Section 1: UML

William Findlay

January 16, 2019

Contents

1 UML Overview

- **unified modeling language**
- what is it?
  - ➢ a tool for **expressing system models**
    - functional
    - dynamic
    - object

1.1 The UML Family

- each notation is for a **specific model**
- models and notations
  - ➢ functional
    - use case diagrams
  - ➢ dynamic
    - state machine diagrams
    - sequence diagrams
    - activity diagrams
  - ➢ object
    - class diagrams

2 Use Case Diagrams

- what is a use case?
  - ➢ behavior observed by **external entities**
  - ➢ entities called **actors**
    - end users
      - ▷ different roles
    - external systems
      - ▷ systems that our system will **interact with**
  - ➢ can also be represented *textually*
    - table-based
- what are use case diagrams?
  - ➢ graphical representation of use cases
- purpose
  - ➢ system boundaries
  - ➢ **always use a box in the drawing**

2.1 Some Rules

- the box is important
- ovals for use cases
  - ➢ use cases are labeled with verb phrases
- actors
  - ➢ draw as stick figures
  - ➢ an actor is a **role**
  - ➢ not necessarily a person
  - ➢ a person can have more than one role
- in our project
  - ➢ SQL and Qt are **not external roles**
  - ➢ they are part of the system

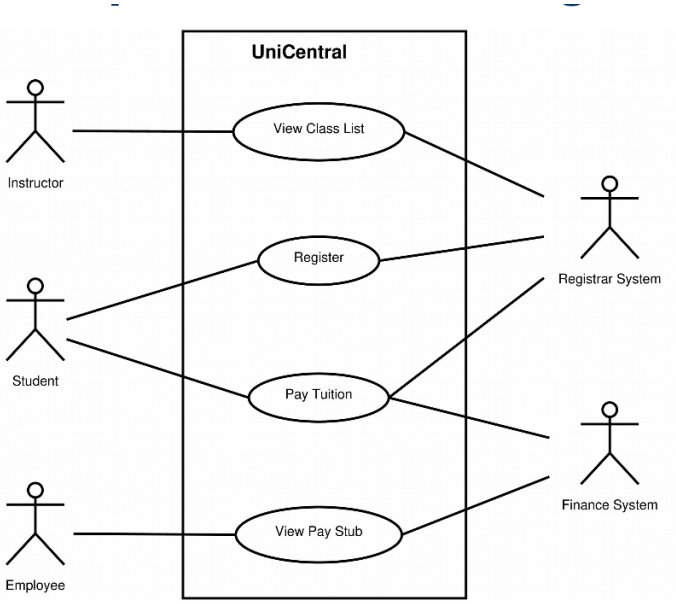

Figure 1: An example of a use case diagram. The stick figures are actors. The bubbles inside the box are use cases. A use case is always labeled with a verb phrase.

3 Class Diagrams

- graphical representation of classes and **objects**
- purpose
  - ➢ describe a system
  - ➢ in terms of **classes**
  - ➢ include
    - attributes
    - operations
    - associations

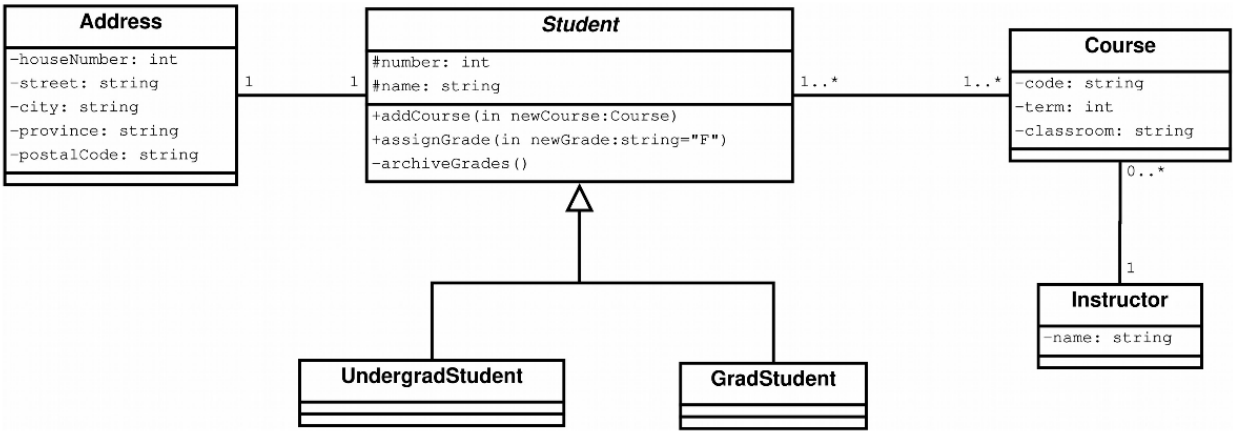

Figure 2: An example of a class diagram. Each class is represented by a box with a name, attributes, and operations and is connected to other classes by associations.

3.1 Some Rules

- three sections
  - ➢ class name
  - ➢ attributes
  - ➢ operations
- attributes
  - ➢ access specifier
    - ■ `+` public
    - ■ `#` protected
    - ■ `-` private
  - ➢ name
  - ➢ `:` followed by data type
- operations
  - ➢ access specifier
    - ■ `+` public
    - ■ `#` protected
    - ■ `-` private
  - ➢ name
  - ➢ parameters
    - ■ input
    - ■ output
    - ■ input-output
- associations
  - ➢ direction
    - ■ directed
    - ■ undirected
  - ➢ types
    - ■ inheritance
      - ▷ aggregation
    - ■ composition
  - ➢ cardinality
    - ■ none-to-many `0..*`
    - ■ one-to-many `1..*`
    - ■ etc.

3.2 Object Diagrams

- underlined $\implies$ specific instance
  - ➢ also include an instance name before a `:` in front of class
  - ➢ sometimes just a `:` if instance name is implied

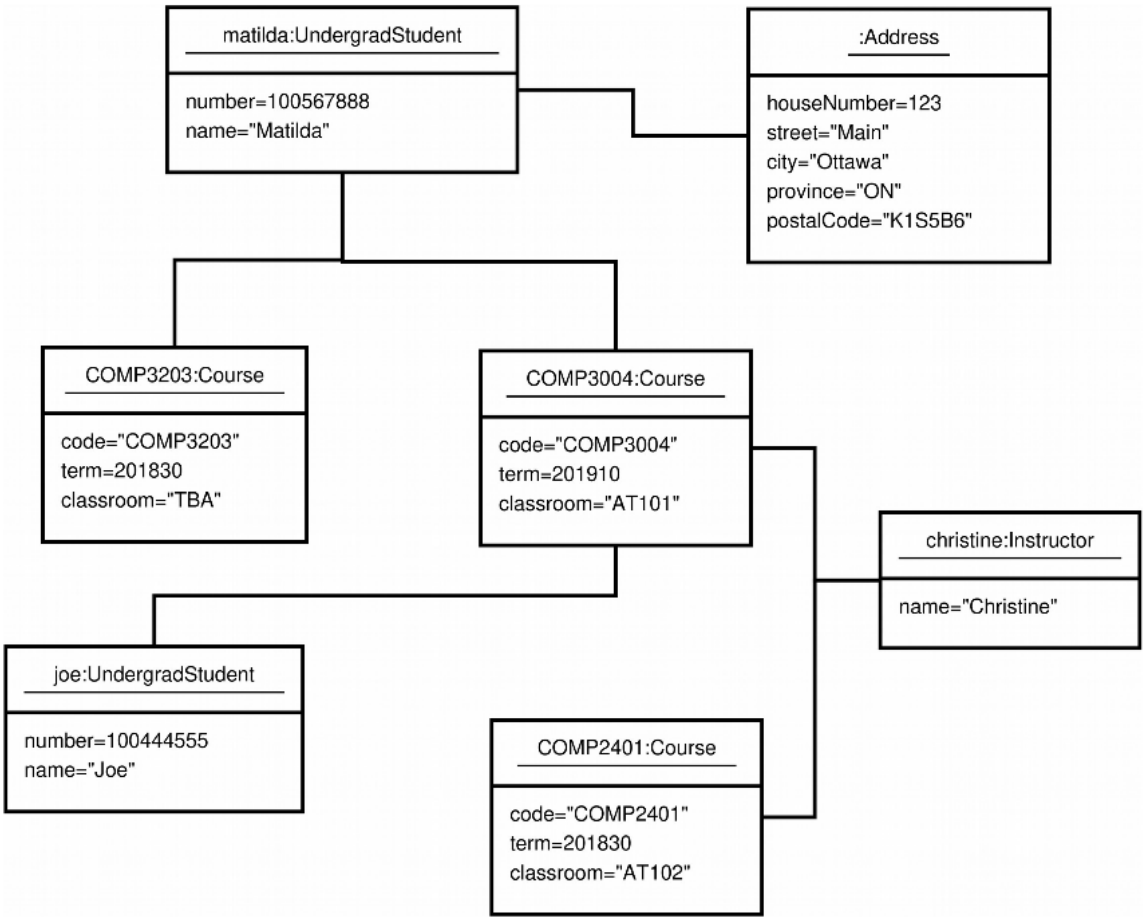

Figure 3: An example of an object diagram. Note that the object name is not always specified if it is obvious.

4 State Machine Diagrams

- graphical representation of the **state** of a **single objects**
  - ➢ only more complicated ones
  - ➢ some may not have any states
- purpose
  - ➢ set of states
  - ➢ transitions from one state to another
  - ➢ state:
    - ■ attribute values for an object
  - ➢ transition:
    - ■ conditions under which an object changes state

4.1 How it Looks

- states in bubbles
- arrows (transitions)
  - ➢ labels are mandatory
    - ■ except labels **from start** or **to end**

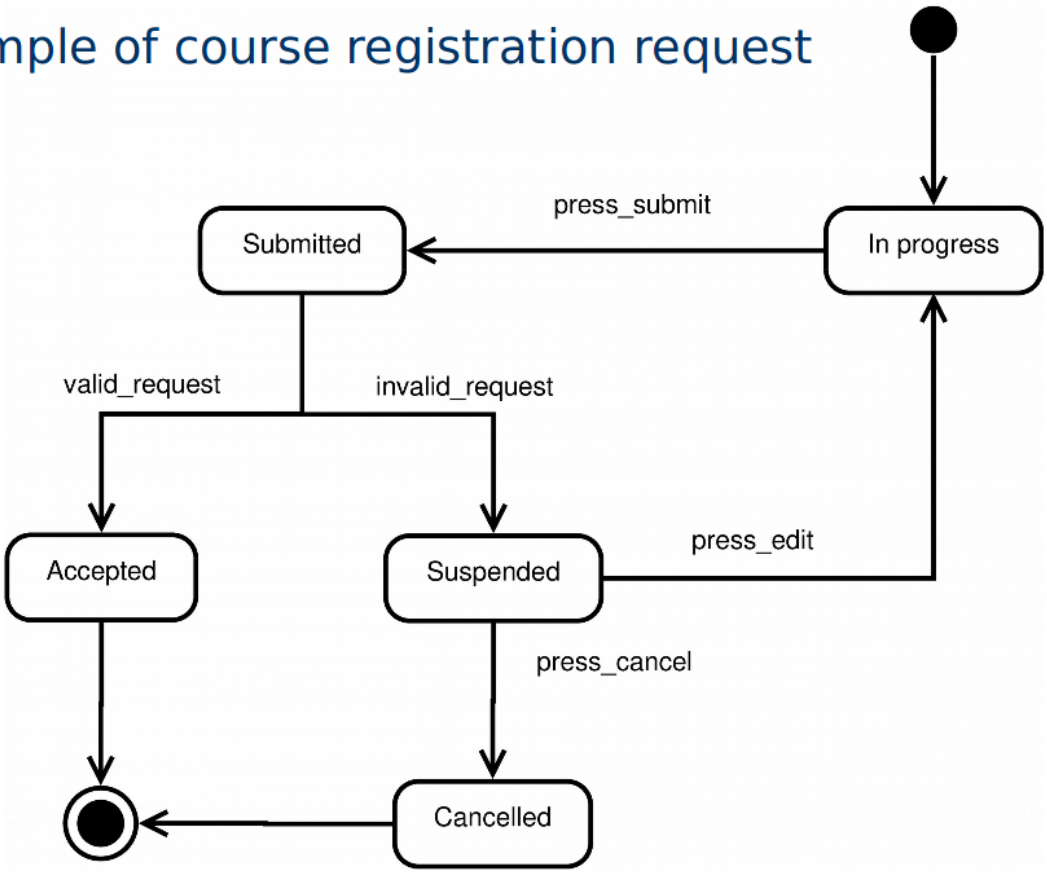

Figure 4: An example of a state machine diagram for a course registration request. The bubbles are the states and the labeled arrows are the transitions. Also note that a state can have one or more transitions to itself.

- ➢ labels are the transitions
- ➢ you can have arrows from a state to itself

5 Activity Diagrams

- we won't use these a lot
- what are they?
  - ➢ system behavior
    - sequencing
    - coordination
- purpose
  - ➢ describe sequential steps in processing
    - control flow
    - concurrency

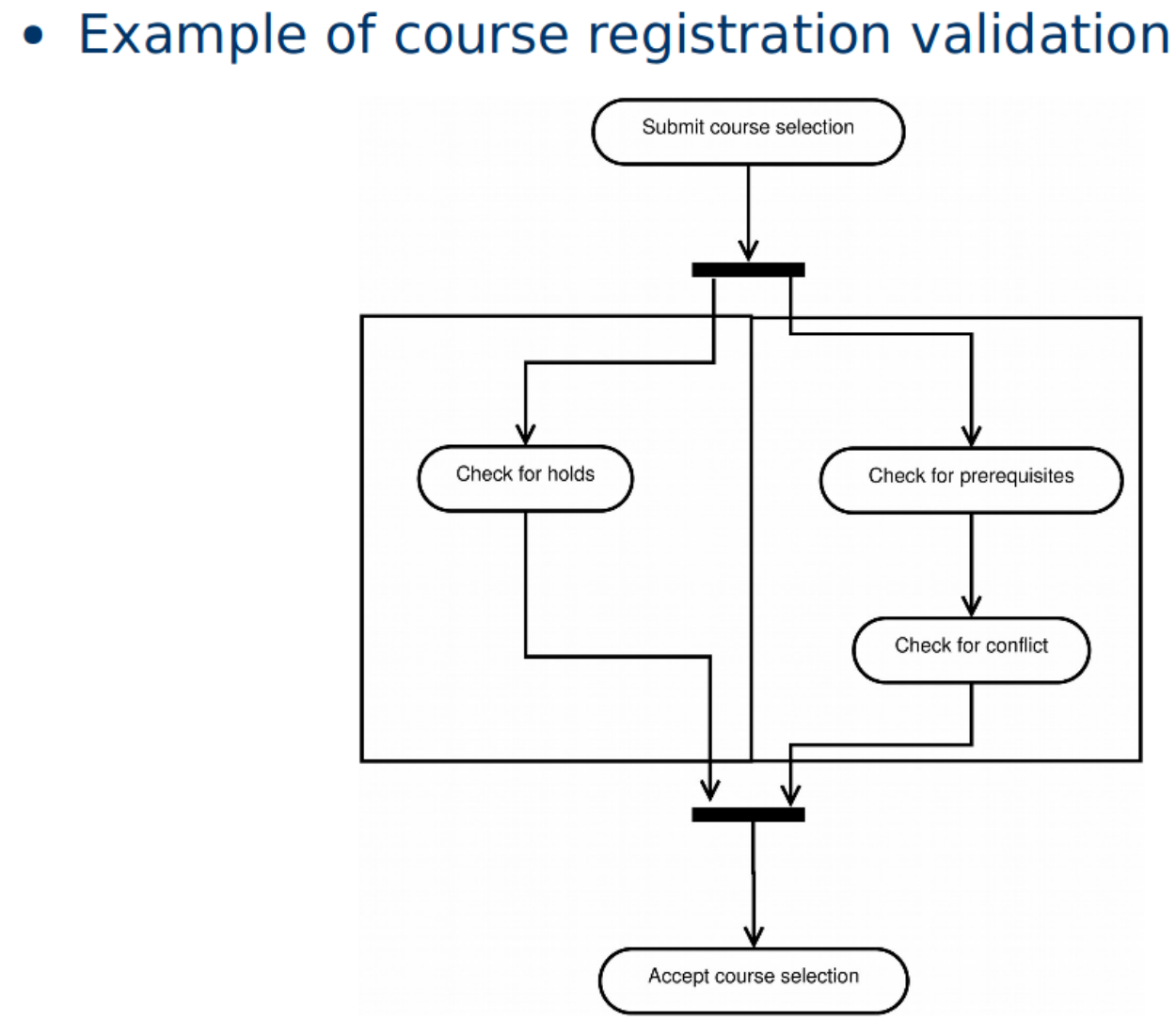

Figure 5: An example of an activity diagram. The two halves of the sqare are called "swim lanes". You can also have one swim lane or more than two swim lanes.

6 Sequence Diagrams

7 Packages