Chapter 6

Activity diagrams

This chapter presents activity diagrams. An activity diagram is a kind of behavioural specification. It is quite suitable to specify workflows or complex algorithms.

Consider the Phone call use case described in Figure 3.2. To understand alternatives, you need to read them simultaneously with the main scenario. Missing alternatives are difficult to spot. The description of the main scenario enforces a sequential execution, missing potential for concurrent executions. These issues can be mitigated by using activity diagrams. We will use the Phone call use case as a running example when introducing the different concepts and elements of activity diagrams.

6.1 Activity diagram elements

6.1.1 Token semantics

The UML standard does not give semantics to all the different elements of an activity diagrams. A way to understand and give semantics to this class of diagrams is to consider that tokens are flowing from one element to another one. Some elements are generating tokens, some others are consuming and most of the elements will be passing tokens to the next element(s). An element is active when it has enough tokens. We will see that some elements require one token to be active and some others require more than one element.

6.1.2 Basic elements

Figure 6.1 shows an activity diagram representing the initial steps of the use case "Place a Call". This figure introduces basic constructs for activity diagrams:

- **Initial node**: this node indicates the start point of an activity diagram. This node is generating one token.

- **Final node**: this node marks the end of an activity. When a token reaches this node, all existing tokens are destroyed and the current activity terminates.
CHAPTER 6. ACTIVITY DIAGRAMS

Figure 6.1: Initial steps of Place a Call.

- **Decision node**: this node specifies a switch between two options. The input token is passed to the next element connected to the branch for which the decision is true.

- **Accept signal**: an accept signal represents an input to the activity. This element waits for some event to occur to pass the token to the next element.

- **Send signal**: a send signal represents an output of the activity. This element generates an event and then passes the token to the next element of the activity.

Coming back to Figure 6.1, this diagram reads as follows. At start, wait for the user to pick up the phone. If the line is free, provide the user with a dial tone, otherwise connect the user to the ongoing conversation and terminates the activity.

6.1.3 Activity final and flow final nodes

As seen in the previous section, the final node terminates the entire activity. It kills all active tokens. It is possible to specify that only one flow must be terminated. This is done using the FlowFinal node (see Figure 6.2).

Figure 6.3 illustrates this difference. The activity starts with "Build Component". Each built component is then installed. Note that the lower flow of the first fork terminates when no more components need to be built. This FlowFinal node only consumes its input token and does not affect the other active tokens. The same holds for the termination of the installation. When no more components need to be installed, the decision will pass a token to the activity "Deliver Application" that when it completes with pass a token to the final node that will terminate the entire activity.
6.1. **ACTIVITY DIAGRAM ELEMENTS**

![ActivityFinal and FlowFinal nodes](image)

**Figure 6.2**: ActivityFinal and FlowFinal nodes.

![Difference between ActivityFinal and FlowFinal nodes](image)

**Figure 6.3**: Difference between ActivityFinal and FlowFinal nodes.

6.1.4 **Time outs**

Figure 6.4 shows a time-out modelles using:

- **an interruptible region**: the dotted lines box is used to identify a region that can be interrupted.
- **an interrupting edge**: the edge identifies where to continue the activity in case of an interrupt.

Figure 6.4 reads as follows. If the event "user dials number B" happens within 5 seconds after starting the activity, forward call request to switch center otherwise send the event "emit a permanent tone" and terminate the activity.

**Internal activity** The square box with the text "Forward the call request to the switch center" is called an **internal activity**. In contrast to accept or send signals, this kind of element represents an action that has no interaction with the environment of the activity.

6.1.5 **Synchronisations**

Figure 6.5 shows the element "fork" and "join". These elements are used to synchronise activities. Their definitions are as follows:

- **Fork**: A fork consumes one token and generates as many tokens as its number of outputs.
CHAPTER 6. ACTIVITY DIAGRAMS

6.1.6 Merge and decision

Figure 6.6 illustrates a merge and then a decision. We already discussed decisions in Section 6.1.2. A merge is the dual of a decision. It works as follows:

- **two tokens at the input**: the merge will select one token and pass it along. It keeps the other token and will pass it along later;
- **one token at one input**: the merge passes the token along.

6.1.7 Place a call diagram

Figure 6.7 shows the complete activity diagram for the use case "Place a call". This diagram shows one of the shortcomings of activity diagram: these diagrams are getting complex!

6.2 Advanced diagrams

There are several techniques to keep activity diagrams compact. We will discuss two of them: the use of connectos and sub-activities.
6.2. ADVANCED DIAGRAMS

6.2.1 Connectors

As shown in Figure 6.8, connectors are a simple way to cut cross-links. Between any two connectors with the same name, there is a link.

6.2.2 Sub-activities

Figure 6.9 illustrates the use of sub-activities, which are means to give structure to activity diagrams. The idea is to create a reference from one element of an activity to another activity diagram. In Figure 6.9, the sub-activity "Start successful communication" is a reference to the activity diagram on the left. The semantics is to replace – in a kind of in-line way – the reference to the sub-activity with the activity diagram describing the sub-activity.

Figure 6.12 gives an overview of sub-activities:

- an activity contains itself actions, activities etc ...
- parameters: an activity can have input parameters of a specific type
- the stereotype "singleExecution" specifies that this sub-activity can only be present in one thread. Using a "concurrent" stereotype would allow the sub-activity to be started in parallel in multiple-threads.

6.2.3 Parameters

As illustrated in Figure 6.11, pins can be used to specify parameters to actions.
6.2.4 loops

Figure 6.13 illustrates the use of expansion regions to traverse sets of elements. Figure 6.13(a) specifies a sequential traversal. Each element of – say, a collection – are processed in a sequential ordering. Figure 6.13(b) adds the stereotype “concurrent” to specify that the traversal is performed in parallel. Finally, Figure 6.13(c) shows how to specify actions on some element. In this element, each paper is either rejected and no longer processed or it is accepted and passed to the next activity, for instance, inclusion the proceedings.

6.2.5 Responsibilities and Swimlanes

It is possible to specify responsibilities to activities. The most general solution is to use annotations as illustrated in Figure 6.14. Each activity is annotated with the entity responsible for the activity.

Another option is to use swimlanes which are limited to two-dimensions but are more visual. Figure 6.15 shows an example.

6.3 Summary of the possible elements

Figure 6.16 summarises the elements of activity diagrams.

6.4 Conclusion

This chapter presented “activity diagrams”. Activities are used to specify different steps in processes, workflows or algorithms. Relations such as ordering, synchronisation, choices between activities and sub-activities can be expressed using merge, deci-
Exercises

Exercise 6.5.1. Consider the activity diagram in Figure 6.17. Which one of the following executions (to completion) is not possible in this diagram?

1. A and B complete, then C completes twice.
2. A completes then C completes. B starts but is killed.
3. B completes then C completes. A starts but is killed.
4. A and B complete, C completes once.
CHAPTER 6. ACTIVITY DIAGRAMS

Figure 6.9: Sub-activity.

Figure 6.10: Sub-activity and parameters.
6.5. EXERCISES

Figure 6.11: Actions with parameters.

Figure 6.12: Sub-activity and parameters.
Figure 6.13: Loops and concurrency.

Figure 6.14: Annotations for responsibilities.
Figure 6.15: Swimlanes for responsibilities.

Figure 6.16: Summary activity diagram.
Figure 6.17: An activity diagram.