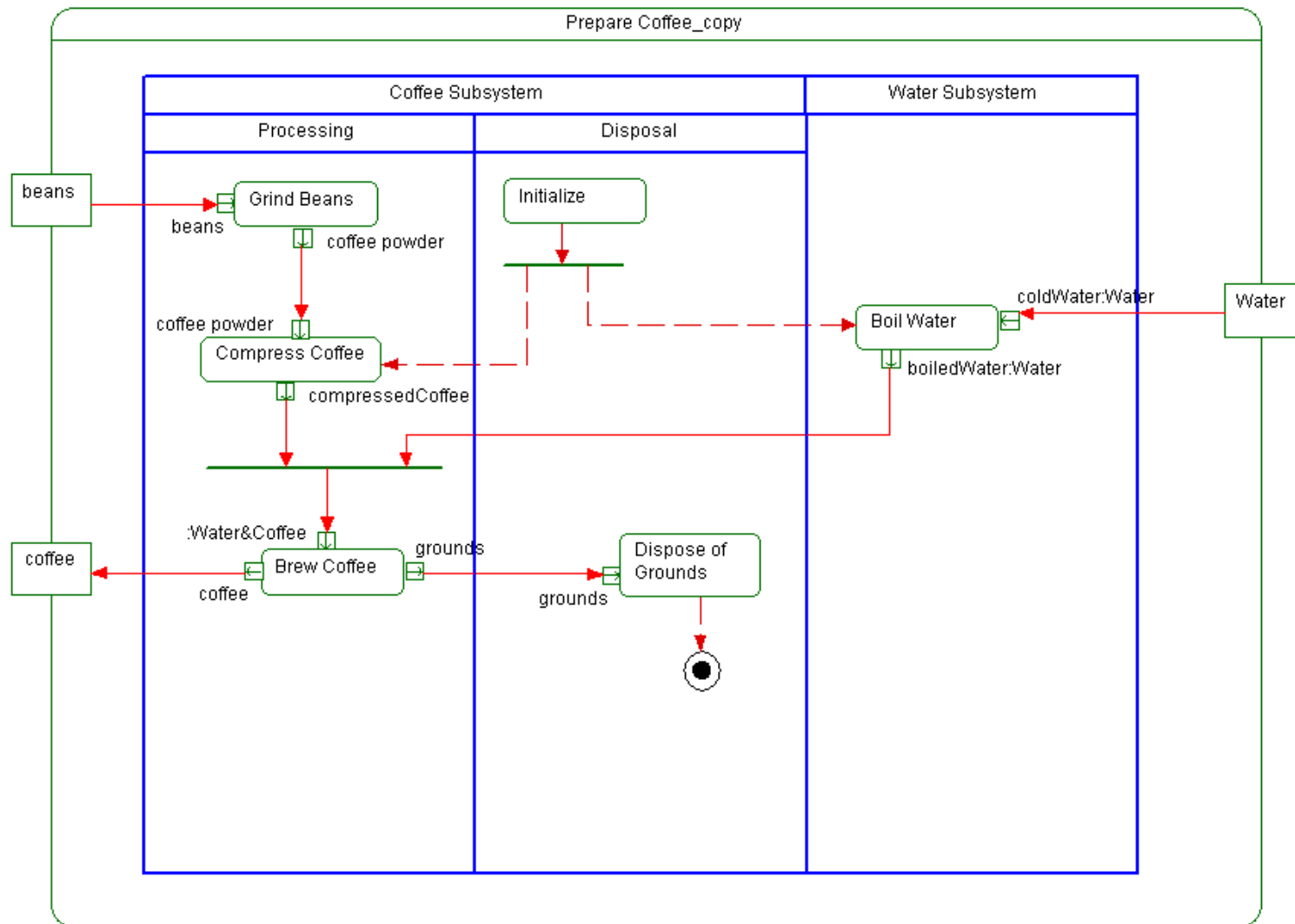


Note the use of a complex type



Nested Swimlanes

- Swimlanes may be nested to show allocation:





IBM Software Group

SysML™ 1.0 for Systems Engineers

Advanced Activity Modeling – Invoking Other Behavior

A horizontal bar containing a series of small, square icons. From left to right, the icons include: a green square, a yellow square, a red square, a purple square, a cyan square, a grayscale image of a highway interchange, a circular arrow icon, a grayscale image of a woman's face, a grayscale image of a hand holding a small object, a grayscale image of a person's arm, and several grayscale squares of varying shades.

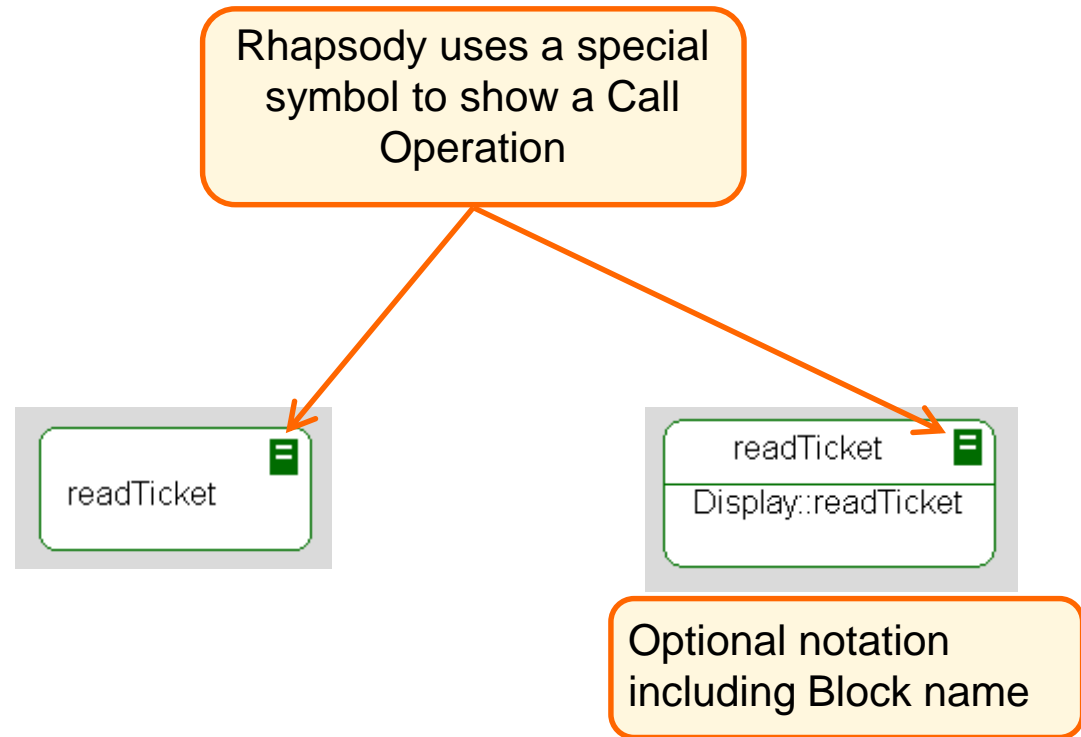
Rational® software

Invoking Behavior

- Activities have two Action types that allow invocation of other behaviors:
 - ▶ Call Behavior
 - ▶ Call Operation

Call Operations

- A Call Operation invokes an Operation call on a target Block / Part.

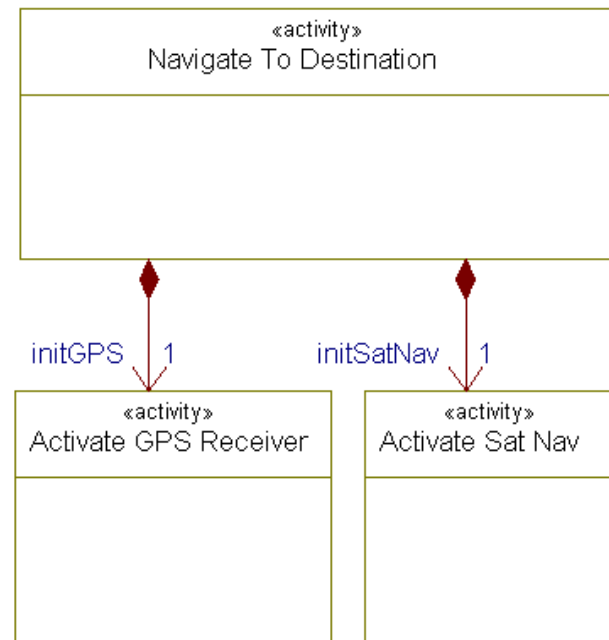


Notes: In Rhapsody, the operation must exist in order for this to work.

Operations can be dragged onto Activity diagrams – they then automatically become call operations.

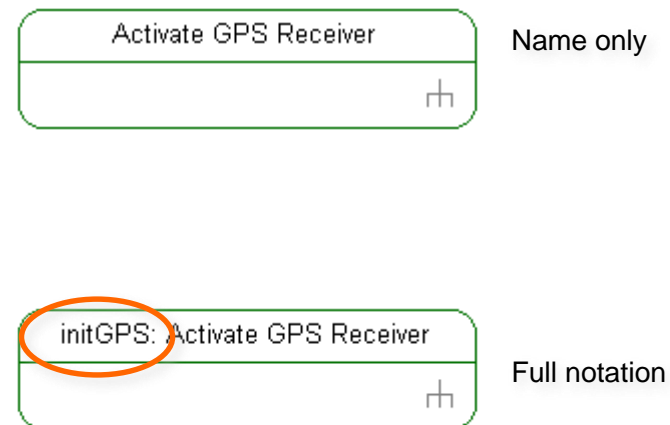
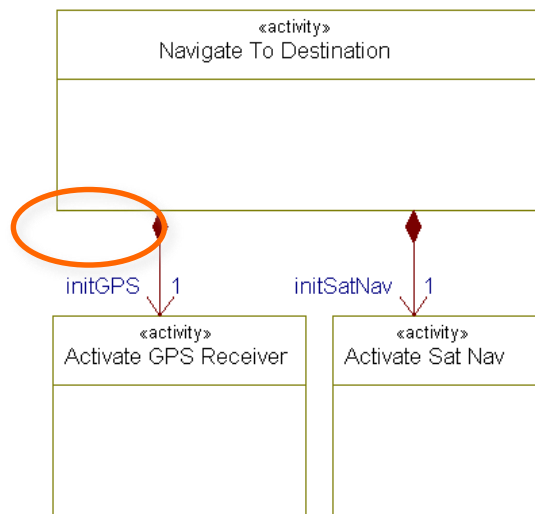
Activities as Classifiers

- Activities are *Classifiers*.
 - ▶ Activities may invoke other Activities
 - ▶ This may be depicted structurally using a Block Definition Diagram
 - ▶ The actual point at which the other Activity is invoked is called a Call Behavior

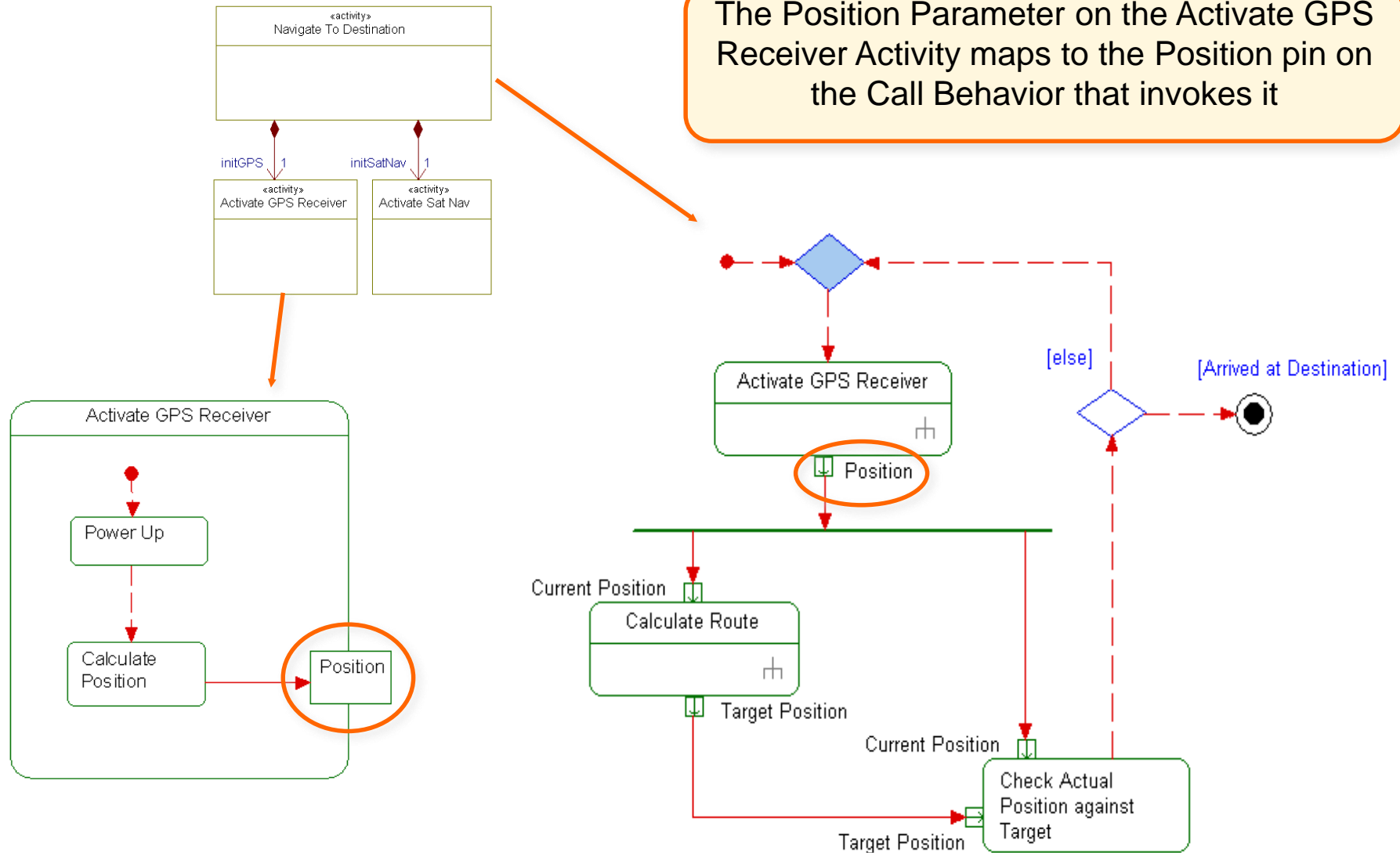


Call Behaviors

- Call Behavior Actions call (invoke) other Activities
- Distinguished from actions by the *rake* symbol in the lower right:
 - ▶ May optionally have the same stereotype as the behavior it invokes.
 - ▶ May optionally contain the action name using the colon notation.

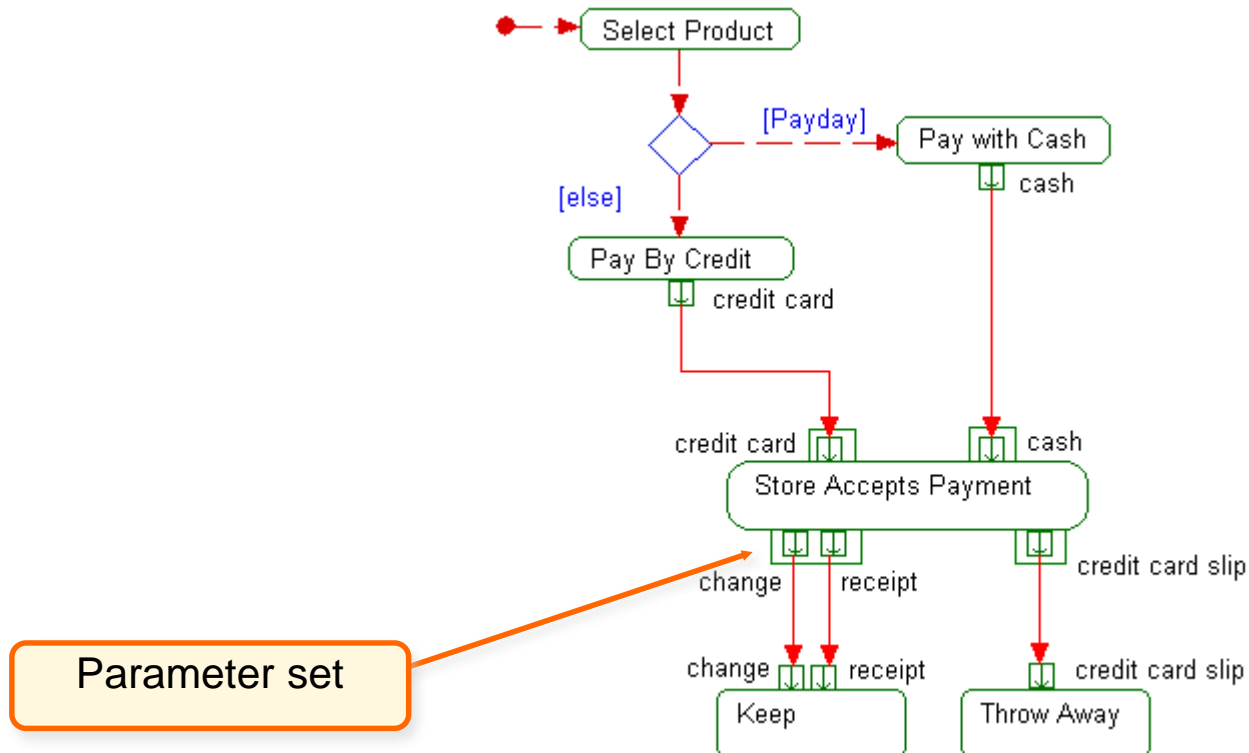


Call behavior pins and parameters





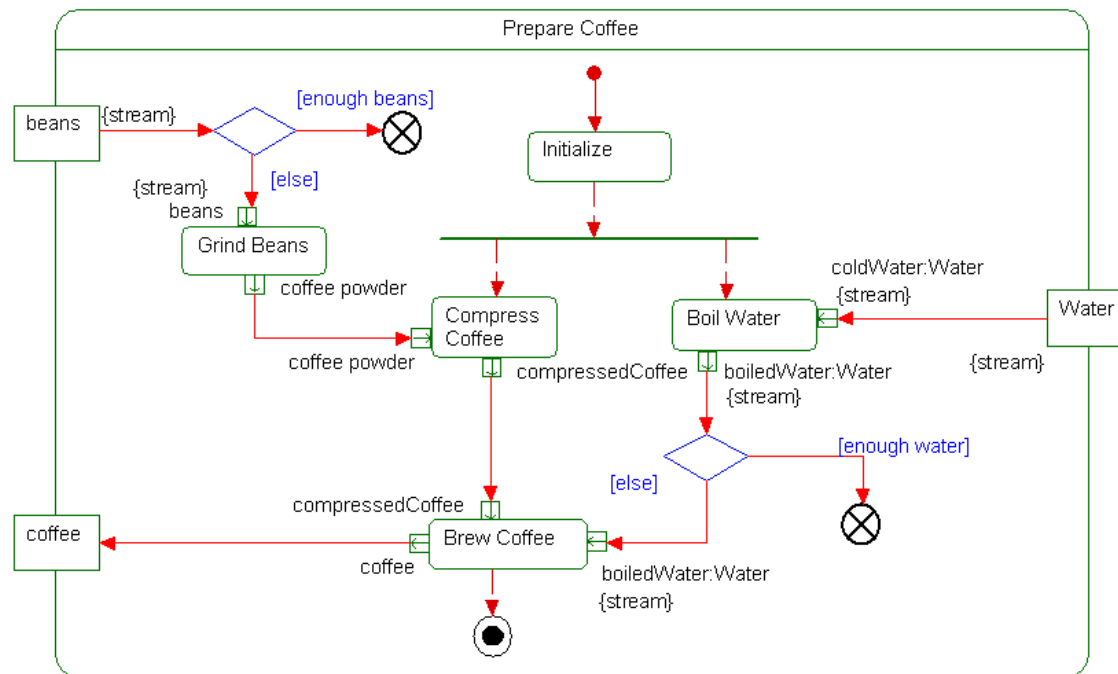
Parameter sets

- Provides alternative sets of inputs or outputs that an Action or Activity may use:
 - ▶ Parameters sets are mutually exclusive entry / exit conditions
 - Although the same pin/parameter may appear in multiple sets



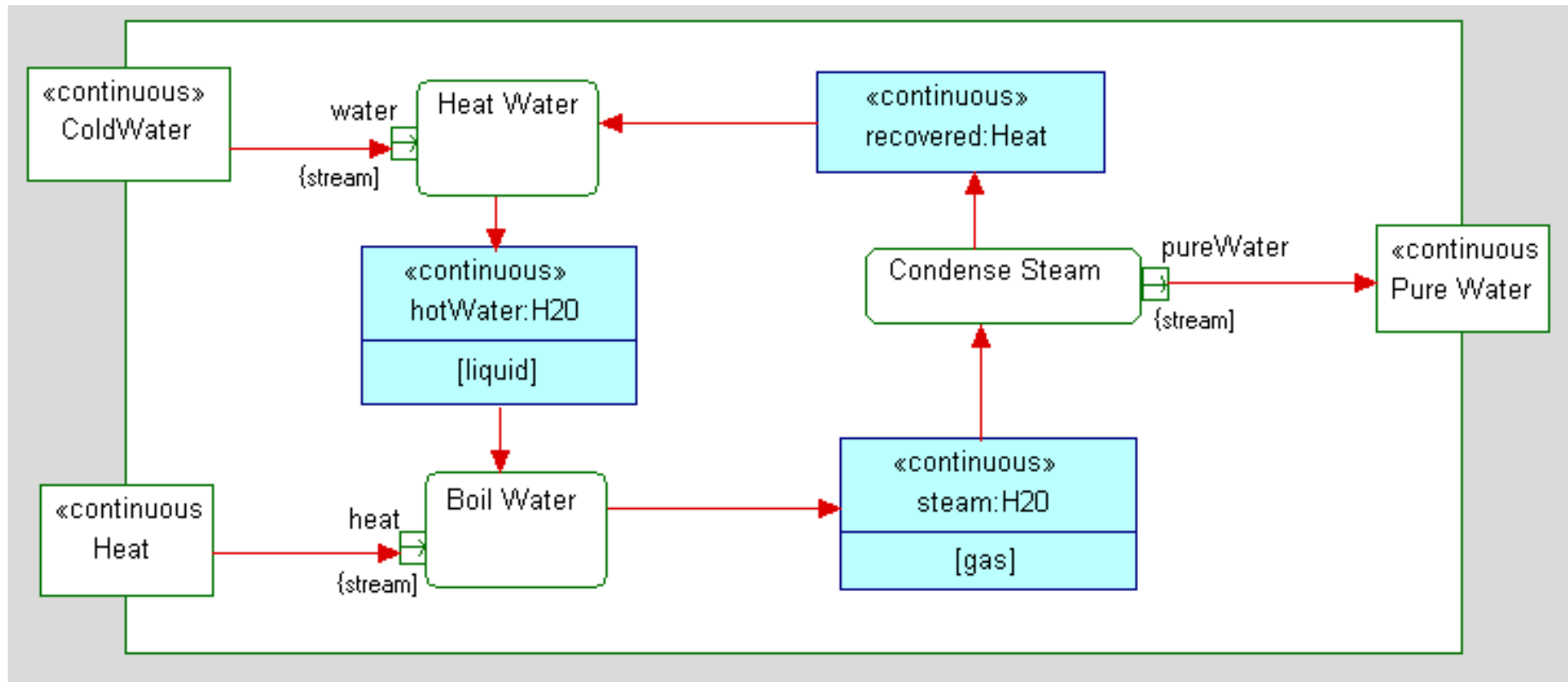
Activity final vs. flow final

- Here the Prepare Coffee Activity can continually accept beans.
 - ▶ When there are enough to make coffee it can still accept them but they 'sink' into a Flow Final. 
 - ▶ The Activity continues until a token reaches the Activity Final 



Continuous modeling

- <<continuous>> means that the arrival time between tokens approaches zero
 - ▶ Representative of many physical processes, for example:

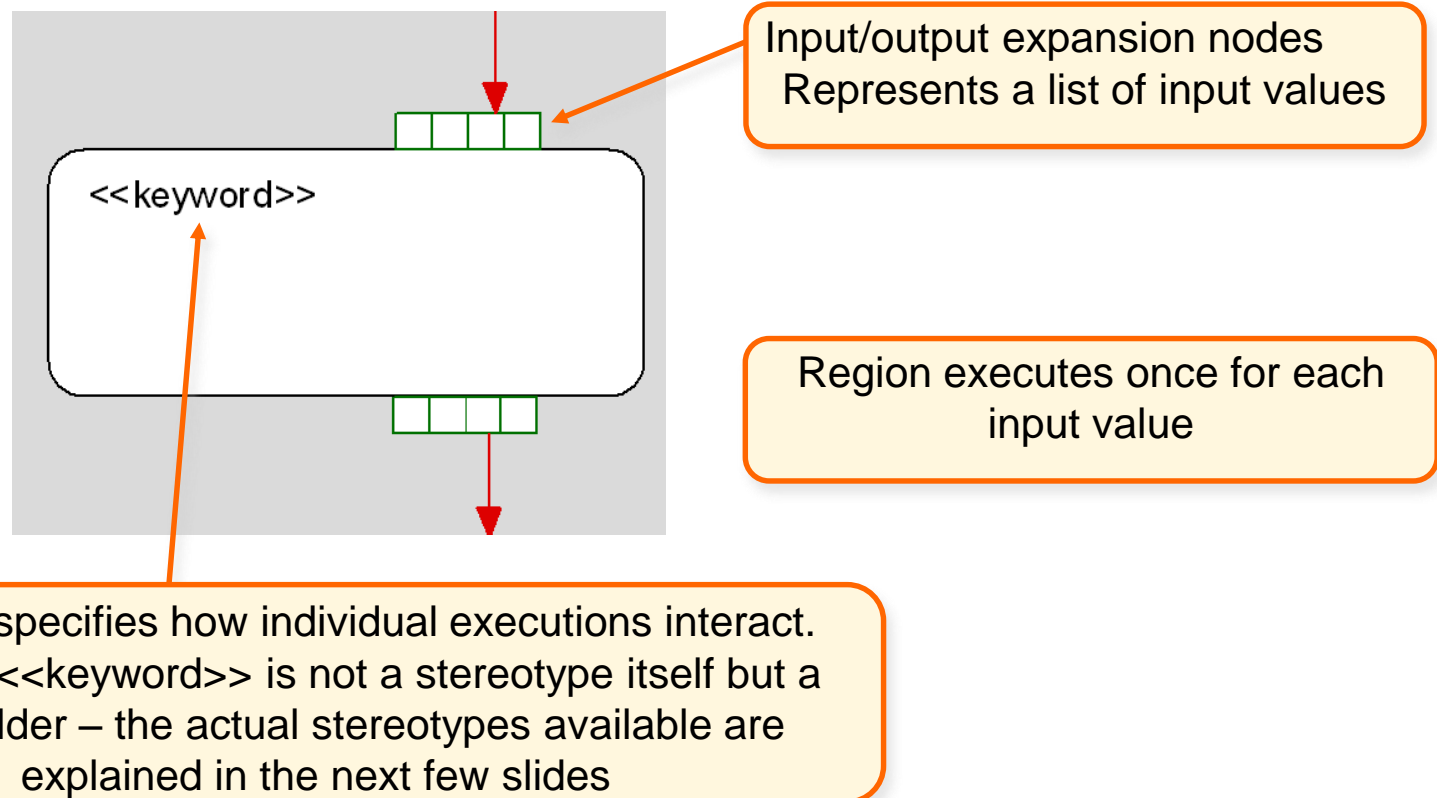


<<rate>>

- <<rate>> specifies:
 - ▶ The expected number of object/data tokens that flow per time interval
- <<rate>> applies to the number of data tokens expected to flow – not the *rate of change* of the actual data itself.
- <<rate>> may be applied to an Edge or a parameter/pin.
 - ▶ Constraint – in order for rate to have any meaning it must be used in conjunction with {stream}.

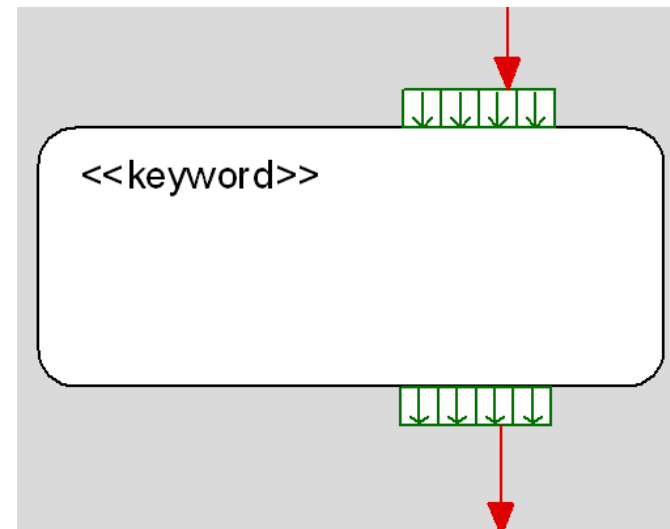
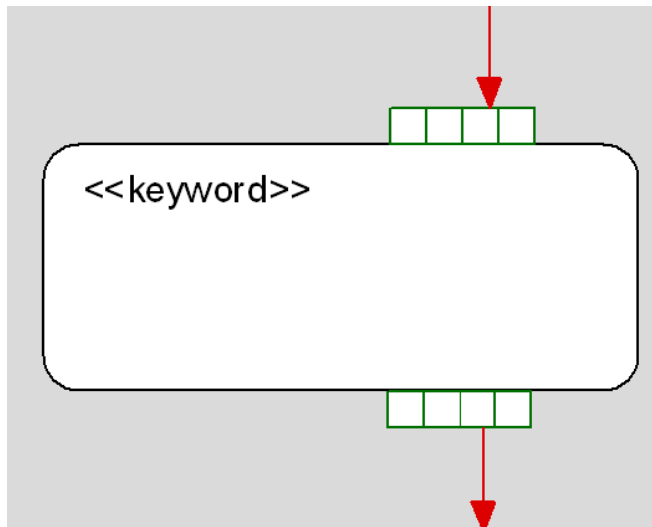
Expansion regions

- An expansion region is a structured activity region that executes multiple times corresponding to the elements of an input collection



Display options – Input/output nodes

- From the SysML specification:
 - ▶ Usually arrows inside and outside the expansion region distinguish input and output expansion nodes.
 - ▶ A small arrow can be used as with pins.

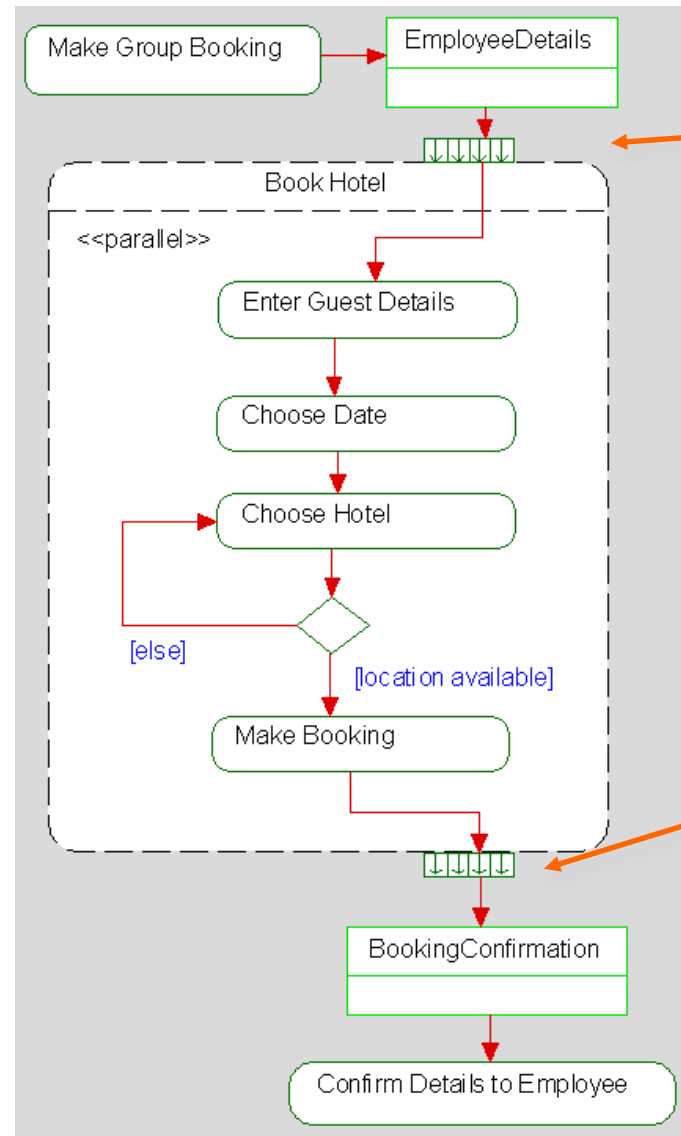


Keywords

- Expansion regions have a keyword indicating how each execution of the region interacts:
 - ▶ **Parallel**
 - Executions may happen in parallel or overlapping in time (but are not required to).
 - ▶ **Iterative**
 - Executions happen in sequence. A token is allowed into the flow when the previous one has completed
 - ▶ **Stream**
 - There is a single execution of the region but the input receives a stream of elements from the collection.
 - Tokens don't have to wait for the previous one to have completed its run.

Example

The Book Hotel sequence of actions occurs many times – once for each guest – each sequence of actions is performed in parallel with the others

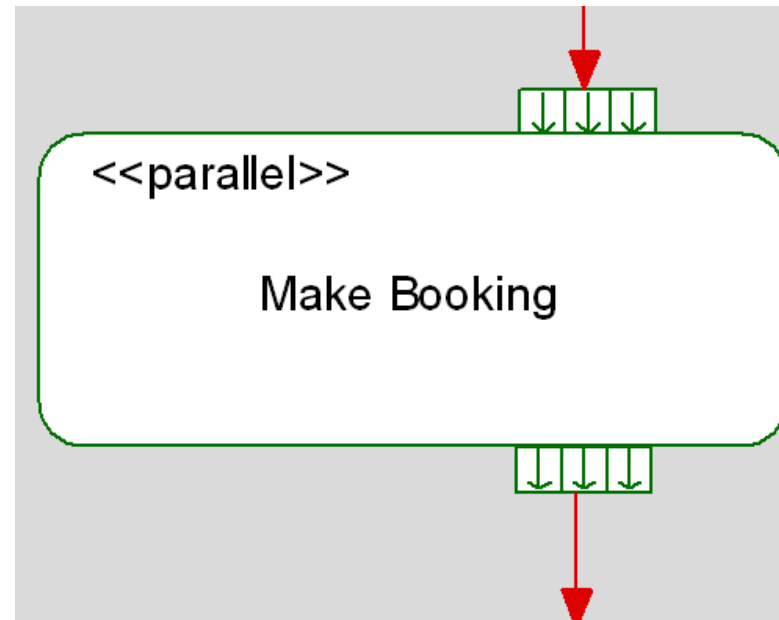


One input – A collection of values

Once all parallel executions have completed, the outputs are made available as a collection

Shorthand for single action

- If the expansion region contains only a single node then the shorthand notation may be used.
 - ▶ The expansion nodes are placed directly on the boundary of the action.





IBM Software Group

SysML™ 1.0 for Systems Engineers

Advanced Activity Modeling – Control Operators



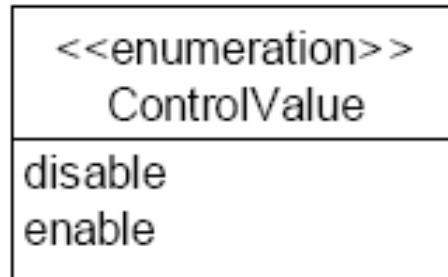
Rational® software

Control operators and values

- A (non streaming) Action:
 - ▶ Begins executing when all of its input tokens are present.
 - ▶ Finishes executing when it has enough tokens for its output conditions to be met.
- SysML extends this notion with the capability to add additional control over when a behavior starts or ends.
- It does this with two new concepts:
 - ▶ Control Operator
 - ▶ Control Value

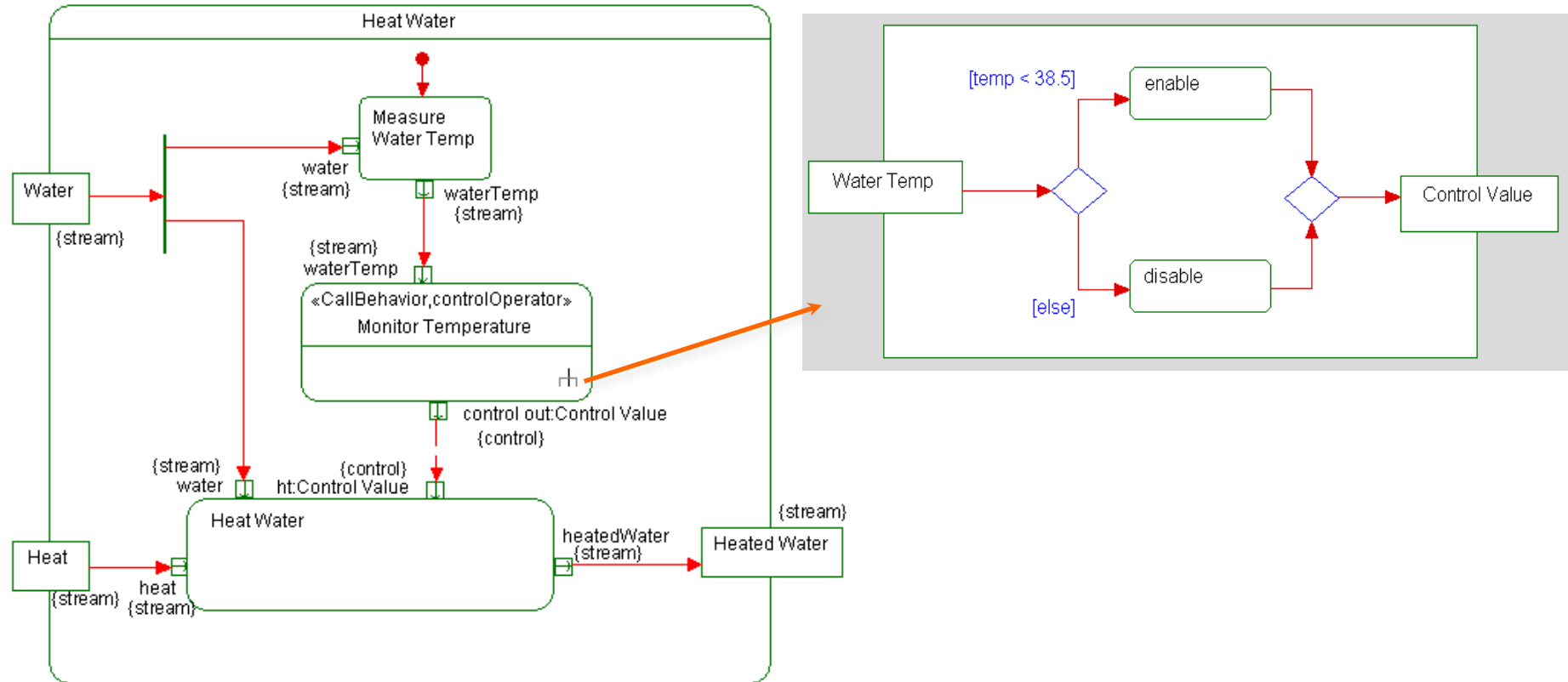
Control Values

- A Control Value is a special kind of token - sent along a control flow - it has the predefined literals:
 - ▶ enable - same semantics as a control token arriving - starts a new execution of the behavior.
 - ▶ disable - terminate the executing behavior.
- Modelers can extend the enumeration with additional literals (for example, suspend, resume).



Control Operators

- A Control Operator is a special kind of behavior
 - ▶ Produces Control Values as its output



Monitor Temperature enables/disables *Heat Water* based on the value of the water temperature. Note that by convention, pins are elided (omitted) inside the Control Operator

Summary

- You are now able to:
 - ▶ Identify how frames are used in Rhapsody Activity diagrams
 - ▶ Identify the different types of tokens and how they are used in Activity diagrams
 - ▶ Describe how nodes, edges, parameters, streams, forks, joins, and swimlanes are used on activity diagrams
 - ▶ Define the two Action types that allow invocation of other behaviors:
 - Call Behavior
 - Call Operation
 - ▶ Confirm that a Control value is a special kind of token
 - ▶ Confirm that a Control operator is a special kind of behavior