Object Oriented Analysis & Design

Week 9

Activity Diagrams

Lecturer: Dr. Msagha J Mbogholi, PhD

Flashback from Lesson 8

- The UML specifies two types of statechart diagrams: behavioral state machines and protocol state machines.
- Statechart diagrams are composed of states, events and complex states.
- A transition is a relationship between two states that indicates that when an event occurs, the object moves from a prior state to a subsequent state.
- Further in both behavioral and protocol state machines if a transition has no event it is termed an automatic transmission; in such circumstances the transition fires/takes place when a guard condition is true, or precondition is true.
- Junction pseudo states represent places/points where transitions merge or branch.
- Branching transitions are used to show a branch without a merge.
- An internal transition is a way to handle events without leaving a state (or activity) and dispatching its exit or entry actions
- Depending on the number of regions there are two types of composite states provided in the UML: simple composite state and orthogonal composite state.

Content

- Introduction
- Activity semantics
- Action nodes
- Control nodes
- Object nodes

Part 1

Introduction

Introduction

- The activity diagram together with the state diagram represent the dynamic model. The dynamic model also supports other diagrams such as sequence diagrams.
- Let us refresh on the views and models that make up the system (Stevens and Pooley, 2006).
- There are 3 models of the system: use case model, static model and dynamic model.
- These are represented by the (4 +1) views of the system, namely:
- Use case view – this is the (+1) view represented by the use case model
- the **logical** view is taken by the class model, and by interaction diagrams and state diagrams to the extent that they are used to specify the logical behavior of the system;

Introduction (cont'd)

- the **process** view is taken by interaction diagrams, state, and activity diagrams, to the extent that they are used to determine the threads of control of the system, and by deployment diagrams;
- the **development** view is taken by the component structure diagrams, and by packages wherever they arise;
- the **physical** view is taken by deployment diagrams.
- Each view will consist of some model elements but not others.
- The activity diagram which is the topic of this lesson steers away from the concept of model elements; it focuses rather on processes (activities) and how they flow in the problem domain.

Introduction (cont'd)

- One of the advantages of activity diagrams is that anybody with some basic knowledge of flowcharting can understand them; suffice to say, even without flowcharting knowledge they are easily explained.
- In this lesson the semantics of activity diagrams are discussed.
- The lesson then discusses the different elements that make up an activity diagram while using simple examples for demonstration.
- By the end of the lesson the learner is expected to see the use of activity diagrams in business modelling and how they simplify the overall understanding of activities in the problem domain.

Part 2

Activity semantics

Introduction

- Activity diagrams can simply be described as OO flowchart diagrams.
- As will be seen they will often remind you of statechart diagrams; this is because in the UML1 they were considered special cases of state machines.
- Subsequently though they have grown their own feet and in UML2 are a construct by their own right. Their semantics are now based on petrinets.
- A Petri Net is a graph model for the control behavior of systems exhibiting concurrency in their operation. (Dennis, 2011). Further petri net consists of four elements: places, transitions, edges, and tokens. Graphically, places are represented by circles, transitions by rectangles, edges by directed arrows, and tokens by small solid (filled) circles. This structure is what activity diagrams follow in their construction.
- Activity diagrams allow modelling of a process as an activity that consist of a collection of nodes that are connected at their edges.
- The use of petri nets has contributed significantly to show the distinction between activity diagrams and state chart diagrams.

Introduction (cont'd)

- From its description an activity diagram can be attached to any modelling element in OO for the purpose of modelling its behavior. This includes:
  - Use cases
  - Classes
  - Interfaces
  - Components
  - Collaboration
  - Operations
  - Business processes
- The activity diagram focuses on communicating a specific aspect of the system's dynamic behavior.

2.1 Activity diagrams and the UP

- Activity diagrams can be used anywhere in the Unified Process (UP) since they model dynamic behavior of any aspect of the system; however, they are used mostly in analysis workflow.
- Since the static structure of the system is not taken into account when designing an activity diagram they are mostly useful in certain areas.
- Not using the static structure means that we don't indicate classes *per se* as part of the activity diagram; in certain cases (like when using swim lanes) a class may be used to show responsibility or where the flow is happening. Nonetheless, the flow from one activity to another is the essence of the activity diagrams and not the classes, actors or objects.

2.1 Activity diagrams and the UP (cont'd)

- Arlow and Neustadt (2013) aver that in their experience activity diagrams have been used in the following ways as far as the UP is concerned:
- Analysis workflow:
  - Modelling flow in a use case to make it easier to understand;
  - Modelling flow between use cases
- Design – modelling details of an operation or an algorithm
- Business modelling – to model a business process
- When designing an activity diagram the modeler should ensure that design should always be done with the audience in mind. The simpler the design the easier it is to understand and by extension, the more effective in making the audience understand it.

2.2 Activities

- In UML 2 activities are networks of nodes connected by edges. This may sound very technical; simply put activities are depicted as nodes and are connected to each other by edges (from one activity to the other).
- The nodes are divided into three categories:
  - Action nodes – represent units of work that are singular (atomic) within the activity
  - Control nodes – as the name implies they control the flow through the activity
  - Object nodes – represent objects used in the activity.
- Before looking at a simple example demonstrating the above (though they will be covered in detail in another section of this lesson) let us first introduce the symbols used in activity diagrams.

2.2 Activities (cont'd)

- Start state – signals the beginning of the activity diagram. It is shown as a filled dot (similar to the statechart diagram). This is shown in fig 1.
- End state – it signals the end of the activity diagram and is denoted as a bullseye, similar to the statechart diagram. It is shown in fig 2.

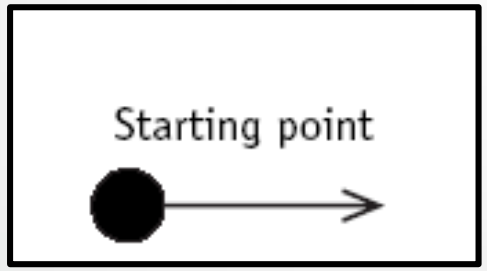


Fig 1. Start point

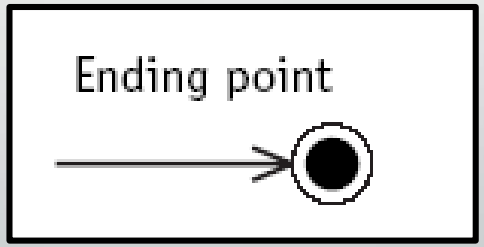


Fig 2. End point

2.2 Activities (cont'd)

- Action state - An action state represents processing as an element fulfills a responsibility. There are various types of action states, including simple, initial, and final action. A simple action state represents processing, and is shown as a shape with a straight top and bottom and convex arcs on the two sides, and is labeled with the name of an operation or a description of the processing (etutorials.org). Fig 3 shows a labeled action.

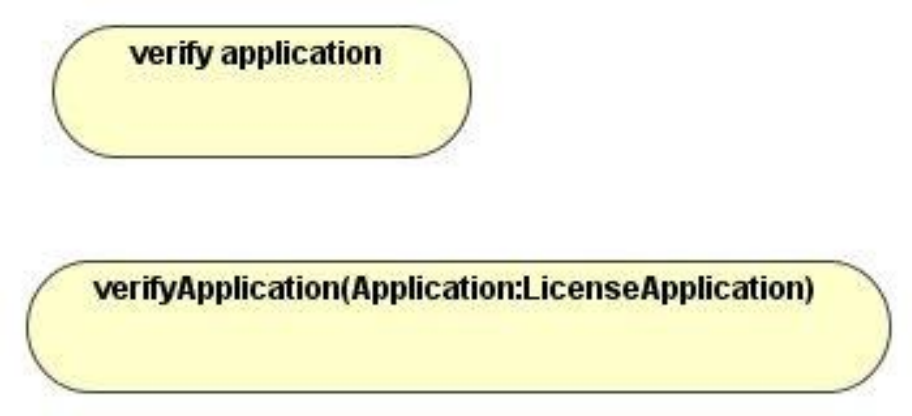


Fig 3. Labeled actions (Ojo and Estevez, 2005)

2.2 Activities (cont'd)

- Activity – is used to represent a set of actions and is labelled by its name.
- Control flow – is represented by an arrow and is used to show the sequence of execution.
- Fig 4 shows an activity diagram for a simple business process *send letter* with pre and post conditions.
- The actions required to fulfill the process of sending a letter are simply to write the letter, address it, and post it.
- The pre condition is that you need to know the topic of the letter (otherwise why write in the first place?) and the post condition is to know where you will send it (the address).

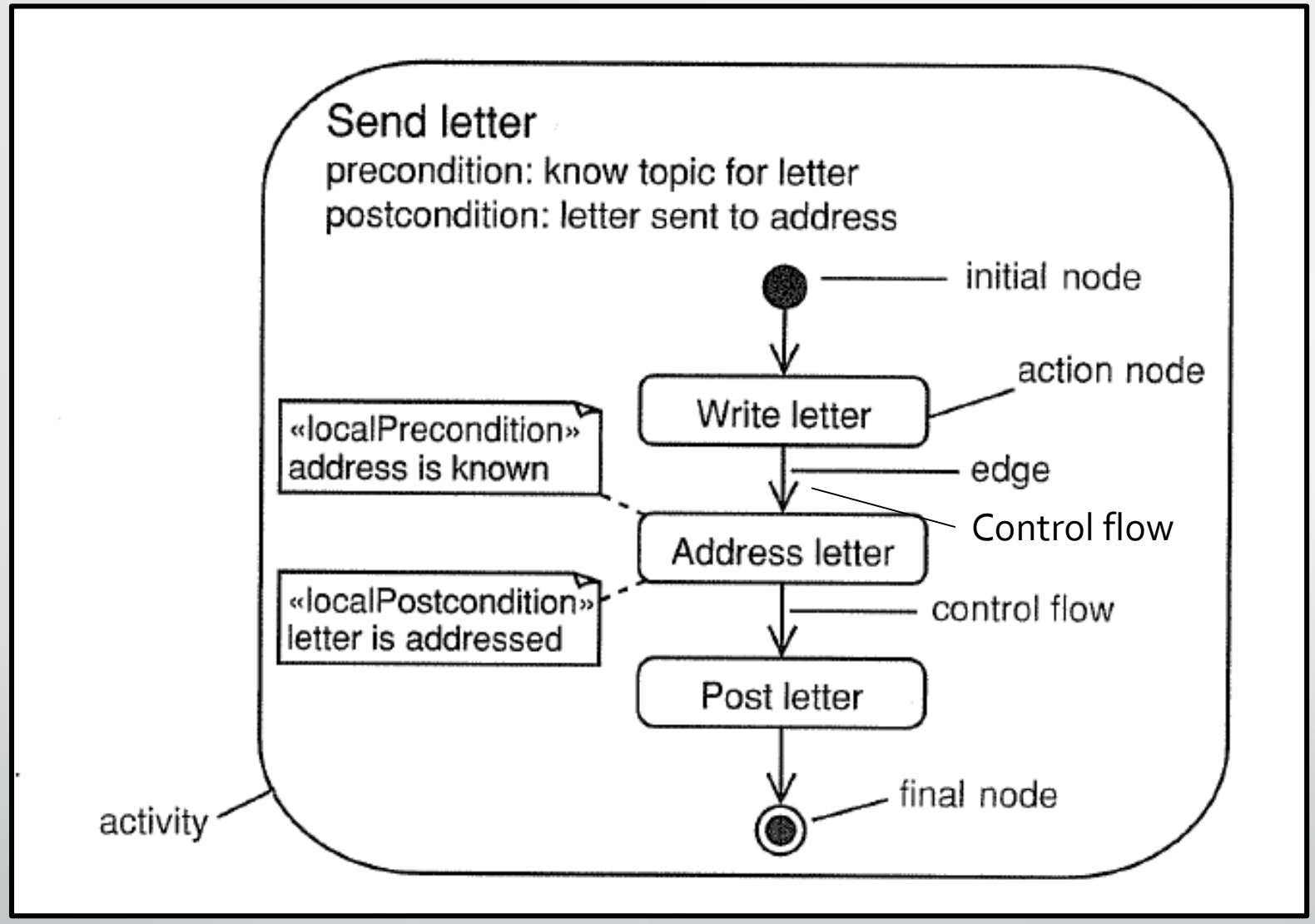


Fig 4. Activity diagram for sending a letter (Arlow and Neustadt, 2013)

2.2 Activities (cont'd)

- Another common use of activity diagrams is to model use cases as a series of actions.
- This makes it much easier to understand the use case.
- As you recall the use case is pivotal in understanding the requirements.
- When a use case is elaborated on using an activity diagram or several activity diagrams expounding on the processes involved, it makes it that much easier to understand the requirements.
- Fig 5 and fig 6 illustrate this concept. In fig 5 describes the use case *PaySalestax* while fig 6 expresses the use case as an activity diagram. It is not lost on the learner that the activity diagram makes the actions required to fulfill the use case much simpler to understand; in this case only two actions are required to fulfill the use case, namely "calculate sales tax" and "send electronic payment".
- Of course another activity diagram may need to be designed to show how to calculate sales tax as a series of atomic, discrete actions; same as for send electronic payment action. However, at this abstract level, the user has a clear idea of how to go about realizing the use case.

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.

Fig 5. PaySalestax use case. (Arlow and Neustadt, 2013)

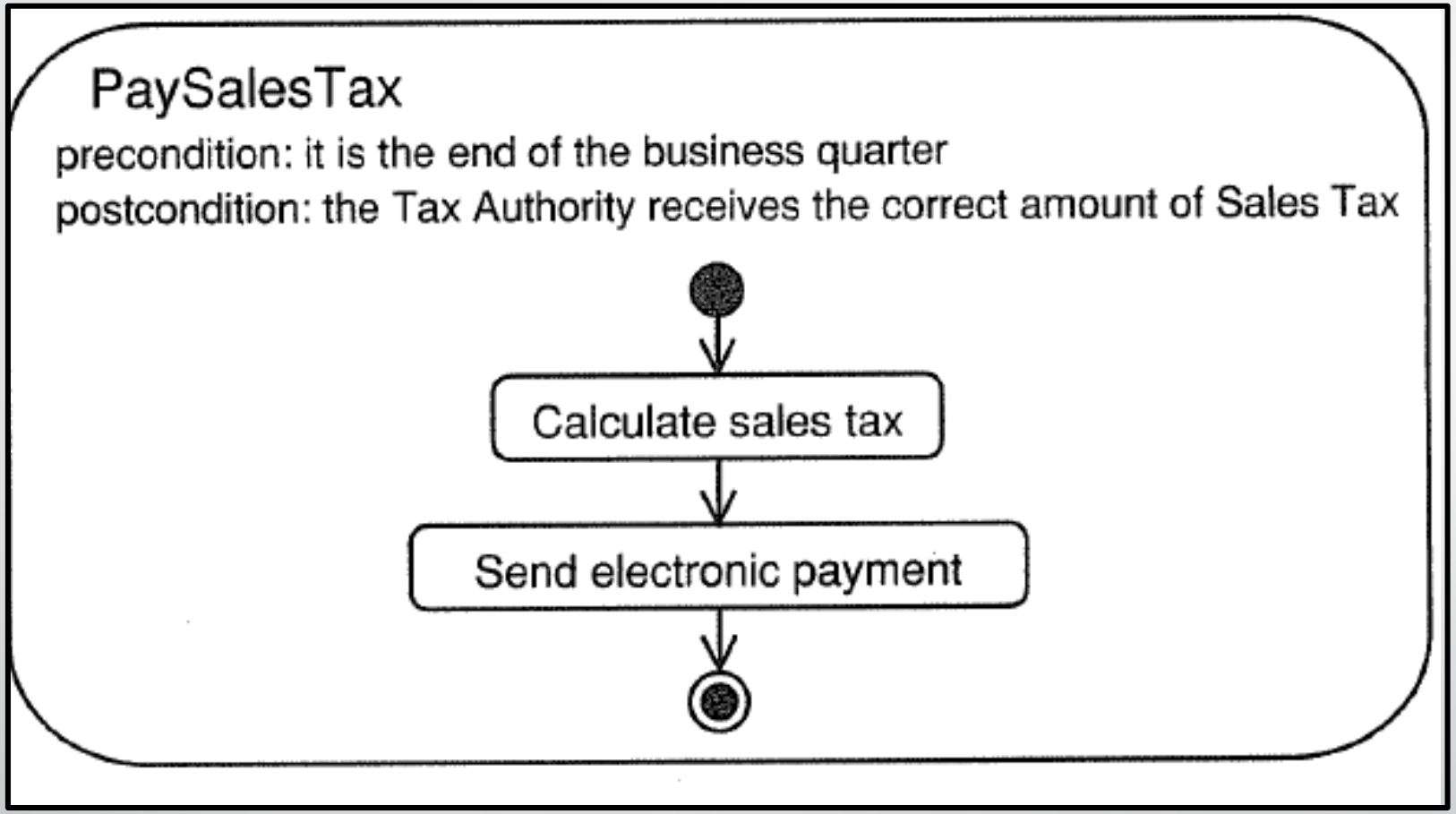


Fig 6. Activity diagram for PaySalestax (Arlow and Neustadt, 2013)

2.3 Activity semantics

- Activity diagrams model behavior based on the use of tokens.
- Tokens flow around the nodes and edges in the network based on some rules. If you are familiar with the concept of token passing in networks then this is not very different; the token in a network passes from one computer to another based on some rules. Whoever, has the token gets to send information while others wait for their turns.
- In activity diagrams the tokens determine the flow based on the rules.
- In UML2 the tokens in activity diagrams can be represented by:
- Flow of control;
- An object
- Some data
- For example in fig 7 the token is the flow of control.

2.3 Activity semantics (cont'd)

- Edges connect source nodes and target nodes.
- Tokens are moved across edges and their movements are subject to certain conditions being fulfilled (satisfied). It is only when these conditions have been satisfied can a token move to the target node.
- In fig 7 the (action) nodes have the following conditions:
  - Post conditions of the source node;
  - Guard conditions at the edge;
  - Preconditions of the target node.

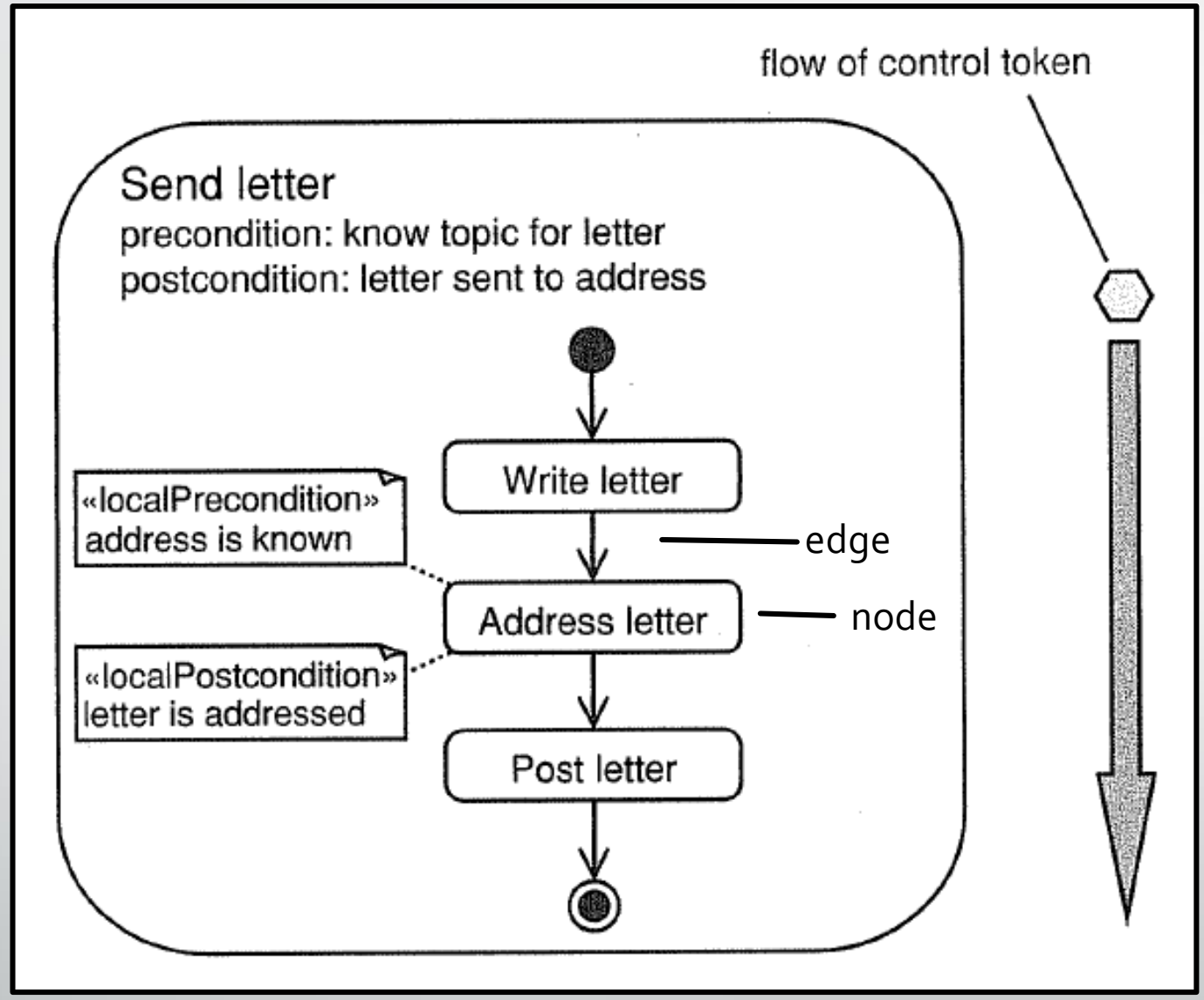


Fig 7. Flow of control token (Arlow and Neustadt, 2013)

2.3 Activity semantics (cont'd)

- Together with action nodes as seen in fig 7 there are also control nodes and object nodes.
- Control node semantics control how a token passes from its input edge to its output edge; for example a join node (described later) will offer a token on its single output edge, only if there are tokens on all its input edges.
- Object nodes represent objects flowing around the system.

2.3 Activity semantics

- Swimlanes (also known as activity partitions) - A swimlane is a way to group activities performed by the same actor on an activity diagram or to group activities in a single thread. The activities can be partitioned vertically, horizontally or using curved lines. The partitioning by grouping related activities is what is referred to as partitioning; the resulting activity diagram is called an activity diagram with swimlanes.
- The UML2 does not restrict how to create the partitions as long as they fulfill the desired objective; consequently, the designer can partition along use cases, classes, departments, or even roles.
- Activities can even be nested. Fig 8 shows hierarchical nesting where activity partitions are done according to location and department (in London); fig 9 shows an activity diagram without swimlanes for the business process of meeting a new client, and fig 10 shows the same activity diagram with swimlanes.
- Lastly Dennis et al. (2015) summarize activity semantics in table 1.

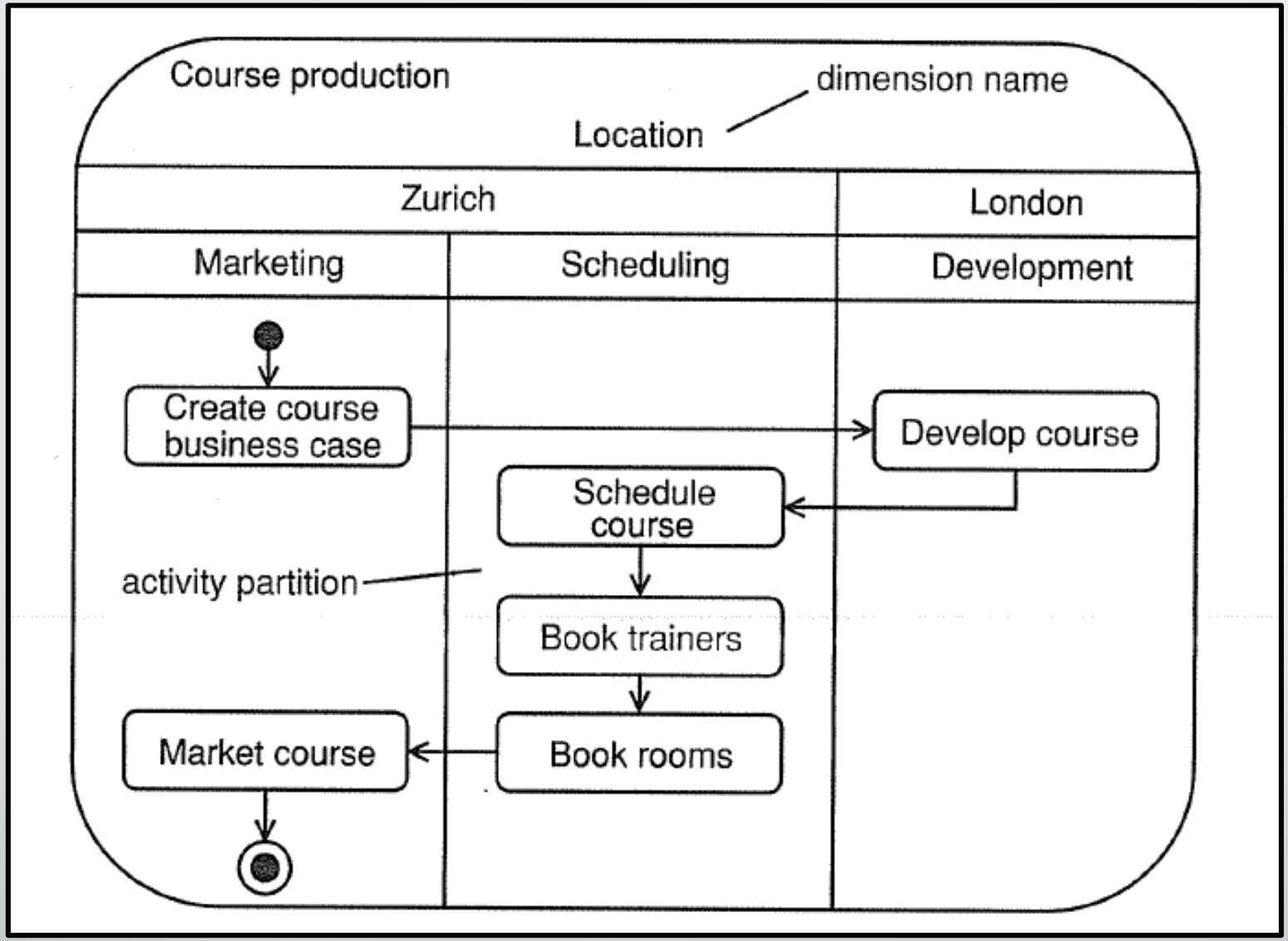


Fig 8. Activity partitioning (Arlow and Neustadt, 2013)

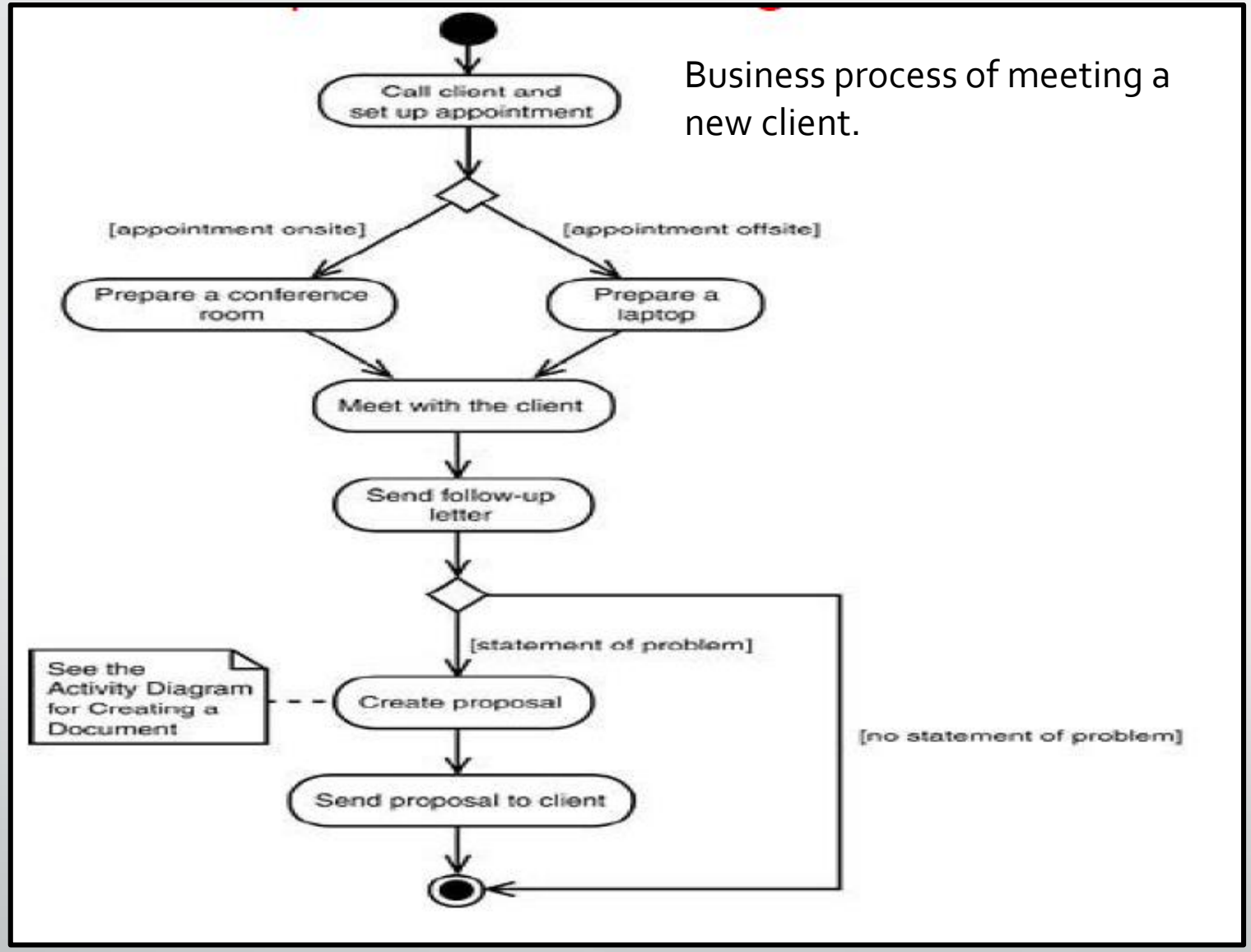


Fig 9. Activity diagram without swimlanes (visual-paradigm.com)

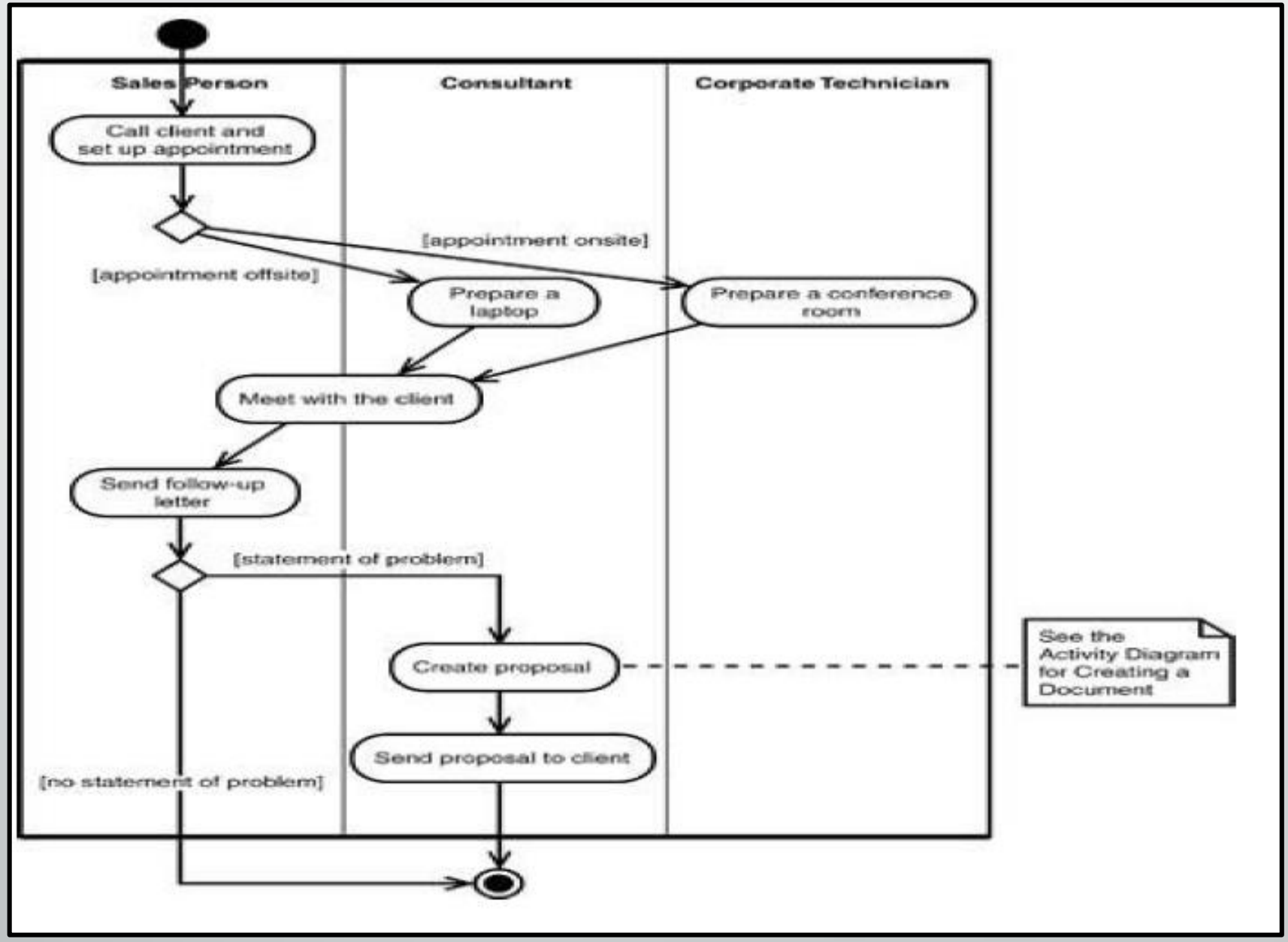


Fig 10. Activity diagram with swimlanes. (visual-paradigm.com)

Table 1. Activity diagram semantics (Dennis et al., 2015)

An action: ■ Is a simple, nondecomposable piece of behavior. ■ Is labeled by its name.	Action
An activity: ■ Is used to represent a set of actions. ■ Is labeled by its name.	Activity
An object node: ■ Is used to represent an object that is connected to a set of object flows. ■ Is labeled by its class name.	Class Name
A control flow: ■ Shows the sequence of execution.	
An object flow: ■ Shows the flow of an object from one activity (or action) to another activity (or action).	
An initial node: ■ Portrays the beginning of a set of actions or activities.	
A final-activity node: ■ Is used to stop all control flows and object flows in an activity (or action).	
A final-flow node: ■ Is used to stop a specific control flow or object flow.	
A decision node: ■ Is used to represent a test condition to ensure that the control flow or object flow only goes down one path. ■ Is labeled with the decision criteria to continue down the specific path.	[Decision Criteria] [Decision Criteria]
A merge node: ■ Is used to bring back together different decision paths that were created using a decision node.	
A fork node: Is used to split behavior into a set of parallel or concurrent flows of activities (or actions)	
A join node: Is used to bring back together a set of parallel or concurrent flows of activities (or actions)	
A swimlane: Is used to break up an activity diagram into rows and columns to assign the individual activities (or actions) to the individuals or objects that are responsible for executing the activity (or action) Is labeled with the name of the individual or object responsible	Swimlane

Part 3

Action nodes

3.1 Introduction

- Action nodes represent some work being carried out and are shown as a rectangle with rounded corners; they execute based on two scenarios:
- There is a token simultaneously on each input edge;
- Input tokens satisfy all the action node local preconditions.
- Action nodes perform a logical AND on all the input tokens; this means that all input tokens must be present (returning a value of true on the logical AND).
- Nonetheless even if all the input tokens are present the node will only execute if the local preconditions are satisfied.
- This is generically represented in fig 11.

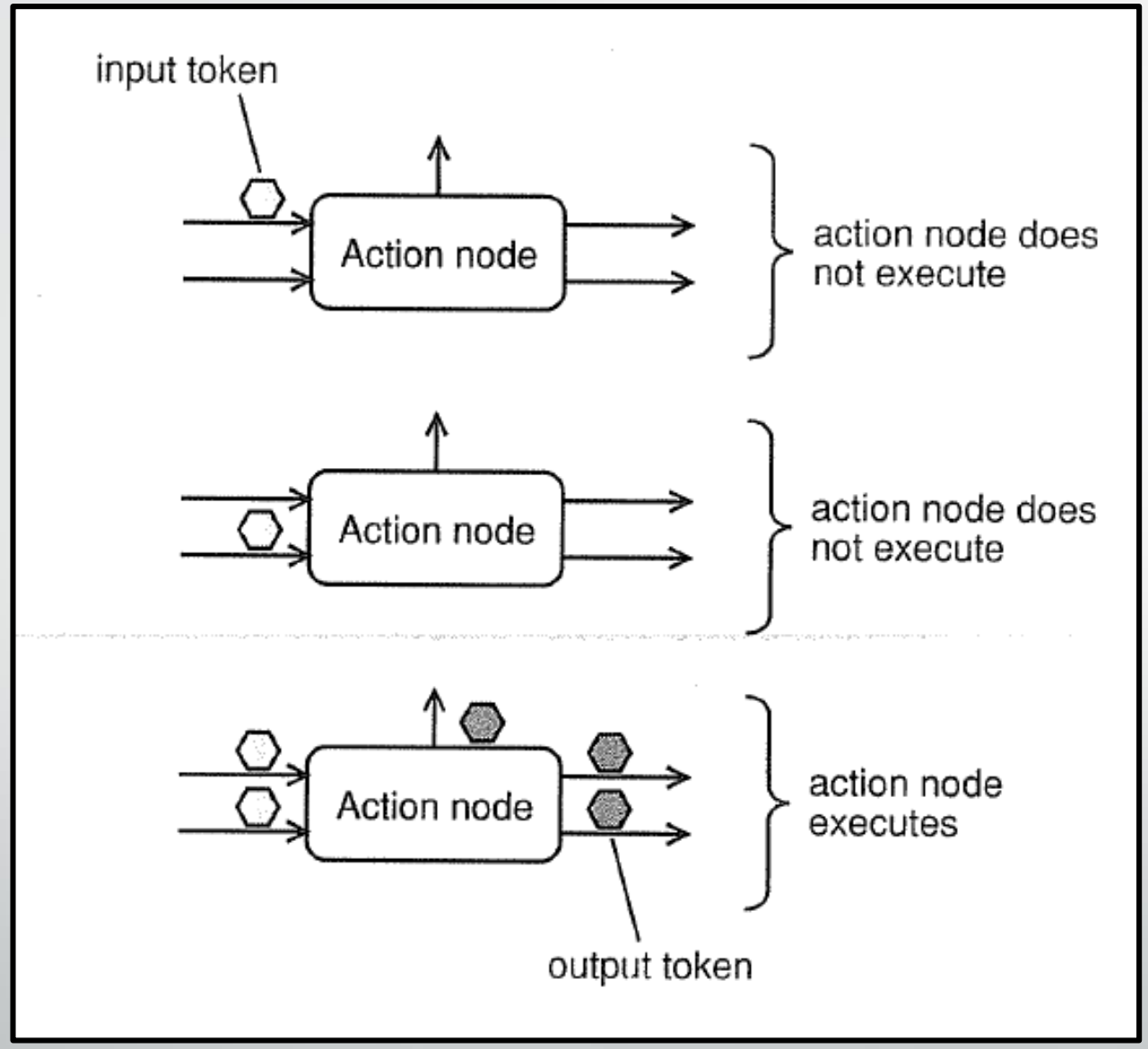


Fig 11. Action node execution. (Arlow and Neustadt, 2013)

3.2 Execution

- Once the action node finishes execution the local post condition is checked (see fig 7 for example); if this is satisfied the node offers tokens simultaneously to all its output edges. This implies a fork (see table 1) which instigates concurrency in turn.
- There is no specific naming convention for action nodes; however, since they are verbs mostly, it is appropriate to use a verb that clearly indicates the action that the node performs.
- There are four types of action nodes.

3.2.1 Call action node

- This type of node can invoke either of the following:
  - An activity
  - A behavior
  - An operation
- Fig 12 shows a few examples of syntax used for call action nodes.
- At analysis level action nodes in activity diagrams tend to specify behavior while operations tend to be invoked (used) in design.

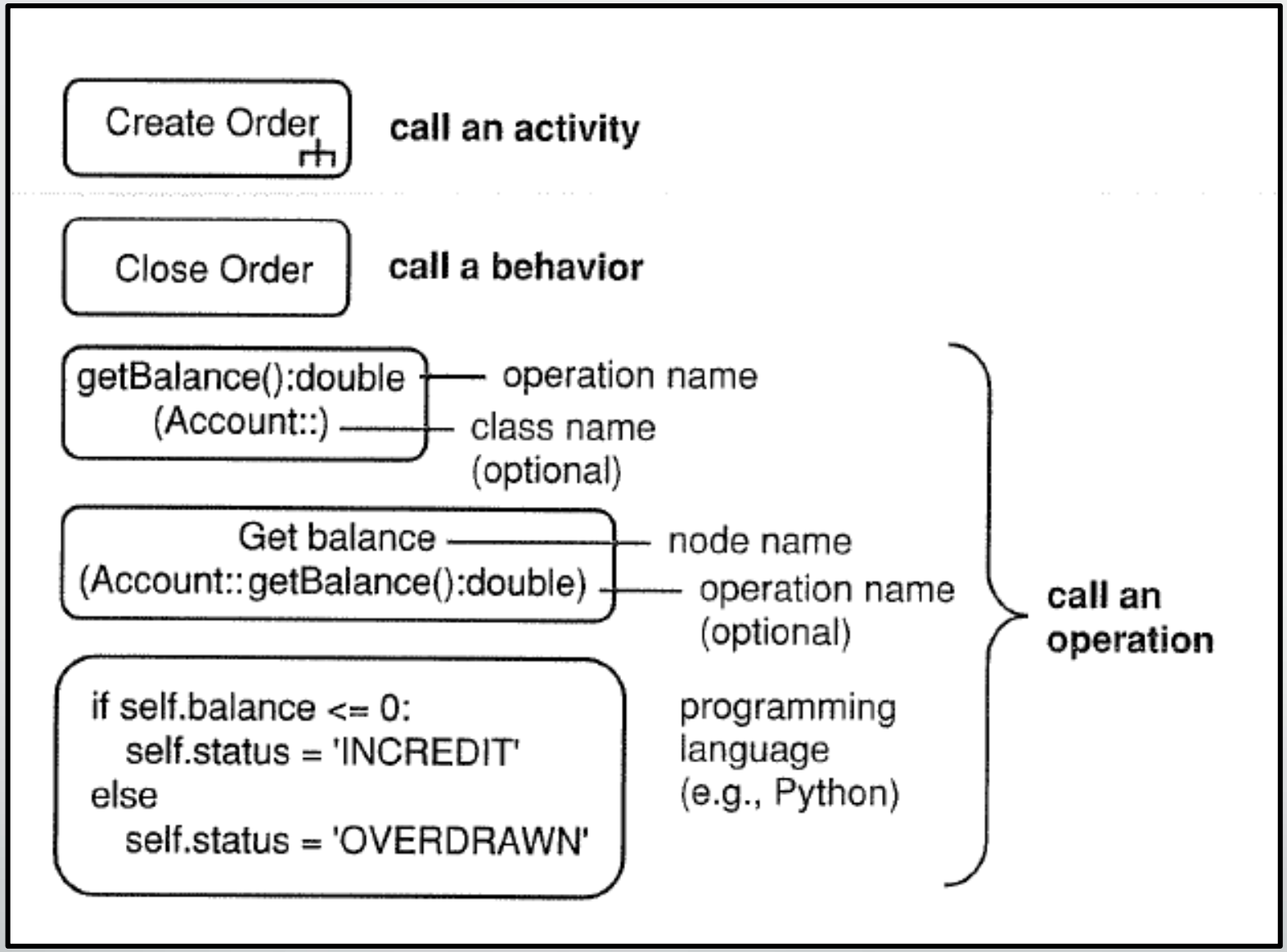


Fig 12. Call action node syntax (Arlow and Neustadt, 2013)

3.2.2 Send signal node

- This node sends a signal asynchronously; that is to say it does not wait for confirmation of signal receipt from the receiver's end
- It may accept input parameters to create the signal. (Arlow and Neustadt, 2013)
- Fig 13 shows the notation for this node.

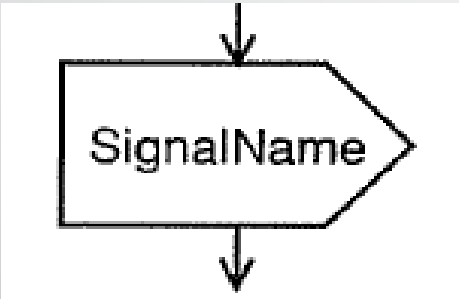


Fig 13. Send signal node

3.2.3 Accept event action node

- Arlow and Neustadt (2013) describe this node as follows:
- Accepts an event – waits for the event detected by its owning object and offers the event on its output edge.
- It is enabled when it gets a token on its input edge
- If there is no input edge it starts when its containing activity starts and it is always enabled.
- Fig 14 shows its notation in the UML.

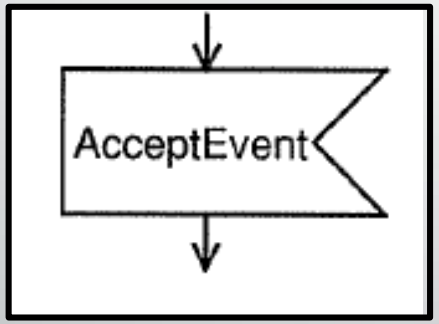


Fig 14. Accept event action node

3.2.4 Accept time event action node

- This node responds to time. It has a time expression and generates a time event when the time expression returns true. It behaves differently dependent on the presence or absence of an input edge.
- Fig 15 (a) shows the notation in the UML while fig 15 (b) shows an example of its use.

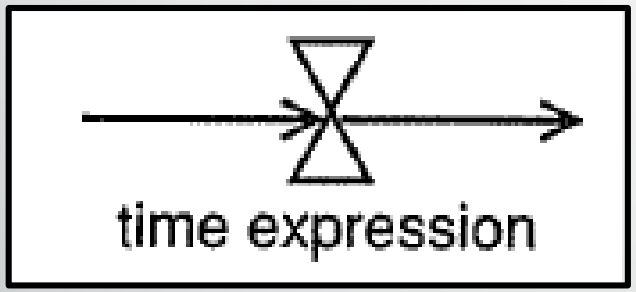


Fig 15 (a) time event action node

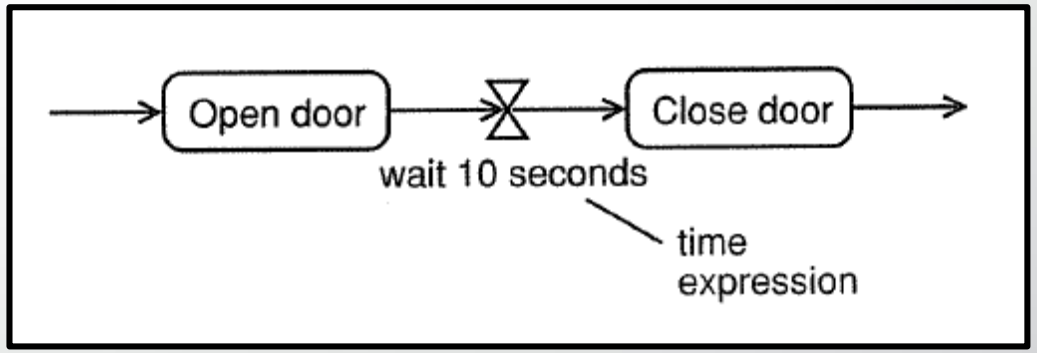


Fig 15 (b) time event action node example (Arlow and Neustadt, 2013)

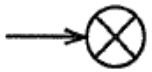
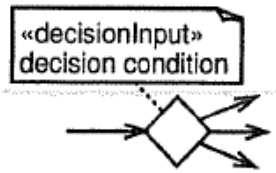
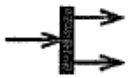
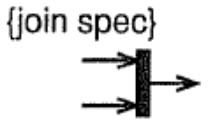
Part 4

Control nodes

4.1 Introduction

- Control nodes manage the flow in an activity.
- In activity diagrams the following constructs are used for control purposes:
  - Initial node
  - Final node
  - Flow final node
  - Decision node
  - Fork node
  - Join node
- Table 2 describes these constructs and their purpose.
- Thereafter we examine a few examples to show their use.

Table 2. Control node constructs (Arlow and Neustadt, 2013)

Syntax	Name	Semantics	
	Initial node	Indicates where the flow starts when an activity is invoked	
	Activity final node	Terminates an activity	Final nodes
	Flow final node	Terminates a specific flow within an activity – the other flows are unaffected	
	Decision node	The output edge whose guard condition is true is traversed May optionally have a «decisionInput»	
	Merge node	Copies input tokens to its single output edge	
	Fork node	Splits the flow into multiple concurrent flows	
	Join node	Synchronizes multiple concurrent flows May optionally have a join specification to modify its semantics	

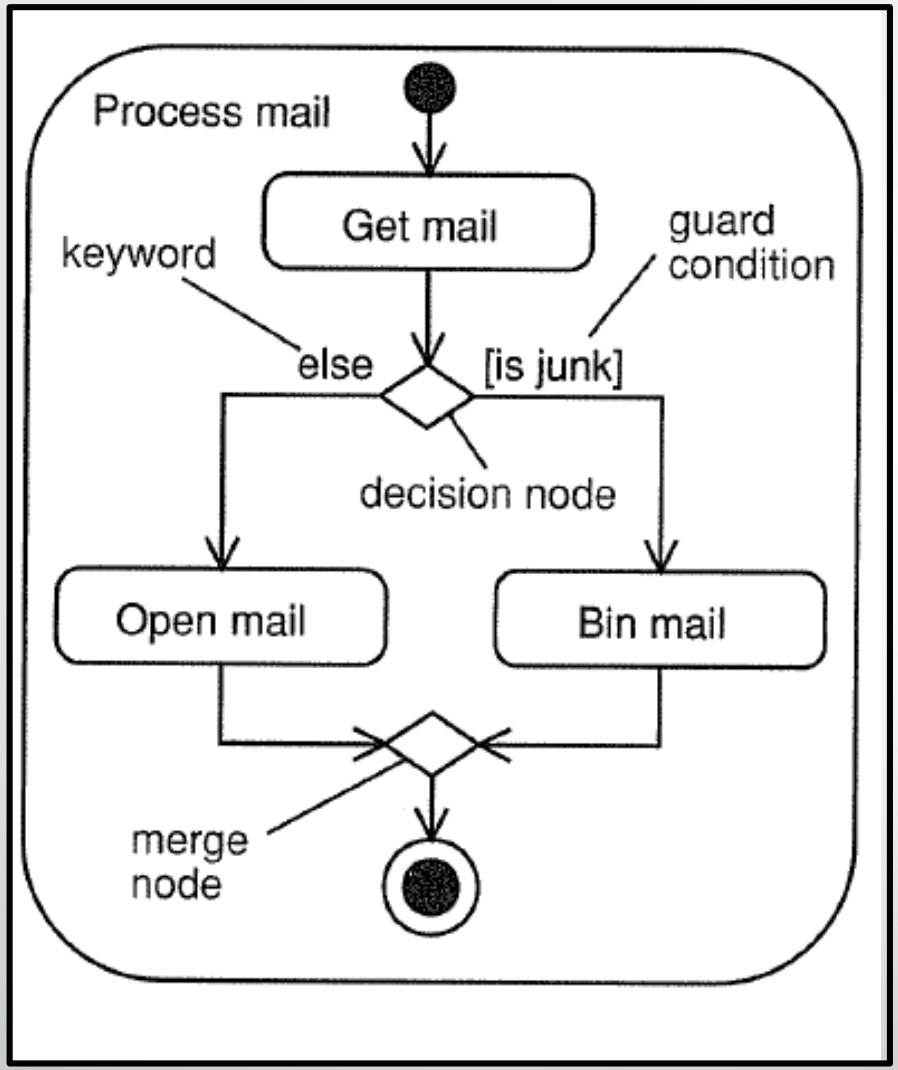


Fig 16. Decision node (Arlow and Neustadt, 2013)

4.2 Control nodes application

- Fig 16 demonstrates the use of a decision node in an activity diagram.
- Once flow has passed from the "get mail" action, the flow will move in one direction depending on the outcome of the classification of mail as being junk or not. This is based on rules in the decision node that make this determination. Thereafter the mail will either be opened or sent to the bin. The two flows then merge via the merge node before the activity terminates.
- Fig 17 demonstrates the use of forks and joins. These are used in situations where parallel and concurrent processes need to be modelled. Two parents work together to perform tasks in parallel (the forks) and then the flow comes back together before the activity finally terminates.

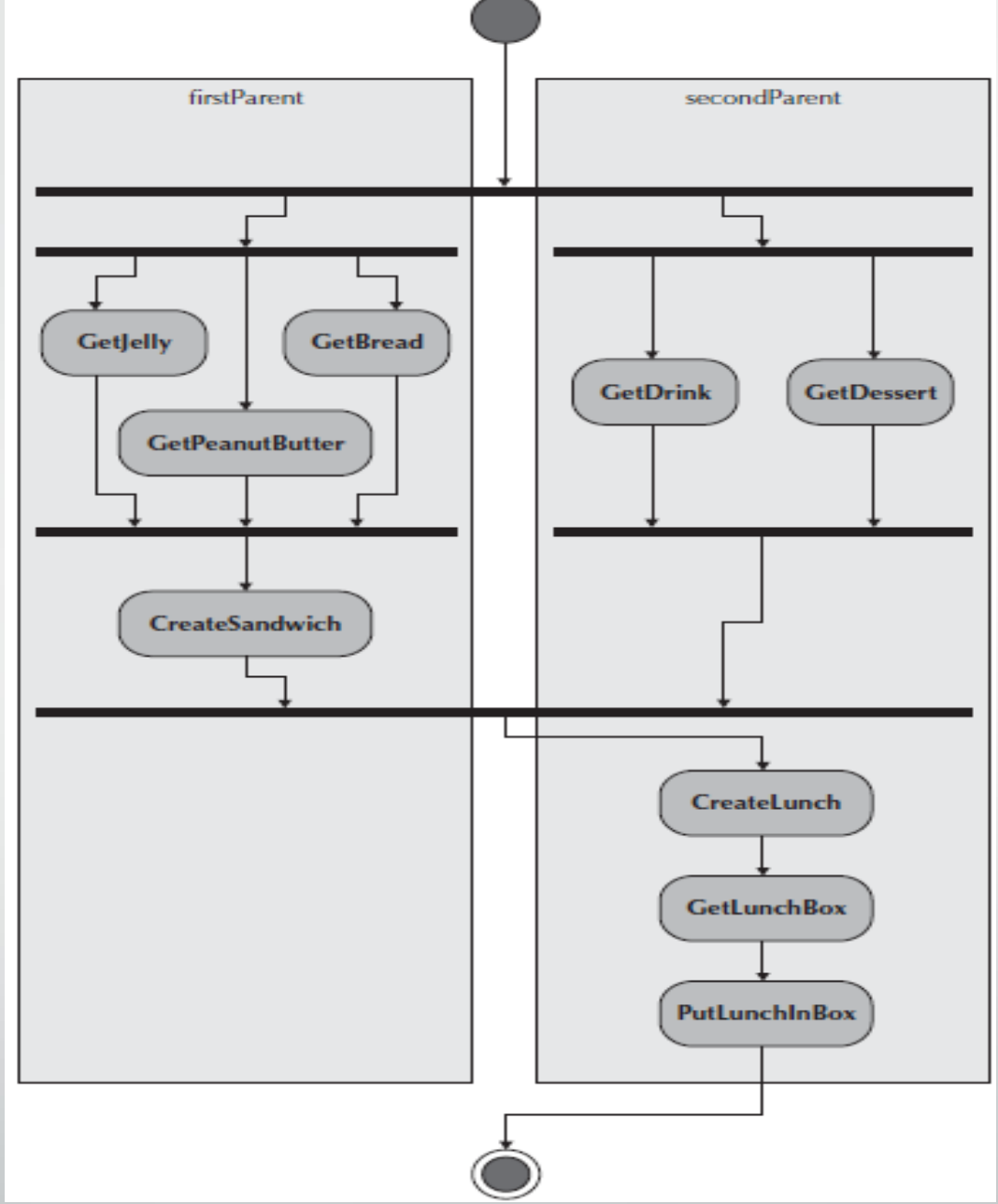


Fig 17. Activity diagram for making school lunch (Dennis et al., 2015)

Part 5

Object nodes

Object nodes

- These are special nodes that are used to show when instances of a class (objects) are available at a specific point during an activity.
- They are labeled with the name of the class to represent its instances or sub classes (where applicable).
- The input and output edges of these nodes are object flows.
- When an object node receives an object token on one of its input edges, it offers this token on all of its output edges simultaneously and these edges compete for the token. As there is only one token, the first edge to accept it gets it.
- Fig 18 shows an example of object flow for a product process activity.
- One last tool that can be used to tidy up the activity diagram is a pin. The reader is encouraged to do further reading on pins.
- A **pin** is an **object node** for inputs and outputs to **actions**. Pin is usually shown as a small rectangle attached to the action rectangle. The name of the pin can be displayed near the pin. See https://www.uml-diagrams.org/activity-diagrams-objects.html#:~:text=A%20pin%20is%20an%20object,from%20the%20Create%20Invoice%20action. For more details

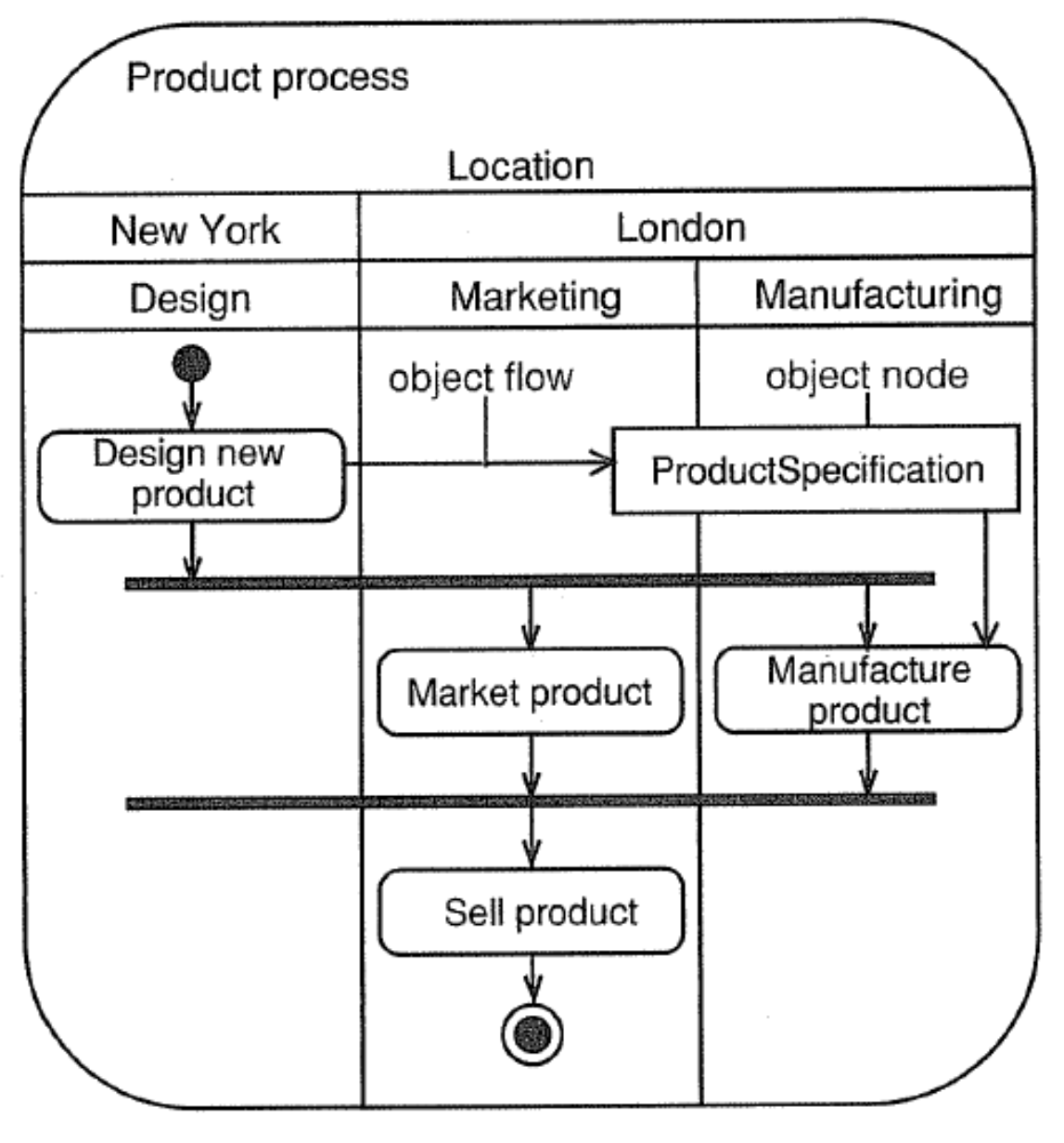


Fig 18. Object flow for product process. (Arlow and Neustadt, 2013)

Summary

- The activity diagram together with the state diagram represent the dynamic model. The dynamic model also supports other diagrams such as sequence diagrams.
- Activity diagrams allow modelling of a process as an activity that consist of a collection of nodes that are connected at their edges.
- Activity nodes are divided into three categories: action nodes, control nodes, and object nodes.
- The UML2 does not restrict how to create activity partitions (swimlanes) as long as they fulfill the desired objective; consequently, the designer can partition along use cases, classes, departments, or even roles.
- Action nodes represent some work being carried out and are shown as a rectangle with rounded corners.
- Control nodes manage the flow in an activity.
- Object nodes are special nodes that are used to show when instances of a class (objects) are available at a specific point during an activity.

References

- Action states.(n.d.). Retrieved November 3, 2022, from http://etutorials.org/Programming/Learning+uml/Part+III+Behavioral+Modeling/Chapter+8.+Activity+Diagrams/8.1+Action+States/
- "Composite State." *Sparx Systems*, https://sparxsystems.com/enterprise_architect_user_guide/15.2/model_domains/compositestate.html#:~:text=OMG%20UML%20Specification%3A&text=A%20composite%20state%20either%20contains,one%20of%20these%20two%20ways.
- Arlow, J., & Neustadt, I. (2013). *Uml 2 and the unified process: Practical object-oriented analysis and Design* (Second). Addison-Wesley.
- Dennis, A., Wixom, B. H., Tegarden, D. P., & Seeman, E. (2015). *System analysis & design: An object-oriented approach with Uml*. Wiley.
- Dennis, J.B. (2011). Petri Nets. In: Padua, D. (eds) Encyclopedia of Parallel Computing. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-09766-4_134
- Ojo, A., & Estevez, E. (2005). (rep.). *Object-Oriented Analysis and Design with UML - Training Course* (Vol. 1). In E-Macao Report 19.
- Stevens, P., & Pooley, R. (2006). *Using UML: Software Engineering with Objects and Components*. Pearson Educaion Limited.
- *What is activity diagaram*. What is activity diagram? (n.d.). Retrieved November 3, 2022, from https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/