ICS 3105
OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter 5.6

Activity Diagrams

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Describe purpose of activity diagrams.
  - Draw activity diagrams given a narrative.
  - Convert state diagrams to activity diagrams.

Activity diagrams

- An activity diagram is like a state diagram, except that it has a few additional symbols and is used in a different context.
- In a state diagram, most transitions are caused by external events.

Activity diagrams

- However, in an activity diagram, most transitions are caused by internal events, such as the completion of an activity.
- An activity diagram is used to understand the flow of work that an object or component performs.

Activity diagrams

- It can also be used to visualize the interaction between different use cases.
- One of the strengths of activity diagrams is the representation of concurrent activities.
- Concurrency is shown using forks, joins and rendezvous, all three of which are represented as short lines, at which transitions can start and end.

Fork

- A fork has one incoming transition and multiple outgoing transitions.
- The result is that execution splits into two concurrent threads.

Join

- A join has multiple incoming transitions and one outgoing transition.
- The outgoing transition will be taken only when all incoming transitions have been triggered.
- The incoming transitions must be triggered in separate threads.

Rendezvous

- A rendezvous has multiple incoming and multiple outgoing transitions.
- Once all the incoming transitions are triggered, the system takes all the outgoing transitions, each in a separate thread.

Activity diagram nodes

- An activity diagram also has two types of nodes for branching within a single thread.
- These are represented as small diamonds:
  - A decision node.
  - A merge node.

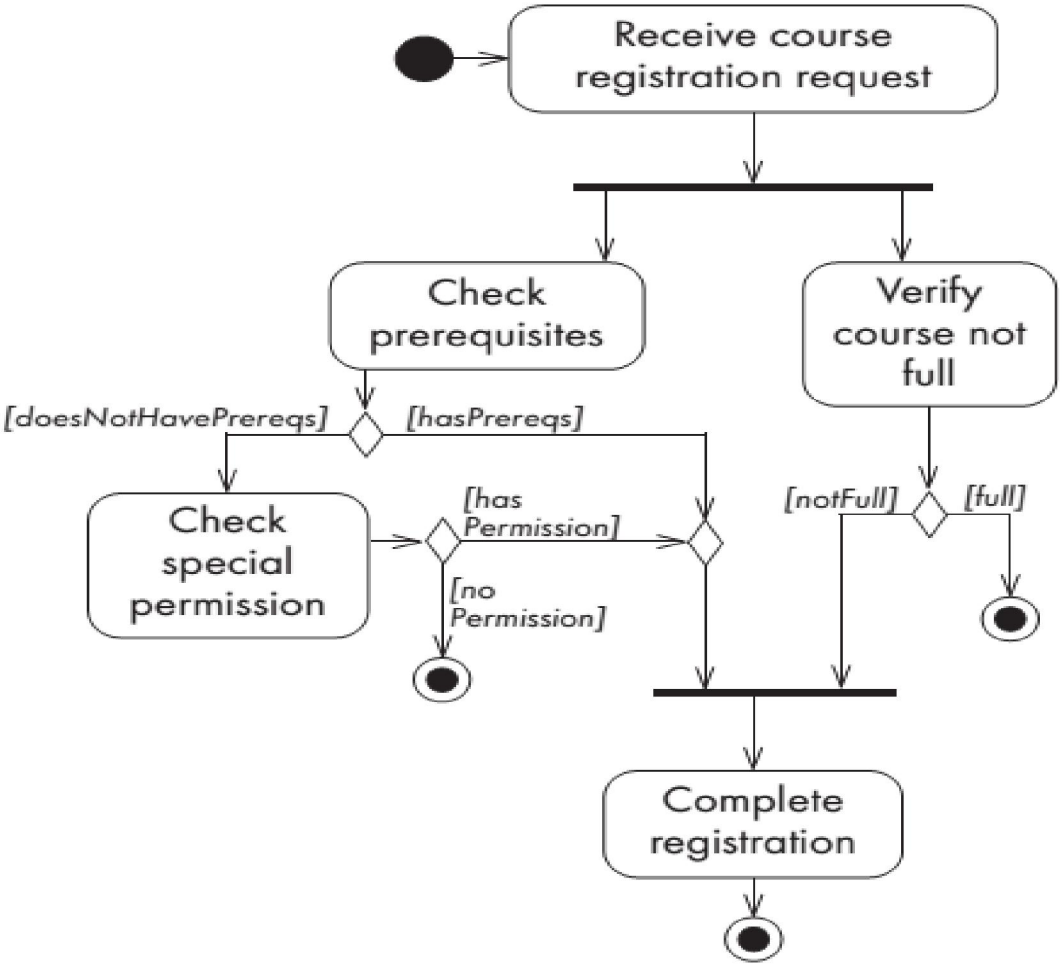
Decision node

- A decision node has one incoming transition and multiple outgoing transitions, each with a Boolean guard in square brackets.
- Exactly one of the outgoing transitions will be taken.

Merge node

- A merge node has two incoming transitions and one outgoing transition.
- It is used to bring together paths that had been split by decision nodes.

Activity diagram for registering in a course section process



Process of registering in a course section

- The first thing that occurs is the reception of the registration request.
- Once this processing is complete, the system immediately takes the outgoing transition and forks the processing into two concurrent threads.

Process of registering in a course section

- The concurrent thread on the right is responsible for checking whether a course is full or not.
- The outcome of its computations is either that the course is not full, in which case execution proceeds to the join, or else the course is full, in which case the whole activity diagram terminates.

Process of registering in a course section

- The concurrent thread on the left, meanwhile, checks whether the student is allowed to register.
- First it checks if the student has the prerequisites for the course; if this check is affirmative, the thread proceeds to the join via a merge node.

Process of registering in a course section

- Otherwise a second check is performed, to see if the student has special permission that can override the lack of prerequisites.
- If this second check is affirmative, the thread proceeds to the join via the merge node, otherwise registration is disallowed and the whole activity diagram terminates.

Process of registering in a course section

- Either concurrent thread may reach the join first, at which time it will wait for the other.
- It may be the case that neither thread, or only one thread, ever reaches the join, due to the course being full or registration being disallowed.

Process of registering in a course section

- However, if both threads reach the join, then they are replaced by a single thread that performs the final activity – completing the registration.

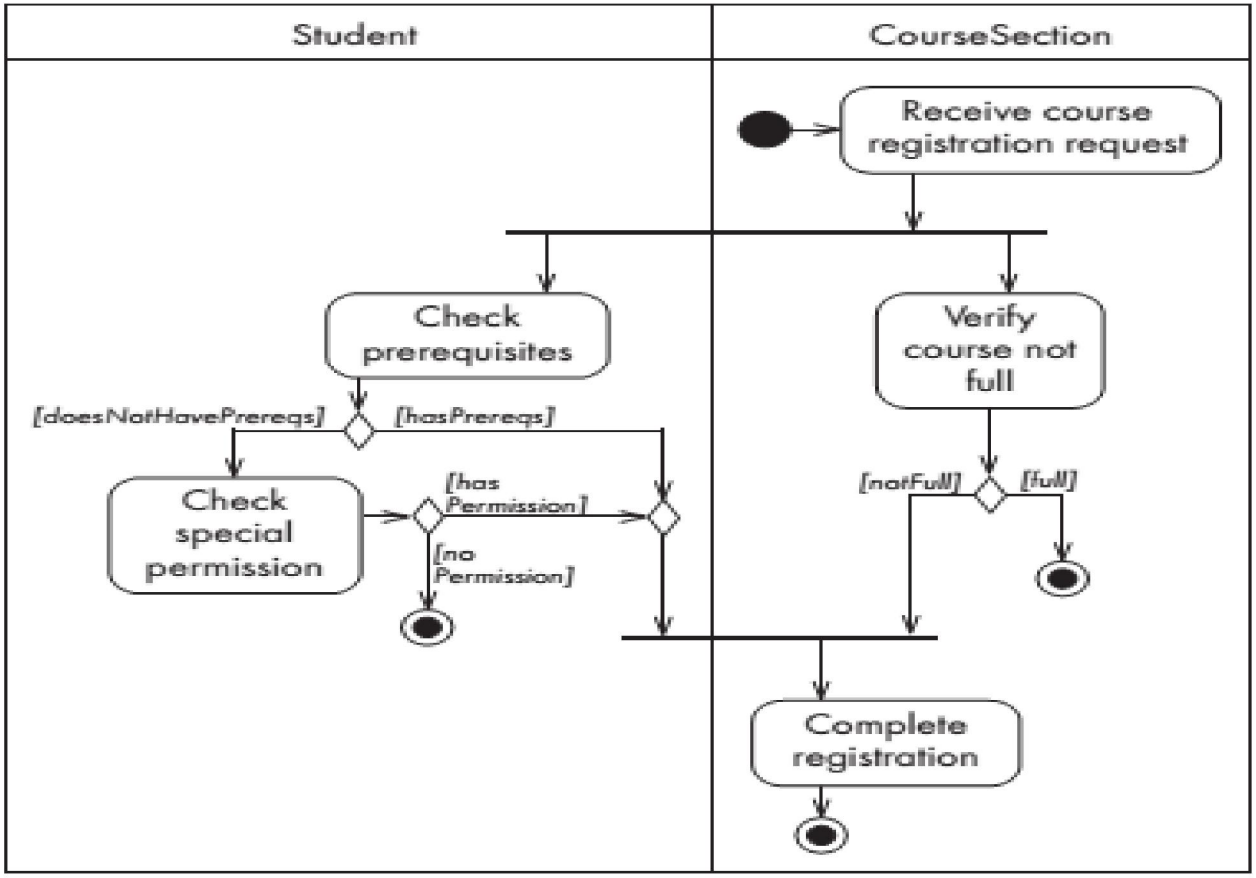
Process of registering in a course section

- The entire process comes to an end as soon as any of the end states is reached.
- This implies that even if one thread is waiting at the join, it will be terminated if the other thread reaches an end state.

Activity diagrams Vs state diagrams

- While state diagrams typically show states and events concerning only one class, activity diagrams are most often associated with several classes.
- The partition of activities among the existing classes can be explicitly shown in an activity diagram by the introduction of swimlanes.

Activity diagram with swimlanes



Activity diagrams Vs state diagrams

- Swimlanes are boxes that form columns, each containing activities associated with one or more classes.

Implementing classes based on interaction and state diagrams

- Drawing sequence, communication, state diagrams for every class, interaction and activity would take too much time.
- However, you should certainly use these diagrams for the parts of your system that you find most complex.

Implementing classes based on interaction and state diagrams

- For example, a state diagram is useful when different conditions will cause the instances of a class to respond differently to the same event.
- This is particularly true when behavior is distributed across several use cases.

End of chapter 5.6