Module 3

Activity Diagrams

Vasco Amaral
vma@fct.unl.pt

A bit of history on Activity Diagrams

Gentle warning… neglecting the next couple of slides has been a source for HUGE misunderstandings concerning Activity Diagrams

On the origins of activity diagrams

- Often called “Object-Oriented flowcharts”
- You model a process as an activity consisting on nodes connected by edges (so, a graph)
- In UML 1.*, activity graphs **were** a special case of state machines (we will discuss these later, this semester)
  - Every state had an entry action that specified some process, or function, that occurred when the state was entered.

Guess what? UML 1.* is no more. The semantics behind activity diagrams changed dramatically. You will find a huge amount of outdated resources online on activity diagrams. Beware. They get to be incredibly wrong, quite often.

Activity Diagrams in UML 2.*

- Completely **new semantics based on Petri Nets**
  - The Petri Net formalism provides greater flexibility in modelling different types of flow
  - There is now a clear distinction in UML between activity diagrams and state machines

Activities can be attached to any modelling element

- Activities model the element's behavior
- Frequently used with
  - use cases
  - classes
  - interfaces
  - components
  - collaborations
  - operations
- Activity diagrams should focus in communicating a particular aspect of a system's dynamic behavior
  - Keep them as simple as possible… but not too simplistic

Common usages for activity diagrams

- During analysis
  - To model the flow in a use case in a graphical way that is easy for the stakeholders to understand
  - To model the flow between use cases, through the interaction overview diagram
- During design
  - To model the details of an operation
  - To model the details of an algorithm
- In business modelling
  - To model a business process

As with any other UML diagram, remember that one of its functions is to communicate to stakeholders. Like use cases, activity diagrams can be effective in communication, if we keep them as simple as possible.

Activities are networks of nodes connected by edges

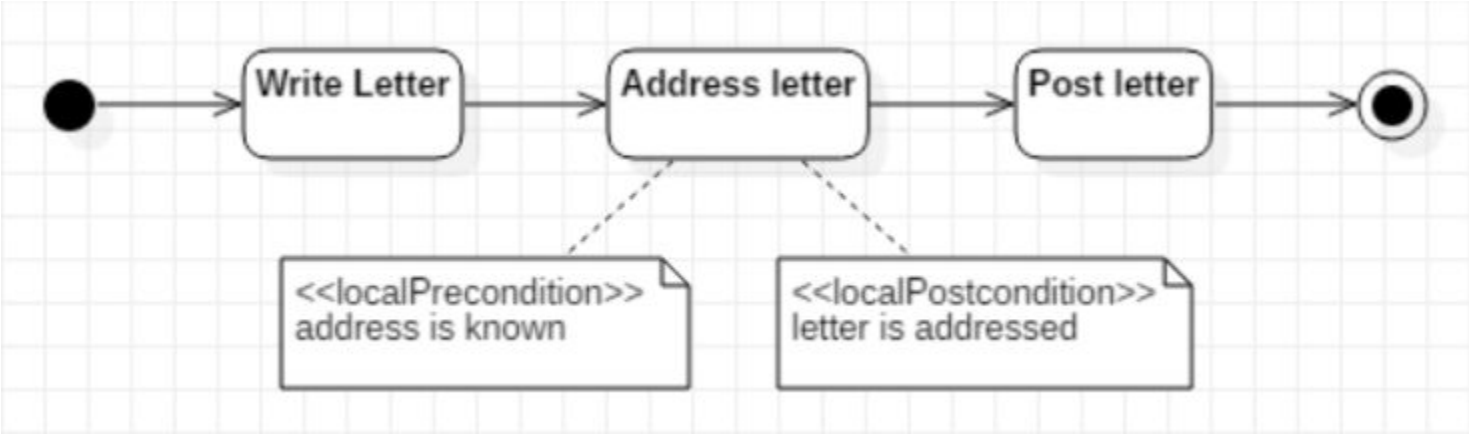
- **Action nodes** - represent discrete units of work that can be regarded as atomic within the activity
- **Control nodes** - control the flow through the activity
- **Object nodes** - represent objects used in the activity

Activities are networks of nodes connected by edges

- **control flows** - represent the flow of control through the activity
- **object flows** - represent the flow of objects through the activity

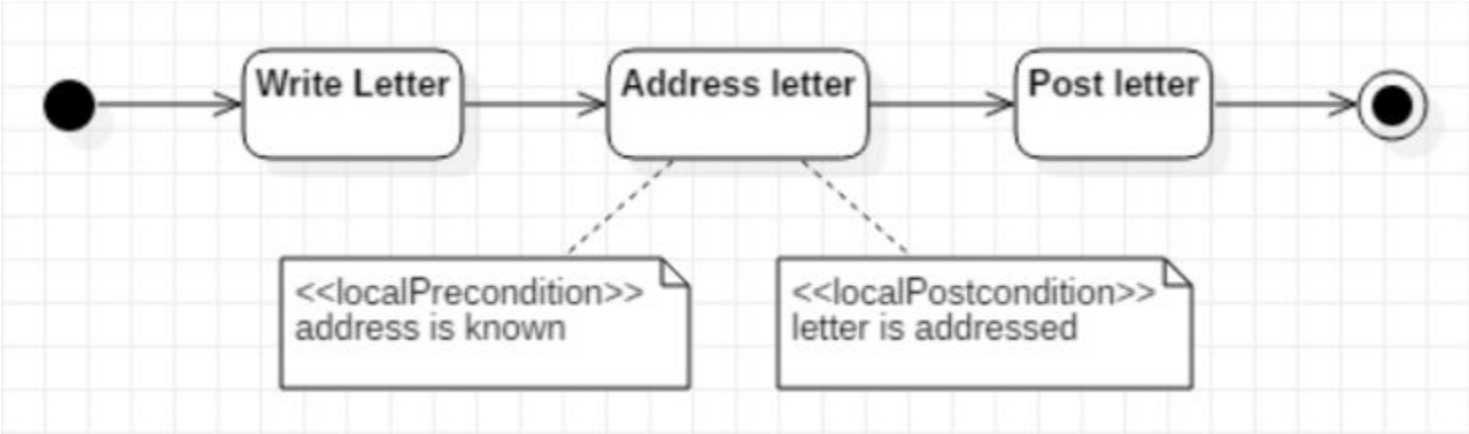
The send letter activity

Send letter
precondition: know topic for letter
postcondition: letter sent to address



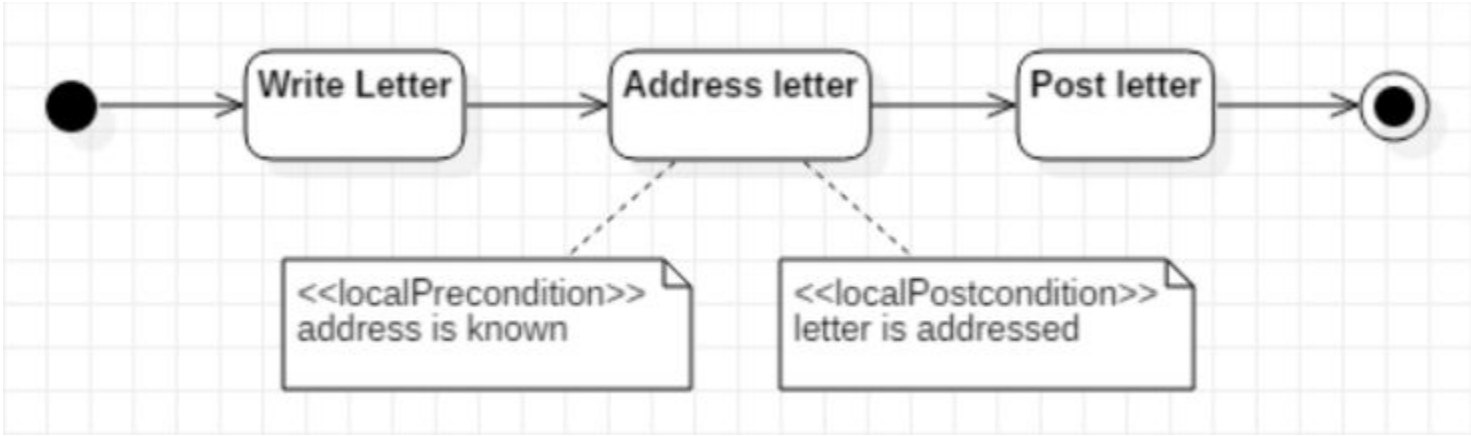
Activities may have preconditions that must hold, before the activity can start

Send letter
precondition: know topic for letter
postcondition: letter sent to address



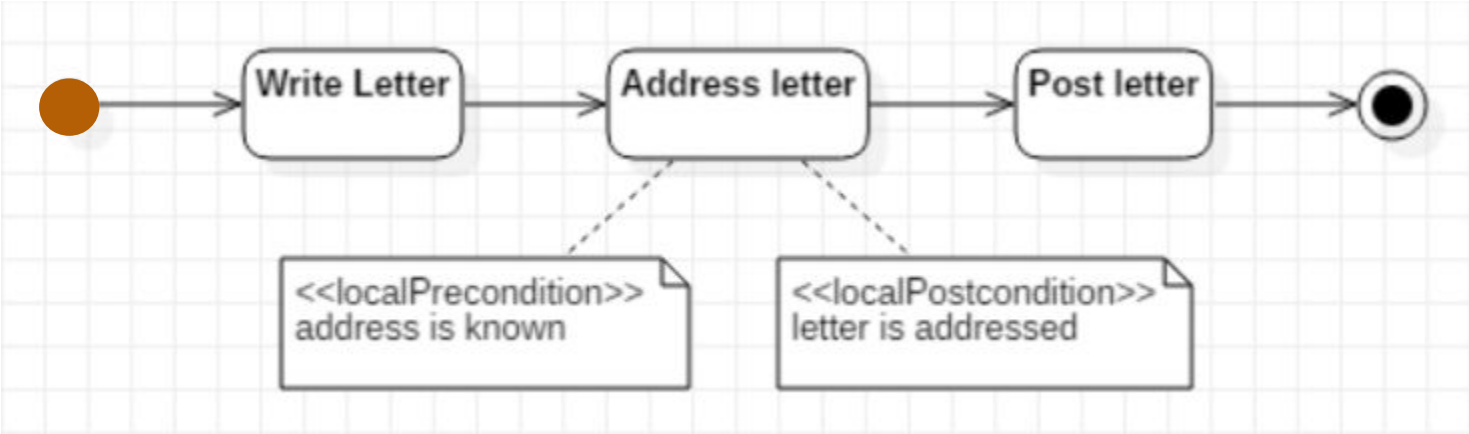
Activities may have postconditions that must hold, after the activity ends

Send letter
precondition: know topic for letter
postcondition: letter sent to address



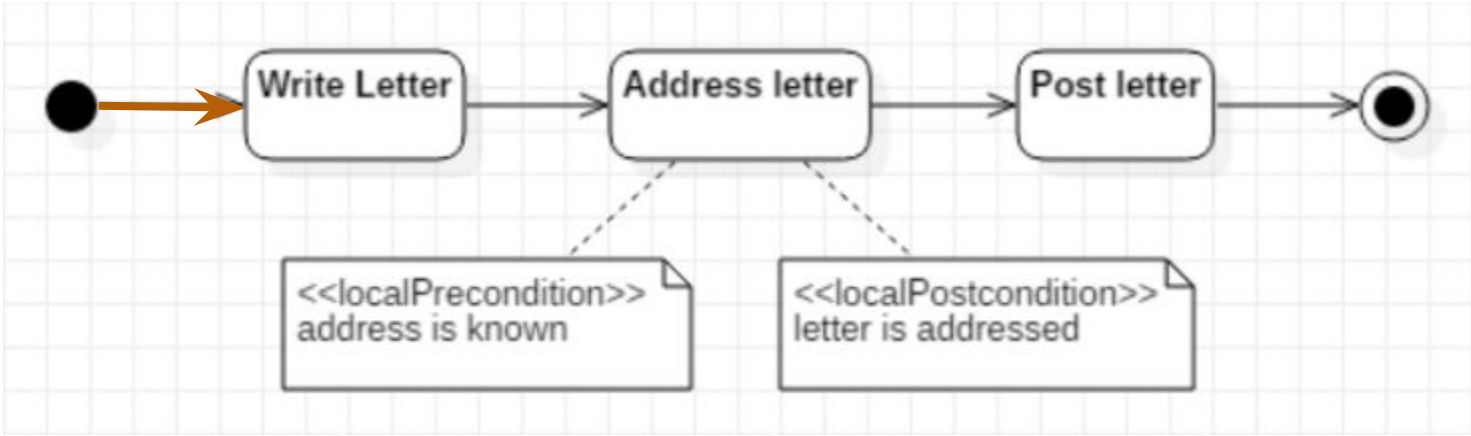
Activities start with a single control node: the initial node

Send letter
precondition: know topic for letter
postcondition: letter sent to address



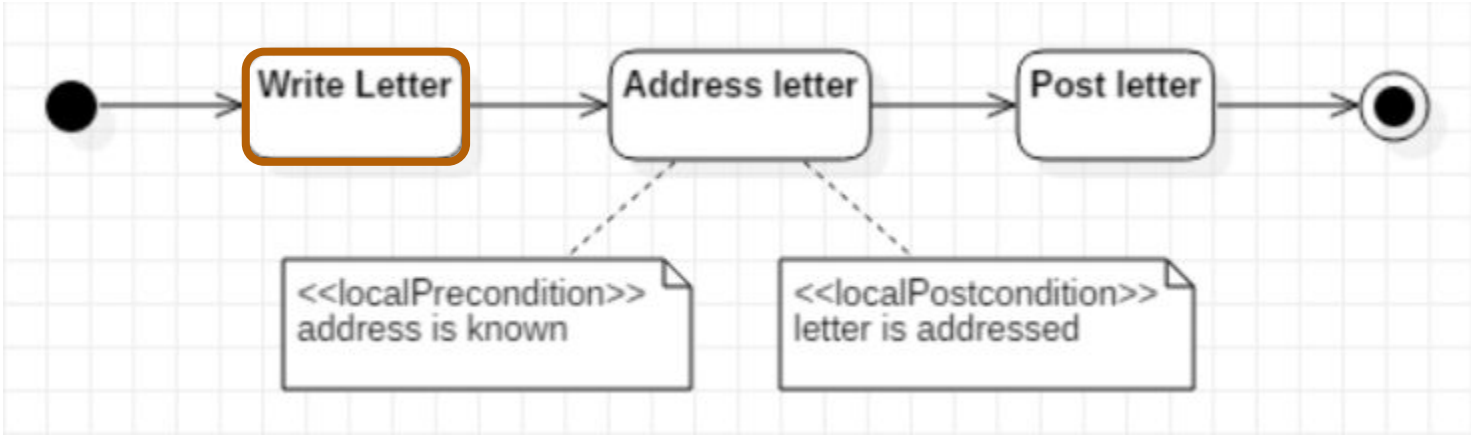
Control transitions from an origin to a destination node through a control flow

Send letter
precondition: know topic for letter
postcondition: letter sent to address



Then, the control transitions no the action node Write Letter

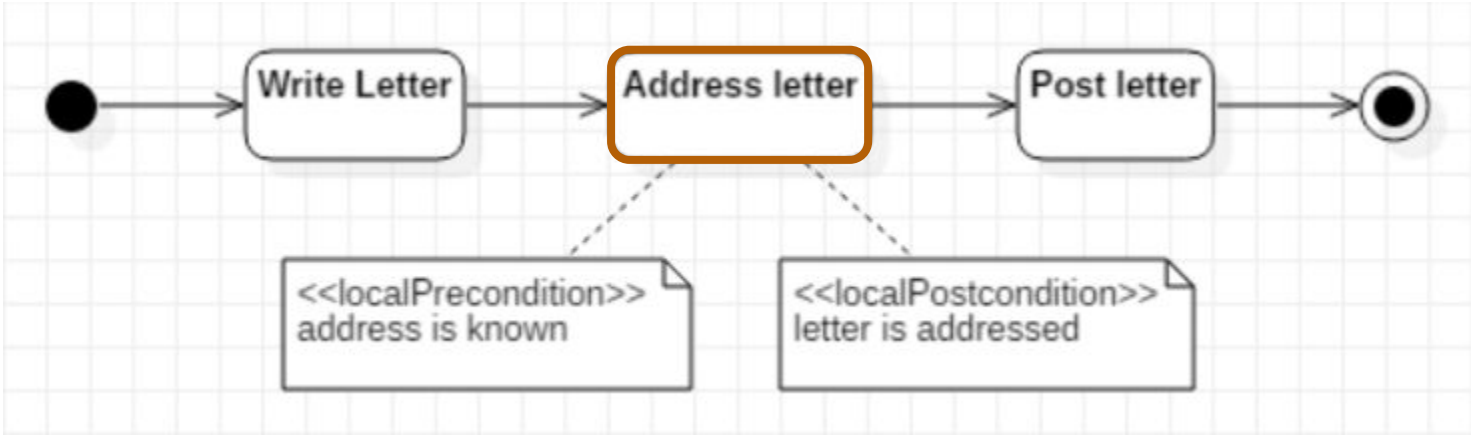
Send letter
precondition: know topic for letter
postcondition: letter sent to address



An action node indicates a piece of work or behavior that is atomic from the perspective of the containing activity. Execution time is considered negligible.

Then, the control transitions no the action node Address letter

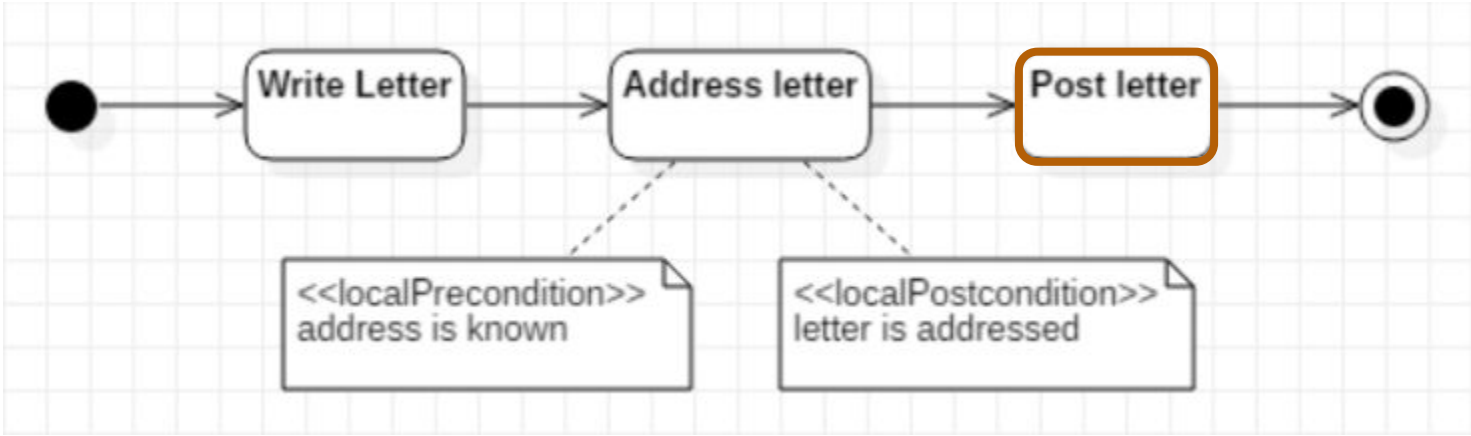
Send letter
precondition: know topic for letter
postcondition: letter sent to address



An action node indicates a piece of work or behavior that is atomic from the perspective of the containing activity. Execution time is considered negligible.

And then, the control transitions no the action node Post letter

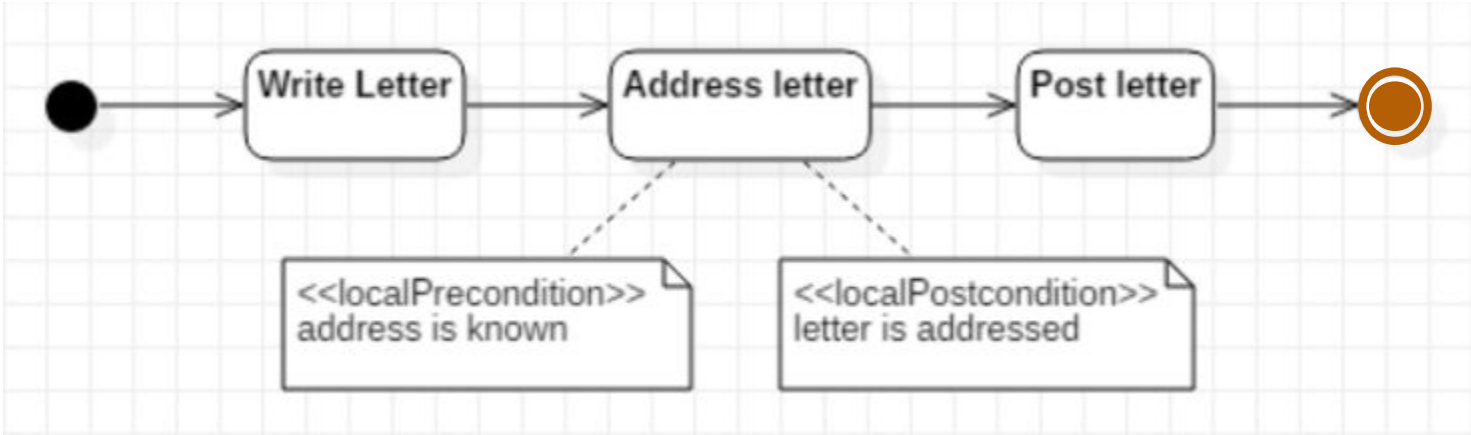
Send letter
precondition: know topic for letter
postcondition: letter sent to address



An action node indicates a piece of work or behavior that is atomic from the perspective of the containing activity. Execution time is considered negligible.

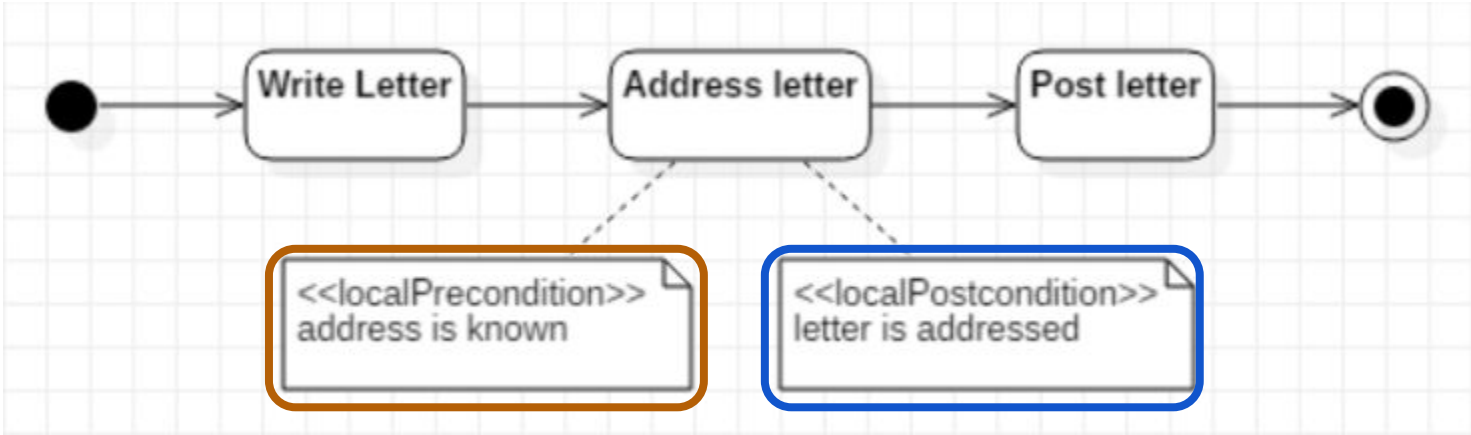
One or more final nodes indicate where the activity terminates

Send letter
precondition: know topic for letter
postcondition: letter sent to address



Each action node may have its own pre- and postconditions

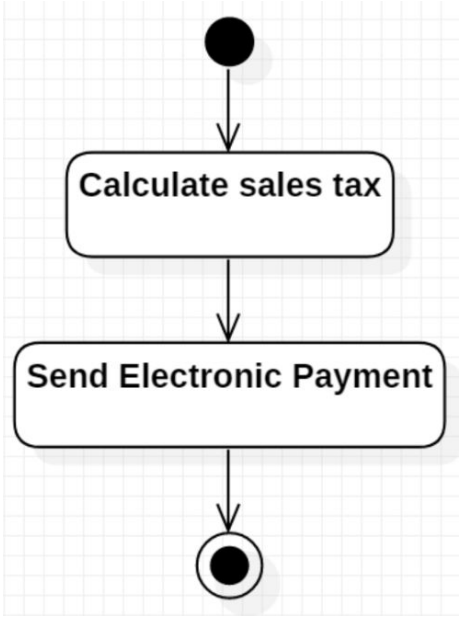
Send letter
precondition: know topic for letter
postcondition: letter sent to address



Use cases express behavior as an interaction between actors and the system

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter
Primary Actors: Time
Secondary Actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter
Main flow: 1. The use case starts when it is the end of the business quarter 2. The system determines the amount of sales tax owed to the tax authority 3. The system sends an electronic payment to the tax authority
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax
Alternative flows: none

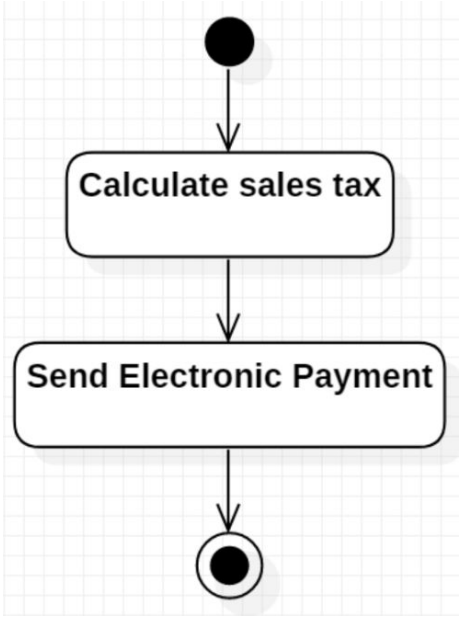
PaySalesTax
precondition: It is the end of the business quarter
postcondition: The Tax Authority receives the correct amount of Sales Tax



Activity diagrams can be used to provide a complementary behavior view to use cases

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter
Primary Actors: Time
Secondary Actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter
Main flow: 1. The use case starts when it is the end of the business quarter 2. The system determines the amount of sales tax owed to the tax authority 3. The system sends an electronic payment to the tax authority
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax
Alternative flows: none

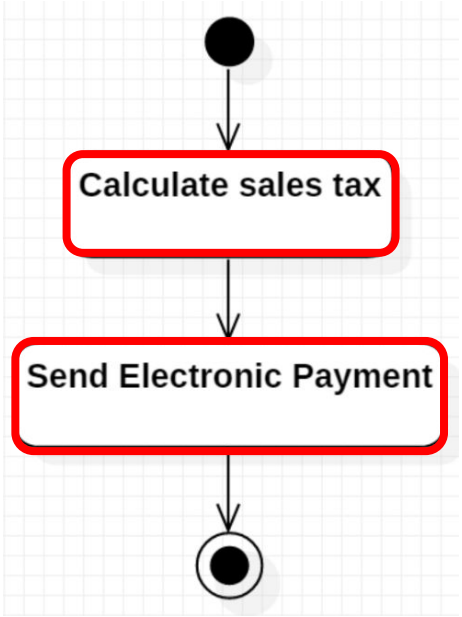
PaySalesTax
precondition: It is the end of the business quarter
postcondition: The Tax Authority receives the correct amount of Sales Tax



Activity diagrams express system behavior as a series of actions

Use case: PaySalesTax
ID: 1
Brief description: Pay Sales Tax to the Tax Authority at the end of the business quarter
Primary Actors: Time
Secondary Actors: TaxAuthority
Preconditions: 1. It is the end of the business quarter
Main flow: 1. The use case starts when it is the end of the business quarter 2. The system determines the amount of sales tax owed to the tax authority 3. The system sends an electronic payment to the tax authority
Postconditions: 1. The Tax Authority receives the correct amount of Sales Tax
Alternative flows: none

PaySalesTax
precondition: It is the end of the business quarter
postcondition: The Tax Authority receives the correct amount of Sales Tax




A GAME OF TOKENS
Generated by Font-Generator.com

A GAME OF TOKENS

Generated by Font-Generator.com

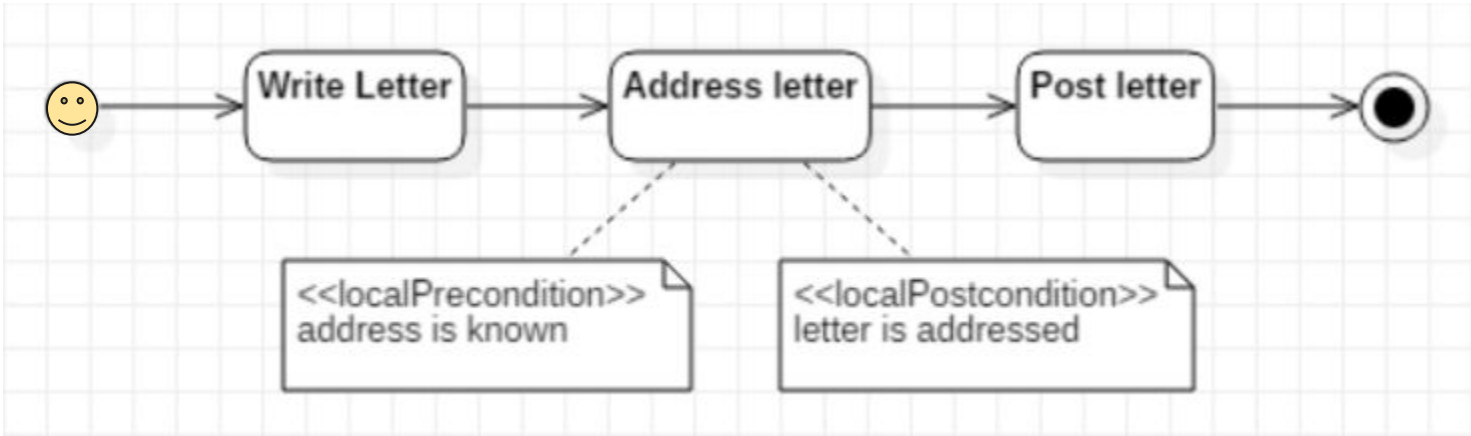
- The **token game** describes the flow of tokens around a network of nodes and edges according to specific rules
- Tokens may represent
  - the flow of control
  - an object
  - some data
- The state of the system is determined by the disposition of its tokens
- Tokens are moved from a source to a target node across an edge
  - Token movement is subject to conditions and only occurs when all those conditions are satisfied

A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

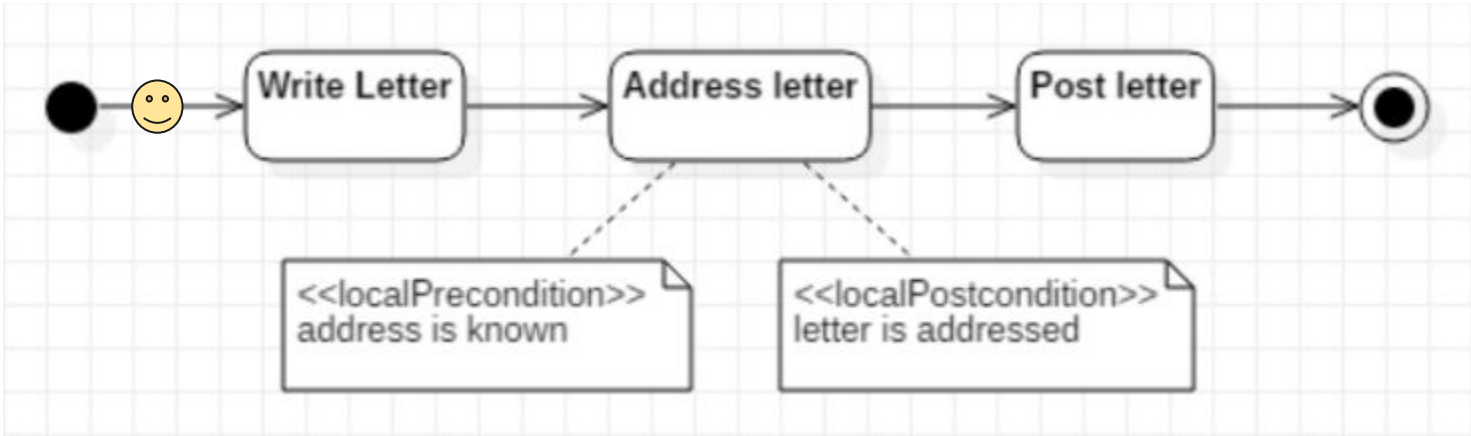


A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

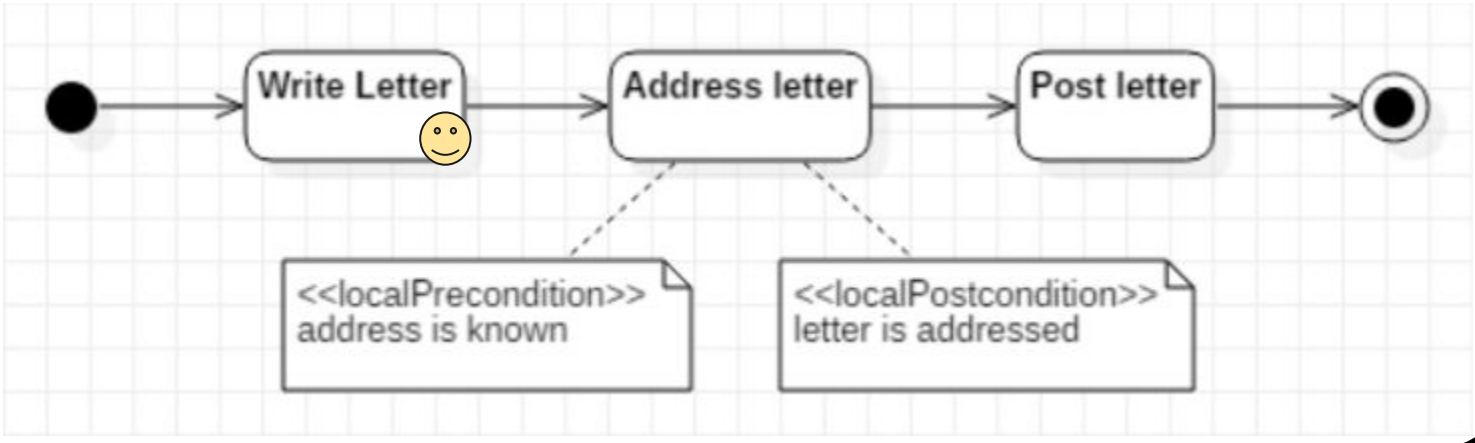


A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

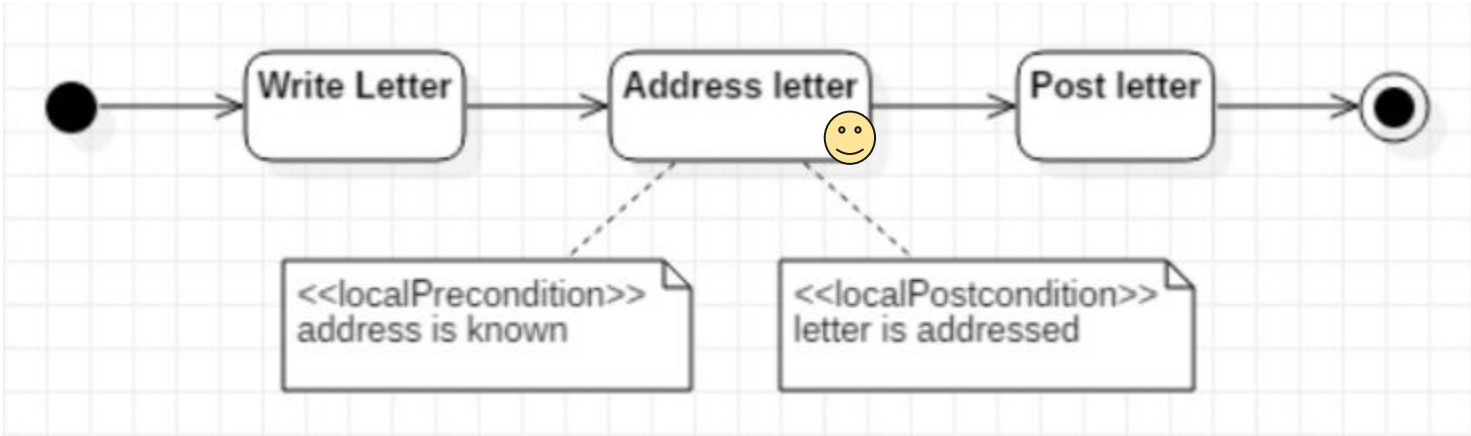


A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

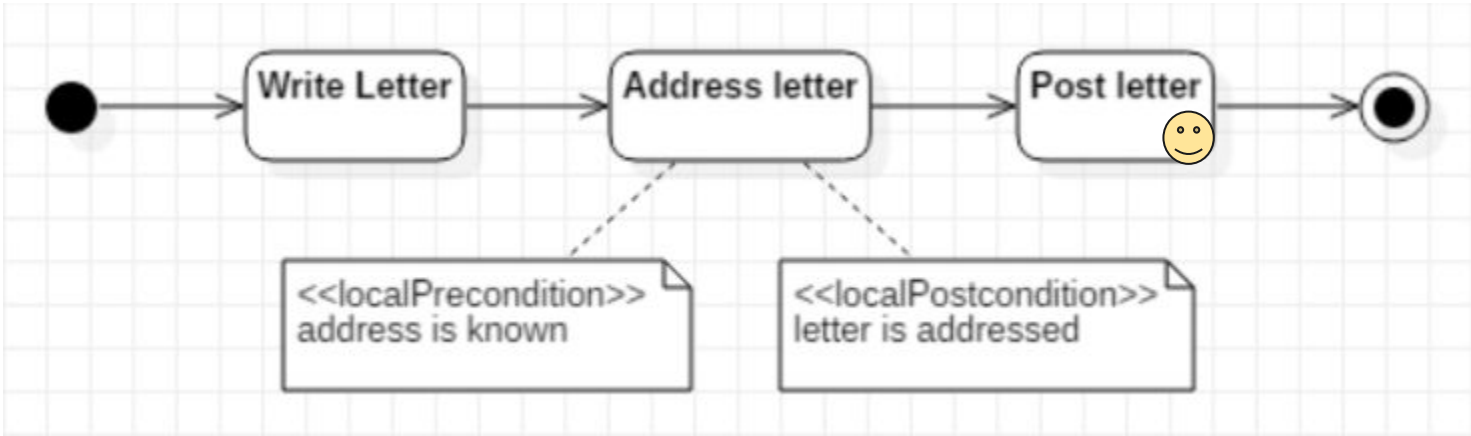


A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

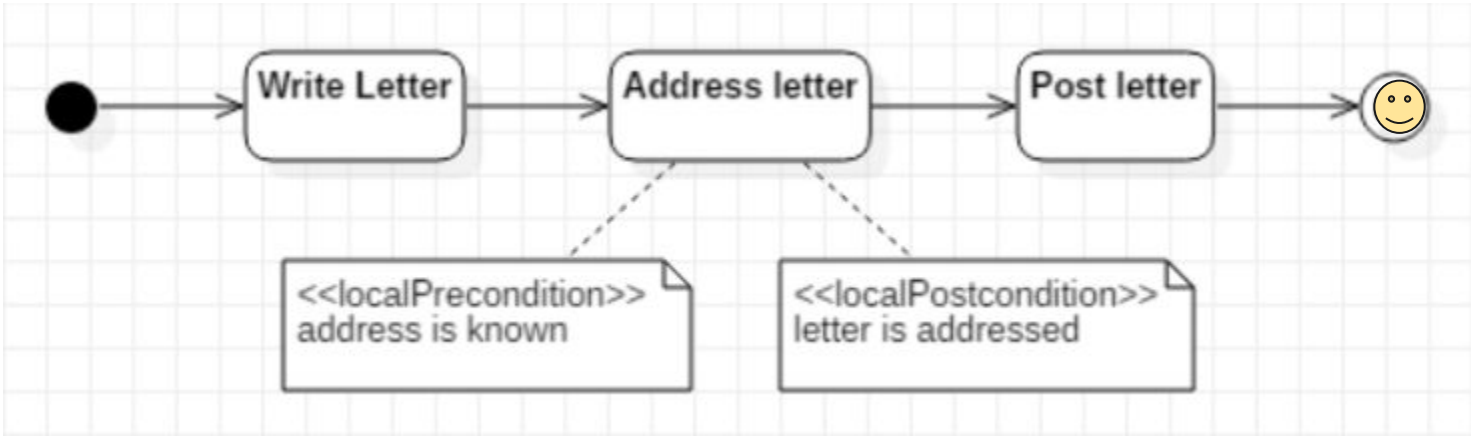


A GAME OF TOKENS

Generated by Font-Generator.com

- Tokens are moved from a source to a target node across an edge
- Token movement is subject to conditions and only occurs when all those conditions are satisfied
- Conditions vary depending on node type

Send letter
precondition: know topic for letter
postcondition: letter sent to address

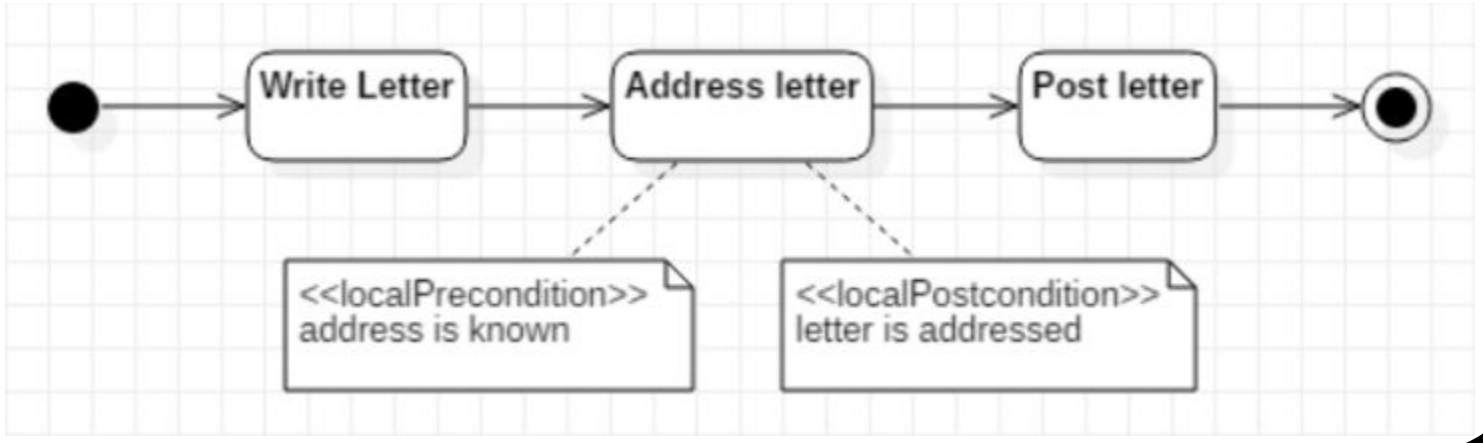


A GAME OF TOKENS

Generated by Font-Generator.com

- Conditions for action nodes
  - postconditions of the source node
  - guard conditions on the edge
  - preconditions of the target node

Send letter
precondition: know topic for letter
postcondition: letter sent to address

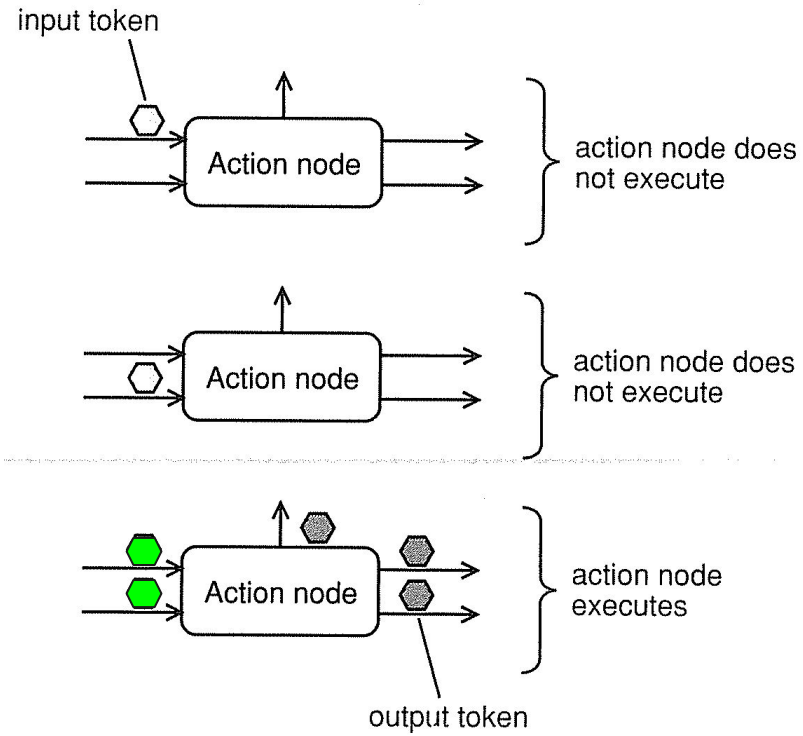


Action nodes

A GAME OF TOKENS

Generated by Font-Generator.com

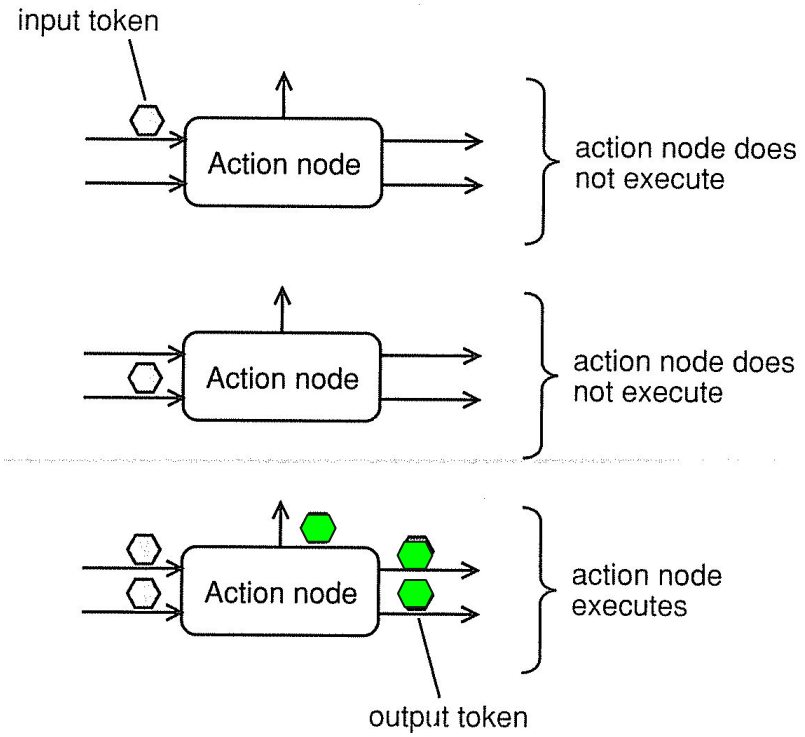
- Action nodes execute when
  - There is a token simultaneously on each of their input edges **AND**
  - the input tokens satisfy all of the action node local preconditions



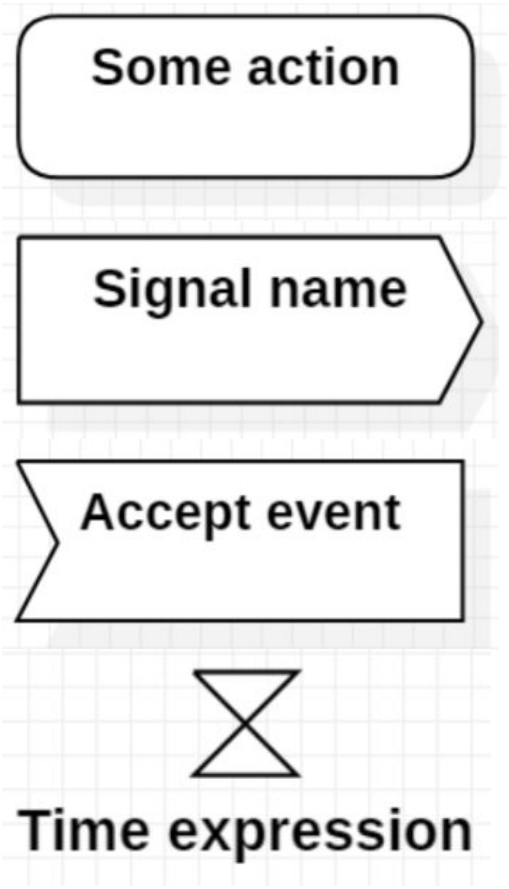
A GAME OF TOKENS

Generated by Font-Generator.com

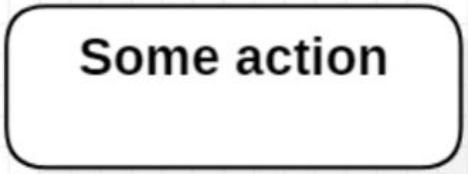
- Action nodes offer control tokens on all their output edges
  - This is an implicit fork
  - Activity diagrams are inherently concurrent

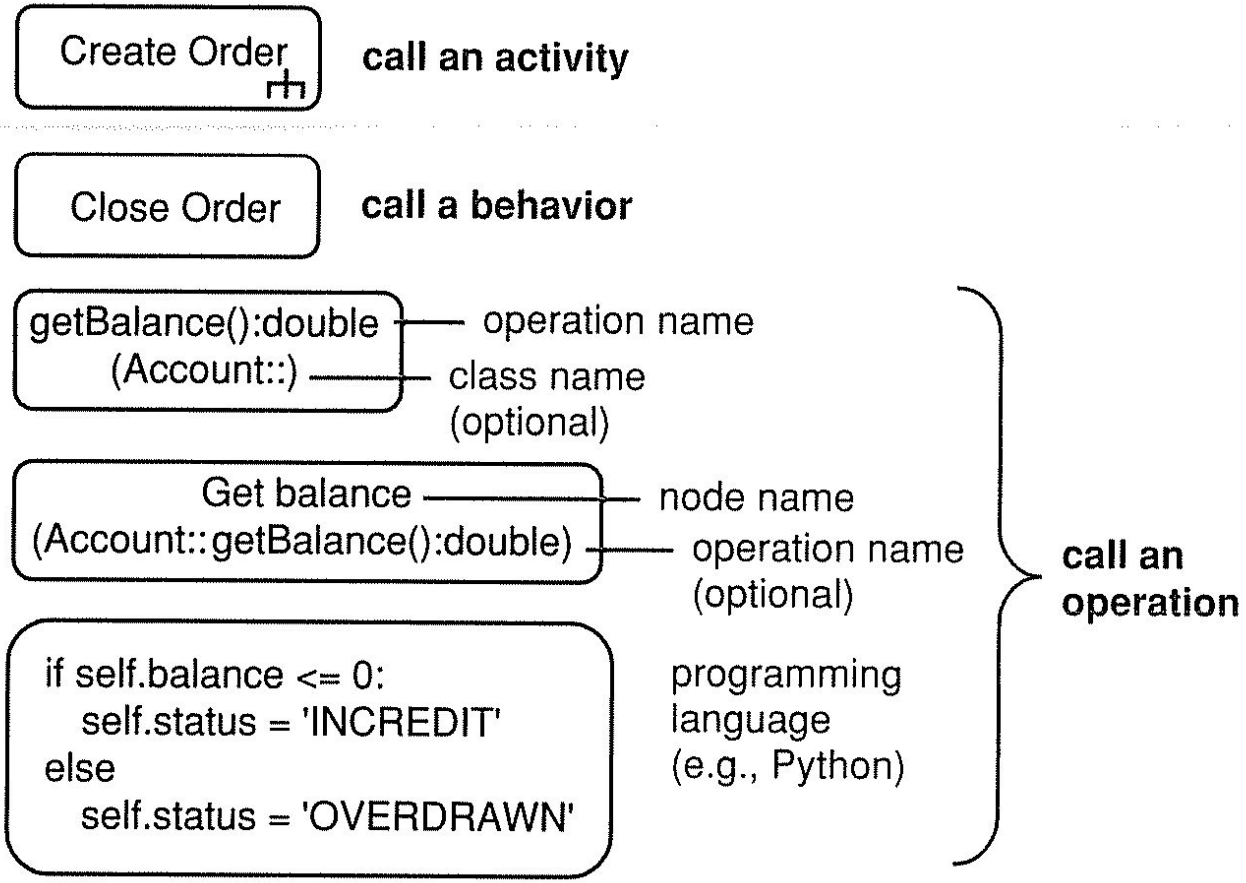


There are 4 action node kinds

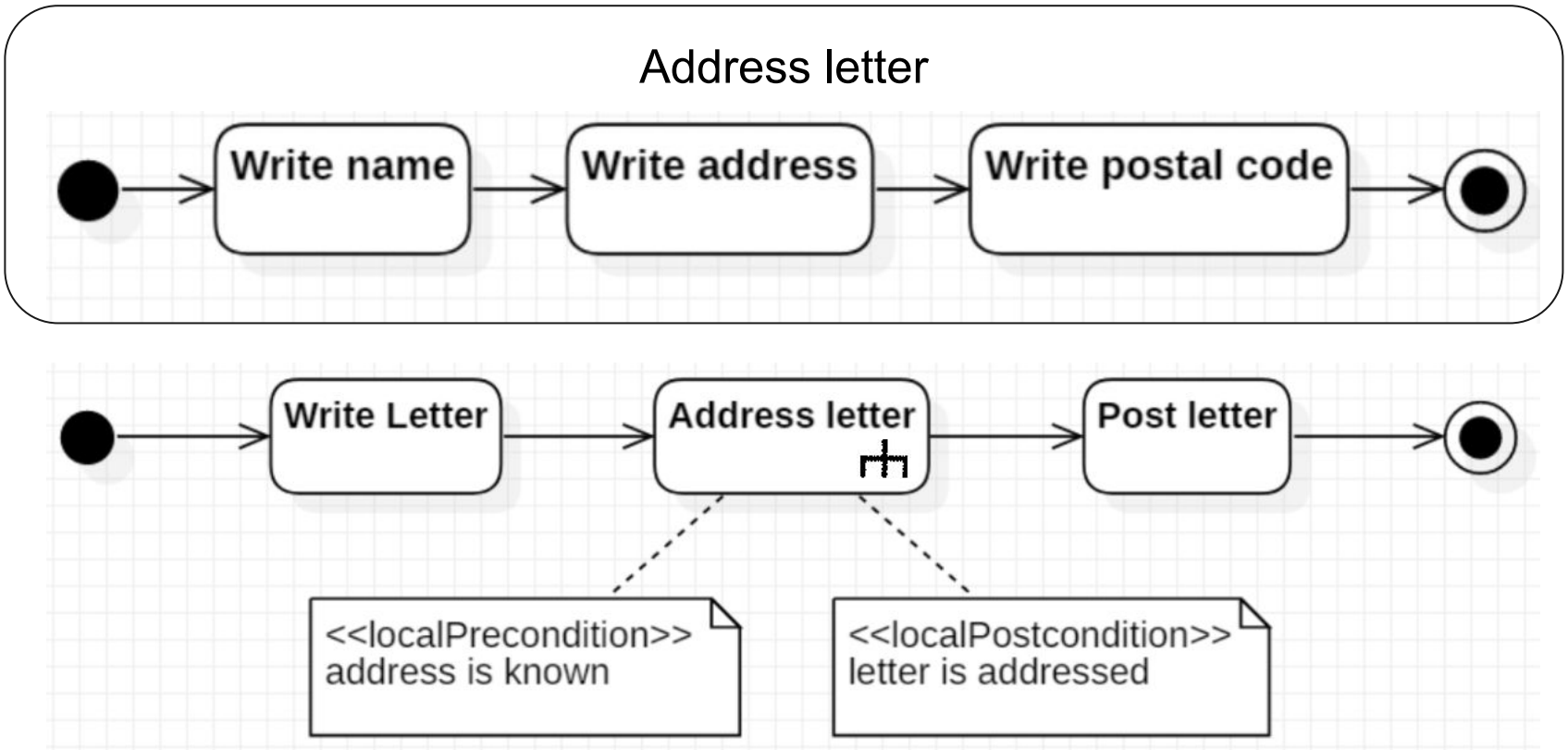


Call action node



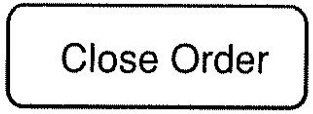


An action can call another activity



Here, **Address letter** invokes the **Address letter activity**

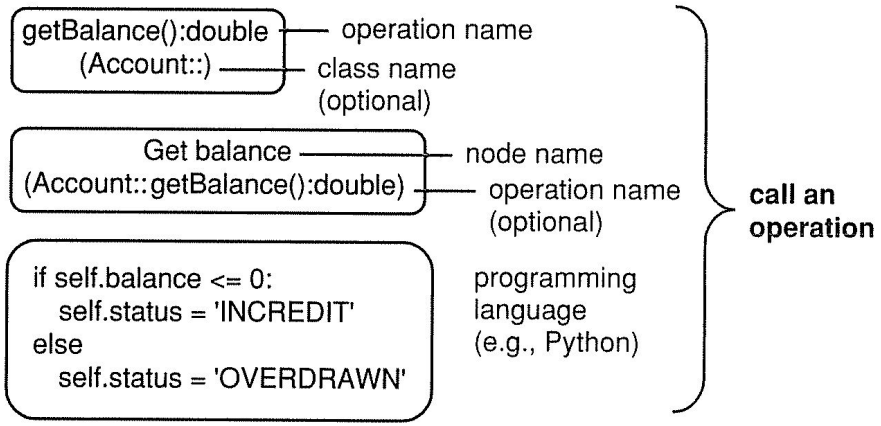
An action can call a behavior



- Direct invocation of a behavior of the context of the activity, without specifying any particular operation

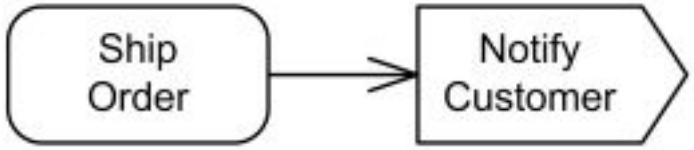
An action can call an operation

- There is a standard operation syntax for it
- You can specify the details of the operation in a particular programming language, which is useful for code generation from activity diagrams
- You can refer to features of the activity by using the keyword `self`

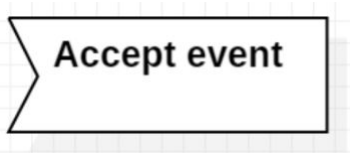


The send signal action node represents the asynchronous sending of a signal

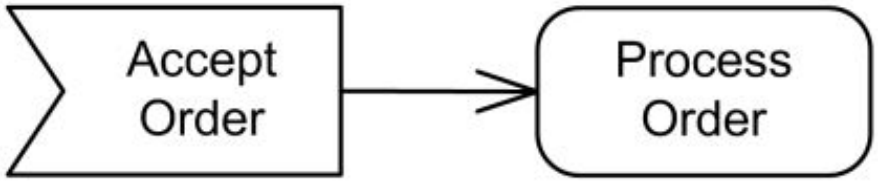
- The send signal action is started when there is a token simultaneously in all its input edges
  - If the signal has pins, it must receive an input of the right type for each of its attributes
- When the action executes, a signal object is constructed and sent. The target is not usually specified, but if you need to specify it, you can pass it into the send signal action on an input pin
- The signal may be parameterized during its creation
- The sending action does not wait for confirmation of signal receipt - it is asynchronous
- The action ends and control tokens are offered on its output edges



The accept event action node waits for the receipt of an event of the right type

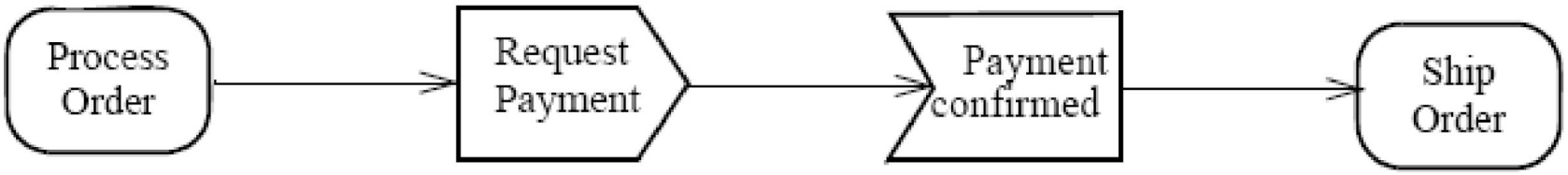


- The accept event action node has zero or one input edges
- Its action is started by an incoming control edge, or, if it has no incoming edge, it is started when its owning activity starts
- The action waits for the receipt of an event of the specified type. This event is known as the trigger.
- When the action receives an event trigger of the right type, it outputs a token that describes the event. If the event was a signal event, the token is a signal.
- The action continues to accept events while the activity executes



Sending/receiving signals

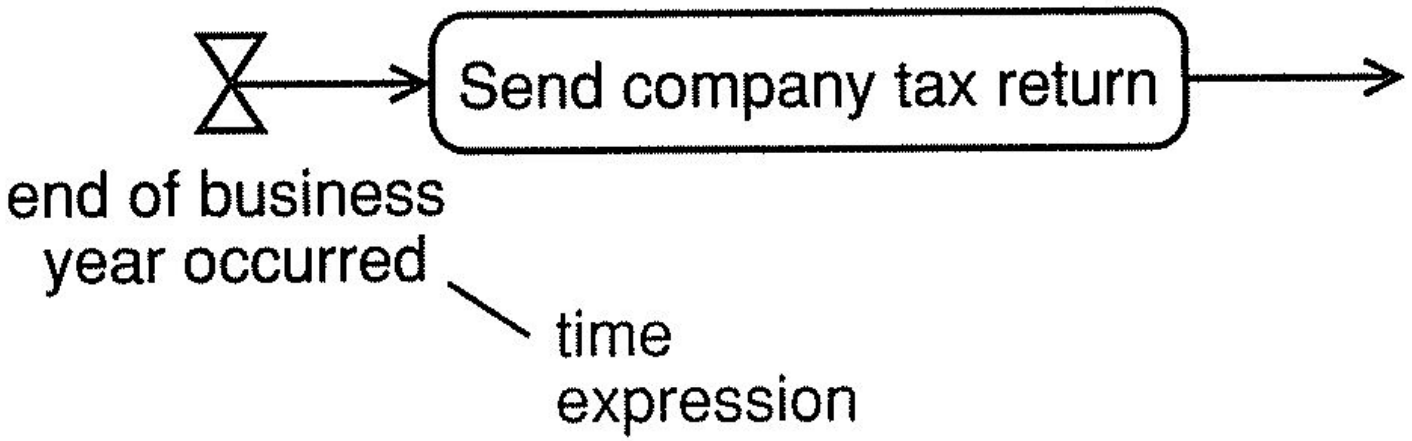
- The Process Order Activity originates the sending of the Request Payment Signal
- The Ship Order waits for receiving the Payment Confirmed Signal



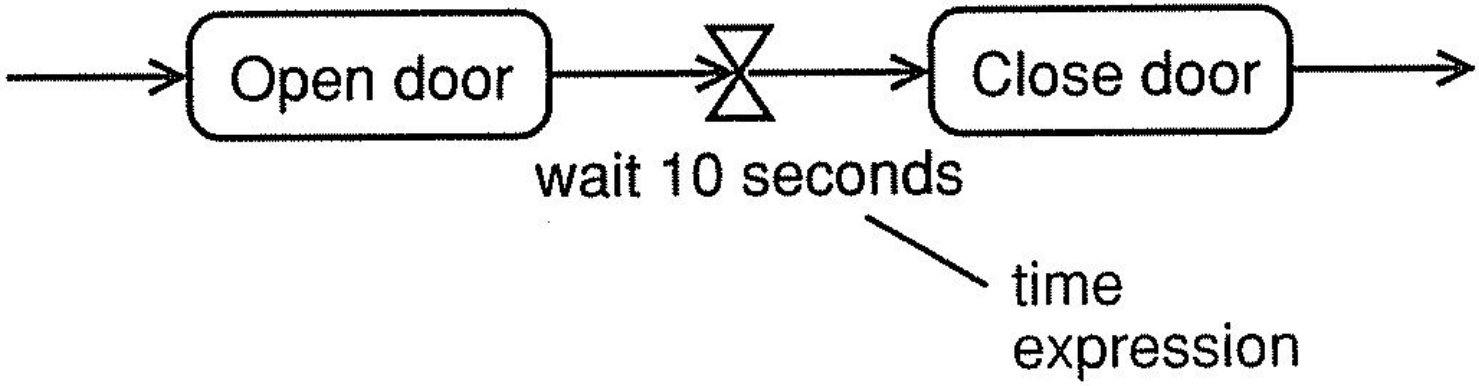
Accept time event action nodes respond to time

- The time event action node has a time expression and generates a time event when the expression becomes true
- A time expression may refer to:
  - an event in time (e.g. end of business year)
  - a point in time (e.g. on July 25, 2004)
  - a duration (wait 5 seconds)

When the owning activity is triggered, the node becomes active and will generate a time event whenever its expression becomes true



If the node has an input edge, it only becomes active when it receives a token and only then will it generate a time event whenever its expression becomes true



Control nodes

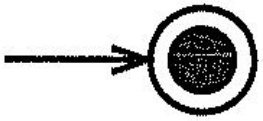
Control nodes

- Initial node
- Activity final node
- Flow final node
- Decision node
- Merge node
- Fork node
- Join node

Initial node indicates where the activity starts

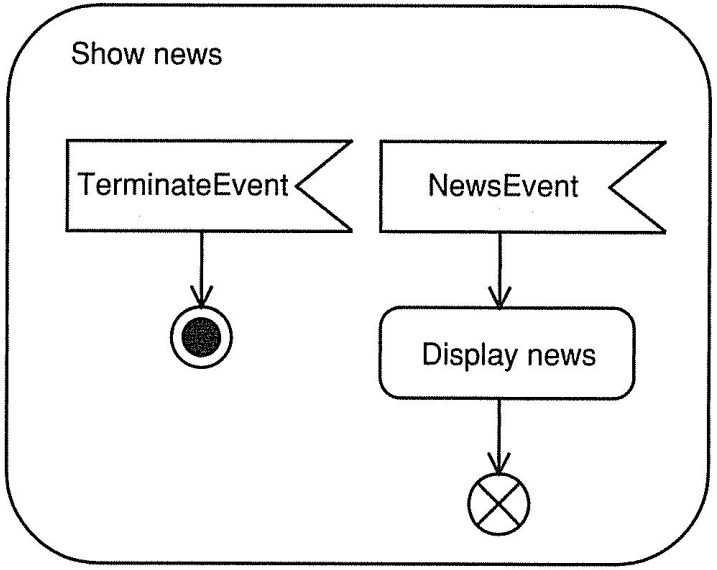
- Indicates where the flow starts when an activity is invoked
- There may be more than one initial node
  - Flows start at all the initial nodes simultaneously
  - Flows execute concurrently
- These nodes are not mandatory, provided there is some other way of starting the activity:
  - Accept event action
  - Accept time event action
  - Activity parameter node

Activity final node terminates an activity

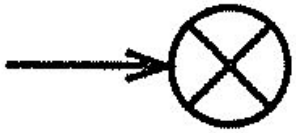


- The activity final node stops all flows within an activity
- An activity may have several activity final nodes
  - The first one to be activated stops all the other flows and the activity itself

In this example, receiving a TerminateEvent signal ends the whole activity (i.e. all existing flows)

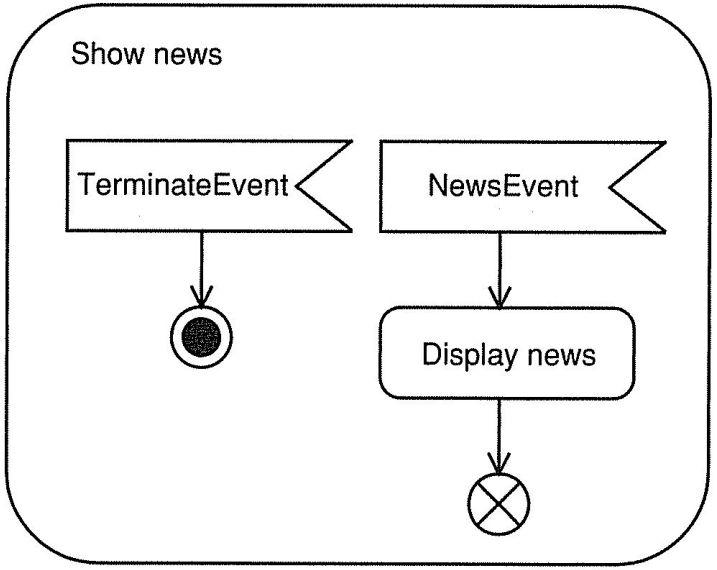


Flow final node terminates a specific flow within the activity

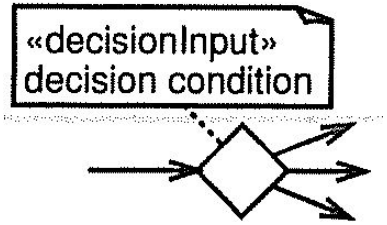


- The activity flow final node stops one of the flows within the activity
  - Other flows are unaffected and continue

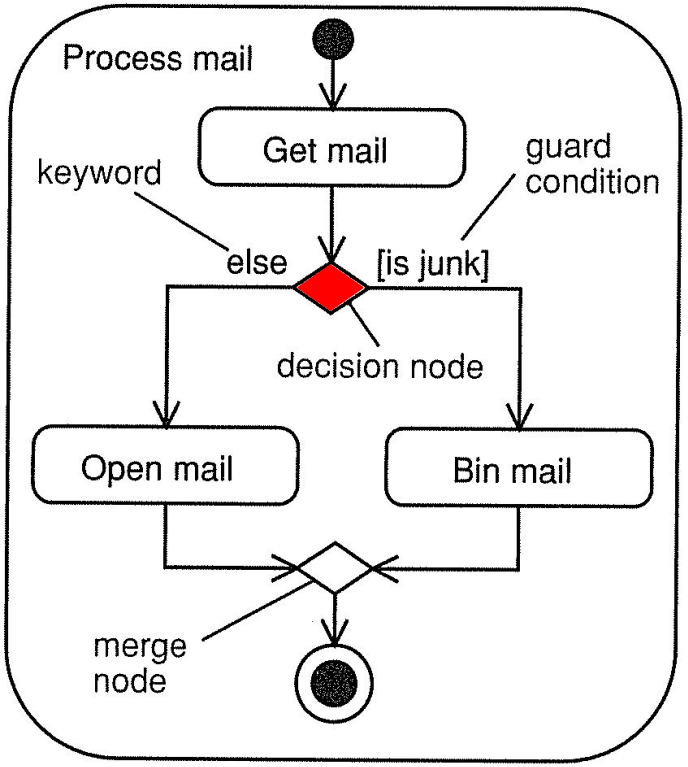
In this example, when receiving a NewsEvent signal, the signal is passed to the action Display news. Control then flows a flow final node, terminating this flow, but not the whole Show news activity.



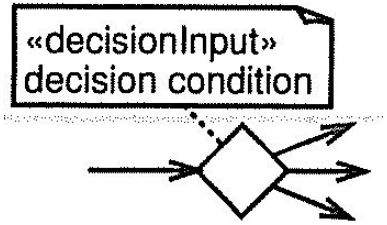
Decision node allows the output edge whose guard is true to be traversed



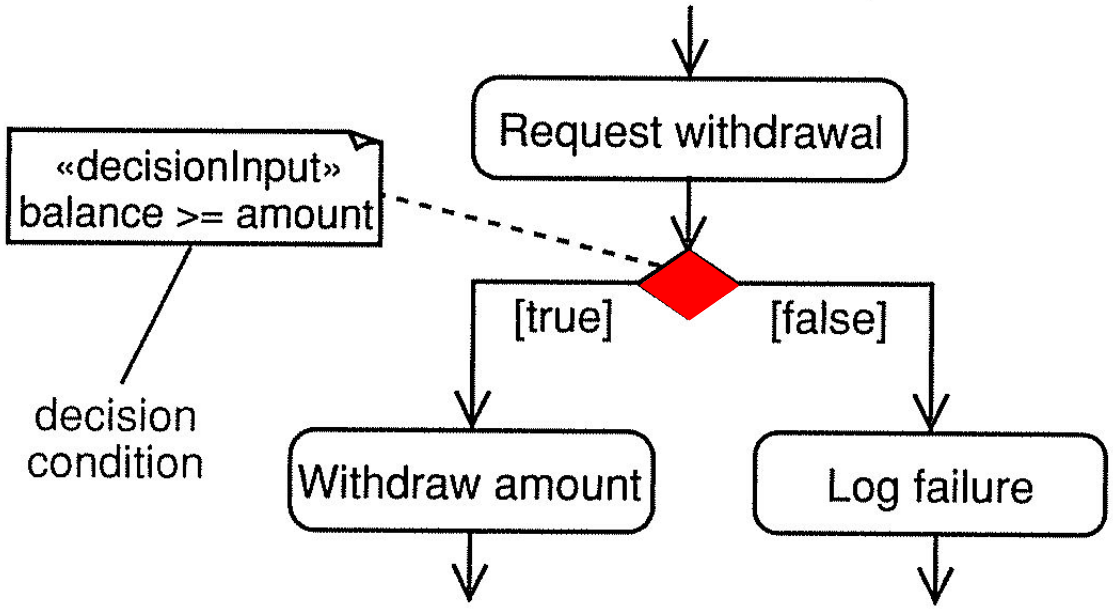
- A decision node has one input edge and two or more output edges
- A token arriving to the decision node is offered to all output edges but it will traverse at most one of them
- Each output edge is protected by a guard condition
  - The edge will accept the token if the guard condition evaluates to true
- The guard conditions must be mutually exclusive, so that only one can be evaluated to true and therefore crossed
  - If they are not mutually exclusive, the behavior of the decision node is undefined.
- The keyword else can be used as an alternative if none of the other guard conditions evaluated to true



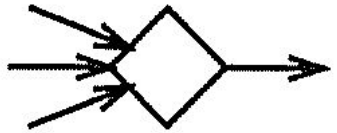
Decision node may have a stereotyped <<decision input>> note



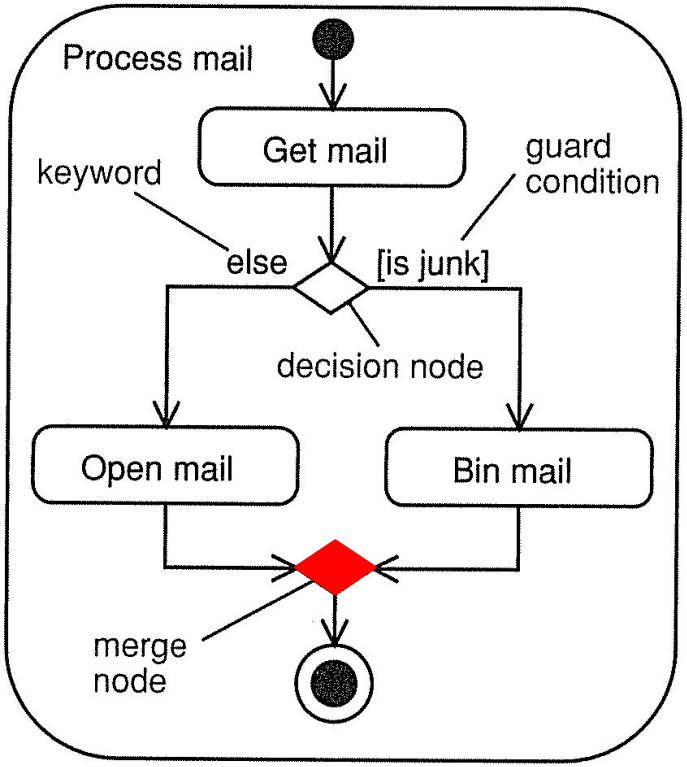
- Optionally, this may have a note stereotyped as <<decision input>> encoding a decision condition whose result is then used by the guard conditions on the edges



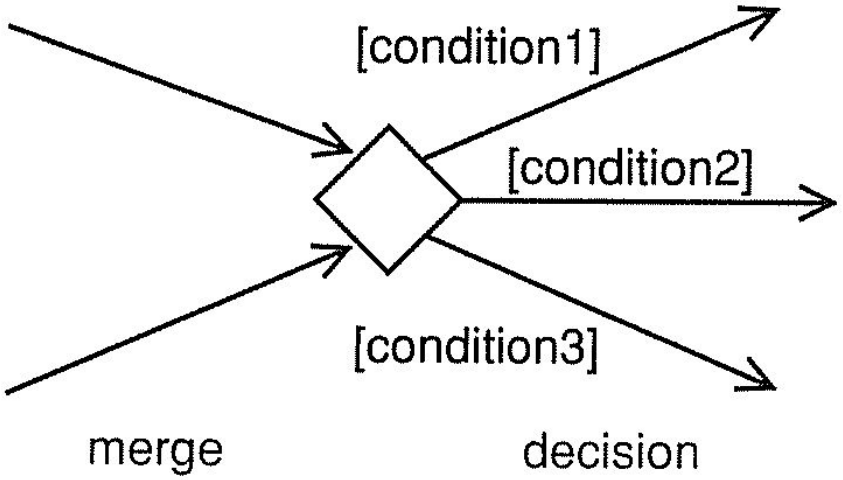
Merge node copies input tokens to its single output edge



- Merge nodes have two or more input edges and one single output edge
- They merge all incoming flows into a single outgoing flow
- All tokens offered on the incoming edges are offered on the outgoing edge, with no modification of the flow, or of the tokens

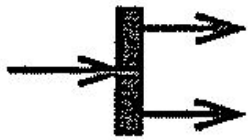


If you want to confuse other stakeholders, try this - breaks the ice in meetings, but not in a nice way

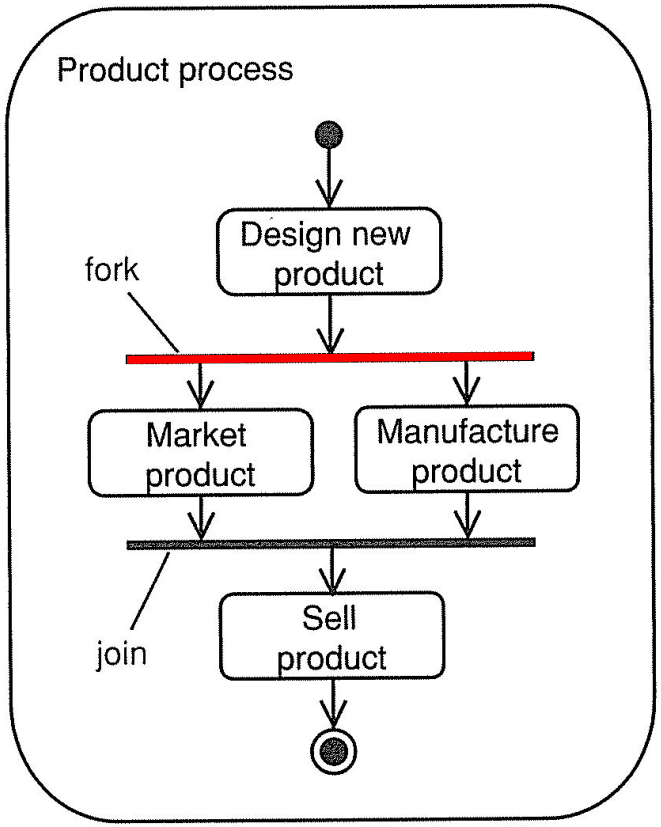


Using a single symbol for a merge, followed by a decision node is legal, but not a good idea. Separate them to increase the understandability of your model.

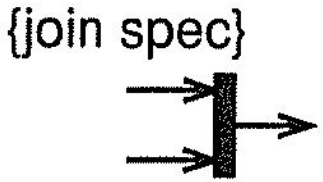
A fork node splits the flow into multiple concurrent flows



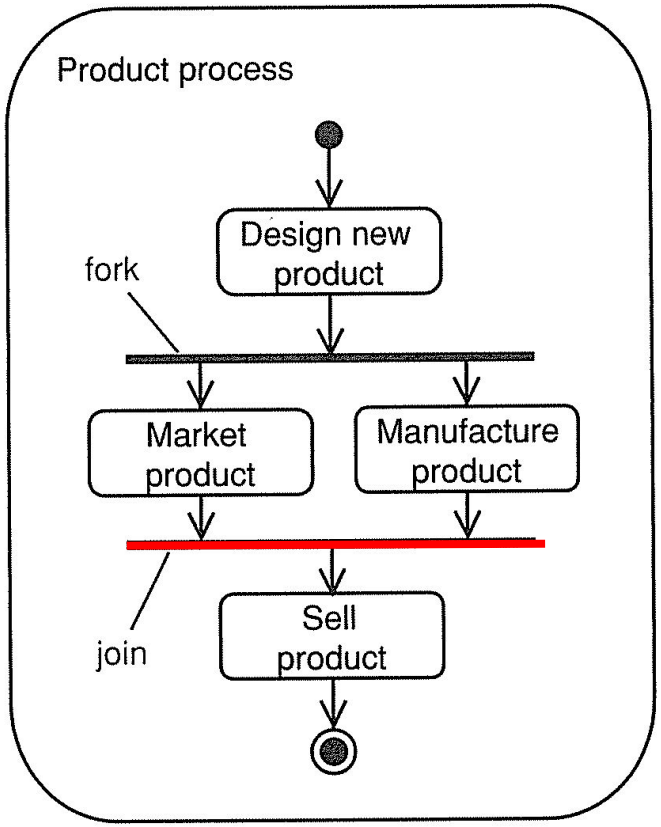
- A fork node has one incoming edge and two or more outgoing edges.
- Tokens arriving from the incoming edge are replicated and offered on all of the outgoing edges simultaneously
- Outgoing edges may have guard conditions
  - Tokens can only traverse the outgoing edges when the guard condition is true
  - **No mutually exclusive guard conditions can be defined in the guard conditions of the output edges of the fork**



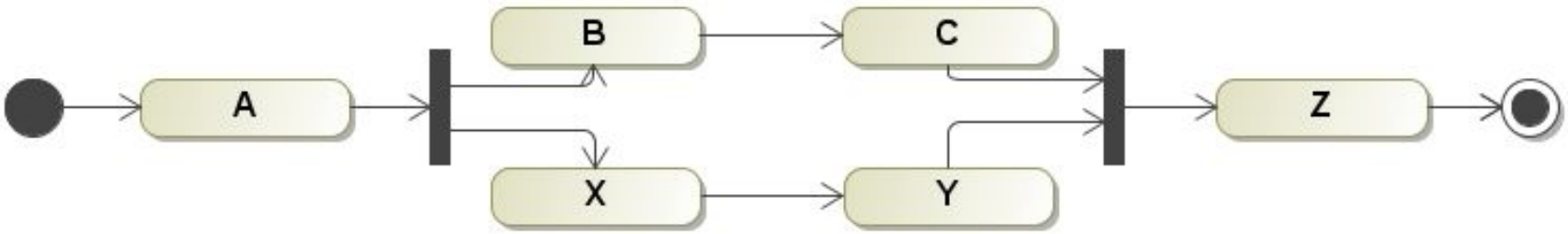
A join node synchronizes multiple concurrent nodes



- Join nodes have multiple input edges and a single output edge
- They synchronize flows by offering a token on their single output when there is a token in all incoming edges
  - This performs a logical AND on the input edges
- Make sure all input edges to the join will receive a token (no mutually exclusive guards, remember?)
- May optionally have a join specification to modify its semantics



Concurrency model



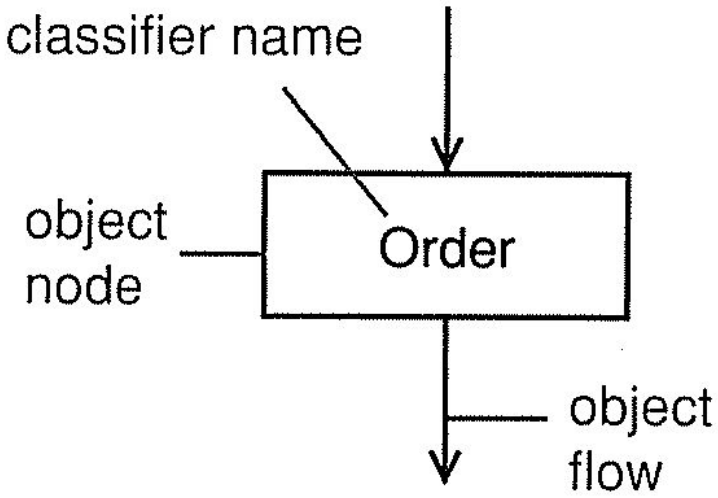
Trace: **A, {(B,C) || (X,Y)}, Z**

where || is the parallel composition operator and , is the sequence composition operator

Object nodes

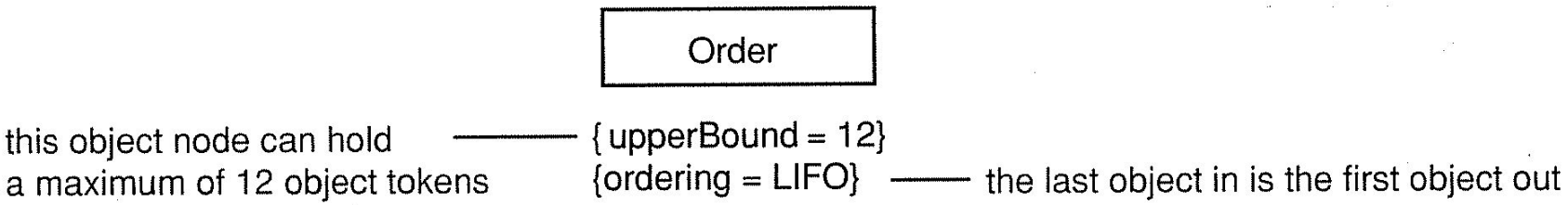
Object flows represent the movement of objects around an activity

- The object node represents an instance of a particular classifier
  - Or of a specialization (subclass) of that classifier
- The input and output edges of an object node are **object flows**



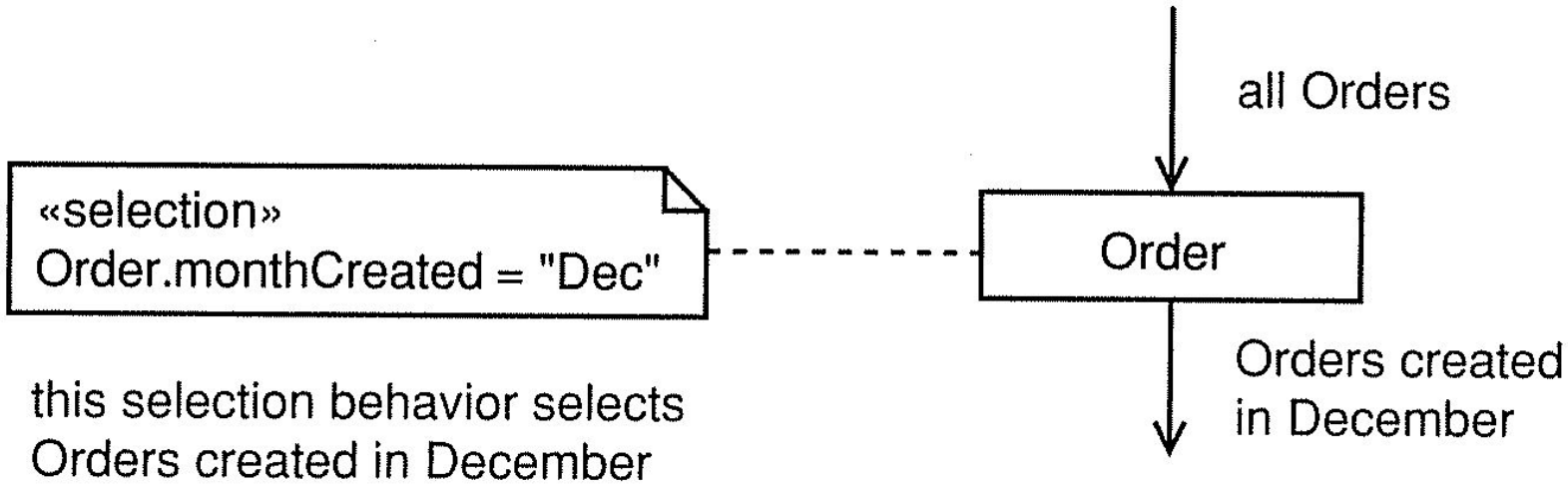
Object nodes as buffers

- Object nodes act as buffers where object tokens can reside while waiting to be accepted by other nodes
- By default, these buffers have infinite capacity
- However, you can define an upper bound to limit this capacity
- You can also define the ordering policy within the object node
  - FIFO (First In, First Out) - this is the default
  - LIFO (Last In, First Out) - this is the alternative



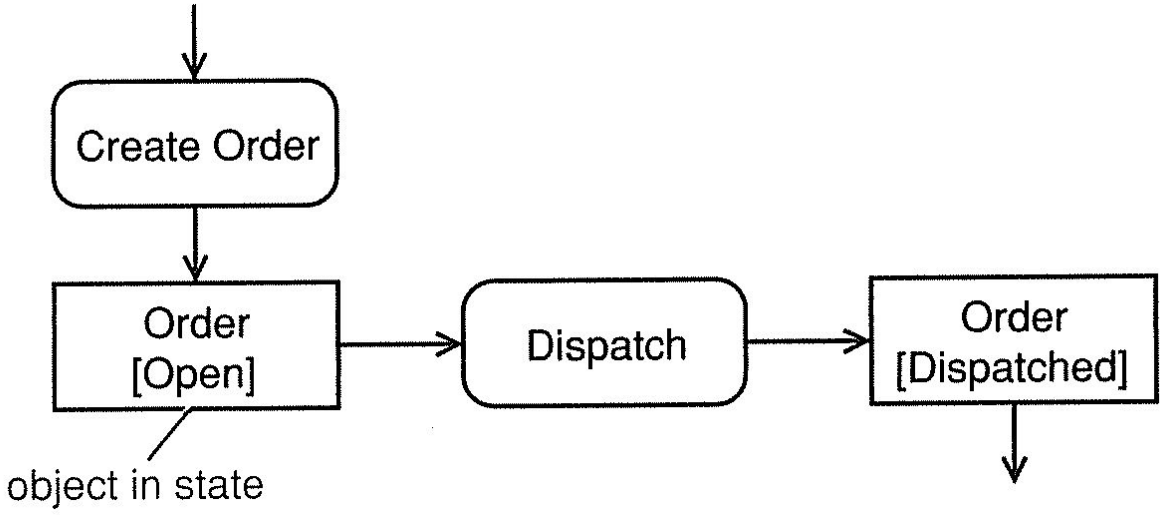
Object nodes as filtered buffers

- Object nodes may have a selection behavior, acting as filters
  - In this example, the object node selects only orders created in December and offers the corresponding object tokens to the output edge using the default ordering (FIFO)



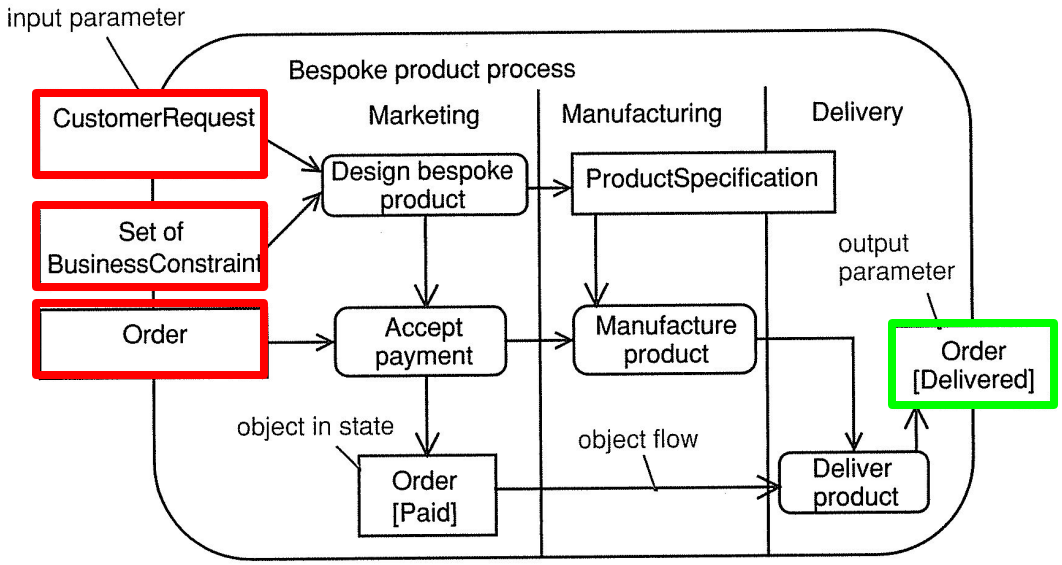
Object nodes can represent objects in a particular state

- In this example, an order processing activity accepts Order objects in the state Open and dispatches them, in the state Dispatched

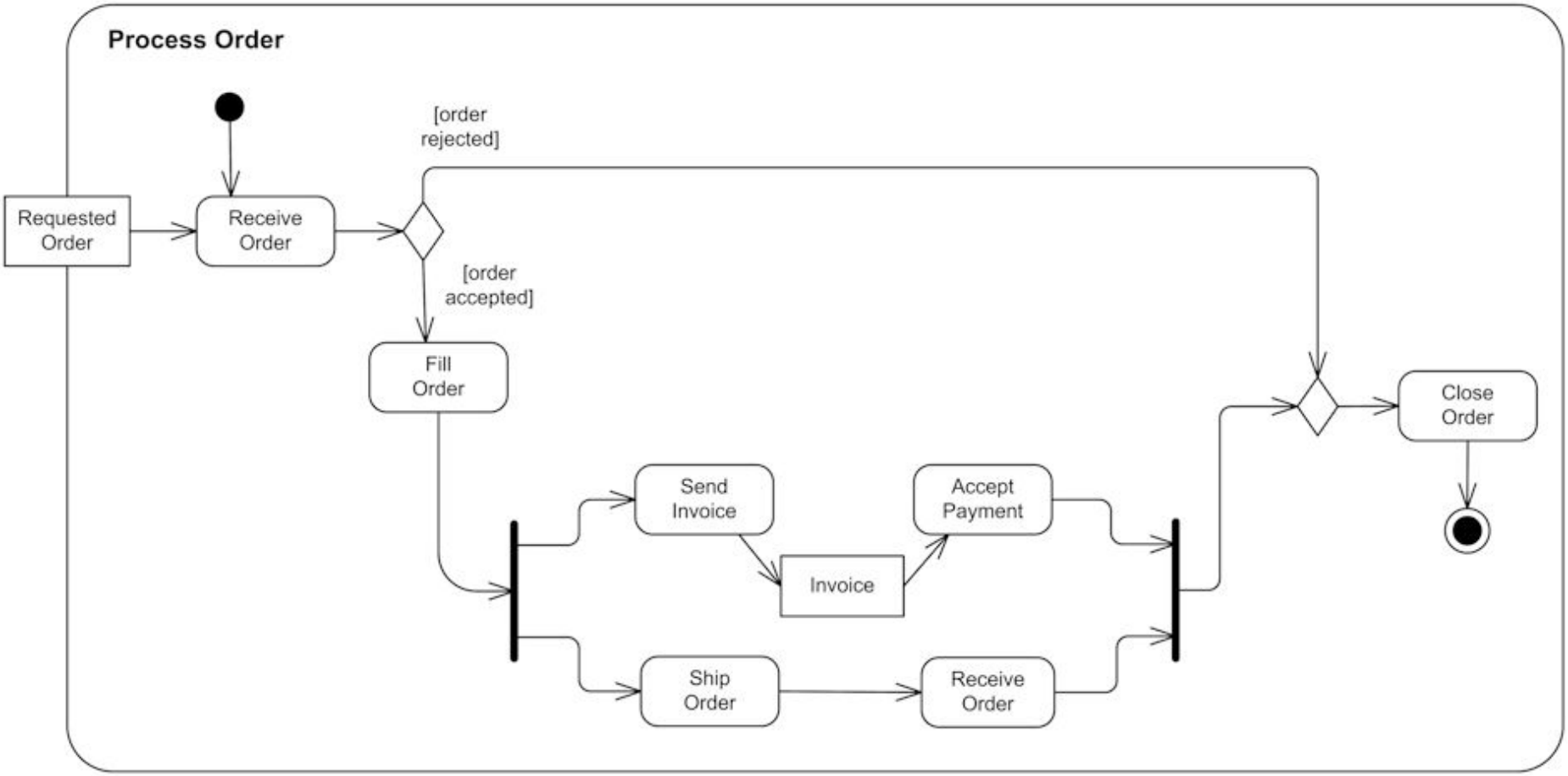


Activity parameters are object nodes input to, or output from, an activity

- Use object nodes to provide inputs or outputs for the activity
- These nodes should overlap the activity frame
- Input nodes have one or more output edges into the activity
- Output nodes have one or more input edges from the activity

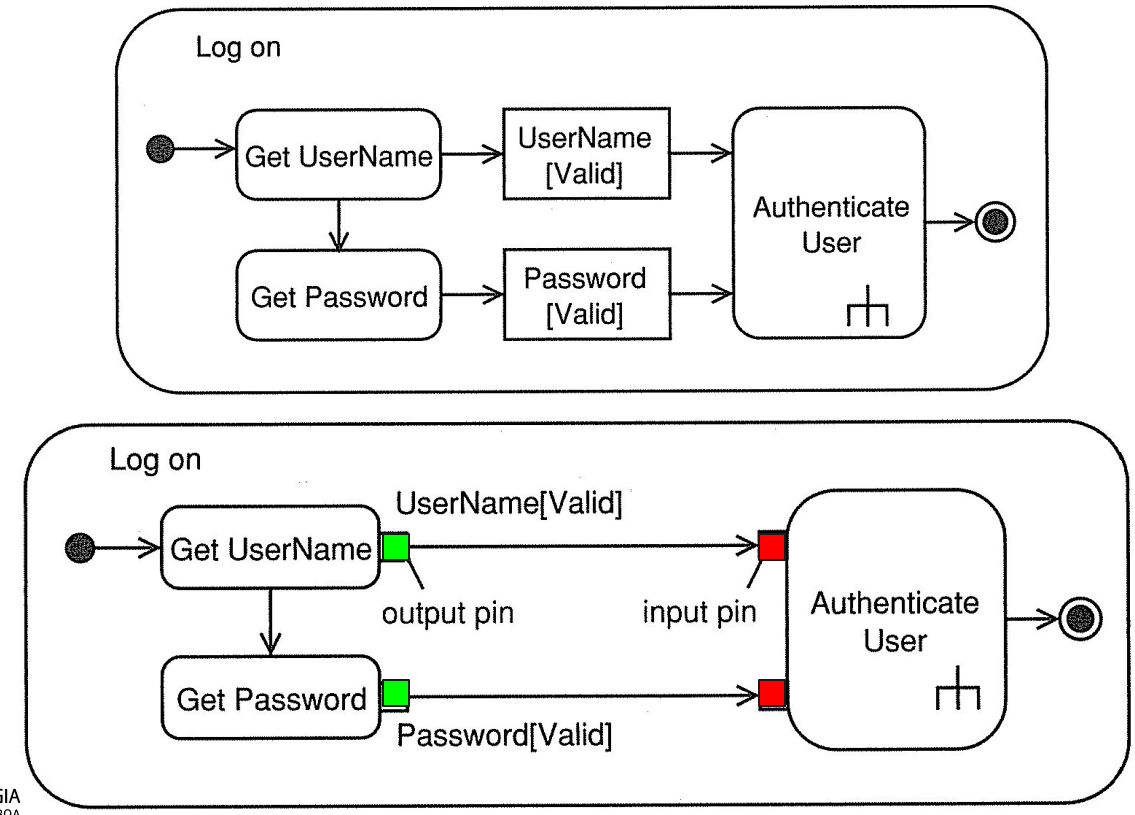


Example

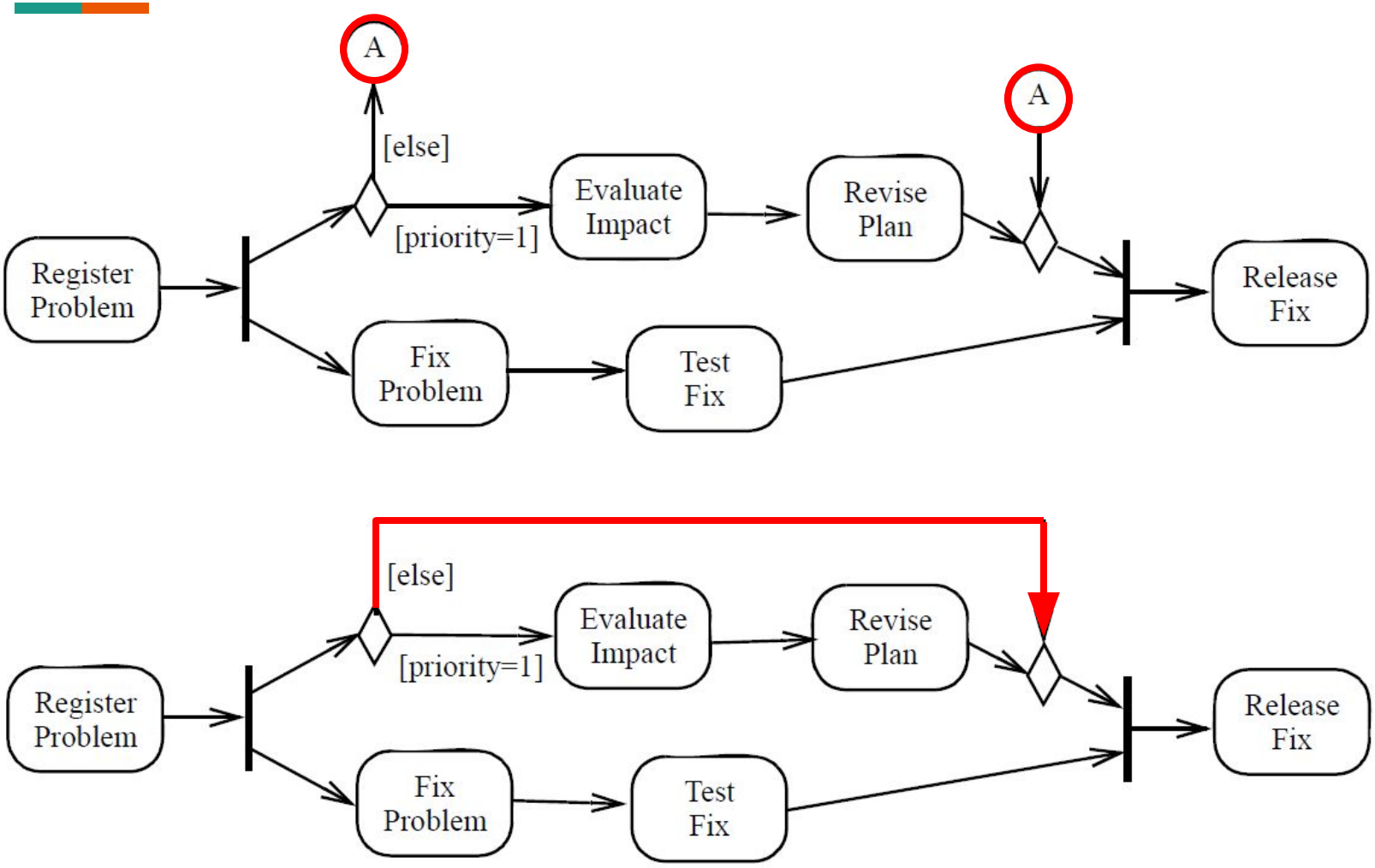

Process Order
Requested Order
Receive Order
[order rejected]
[order accepted]
Fill Order
Send Invoice
Invoice
Accept Payment
Ship Order
Receive Order
Close Order

Pins are object nodes that represent one input, or one output, from an action

- Presentation aside, they have the same semantics as object nodes



Activity edged connectors can be used to prevent cluttering - but avoid this if possible!



Activity partitions

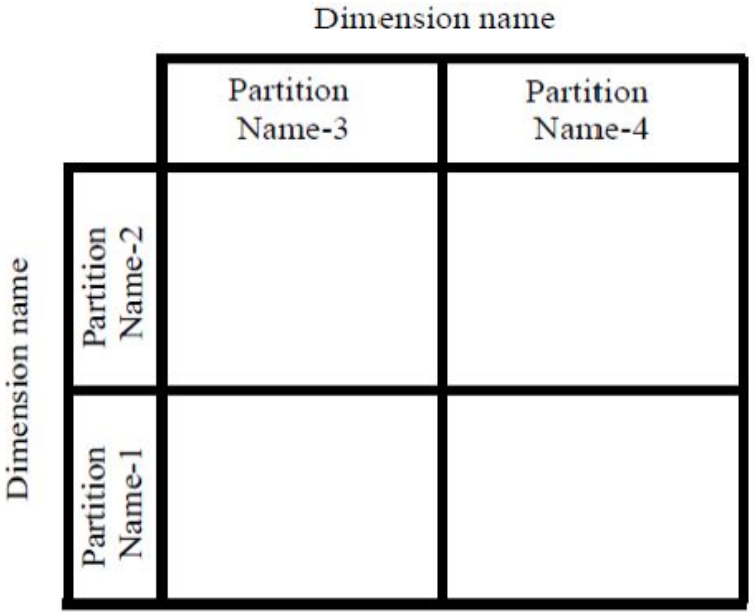
An activity partition (aka swimlane) represents a high-level grouping of related actions

- Activities can be divided into partitions by using horizontal, vertical, or curved lines
- The semantics is defined by the modeller; common examples include:
  - use cases
  - classes
  - components
  - organizational modeling
  - roles
  - ...

Several organization alternatives for swimlanes partitioning

a) Partition using a swimlane notation

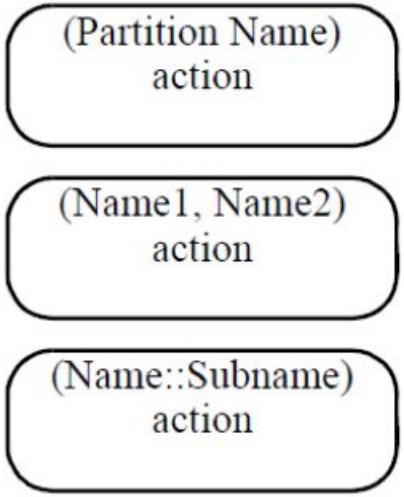
b) Partition using a hierarchical swimlane notation



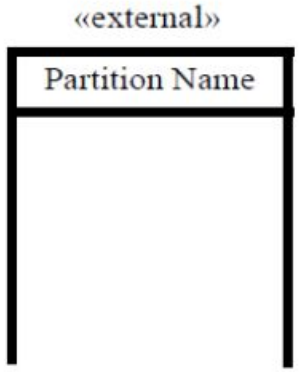
c) Partition using a multidimensional hierarchical swimlane notation

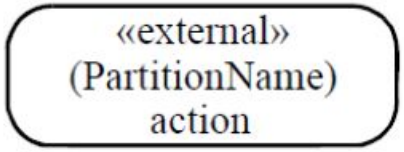
You can also annotate the nodes

(use this only if the visual swimlanes are impractical for your model)



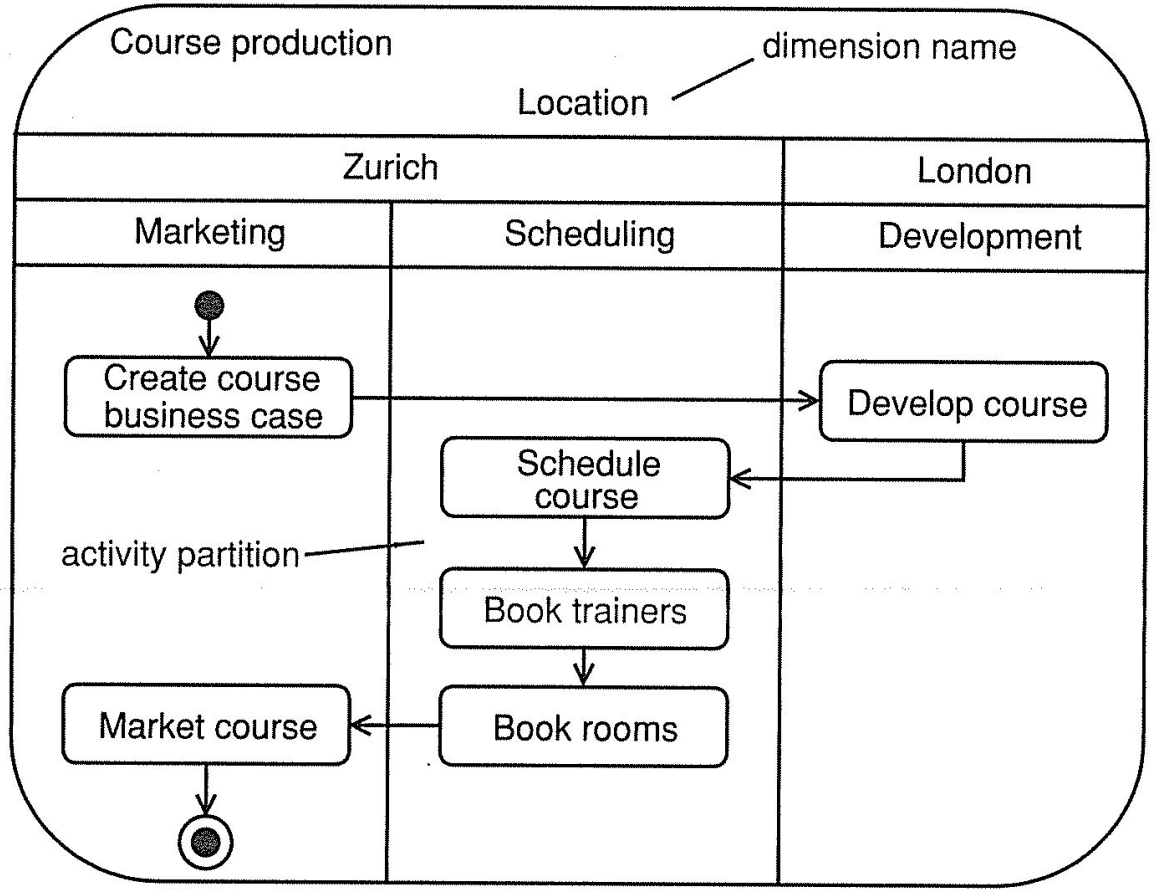
a) Partition notated on a specific activity node



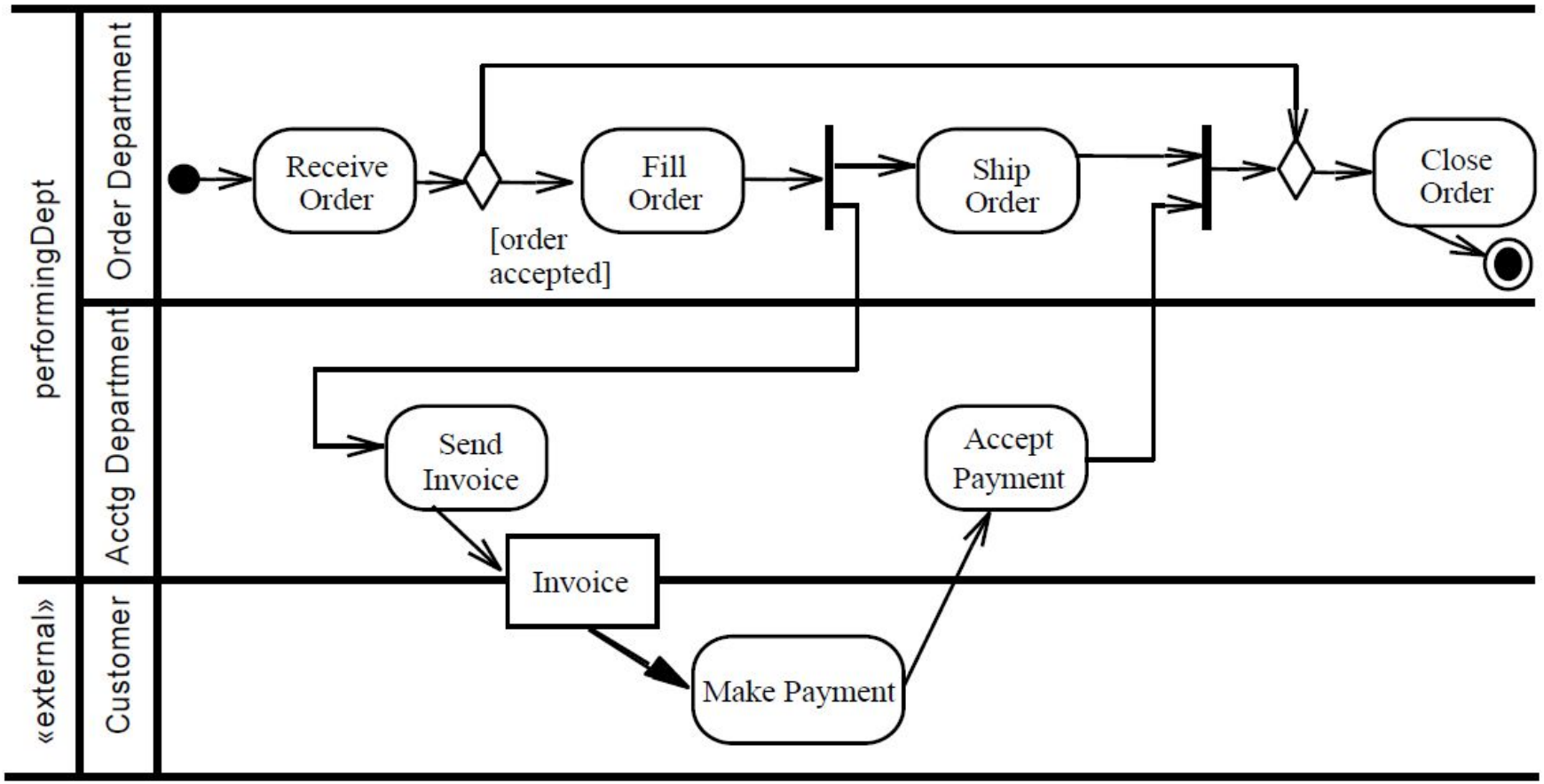


b) Partition notated to occur outside the primary concern of the model.

Each set of partitions should have a single dimension



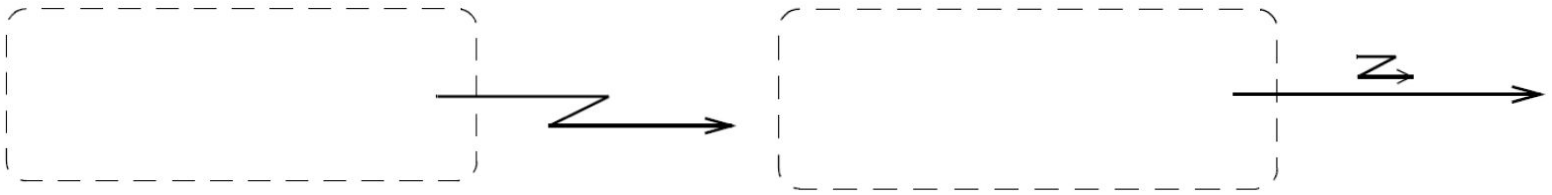
We can also organize swimlanes horizontally



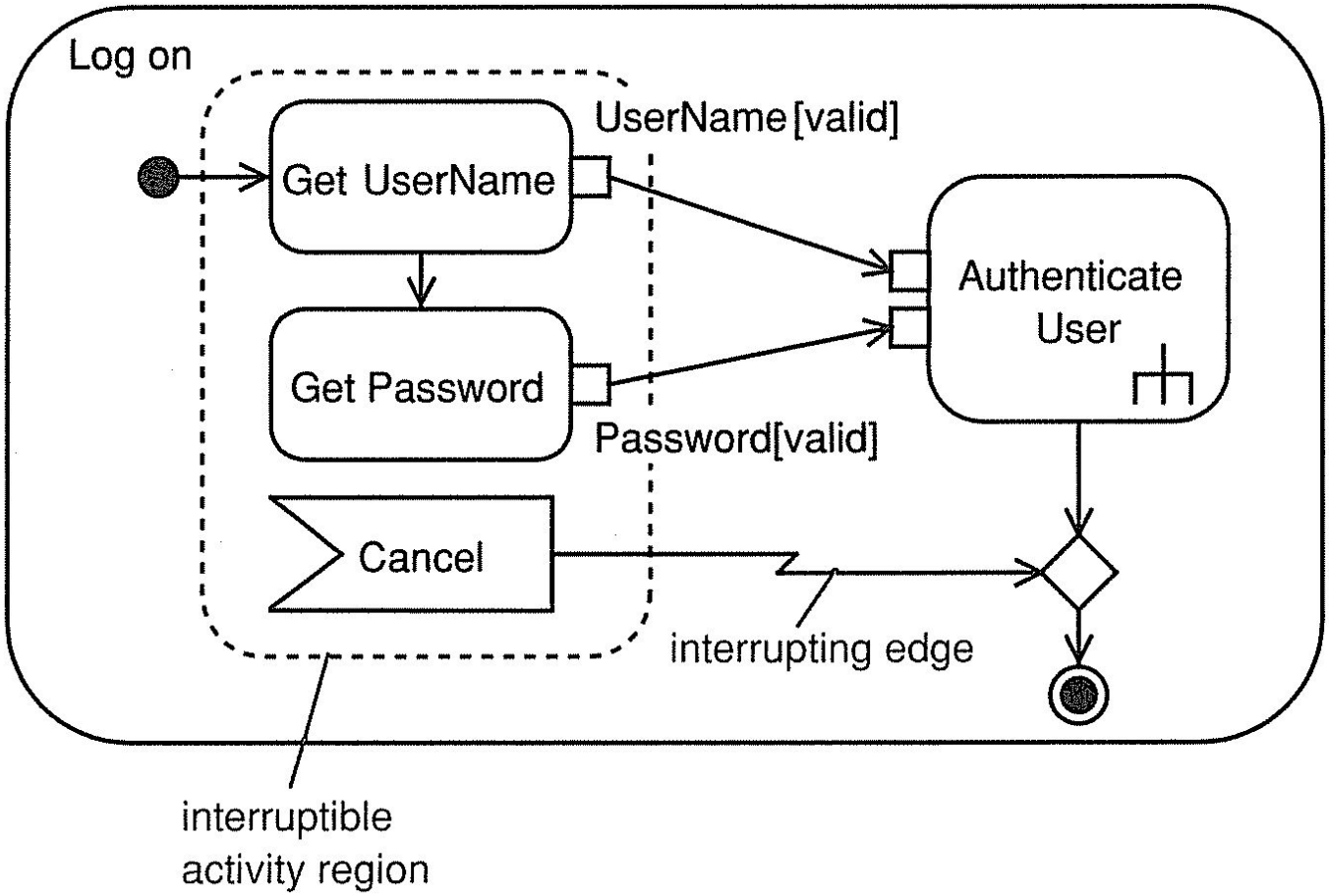
Interruptible Activity Regions

Interruptible activity regions

- Regions of an activity that are interrupted when a token traverses an interrupting edge
  - When the region is interrupted, all flows within it are immediately aborted
- Mechanism for modelling interrupts and asynchronous events
- Two alternative notations:



Example: Log on

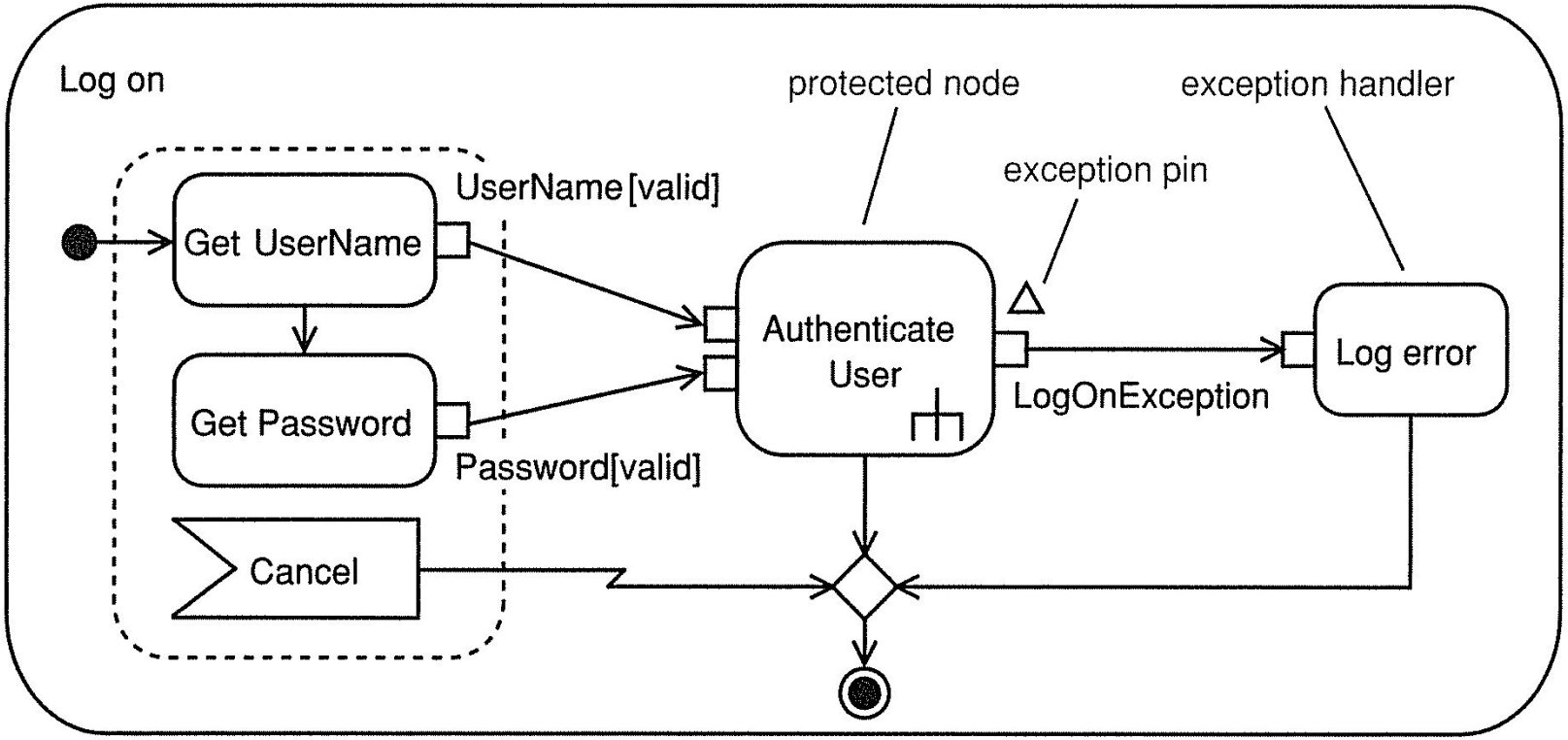


Exception handling

Exception handling

- If an error is detected in protected code, an exception object is created and the flow of control jumps to an exception handler that processes the exception object adequately
  - The exception object contains information about the exception that can be used by the exception handler
  - The exception handler may terminate the application or try to recover from the error, somehow
  - A node covered by an exception handler is known as a protected node

Exception handling example



Expansion regions and nodes

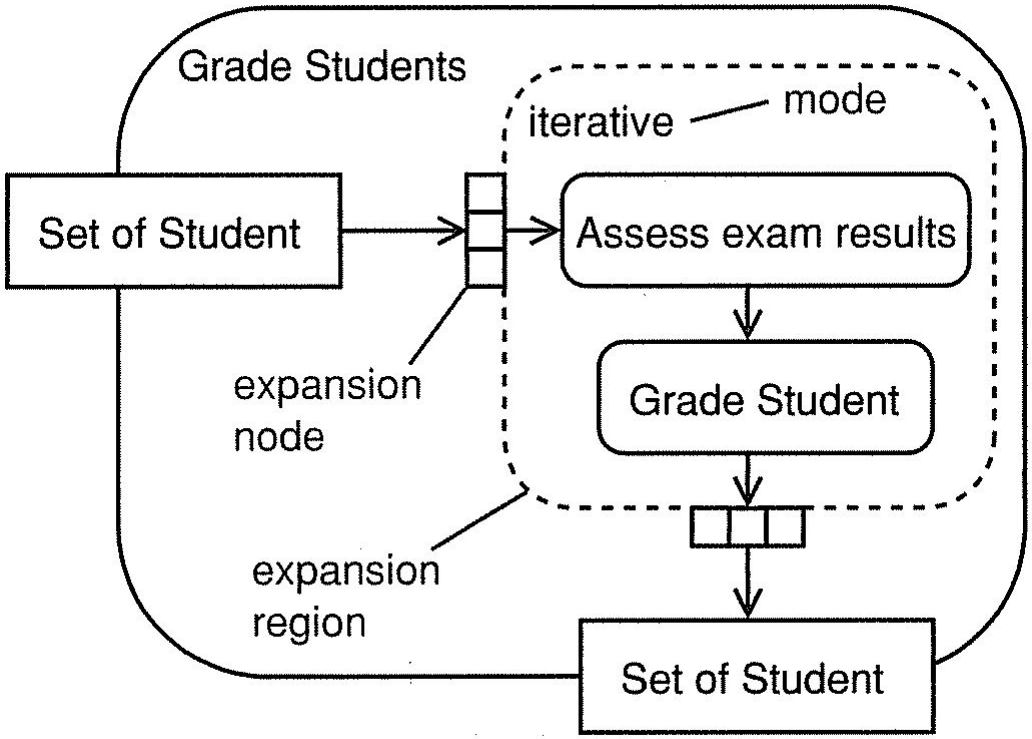
Expansion regions and nodes

- An expansion node is a collection of objects flowing into or out of an expansion region that is executed once for each object
- The notation is similar to a pin, but with 3 boxes, to denote that the node accepts a collection, rather than a single object
- The type of the output collection must match the type of the input collection
- The type of object held in the input and output collections must be the same
  - Expansion regions can’t transform input objects of one kind into output objects from another
- The number of output collections can be different from the number of input collections
  - Expansion regions can combine or split collections

Expansion regions have a mode that must be explicitly defined by the modeller

- Iterative: process each element of the input collection sequentially
  - The output set is only offered after all elements were processed
- Parallel: process each element of the input collection in parallel
- Stream: process each element of the input collection as it arrives to the node
  - The output set is offered, one element at a time, as soon as that element is processed

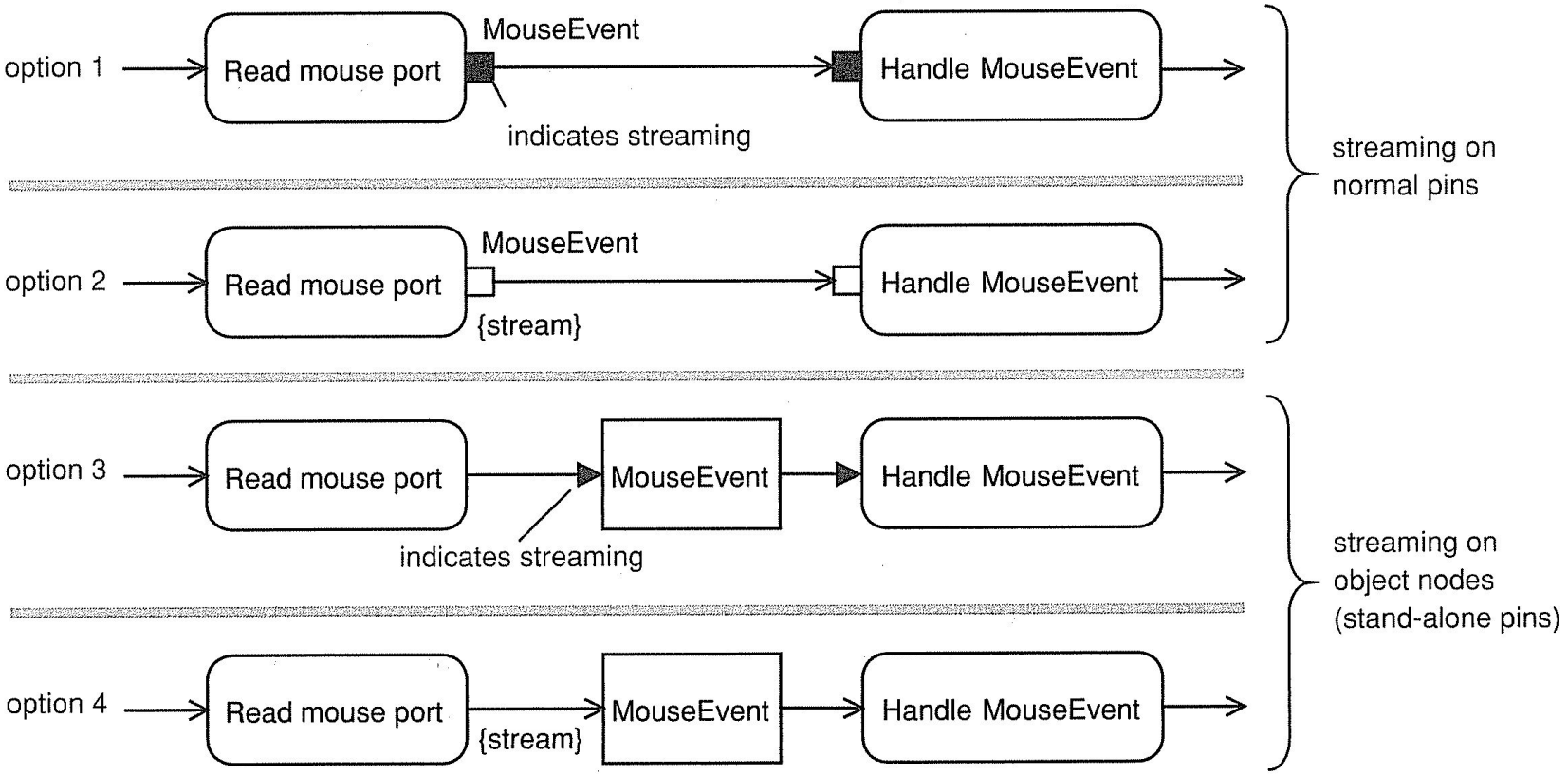
Expansion regions and nodes - example



Streaming

Streaming - an action executes continuously, while accepting and offering tokens

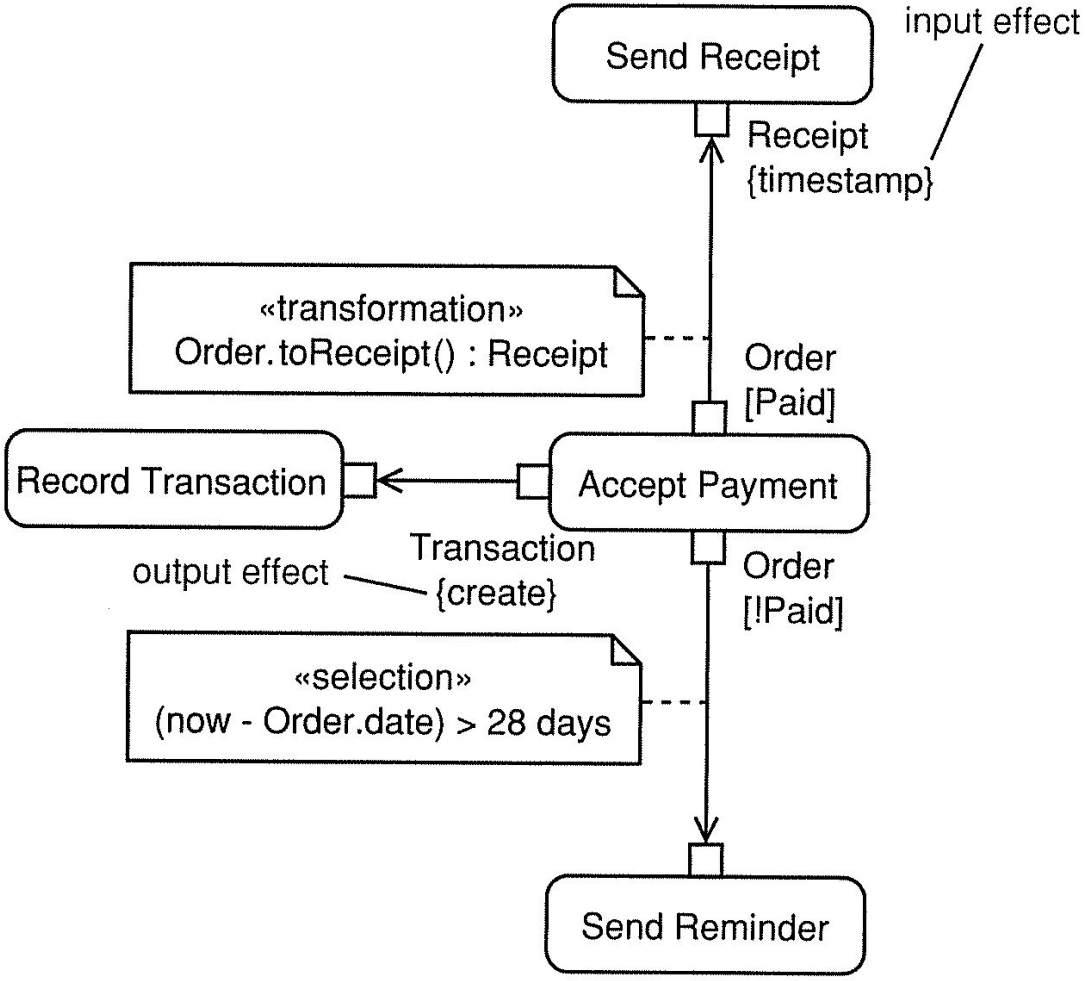
All these are legal. Use just one of the alternatives to avoid confusing stakeholders. We recommend the first option.



Advanced Object Flow features
(these are not used often)

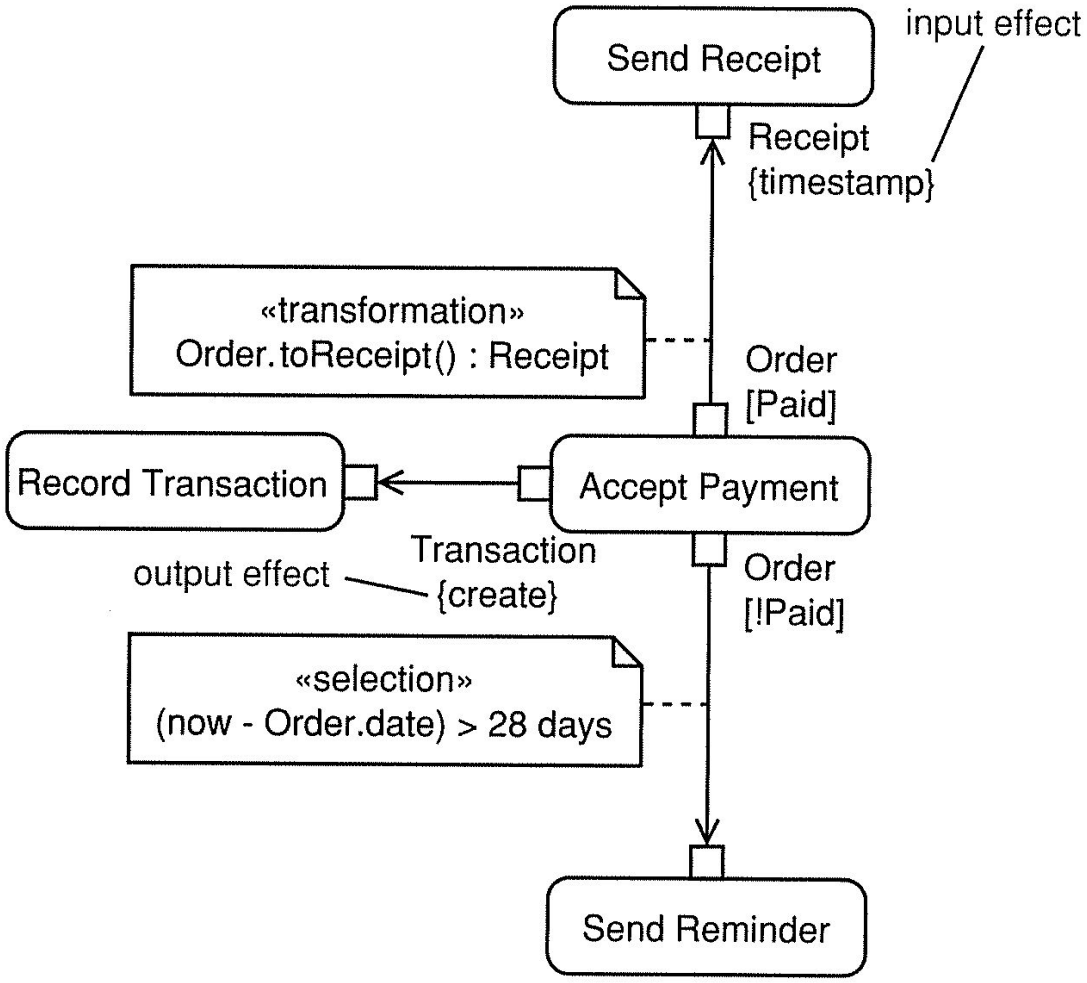
Input and output effects

- Show the effects an action has on its input and output objects
- Effect description between braces, next to the input or output pin



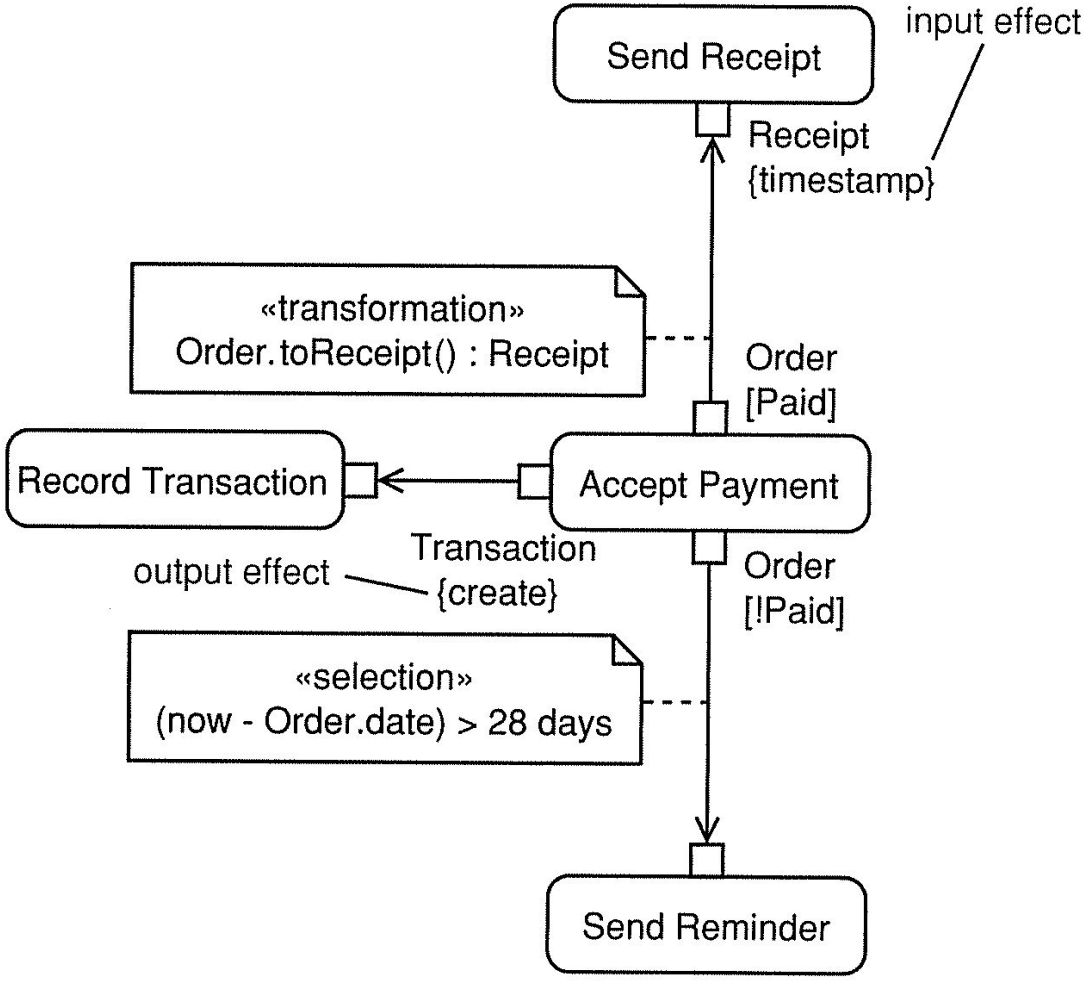
<<selection>>

- A condition on an object flow that causes it to accept only those objects that satisfy the condition



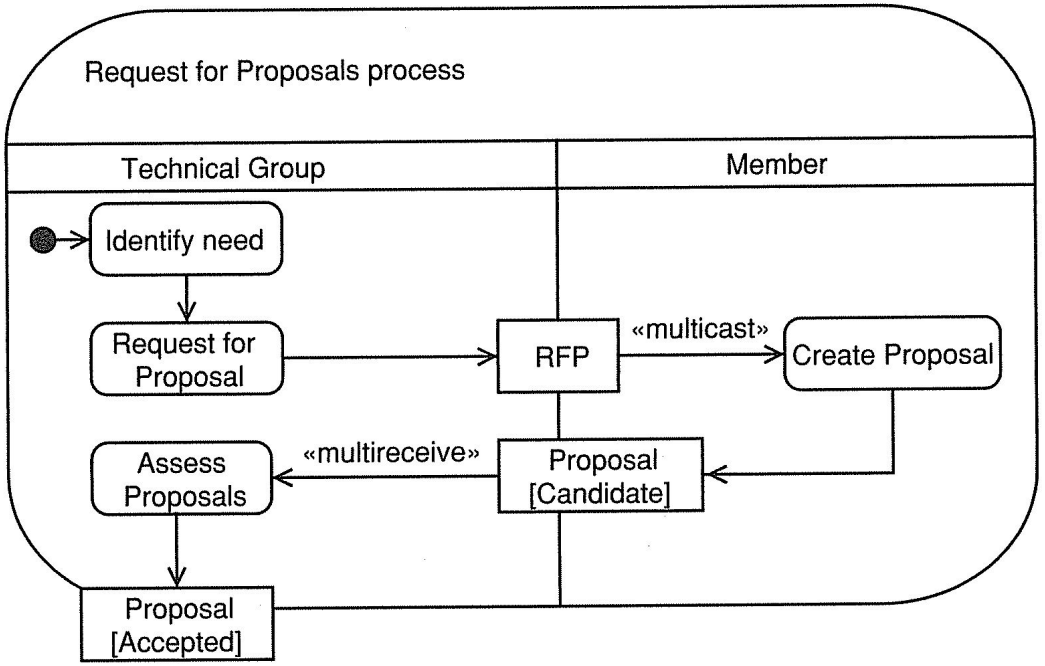
<<transformation>>

- Transforms objects in an object flow to a different type
- Use this when you need to connect an output from a pin of a given type to the input for a pin of a different type



Multicast and multi receive often occur in symmetrical pairs

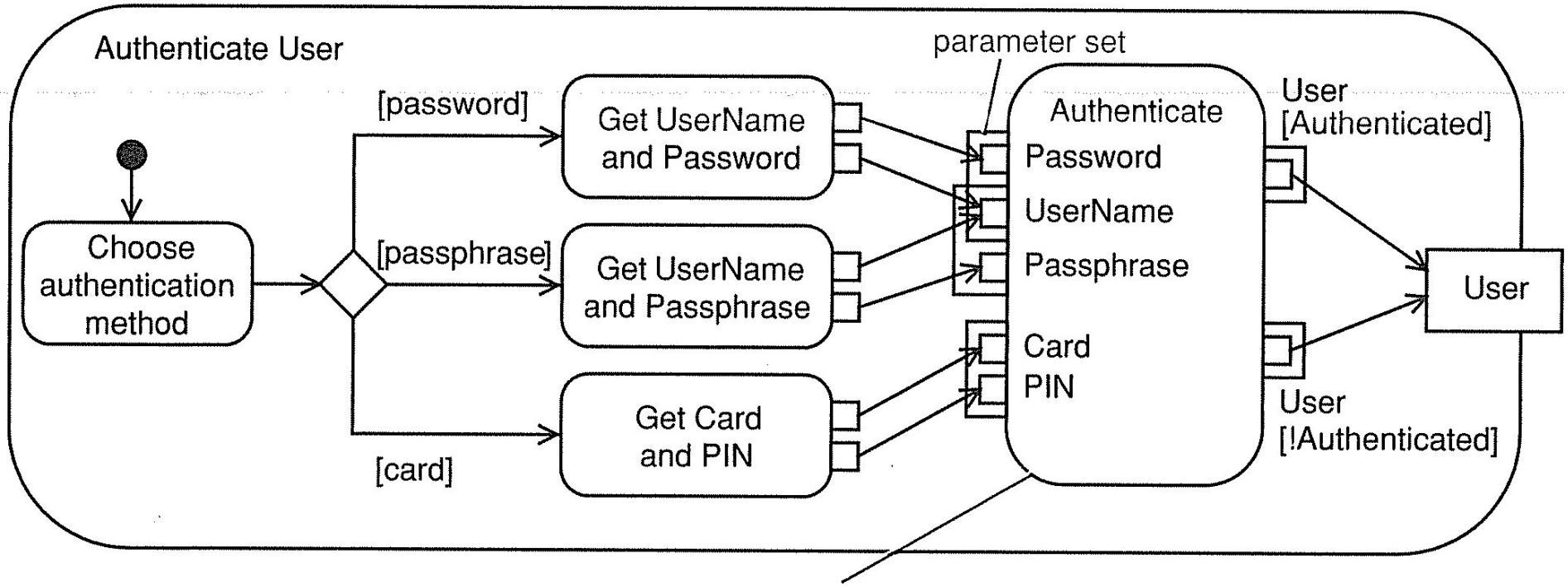
- In general, objects are sent to exactly one receiver
- Sometimes, you need to send them to many receivers - use a multicast
- Conversely, if an object is to be received from multiple senders, use a multi receive.



Parameter sets

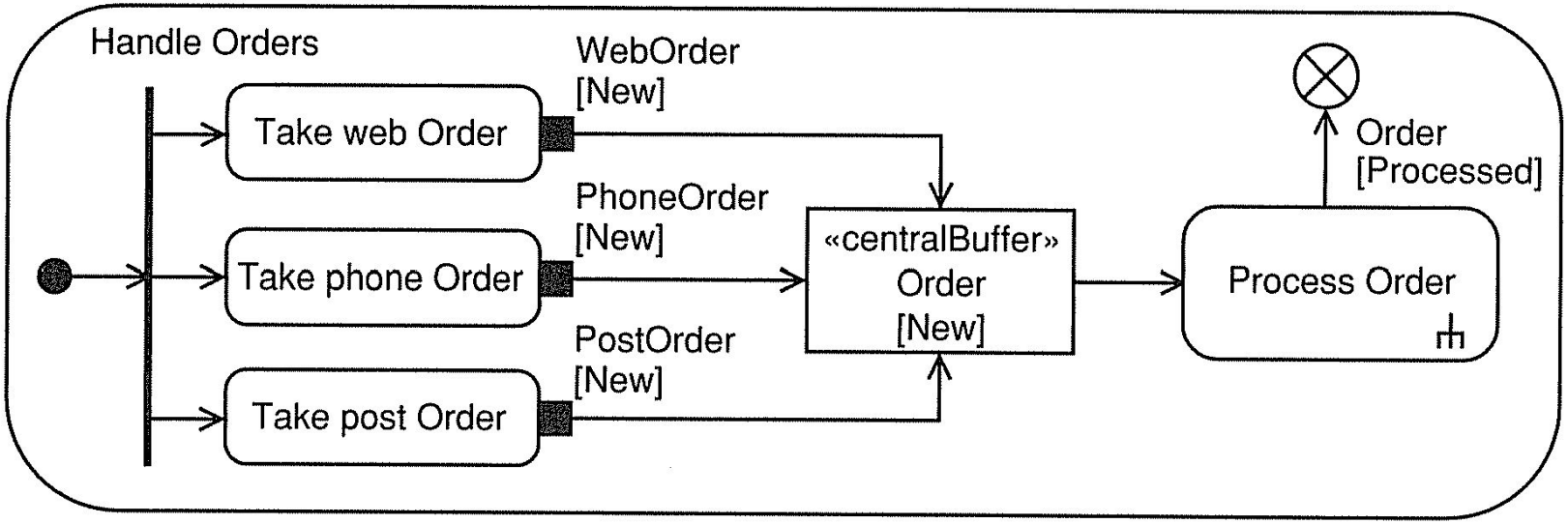
- Allow an action to have alternative sets of input and output pins
  - Input sets only have input pins
  - Output sets only have output pins
  - No mixed sets

Example: Parameter sets to provide alternative methods of authentication



<<centralBuffer>> node

- An object node used specifically as a buffer
- Allows to combine multiple input objects and distribute the objects among multiple output object flows



Style matters

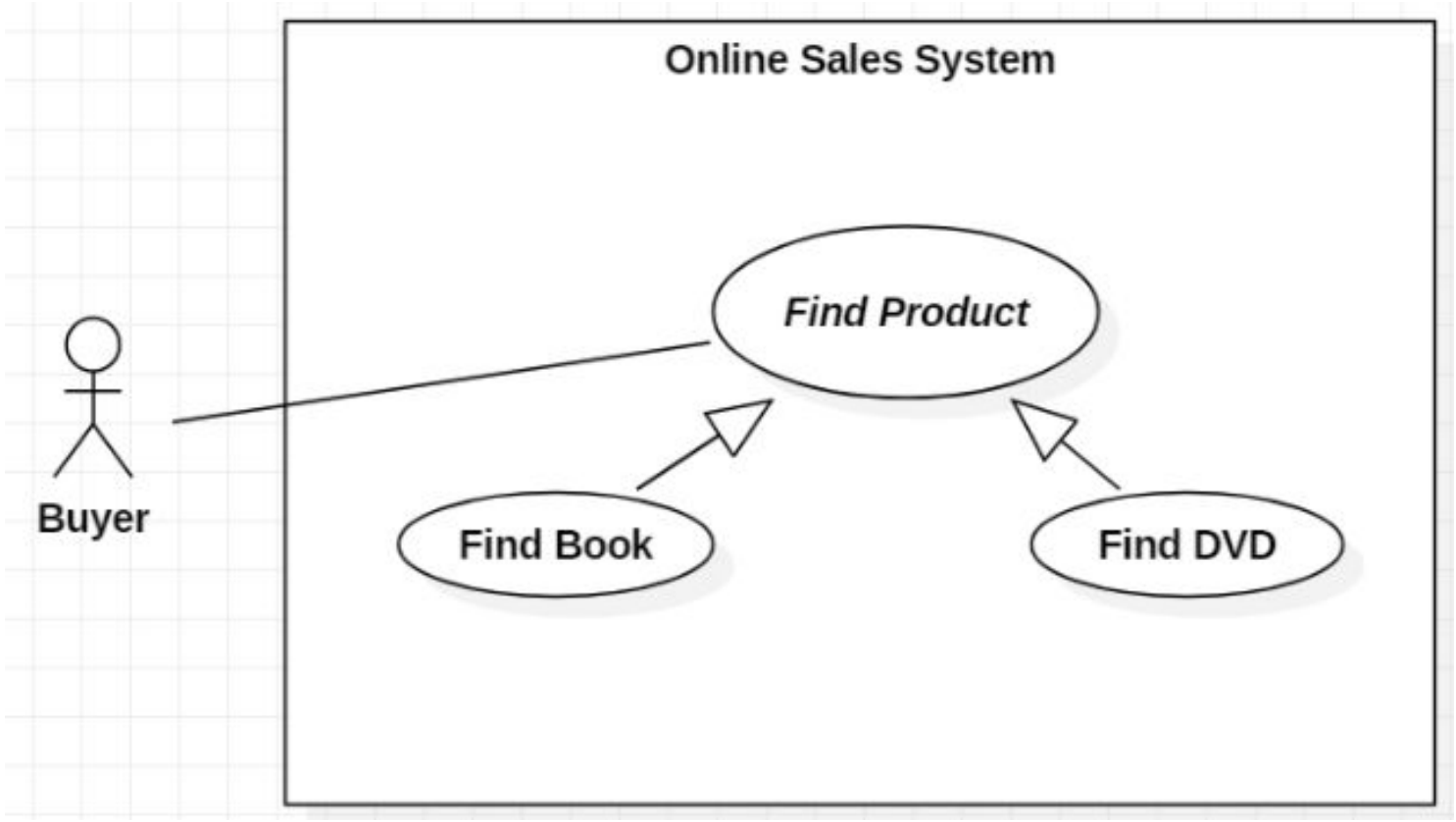
A few notes on style

- A good activity diagram
  - Communicates a particular perspective of the system's dynamics
  - Shows only the elements which are relevant for understanding that perspective
  - Offers only the level of detail to be understood
  - Is not excessively minimalistic to a point where the stakeholders reading it are uninformed about the activity's semantics

Hints and tips

- Use a name for the activity that communicates well its purpose
- Start by modelling the main purpose of the system
- Use the layout of your diagram to minimize lines crossing each other, where possible
- Make sure your activity diagram is consistent with all the other views of your model

Let us get back to Use cases for a moment



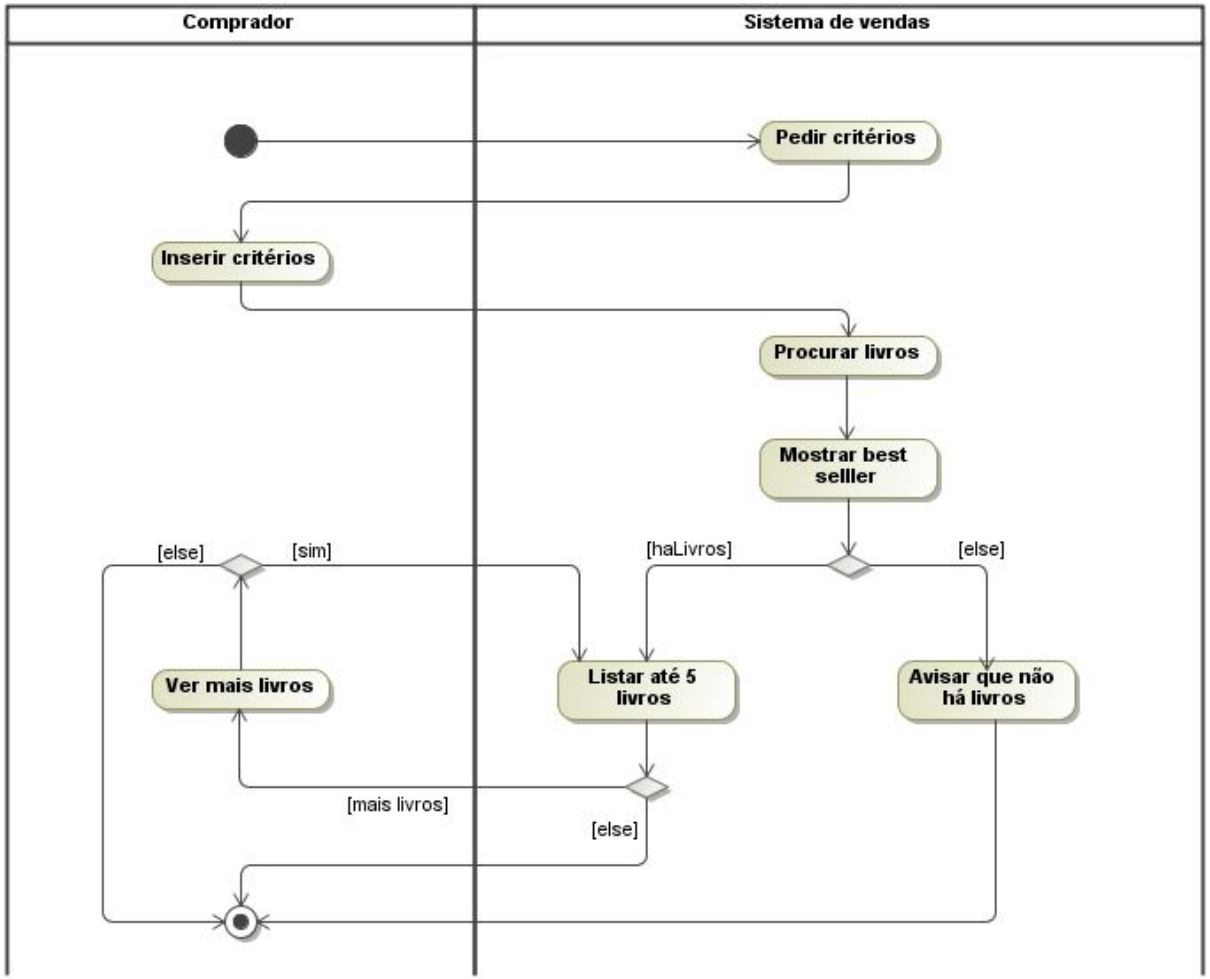
Specifying the *abstract* use case *Find Product*

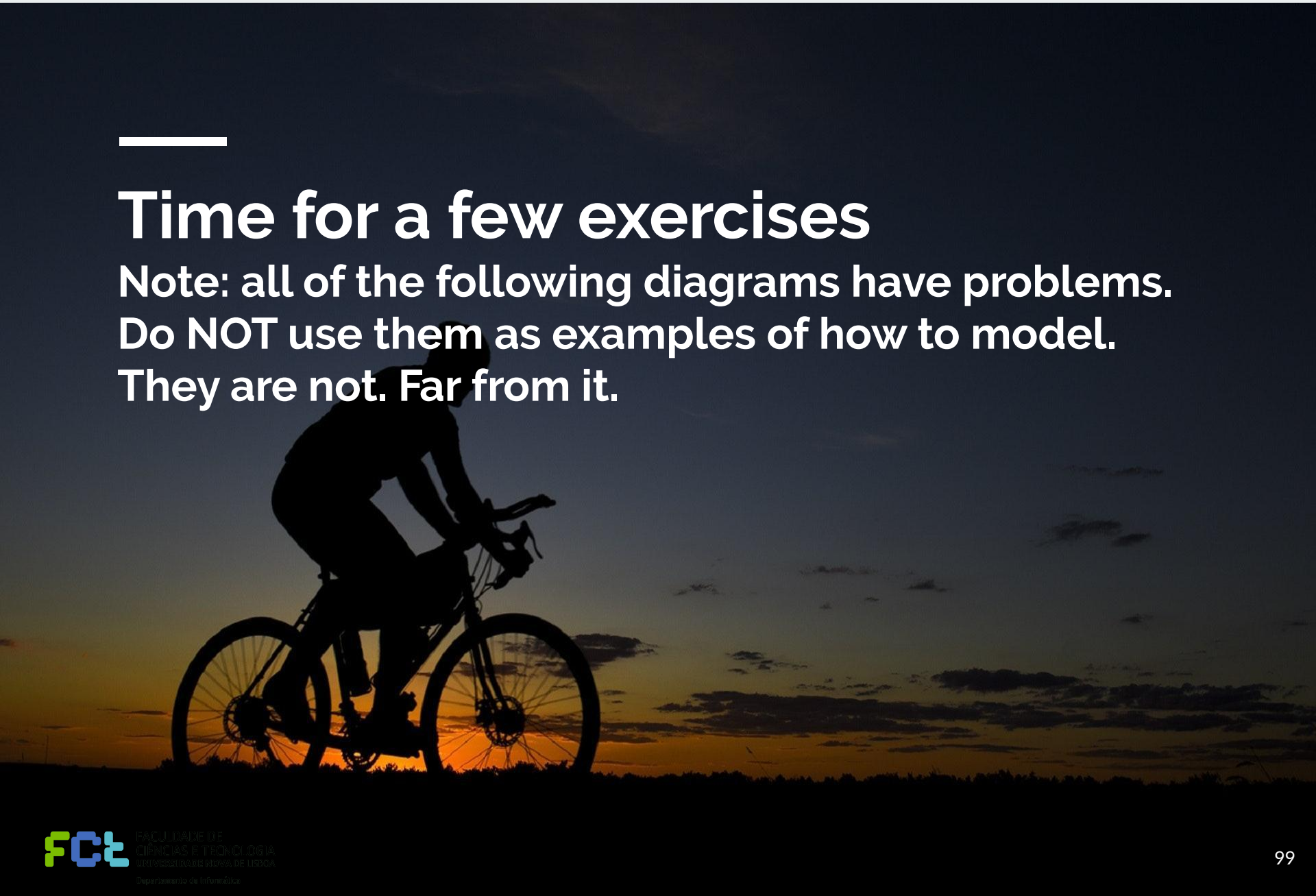
Use case: *Find product*
ID: 1
Description: The buyer tries to find a product in the system.
Main actor: Buyer
Secondary actors: None
Pre-conditions: None
Main flow: 1. The case study starts when the buyer selects the option to find a product. 2. The system asks the buyer to define the search criteria to use. 3. The buyer introduces the search criteria. 4. The system searches products satisfying the buyer's search criteria. 5. If the system finds products matching criteria for products 5.1. The system shows the user a list of products matching the search criteria. 6. Else 6.1. The system shows the buyer a message indicating no product was found.
Post-conditions: None
Alternative flows: None

Specifying the Find Book specialization

Use case: Find Book
ID: 2
Specializes: ***Find Product***
Description: The buyer searches a book in the system.
Main Actor: Buyer
Secondary Actors: None
Pre-conditions: None
Main flow: **1. (o1.) The use case starts when the buyer selects the option to find a book.** **2. (o2.) The system asks the buyer to to define the search criteria to use, which should include the author, title, ISBN, or topic.** 3. The buyer introduces the search criteria. **4. (o4.) The system searches for books satisfying the buyers search criteria.** **5. (o5.) If the system finds books matching the search criteria for books** 5.1. The system shows its best seller. **5.2. (o5.1.) The systems shows a list with up to 5 books matching the search criteria.** 5.3. For each book, the system presents the author, title, price and ISBN 5.4. While there are more books to show, the system offers the buyer the possibility of requesting the next page with more books. 6. Else 6.1. The system shows its best seller. **6.2. (6.1.) The system shows the buyer a message indicating no product was found.**
Post-conditions: None
Alternative flows: None

Create an activity diagram for the Find Book use case

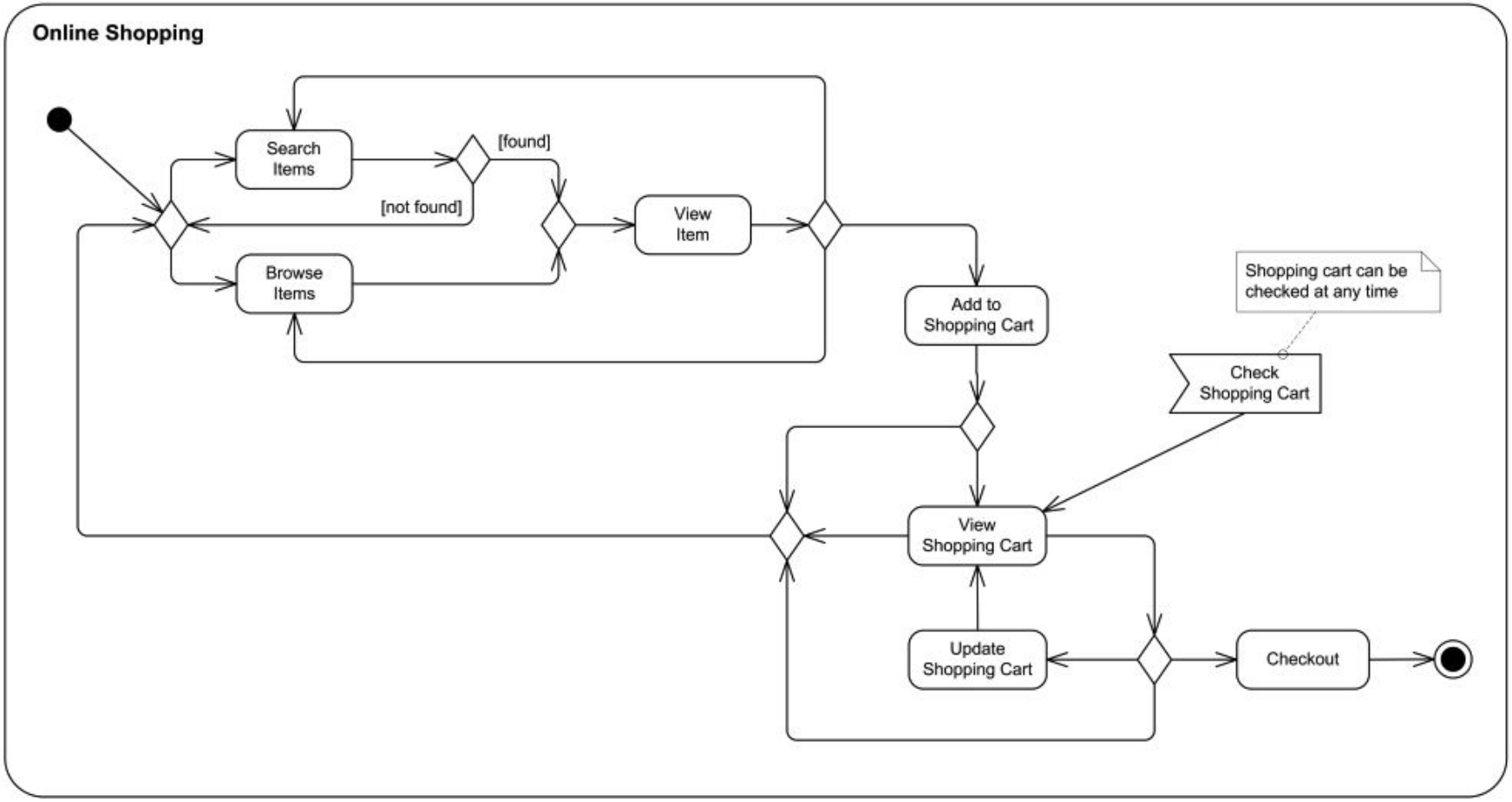




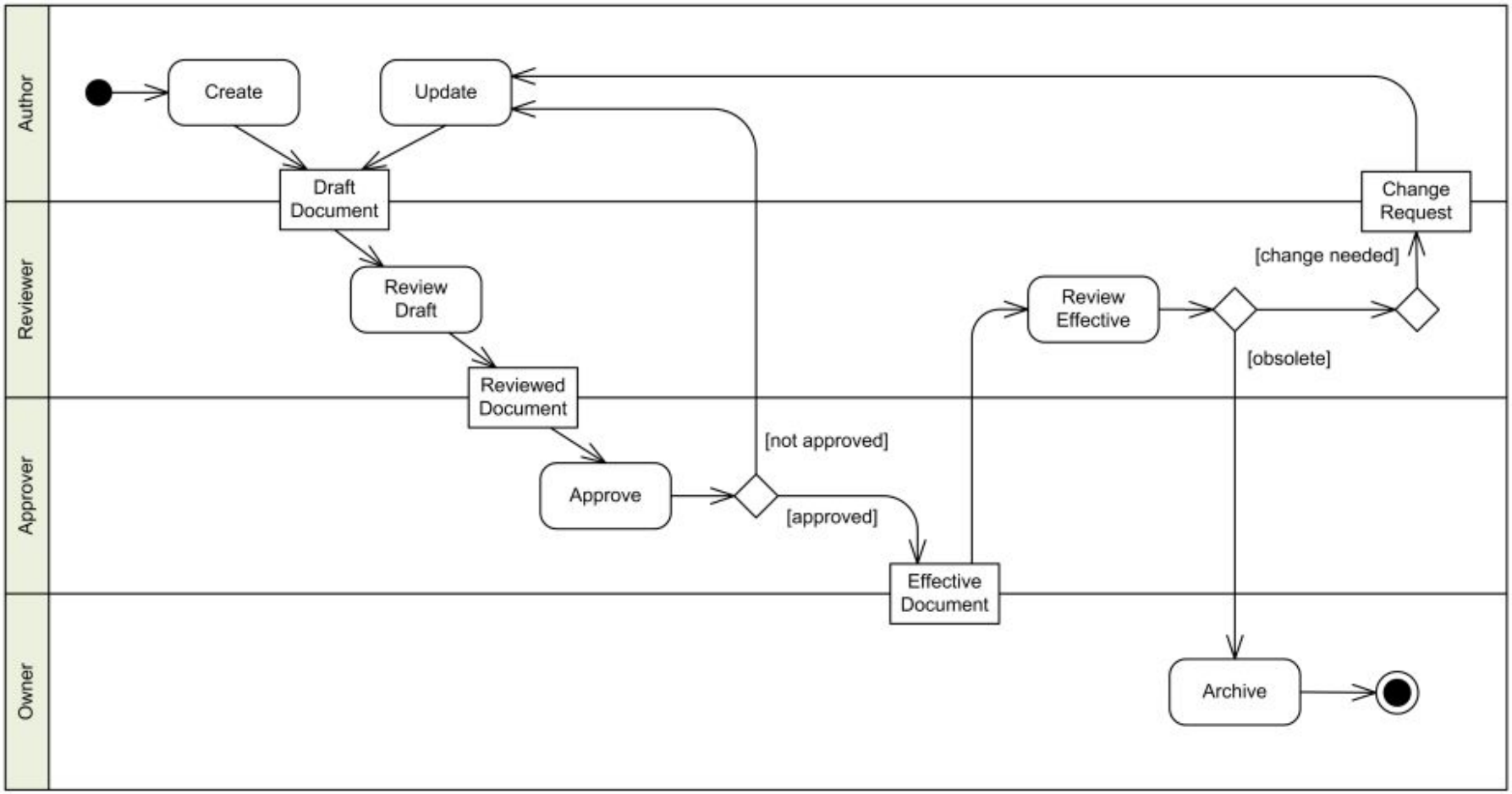
Time for a few exercises

Note: all of the following diagrams have problems.
Do NOT use them as examples of how to model.
They are not. Far from it.

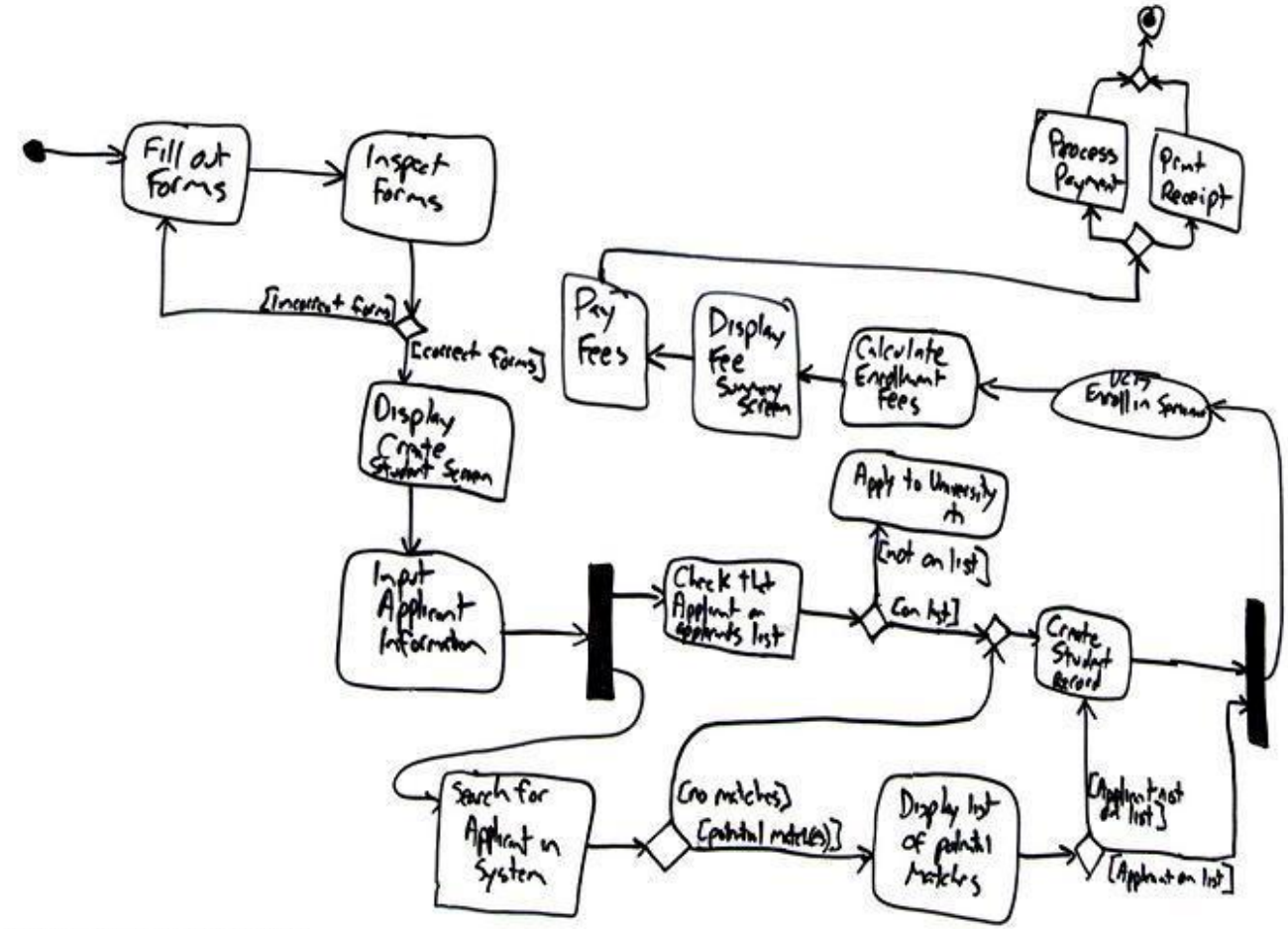
What is wrong with this activity diagram?



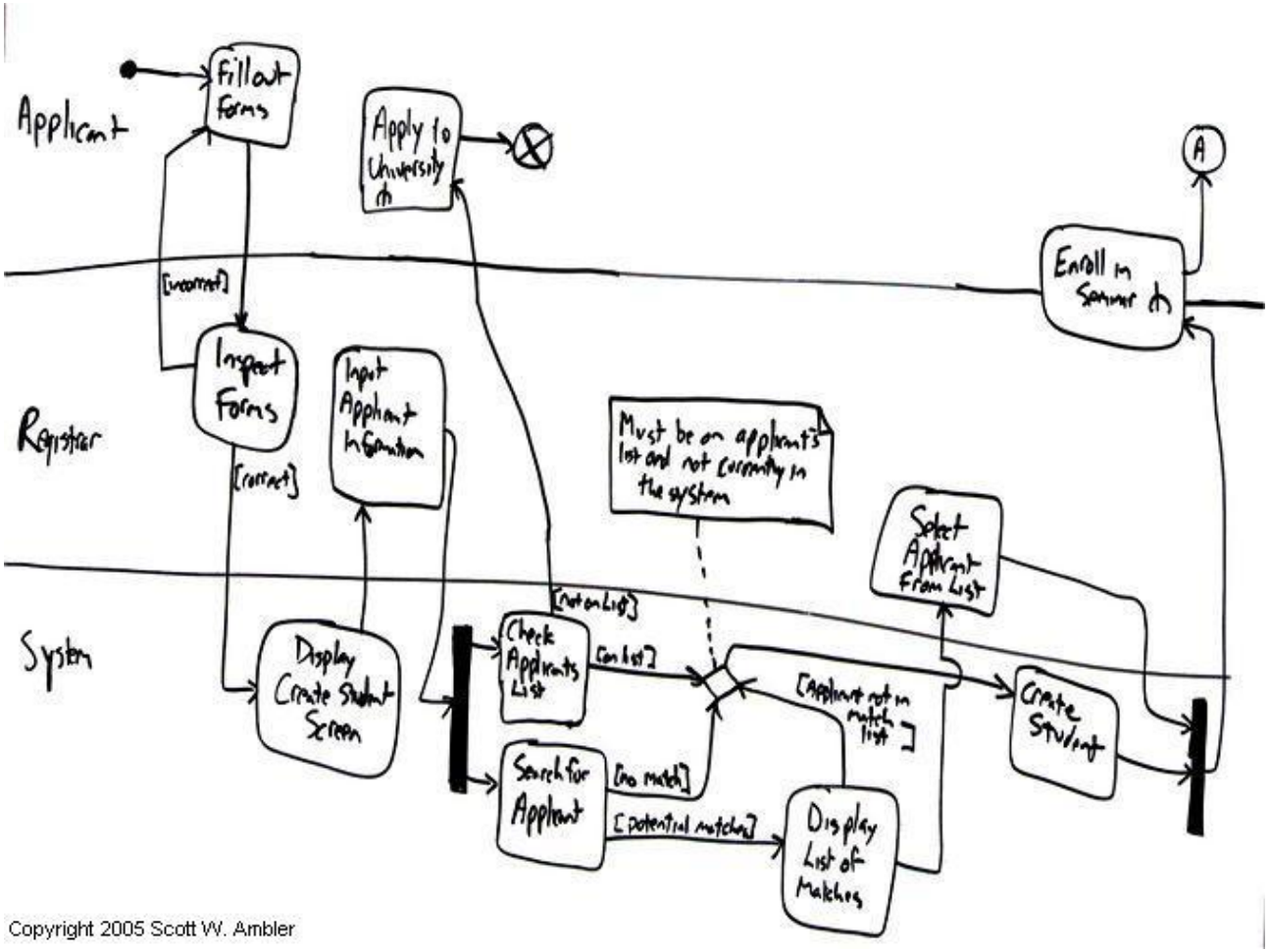
What is wrong with this model?



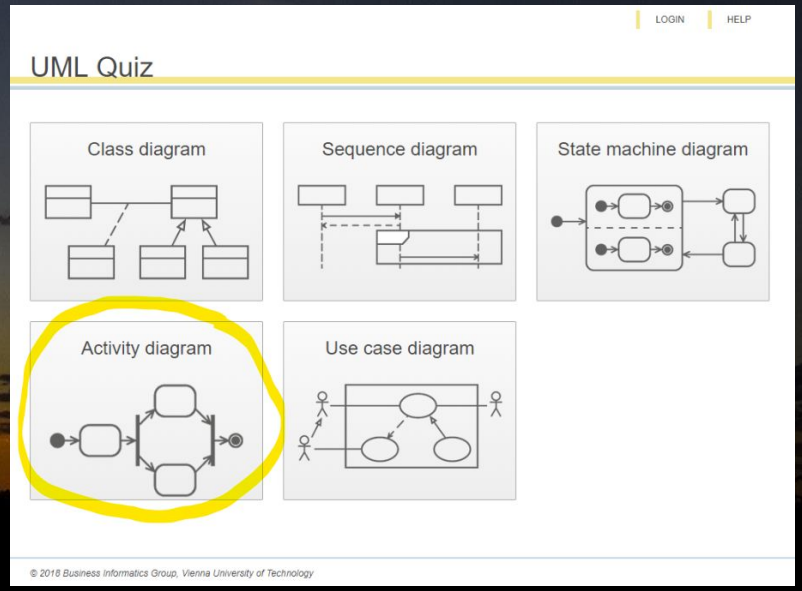
What is wrong with this model?



What is wrong with this model?



Did you really understand Activity Diagrams? Test yourself at: http://elearning.uml.ac.at/



Bibliography

Jim Arlow and Ila Neustadt, “UML 2 and the Unified Process”, Second Edition, Addison-Wesley 2006

- Chapters 14 to 15