Modelling Dynamic Behaviour - From Use Cases to Classes

MechEng SE3

Simon Gay

24 February & 3 March 2010

Reminder: Assignment

Remember to hand in the assignment on Friday.

Announcement

On Friday 26th February we will have a guest lecture
by Dr Tim Storer, Computing Science Department.

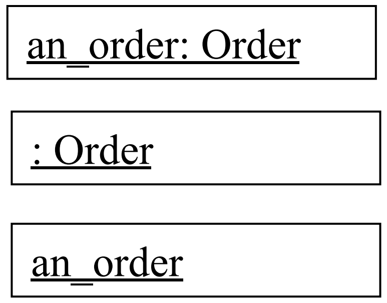
On Wednesday 17th March we will have a guest lecture
by Iain McGinniss, Computing Science Department (formerly
of Sword Ciboodle).

Modelling Dynamic Behaviour

- The dynamic behaviour of the system is what happens when the system is running.
- We have already seen one way of modelling this: activity diagrams.
- The dynamic behaviour involves objects rather than classes.
- This is not really highlighted by the activity diagrams, and so the activities also need to be expressed in different ways that involve objects.
  - » ***Sequence diagrams***, which concentrate on the time sequencing of operations.
  - » ***Communication diagrams***, which group all of the methods associated with an object in one place. (Called ***Collaboration diagrams*** *in UML 1.x)*
- These two types of diagram are collectively called ***Interaction diagrams***.

Object Notation in UML

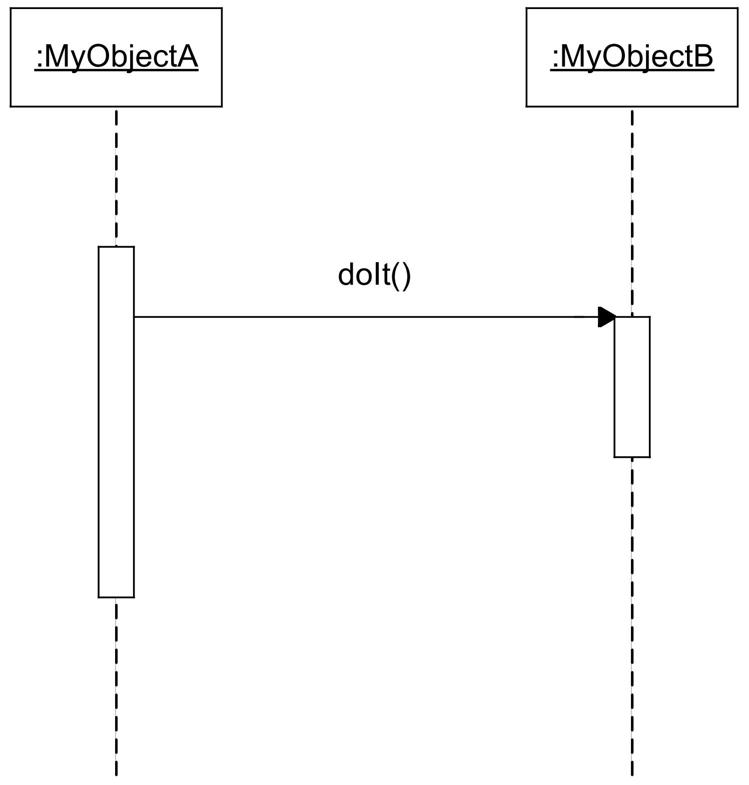
- An object is represented by a rectangle containing the object name, a colon and the class name, all underlined.
- The object name can be omitted, in which case the colon must be present, to show that the word is a class name.
- The class name can be omitted, in which case the colon is not needed.



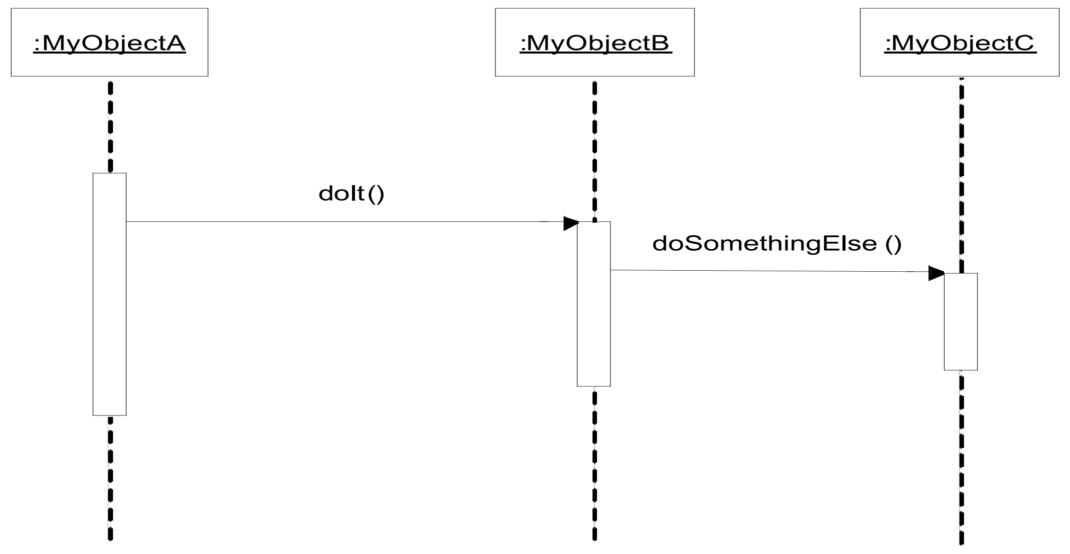
Sequence Diagrams

- Objects are shown at the top of dashed vertical lines.
  - The vertical line is called the object's lifeline.
  - Time sequencing moves downwards.
- Messages are sent from one object to another, and are represented by solid horizontal arrows.
  - They correspond to method calls.
  - The ***active object*** is the one at the blunt end of the arrow. Control starts in this object and is transferred to the passive object.
  - The ***passive object*** is at the sharp end of the arrow.
  - The method is a method of the passive object's class.

One object sends a message to another



An object sends a message in an activation



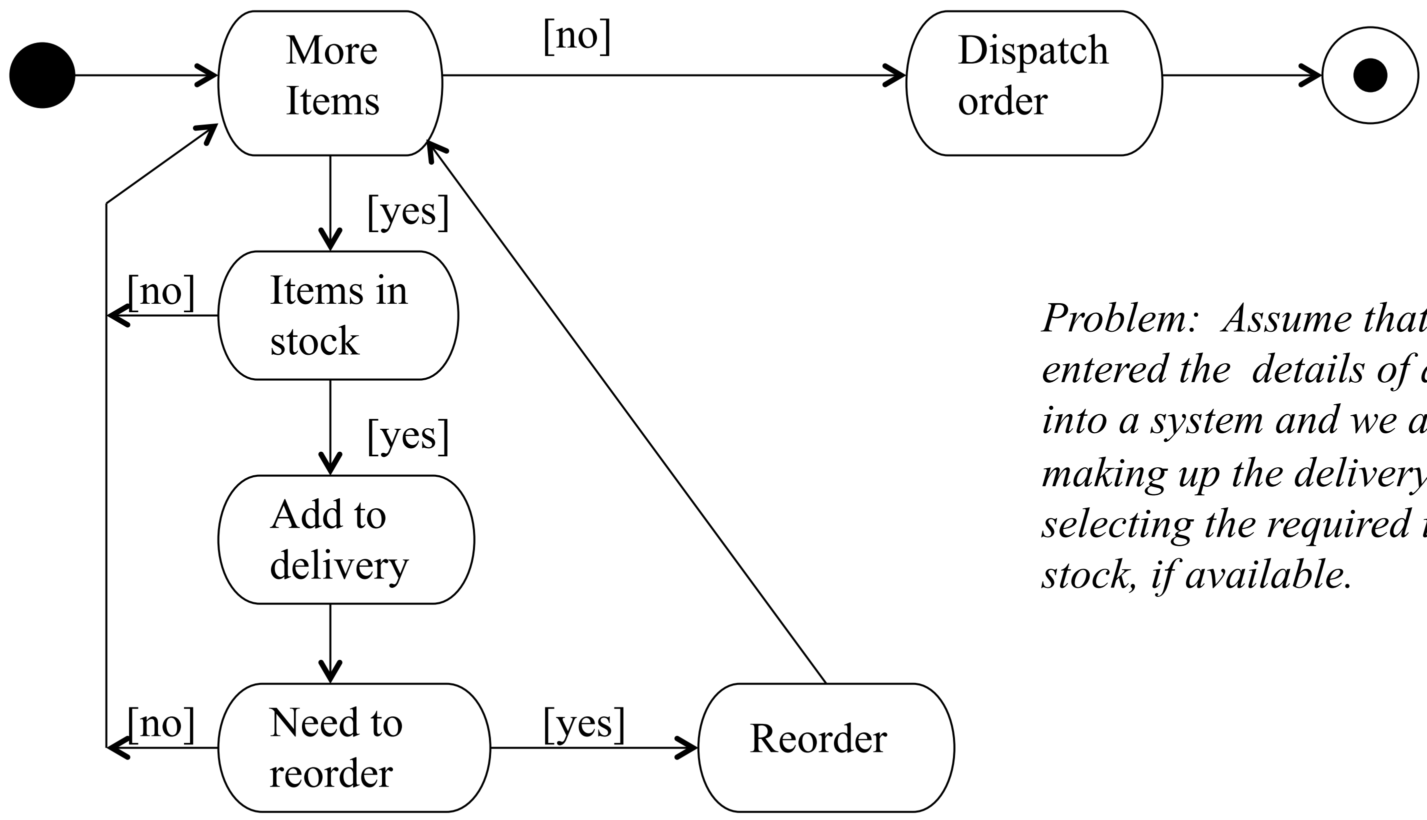
Sequence Diagrams (2)

- Arrows represent flow of control, not flow of information.
  - Information can flow into the method via parameters and back via reference parameters and return values even though the arrow goes in one direction.
- Method activations are shown as rectangles on the lifeline.
  - The code associated with the method is being processed at this time.
- Object creation is done with a method called `new`.
  - The new object appears at the corresponding place in the diagram, complete with its own lifeline.
- Returns from the method can be shown but are usually omitted, since they just clutter the diagram.

Sequence Diagrams (3)

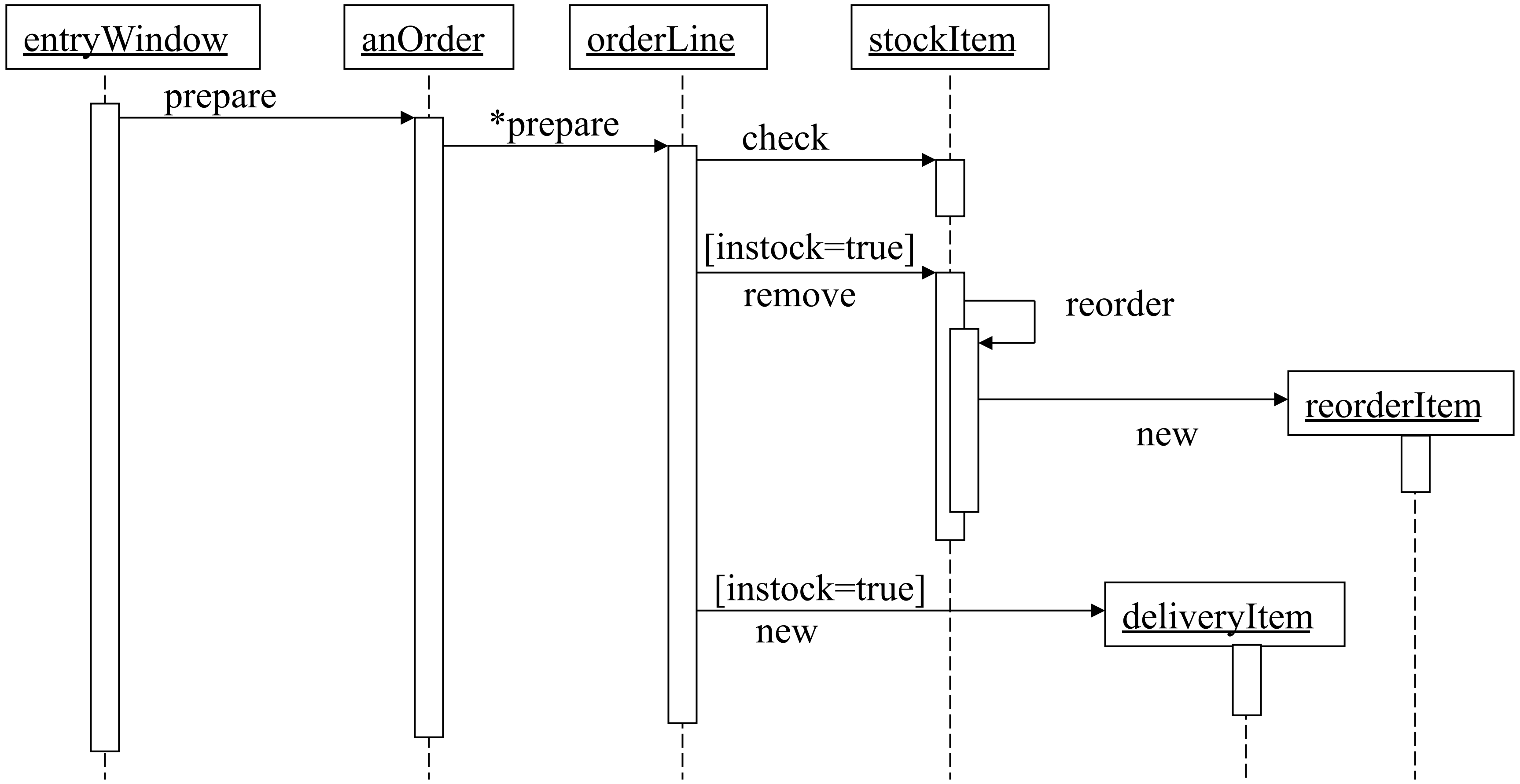
- The object at the extreme left starts off the processing
  - It initiates the activity that the sequence diagram is describing.
- An object can call one of its own methods (self delegation).
  - The new method activation is overlaid on the original.
- Object deletion is indicated by a large cross at the end of the lifeline.
- A conditional method call is shown using a guard on the top of the arrow.
- Repetition is shown by an asterisk in front of the method name.

An Example: Activity Diagram



Problem: Assume that we have entered the details of an order into a system and we are now making up the delivery by selecting the required items from stock, if available.

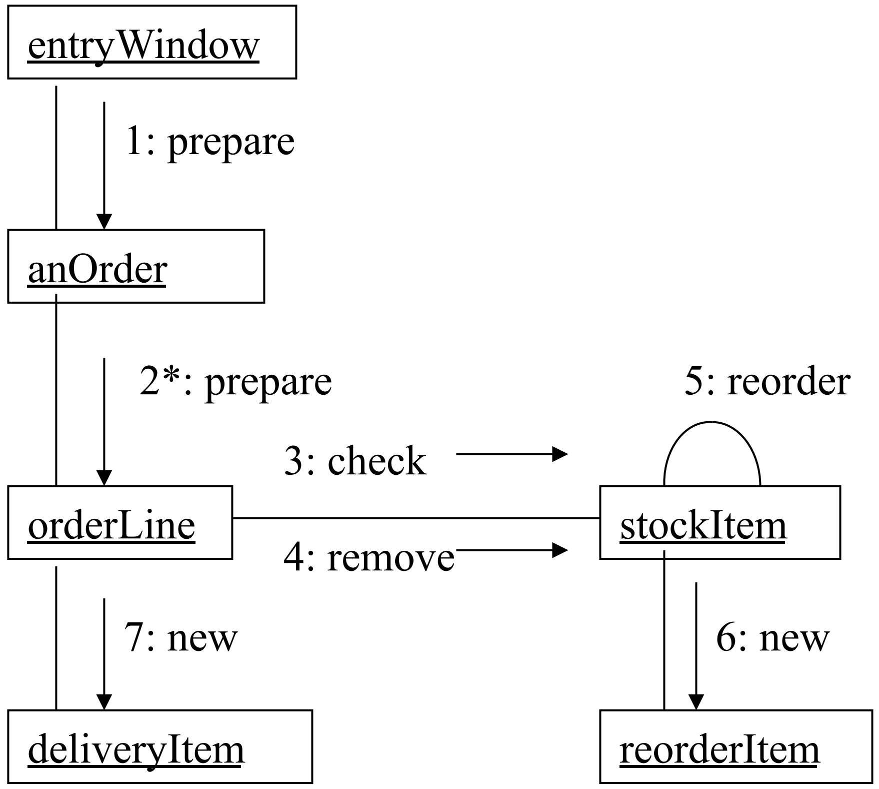
An Example: Sequence Diagram



Communication Diagrams

- These group the messages together with the objects by doing away with the lifelines.
- It is easier to see all the methods that belong to an object.
- It is harder to see the time sequencing information.
  - A sequence number is placed in front of the method.

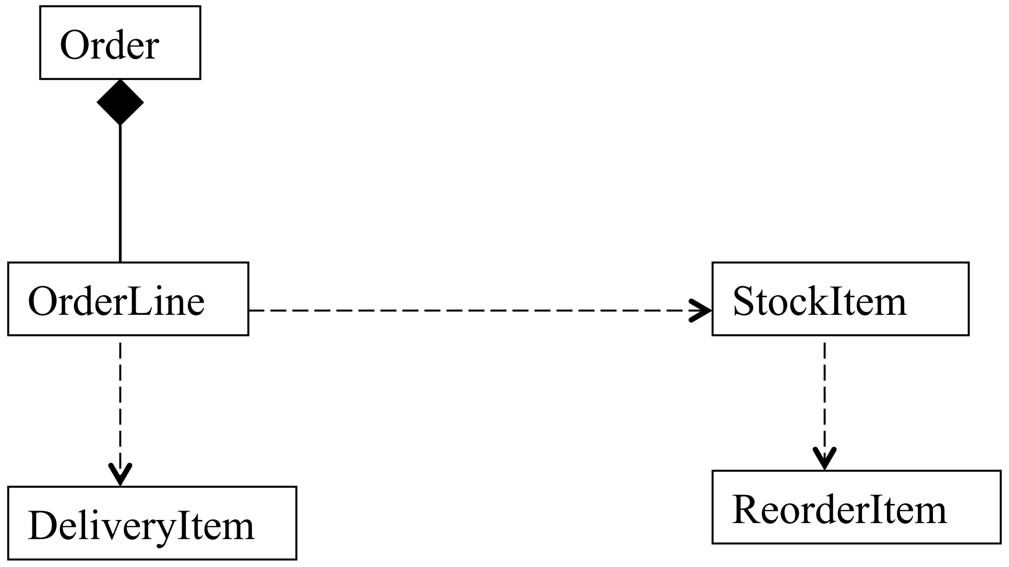
An Example: Communication Diagram



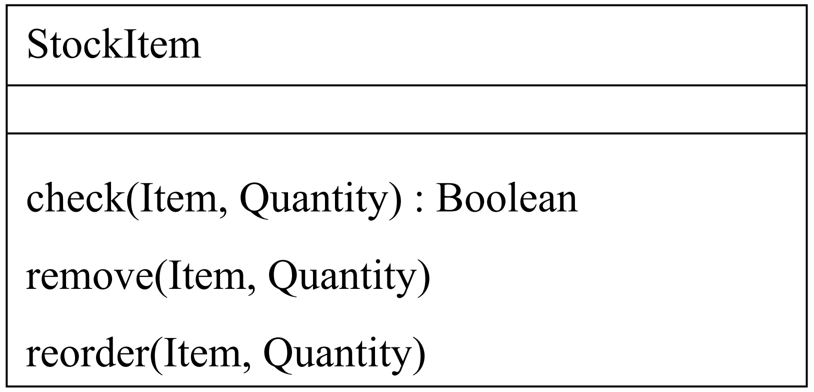
Relationship with Other Diagrams

- Methods
  - Both sequence and communication diagrams will generate methods.
  - These methods will have a corresponding entry in the individual class diagram.
  - We do not usually record method parameters and return values in interaction diagrams.
  - This information is added in the individual class diagram.
- Class Interactions
  - If an object calls a method of another object, then the class corresponding to the first object must know about the class corresponding to the second object.
  - A dependency, part of or inheritance relationship.

Class Structure Diagram



One Detailed Class



From Use Cases to Classes

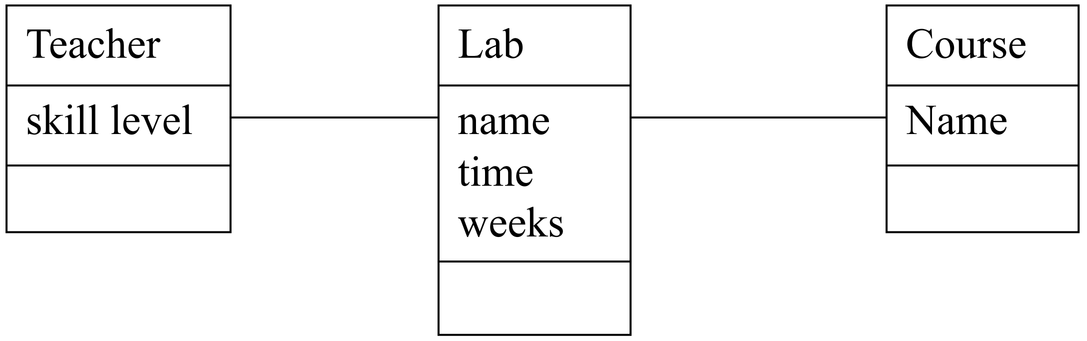
1. Start with the use case details
   - Numbered list or
   - Activity diagram
2. Identify classes and associations
   - Conceptual class diagram
3. Convert use case details to a sequence diagram
   - Objects and methods
4. Convert methods on sequence diagrams to methods on class diagrams
   - Interface class diagram
5. Convert associations to directed dependencies

Request Teachers Flow of Events

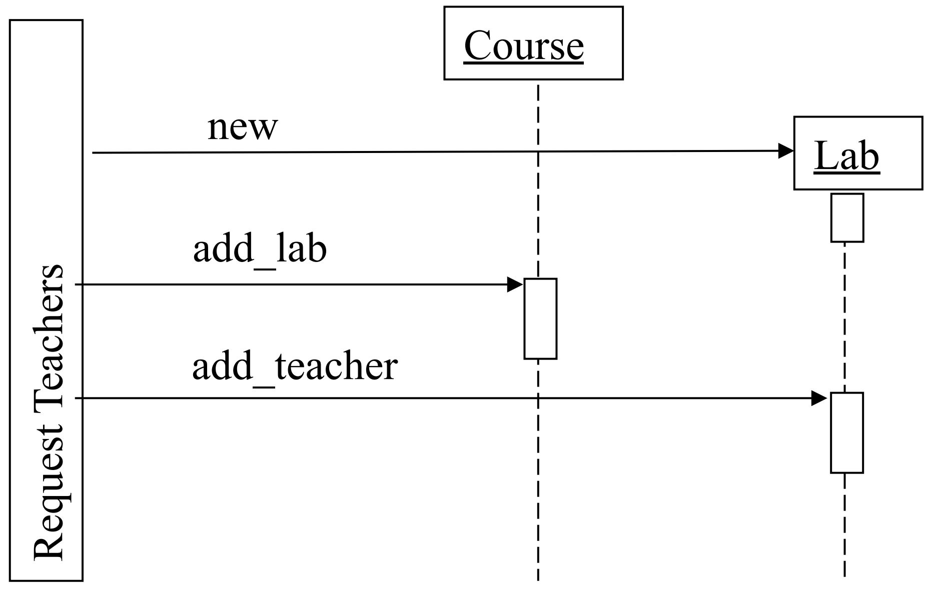
1. For each lab associated with the course, enter course and lab name, day and time, and weeks during which it will run.
2. Enter skills level required for each part-time teacher. A lab may be staffed by more than one person. Skills levels are tutor, graduate demonstrator, undergraduate demonstrator.
3. Enter suggested teachers, to help the recruiter.
4. Notify PTT Director.

Example: PTT Courses

- Initial conceptual class diagram for Request Teachers.

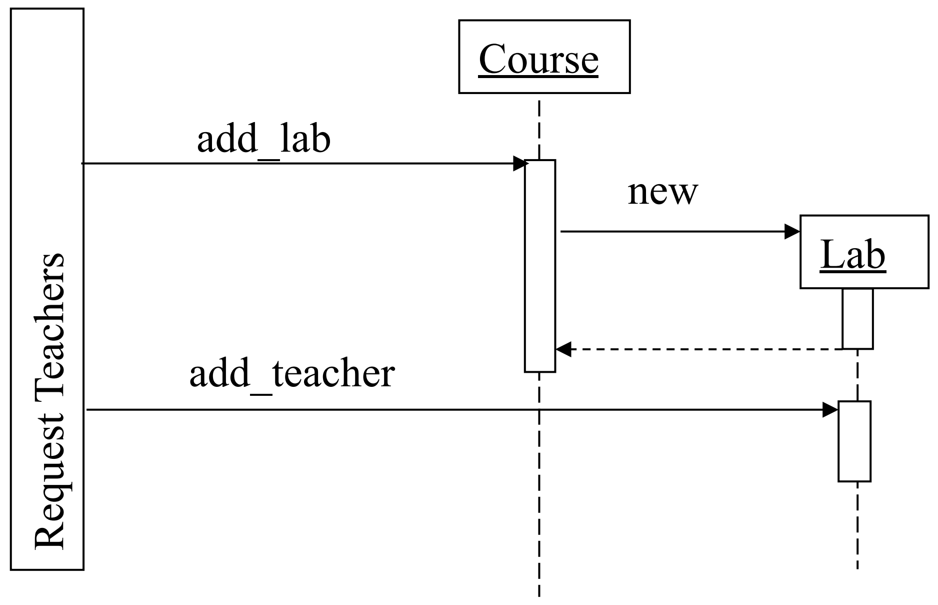


Sequence Diagram 1



Sequence Diagram 2

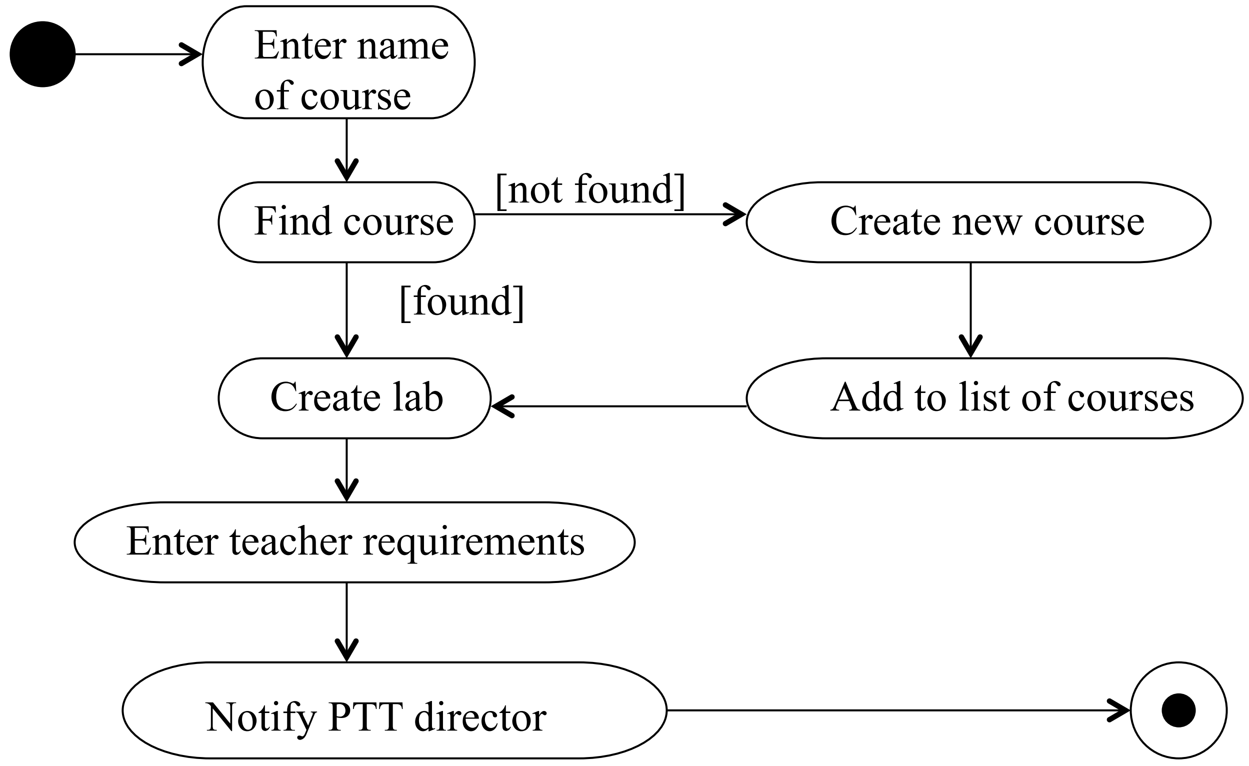
- Looking at this diagram raises a question. We have used two steps to create the lab where one step would be best.
- We can modify the diagram to use a one step creation method.



Activity Diagram

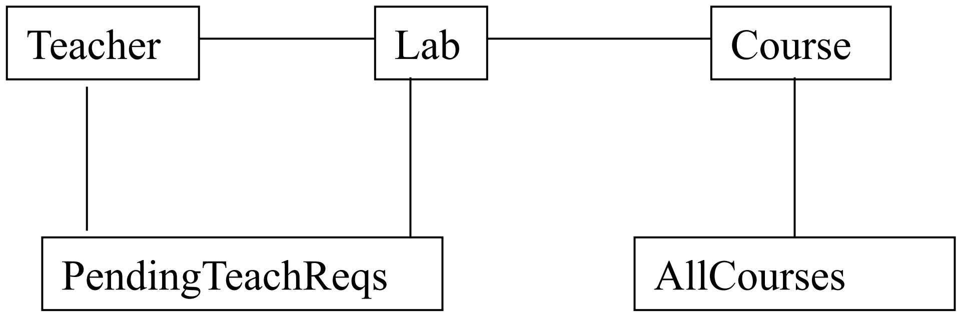
- The sequence diagram raises the question of where the course objects comes from.
  - We will create a new object and class called `AllCourses` which stores all the current courses.
  - We also need to amend the use case details: flow of events, which are now complex enough for an activity diagram to be useful.

Activity Diagram (2)

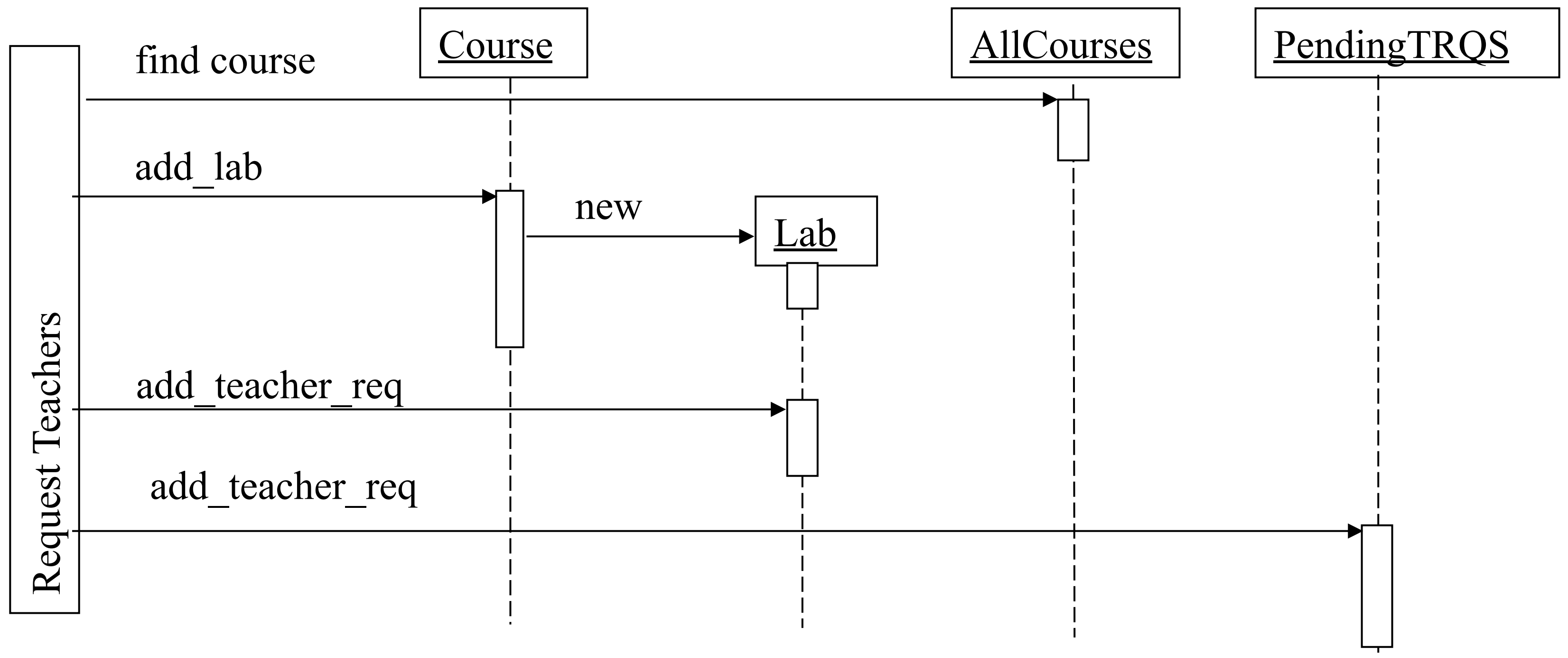


Conceptual Class Diagram 2

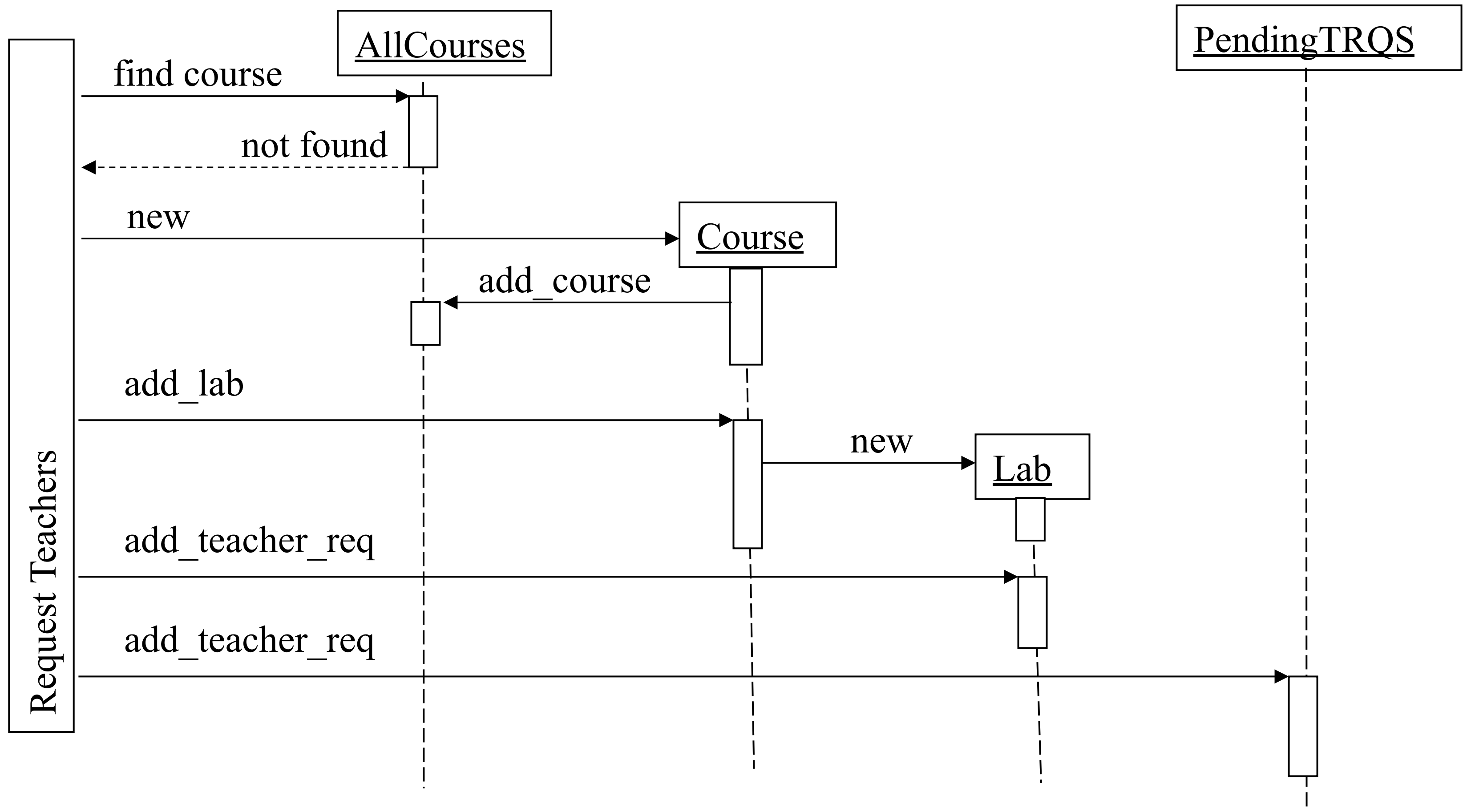
- We have another question, how do we record the suggested teachers.
  - Use a `PendingTeachReqs` object.
  - It is about time we amended our class diagram.



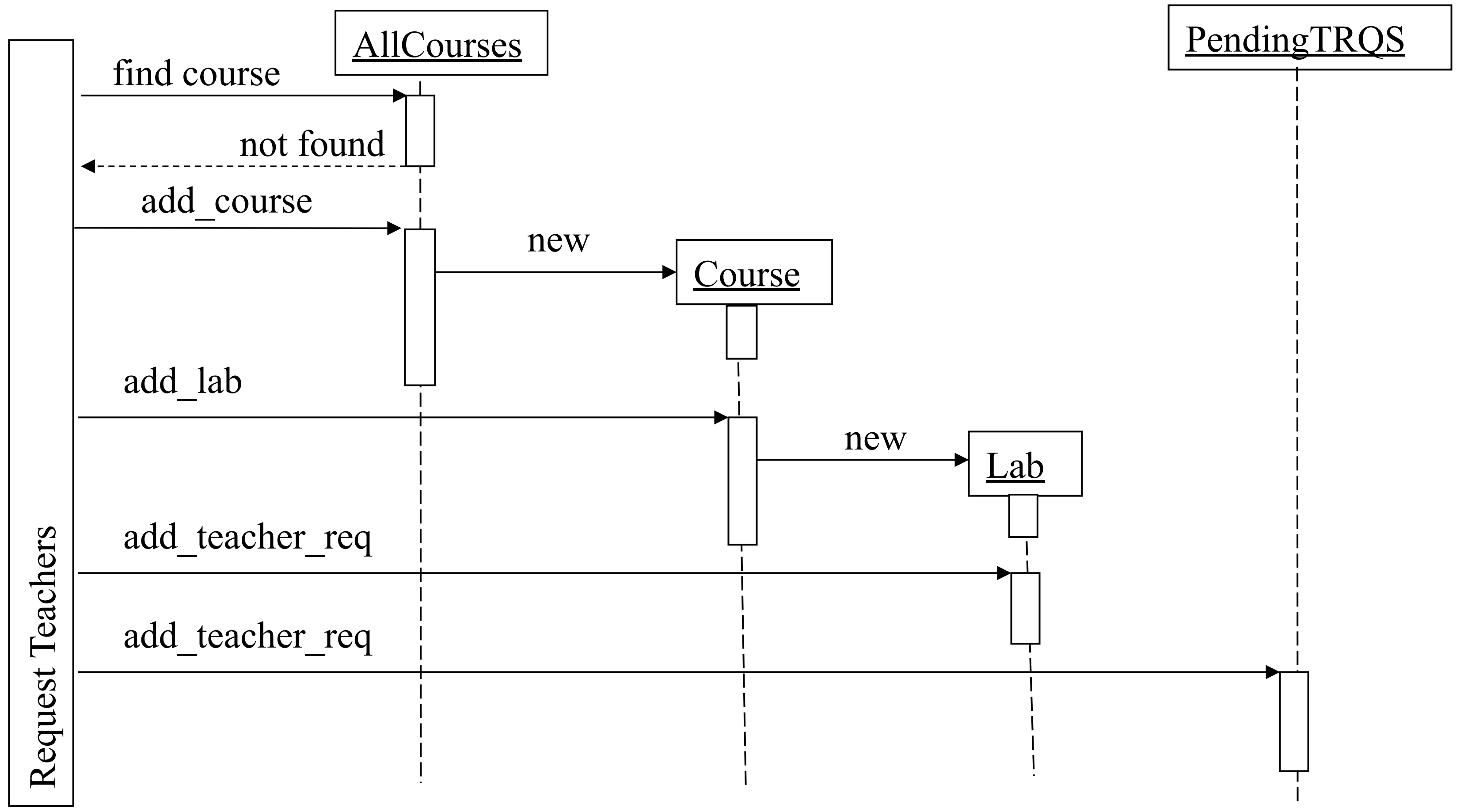
Sequence Diagram 3



Sequence Diagram 4



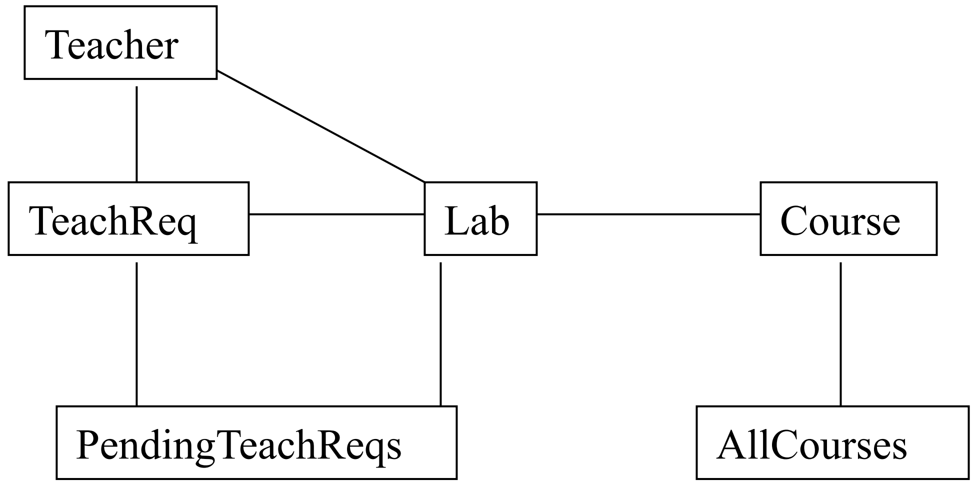
Sequence Diagram 4A



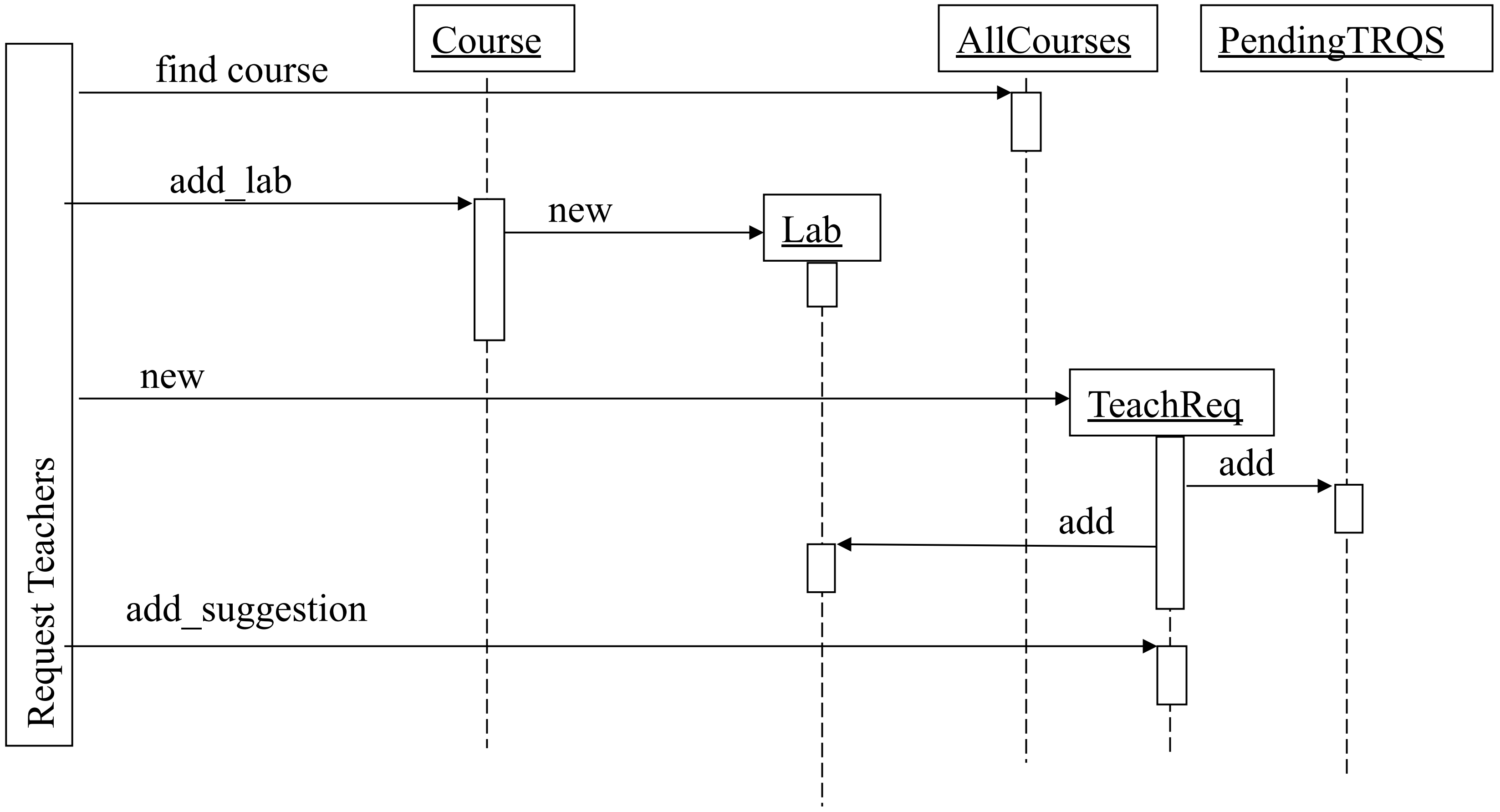
More Refinements

- There will be a similar diagram for the separate case when a new course is created (shown last slide).
- We notice that we add a teaching request in two stages. We would like to do this in one operation.
- The `add_teacher_request` method will have three parameters, the skill level, the lab and the teacher itself.
- This suggests we need a new class of objects, that of teacher request.
- This solves the atomic operation problem since all of this work will be achieved when we create a `TeachReq` object.

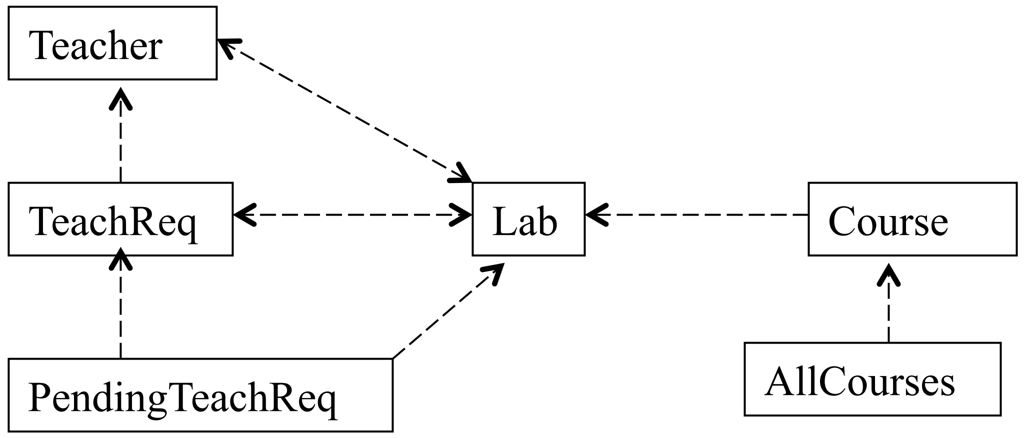
Conceptual Class Diagram 3



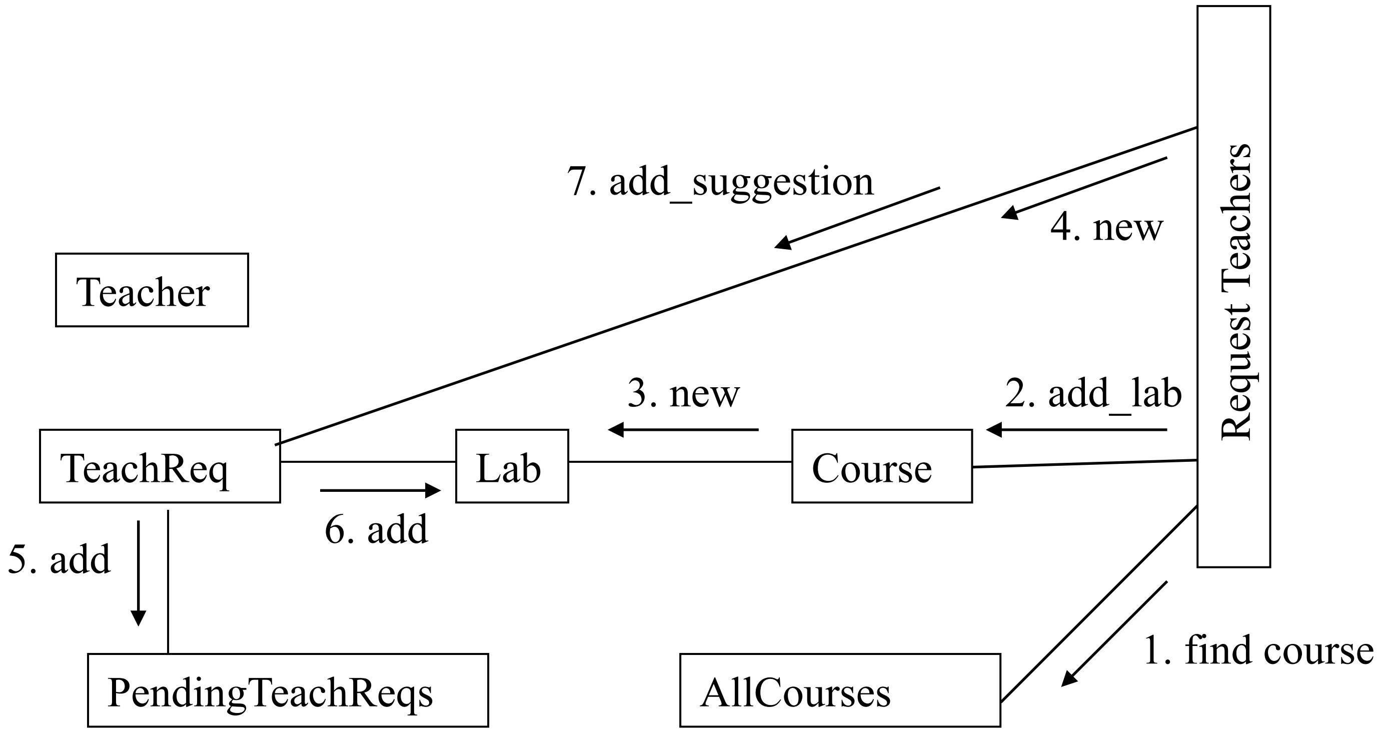
Sequence Diagram 5



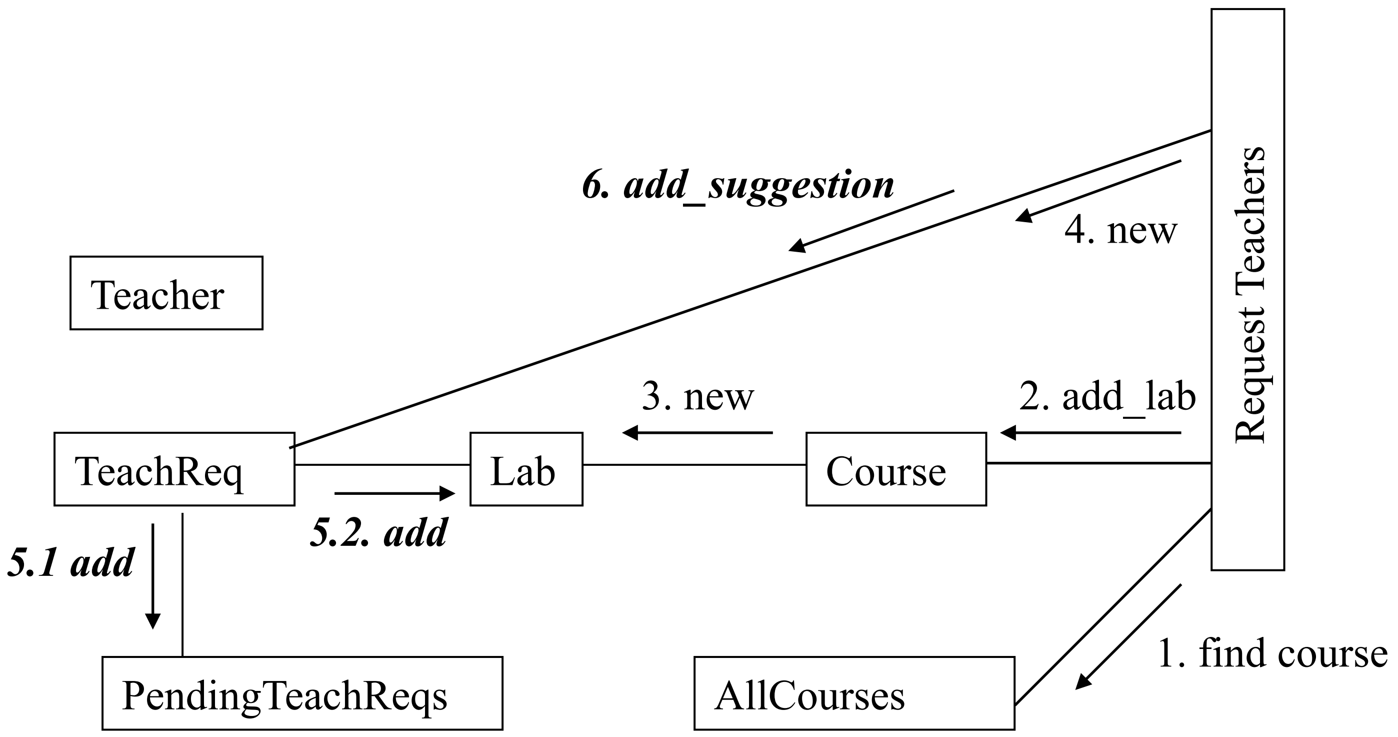
Classes with Dependencies



Communication Diagram



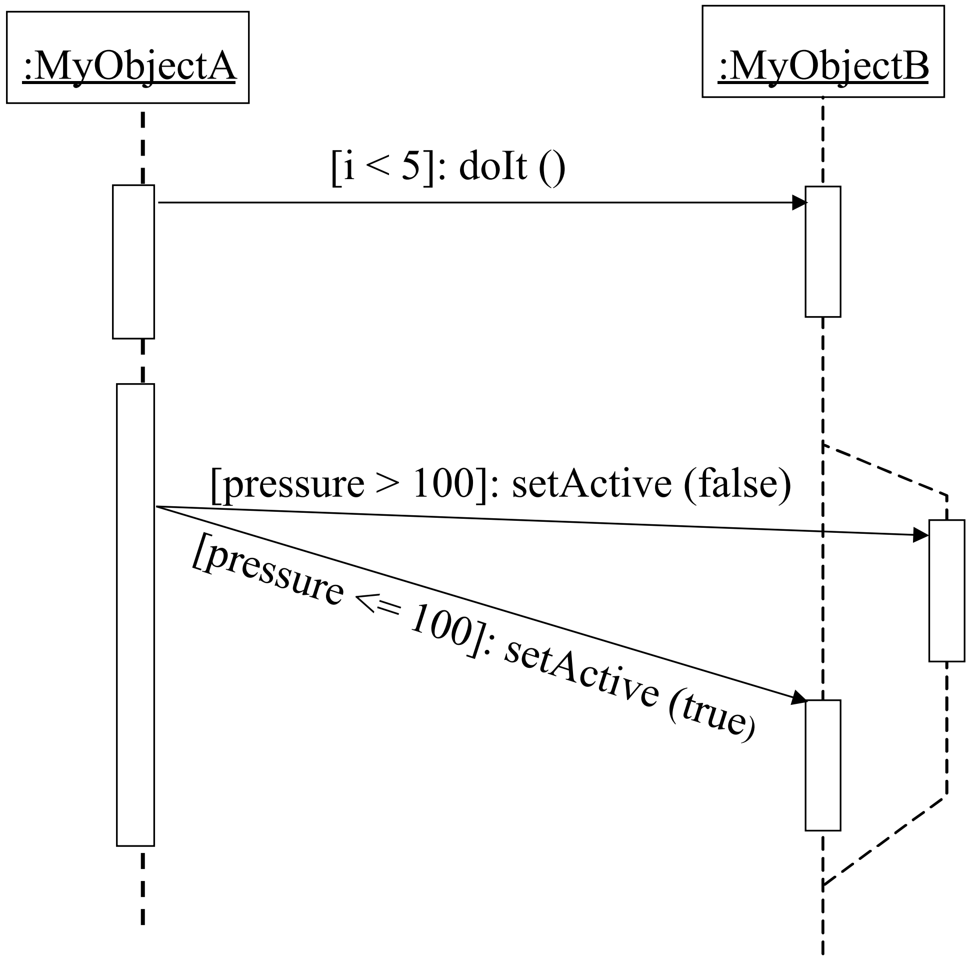
Communication Diagram (Revised)



Extending the Notation

- What we covered so far is enough for most problems you are likely to deal with
- There are various extensions that can add extra information to the sequence diagrams
- These are included in case you meet them in examples or other courses; you don’t really need them at this stage

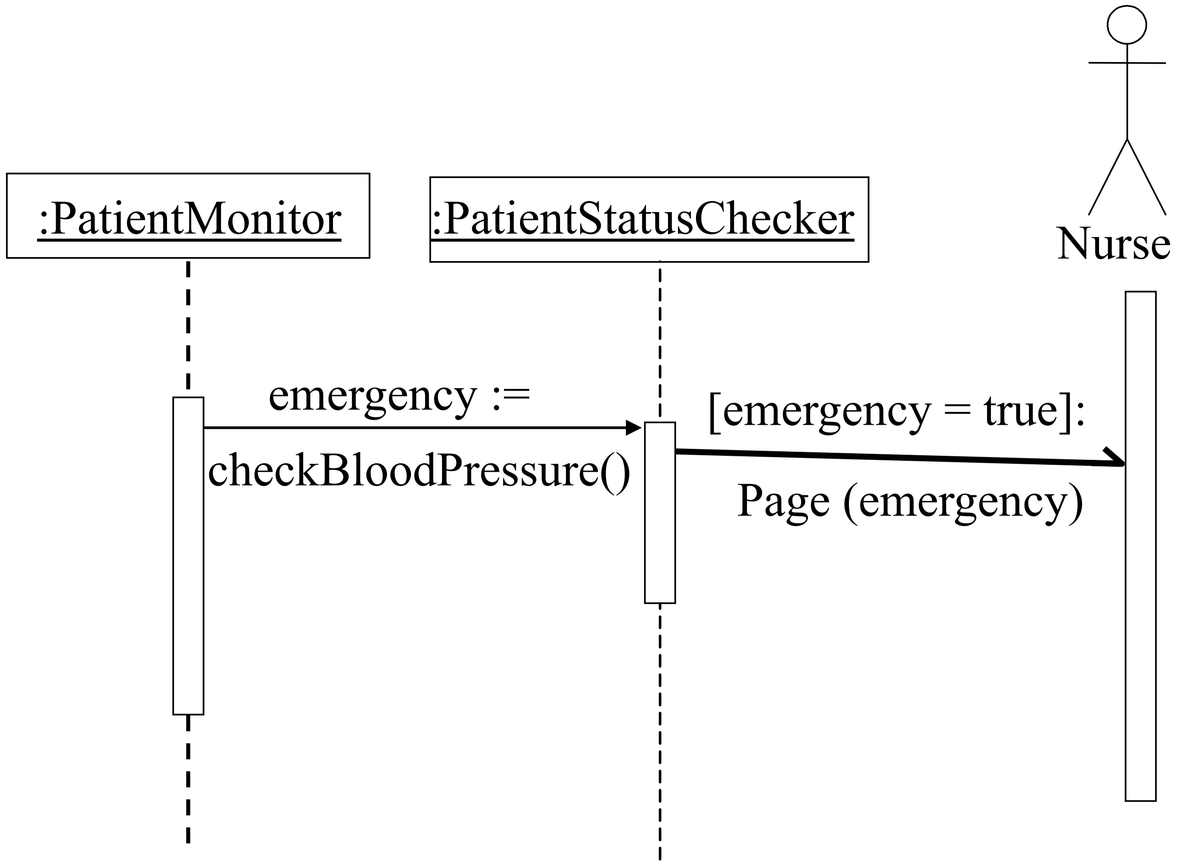
Conditionals


:MyObjectA
:MyObjectB
[i < 5]: doIt ()
[pressure > 100]: setActive (false)
[pressure <= 100]: setActive (true)

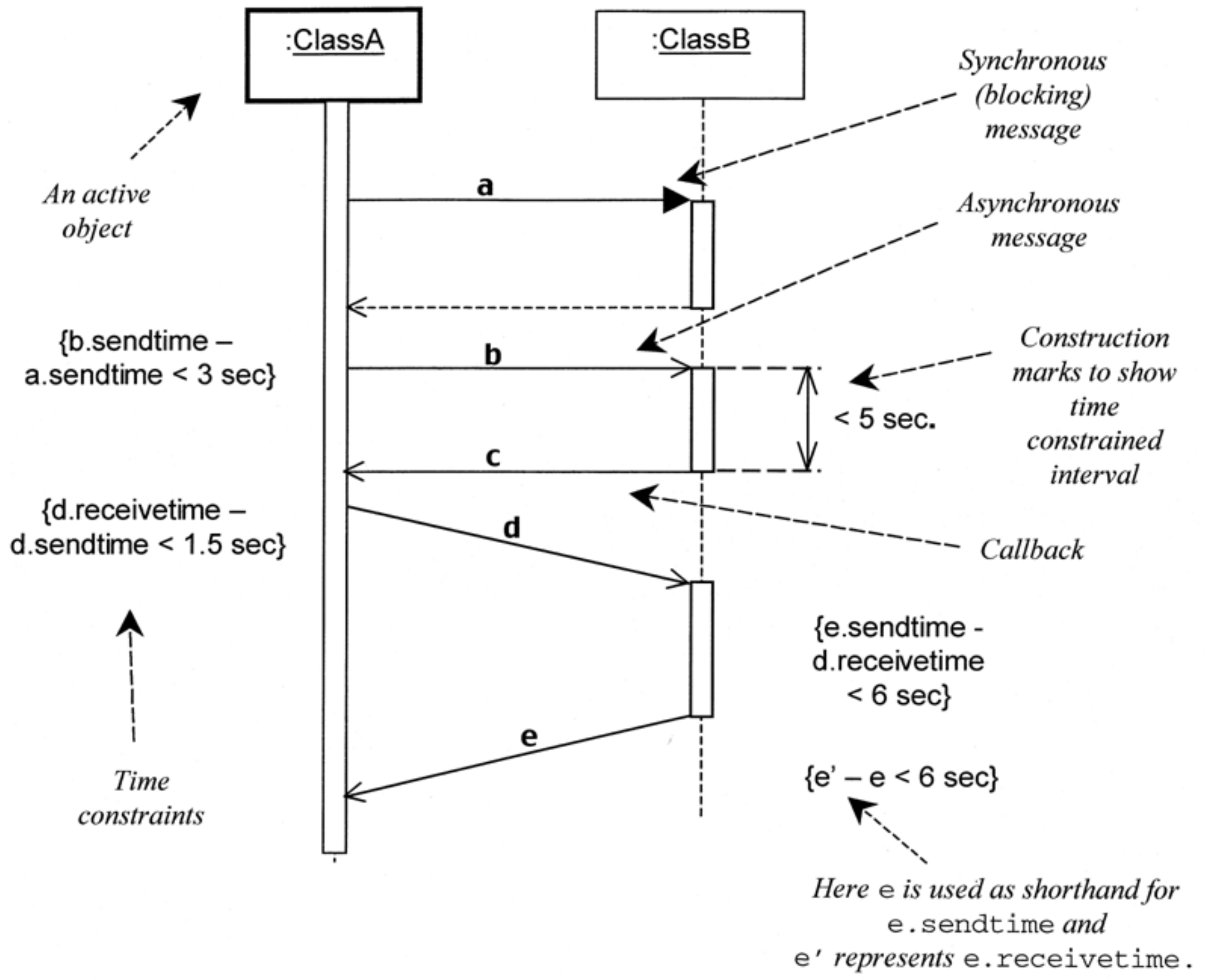
Asynchronous Messaging

- Normal assumption is that messages are synchronised; after sending message to object wait for response before proceeding.
- Asynchronous messages don’t wait for a reply
- May continue to be active and send messages
- Shown with a “half headed” arrow

Asynchronous Messaging Example



Timing Constraints


:ClassA
:ClassB
An active object
Synchronous (blocking) message
Asynchronous message
a
b
c
d
e
{b.sendtime – a.sendtime < 3 sec}
< 5 sec.
Construction marks to show time constrained interval
Callback
{d.receivetime – d.sendtime < 1.5 sec}
{e.sendtime - d.receivetime < 6 sec}
{e' – e < 6 sec}
Time constraints
Here e is used as shorthand for e.sendtime and e' represents e.receivetime.