University of Dublin
Trinity College

CS7CS3: Software Architecture - Functional

Prof. Siobhán Clarke
Ext. 2224 – L2.15
www.scss.tcd.ie/Siobhan.Clarke/

The Session Will Begin Shortly

1. Click on the pink Collaborate button (bottom right) to open the chat window and enter your message.

2. You can close the Collaborate panel at any time so you can see more of the current presentation.

3. Click on the menu icon at the top left when you want to exit the online lecture

→ A recording will be made available afterwards in case you get disconnected or have technical issues.

Student Online Teaching Advice Notice

Questions/Comments?

Last week we covered:

- Course structure and assessment criteria
- Challenges encountered in class members' experience with engineering software in teams

Before we start…
(for 5 minutes)

Testing shared document editing:

- I have shared a Google Drive folder with everyone called:

  CS7CS3 BB Breakout Test

- When you are randomly assigned to a group, go to the folder in CS7CS3 BB BreakoutTest for the group to which you have been assigned: Group 1, Group 2, ….

- Create a document there to which you should all be able to edit (including drawings). Try adding a new Drawing (under “Insert”) to the document.

- If you want to use a different shared drawing application, such as draw.io, then that’s fine too. Whatever works for collaborative drawing.

Software Architecture

What is it?

- Defines the basic components and important concepts of a system
- Describes the relationships between components/concepts

Different Views:

- Functional Architecture
  - *view of software components*
- Technical Architecture
  - *view of where software components reside*

Cohesion – 1

As systems grow, it’s important to know where functionality is provided

The simplest way to achieve this is if all aspects of one feature are provided in one component

- Keep a function together with its parts, and keep functions separate

A class/package/application/component that provides a single abstraction is said to be *cohesive*

- `java.lang.reflect` is strongly cohesive – provides reflection into classes: if you import it, that’s what you’re doing
- `java.util` is quite weakly cohesive – provides a load of stuff that doesn’t fit anywhere else: if you import it, that tells you nothing

Cohesion – 2

Different kinds

- Co-incidental – functions just happen to be collected
- Logical – all perform similar tasks, like a set of I/O routines
- Sequential – all involved in modifying some state in strict sequence
- Functional – all contribute towards a single well-defined task
- Temporal – all get called together, such as all initialisation routines
- Communication – all share common data

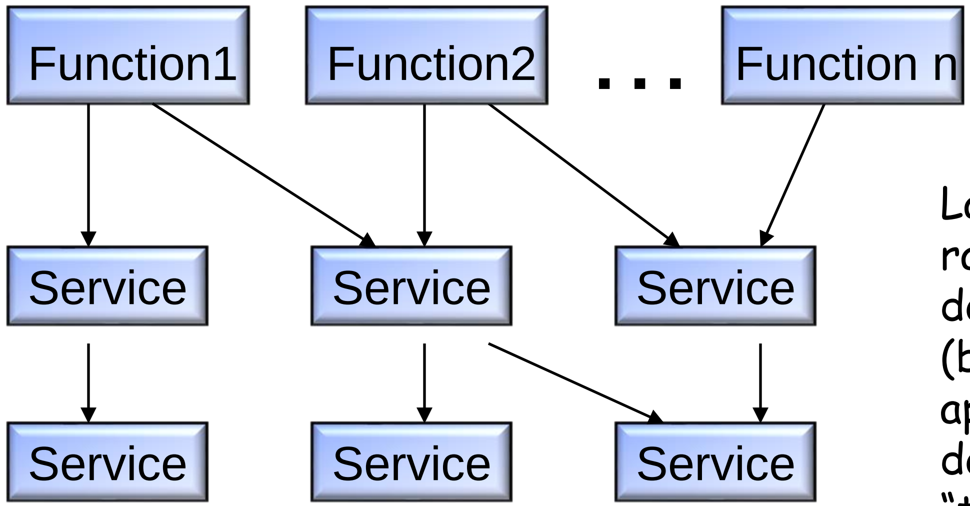
Good architecture and design will generate as many of these as possible (apart from the first)

- `java.lang.reflect` – functionally cohesive
- `java.util` – largely co-incidental, although the collections classes are somewhat logically cohesive

Functional Architecture – Principles 1

Function modularity. Classes (or equivalent design elements) grouped into functional blocks or service subsystems

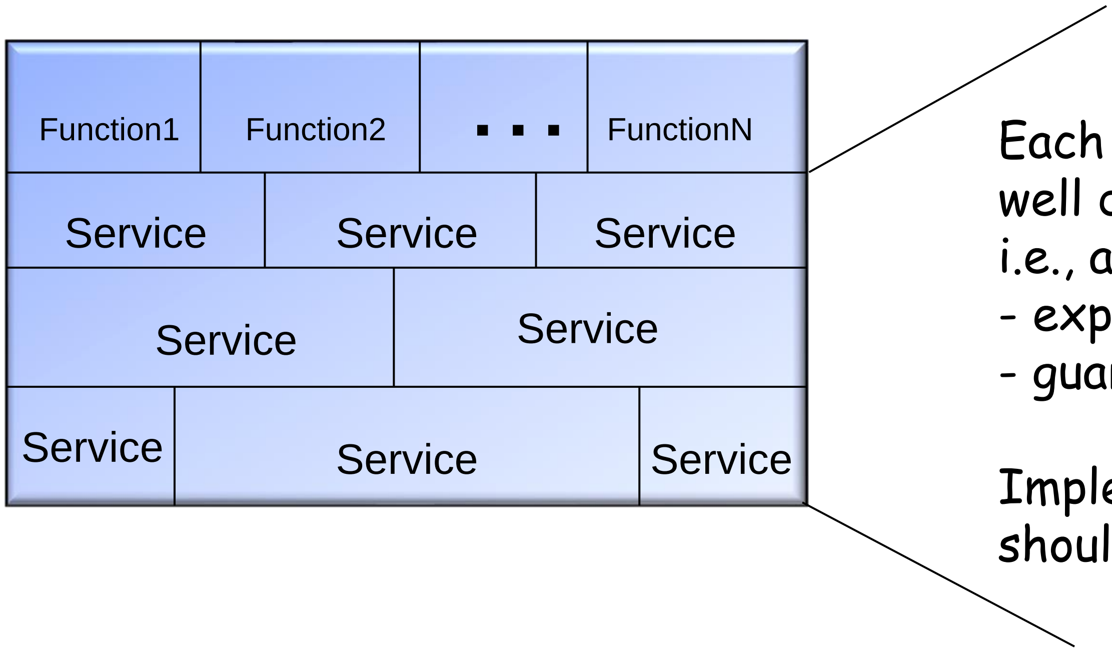
coarse-grained
e.g. "Billing"
"Ordering", etc.



Layered services ranging from domain-specific (but not necessarily application specific) down to more "technical" kinds of services, usually not even domain specific

Functional Architecture – Principles 2

Separate the design of interfaces from the design of service systems – goal is to achieve "plug-able" designs



Each service should have a well defined interface:
i.e., a set of APIs that clearly say:
- expected inputs
- guaranteed outputs

Implementation of these interfaces should be separate.

Functional Architecture – Principles 3

Map service subsystem in the design directly to one or more components in the implementation – (one for computational node), allowing distribution to different computational nodes. This makes managing changes in software on different installations more straightforward

Loose coupling between service subsystems – for example, (asynchronous) signals as the only means of communication between service subsystems

An example: Middleware for SOA for Internet of Things

A Service-Oriented Architectures (SOA) middleware needs a means for software services to:

- Be **registered** to the system
- Be **discovered** by consumers **requesting** services
- Be **composed** with other services
- Be provided to end consumers for **execution**
- **Negotiate** a Service Level Agreement with consumers

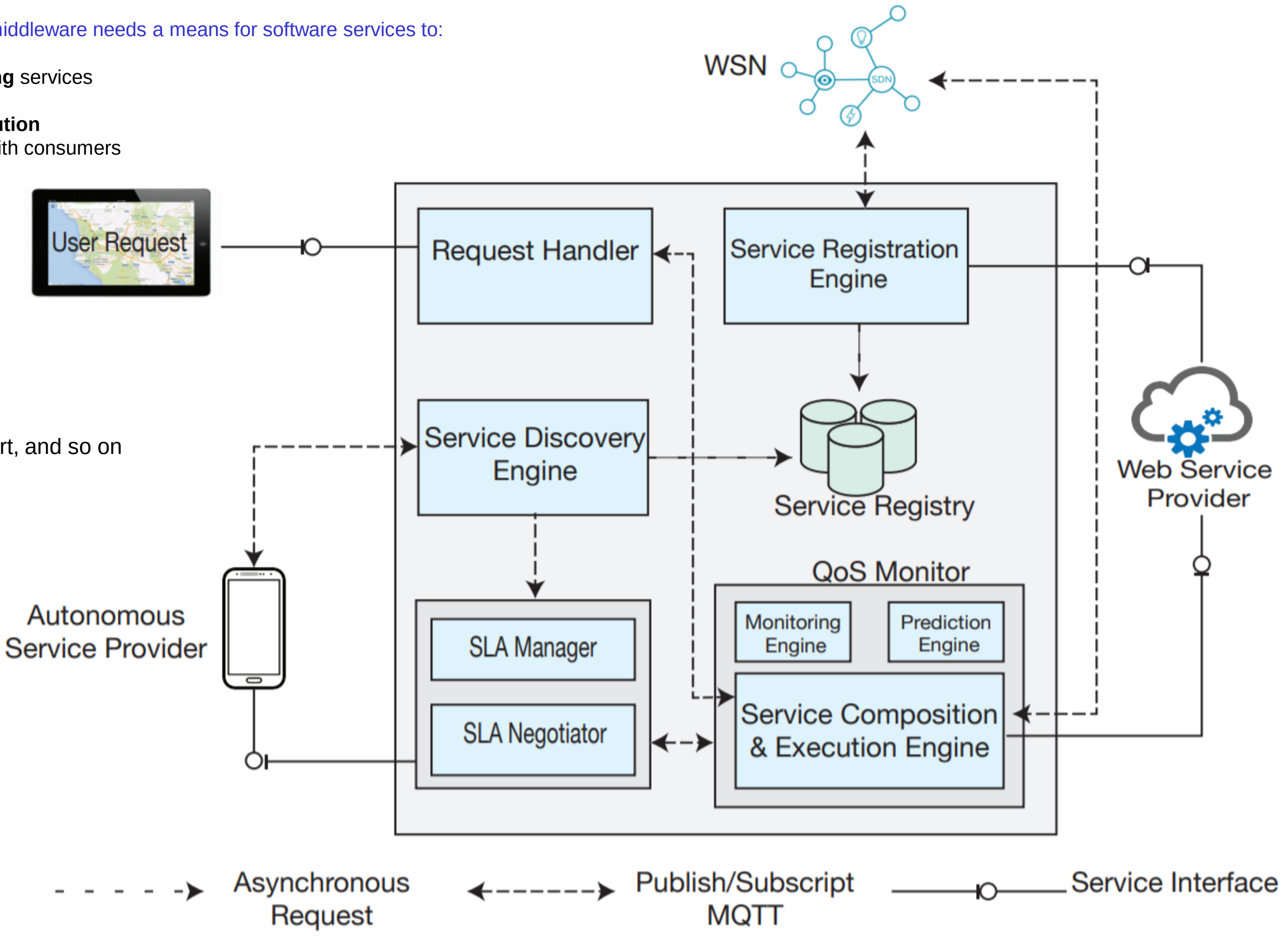
The Internet of Things includes:

- Sensors and networks of sensors
- Mobile devices (personal, on transport, and so on
- Traditional Internet

An example: Middleware for SOA for Internet of Things

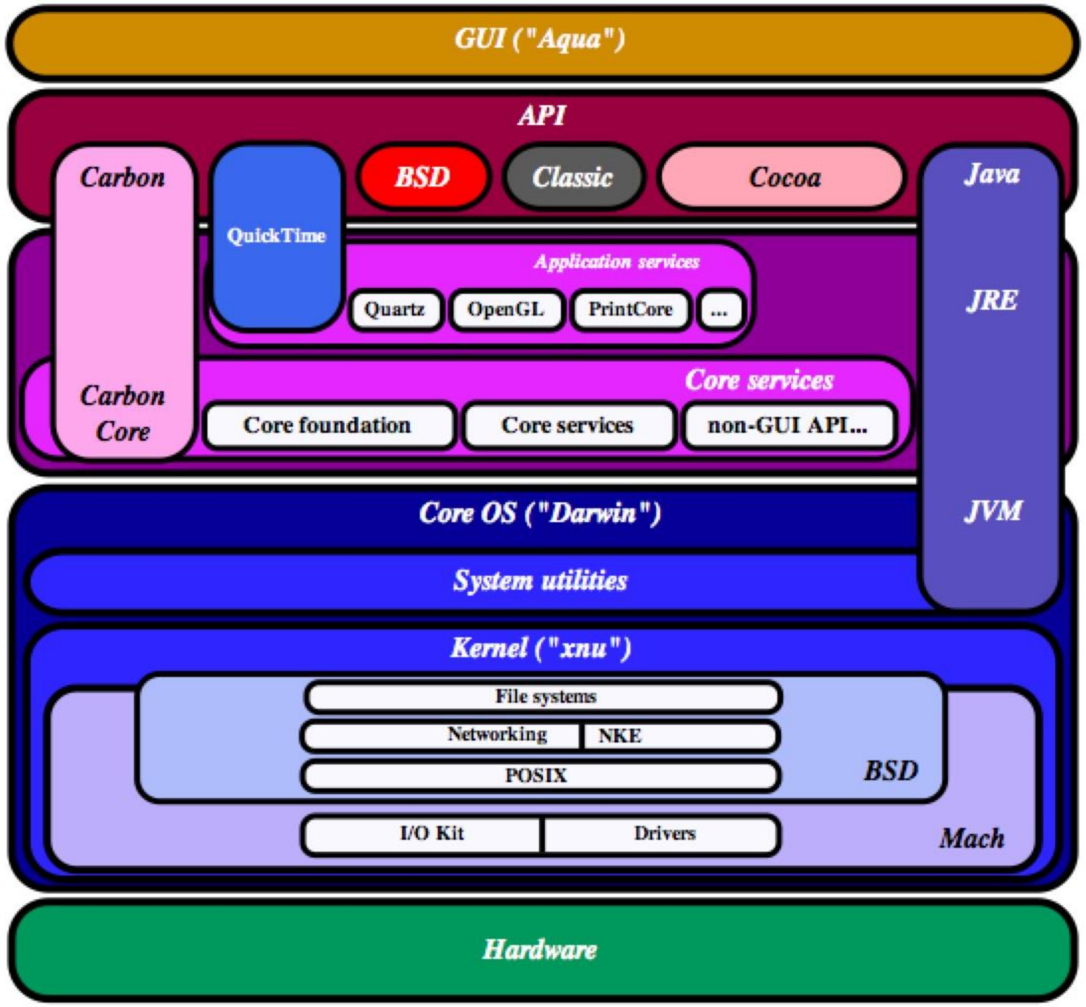
A Service-Oriented Architectures (SOA) middleware needs a means for software services to:
- Be **registered** to the system
- Be **discovered** by consumers **requesting** services
- Be **composed** with other services
- Be provided to end consumers for **execution**
- **Negotiate** a Service Level Agreement with consumers

The Internet of Things includes:
- Sensors and networks of sensors
- Mobile devices (personal, on transport, and so on
- Traditional Internet

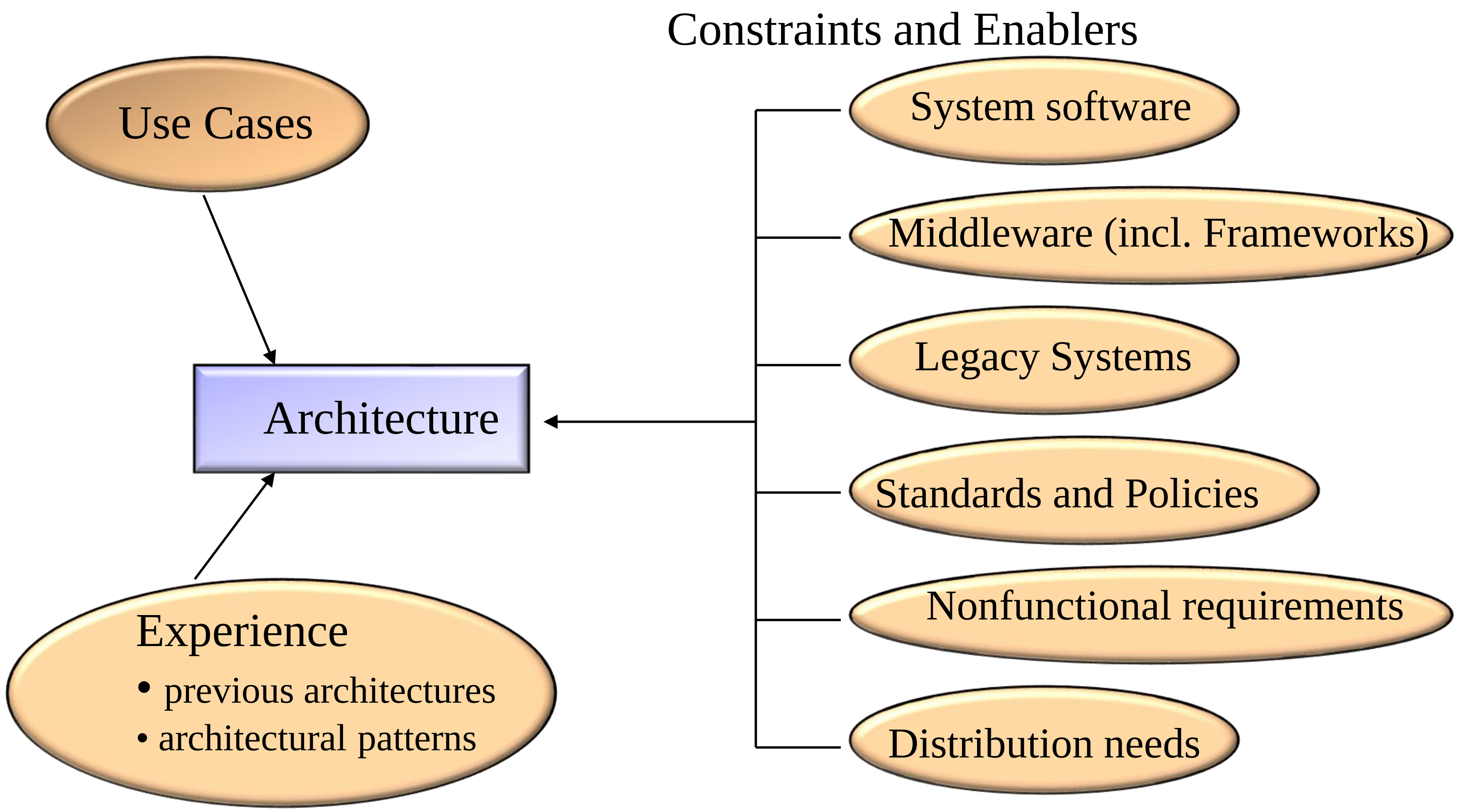


Another example: Mac OS X System Architecture

NOTE MULTIPLE LAYERS



Functional Architecture: inputs to its design



Example: Model-view-controller

One popular architecture for applications is the *Model-View-Controller* architecture

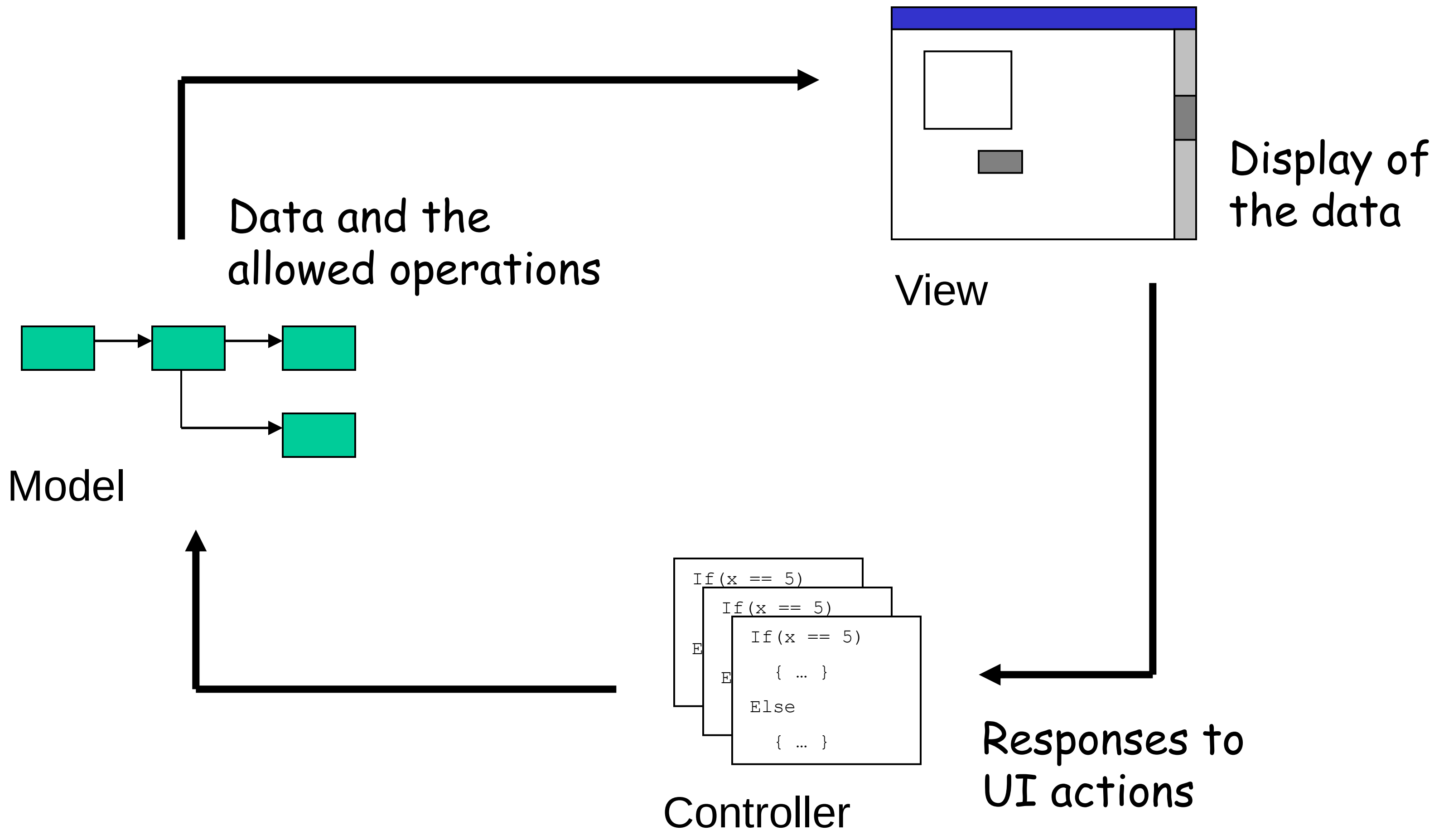
- Defined by Xerox in building the Smalltalk-80 system – perhaps the earliest widely-used object-oriented system

Separate concerns

- The model of the data in memory
- …from the presentation of that data to the user
- …from the way the user manipulates that data

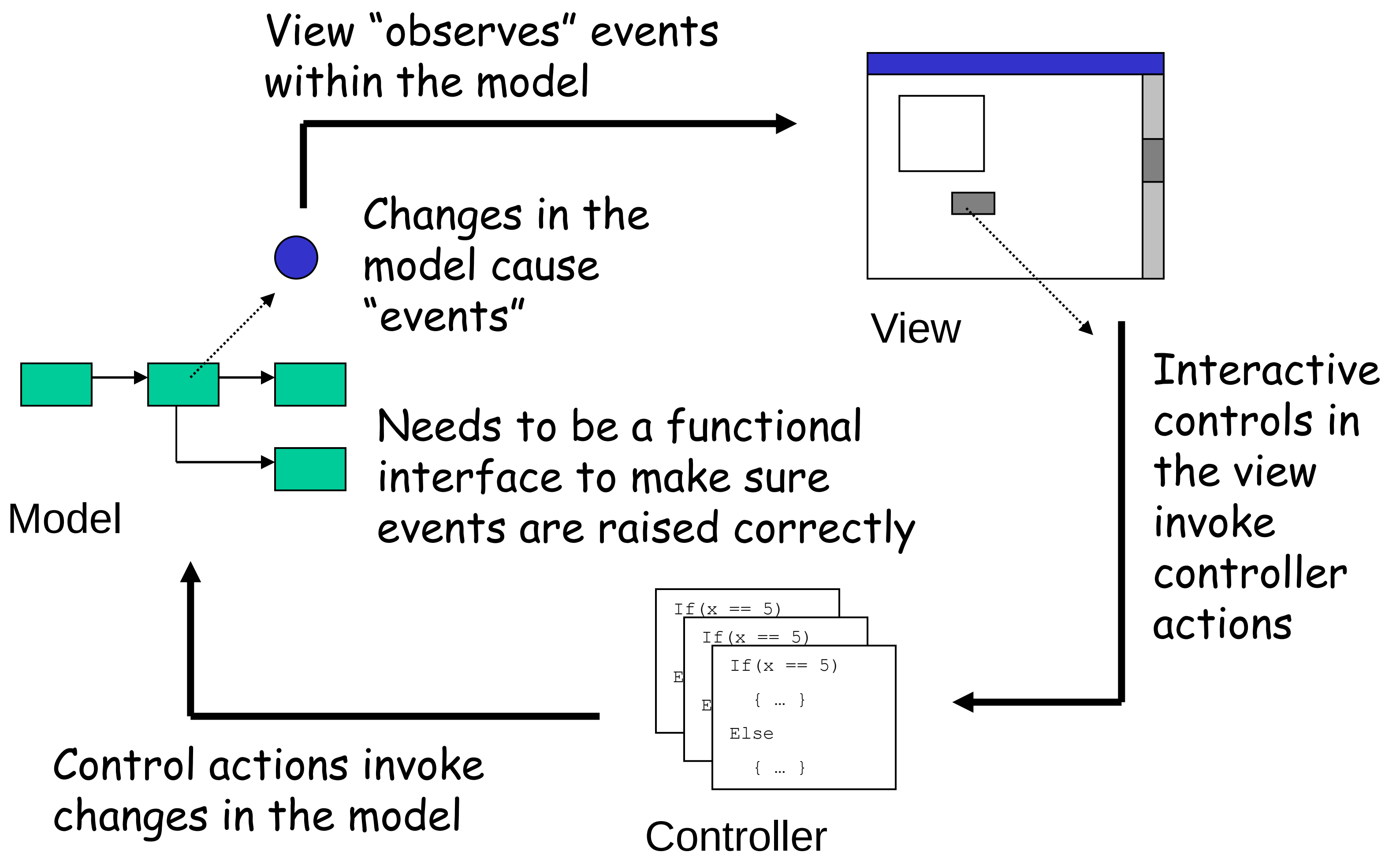
Many of the ideas are also in Java – for example *listener* classes for handling events in the GUI libraries – but Java doesn't mandate the full MVC separation

MVC



Data and the allowed operations
Model
Display of the data
View
If(x == 5)
{ … }
Else
{ … }
Controller
Responses to UI actions

Communication in MVC



View "observes" events within the model

Changes in the model cause "events"

Needs to be a functional interface to make sure events are raised correctly

Interactive controls in the view invoke controller actions

Control actions invoke changes in the model

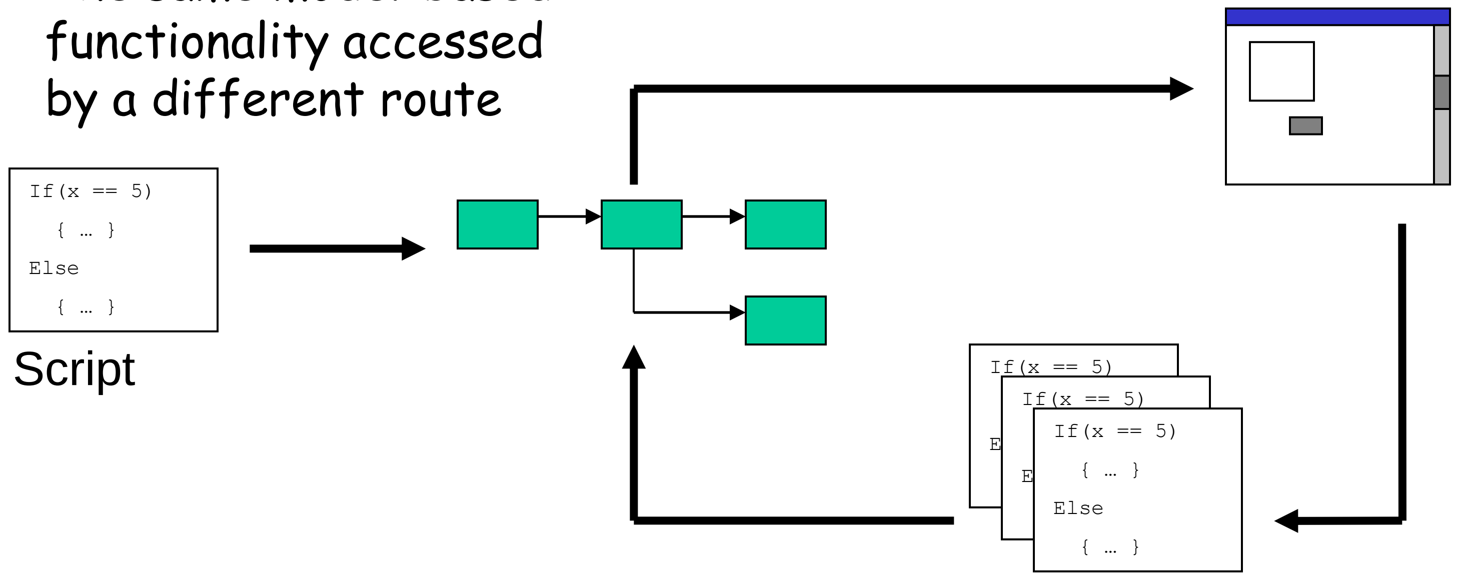
```
If(x == 5)
  { … }
Else
  { … }
```

What does this do?

It encourages the separation of interface from data

- Provide a new view and controller onto the same data
- Provide a new way of controlling the model
- Changes in model will propagate to the view, no matter how they occurred

The same model-based functionality accessed by a different route



Example

User interface specialists talk about "first-person" and "third-person" interactions. What if a user (who needs a nice interface) and another program want to work with the same data?

In many ways, user-driven and program-driven interaction are "the same" in some sense

- MVC-like architectures encourage allowing access to the same functionality through different modalities

Often we encounter options which are "the same, but different" within a single modality

- Different styles of graphical user interface
- Different scripting languages

How can we best deal with these?

Cohesive modularisation is key

- Functional modularity
- Clearly defined interfaces
- Separation and layering of useful services

Functional Architecture Exercise

- Go to Google Drive shared directory using **your TCD email address**:
    - "CS7CS3 Functional Architecture Exercise/"
- You will be randomly placed in a group number N:
    - Go to your group's subdirectory: "Group N"
- Activity: Create a FunctionalArchitectureExercise doc in your group's directory and add a new Drawing (under "Insert"). (note: you can use any drawing tool that allows concurrent editing.
    - Work with your team to devise an architecture for the exercise on the next page.
- Time Allotted: 20 minutes

- **Slides**: In the breakout room, switch between the exercise slide in BB, and your group's Google Doc workings

- Reporting Back: At the end of the activity, decide who will report back to the main room with a description of your architecture

Functional Architecture – Exercise 1

Define Functional Architecture for Hotel System:

The new software application will handle the process of reserving a hotel room. The reservation process is initiated by an enquiry from a potential customer, who states his needs. Room availability is checked and if a suitable room is available the customer makes a reservation. Details of the reservation are confirmed to the customer by e-mail.

The following five use cases must be catered for:

- The customer might arrive and take up his reservation (increasing his “preferred customer” points);
- he might cancel the reservation;
- he might amend some details of the reservation, which will require another confirmation;
- he might not turn up (no-show), but he’s going to get a bill anyway;
- The hotel may provide a complimentary room if the guest has enough customer points.

Functional Architecture exercise 2

- Assign yourself to the same group you were in for the previous session.
- Go back to Google Drive shared directory using **your TCD email address**:
    - “CS7CS3 Functional Architecture Exercise”
- Go to your group’s subdirectory: “Group N”
- Activity: Add to your previous architecture for the exercise on the next page.
- Time Allotted: 20 minutes

- **Reporting Back**: At the end of the activity, decide who will report back to the main room with a description of your architecture

Functional Architecture – Exercise 2

Functional Architecture for Hotel System:

possible components:

- Reservation UI
- Reservation system
- Billing
- Hotel manager
- Customer manager

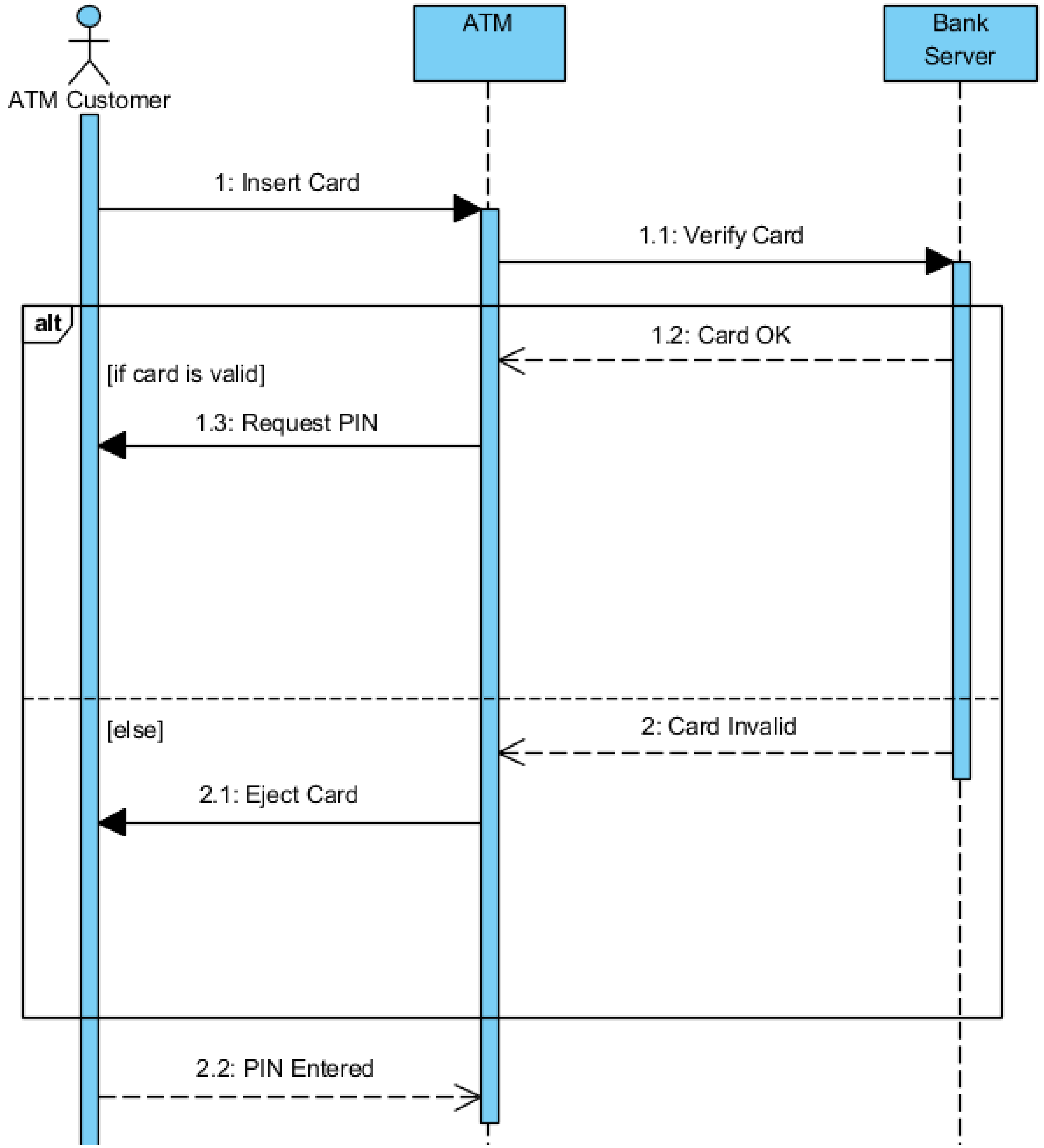
For each of the five previous use cases, indicate how the components in your architecture interact to achieve the goals of the use case. Use UML Sequence Diagrams to show the interactions.

From the Sequence Diagrams, you should also be able to list, for each component:

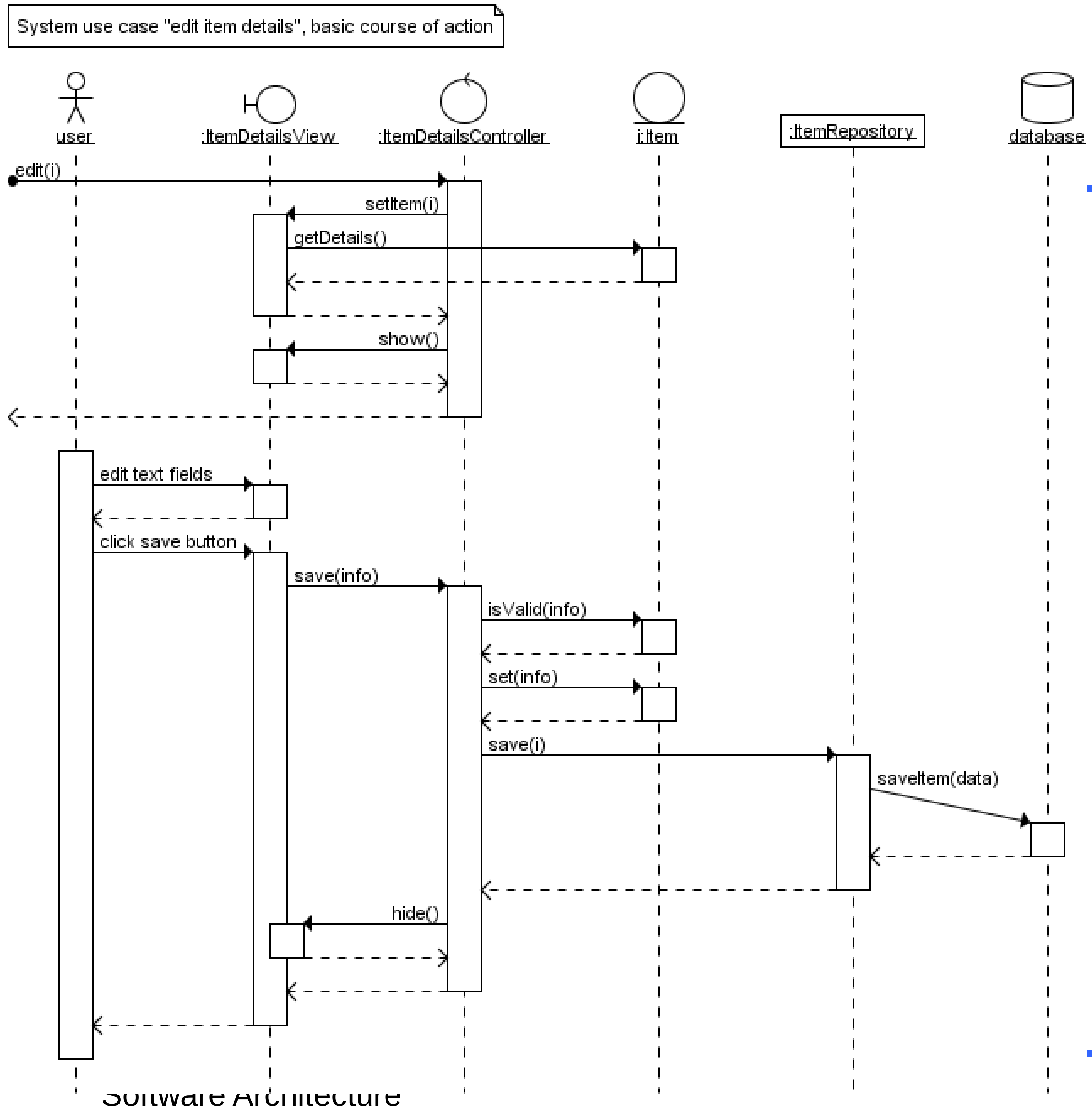
Names of the APIs

Expected inputs for each API (input parameters)

Guaranteed outputs under normal conditions (return parameter)

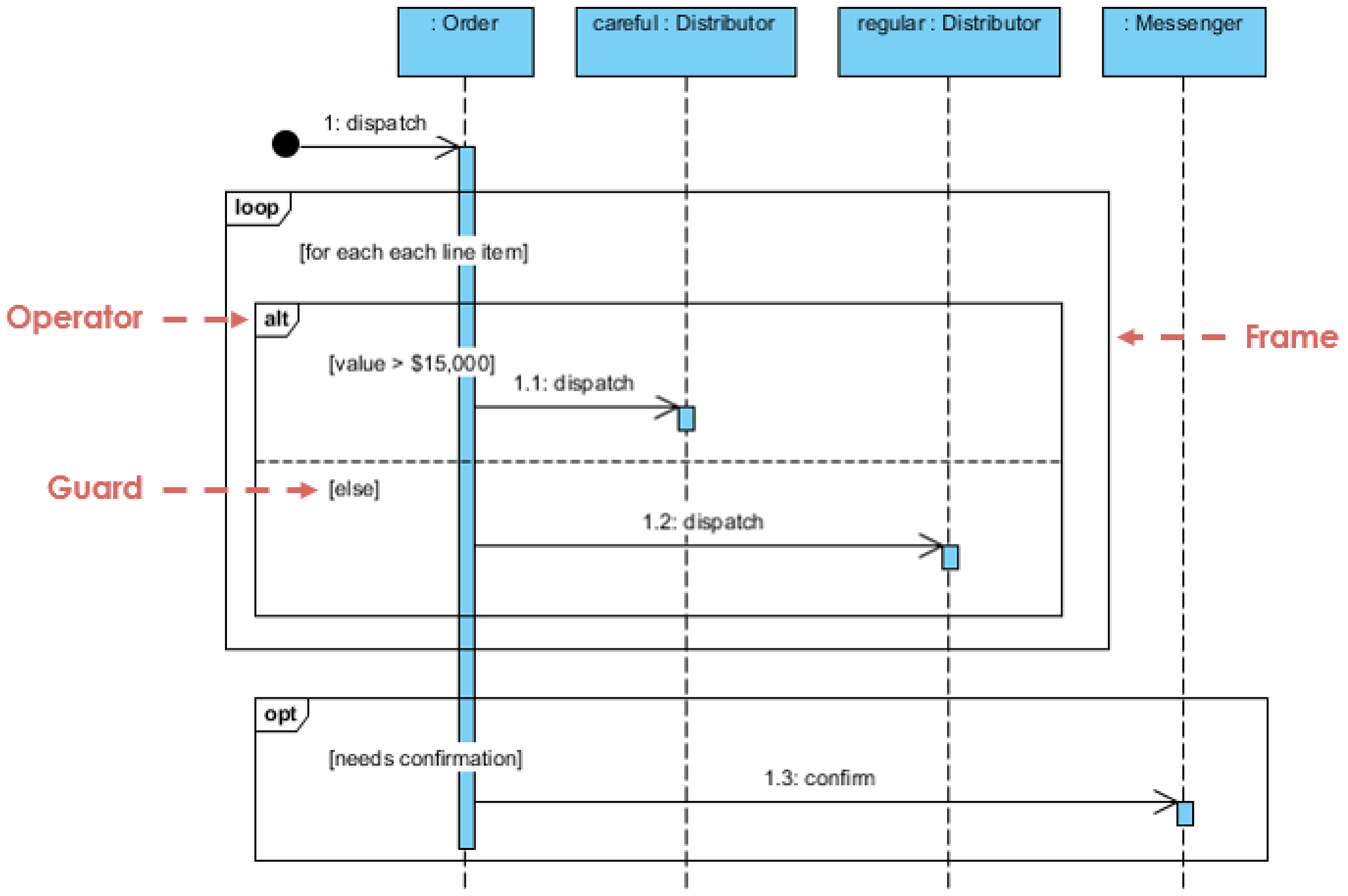


Sequence Diagram Example 1

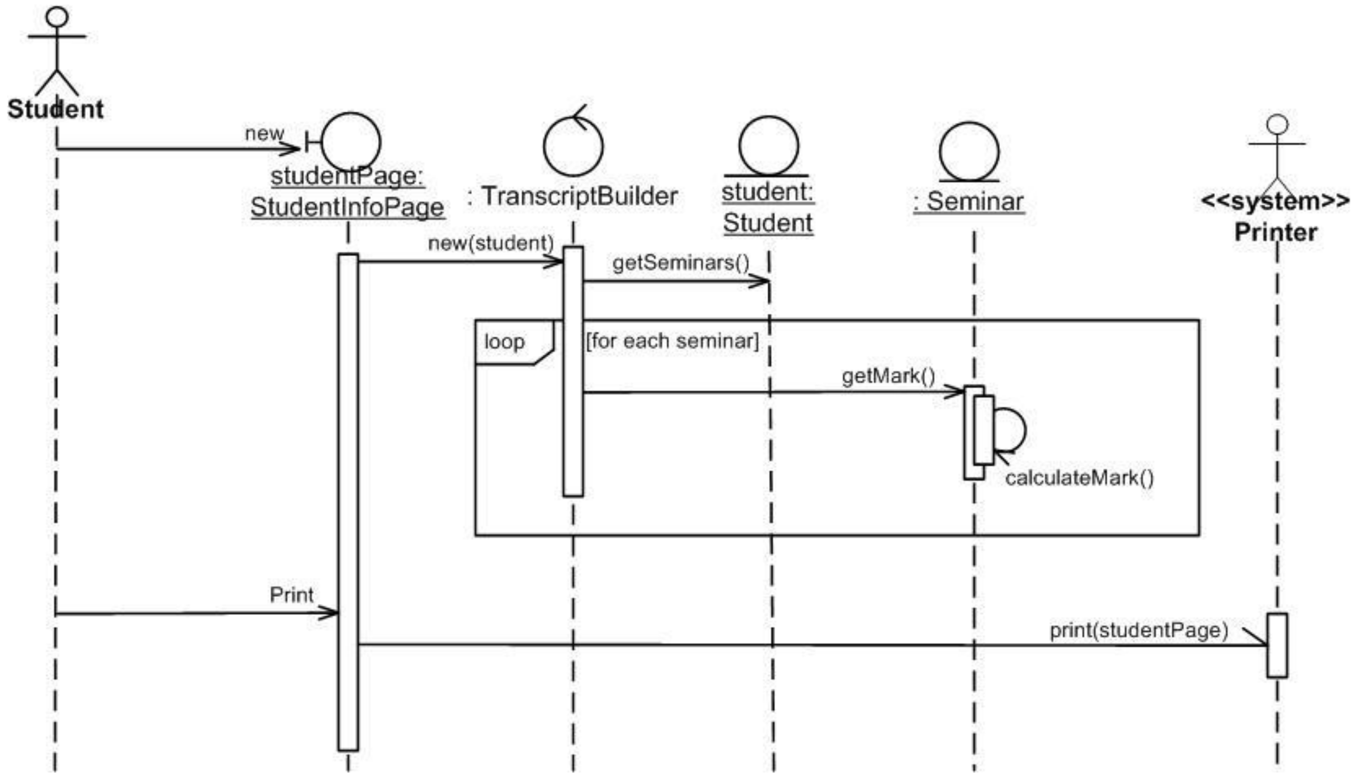


Sequence Diagram Example 2

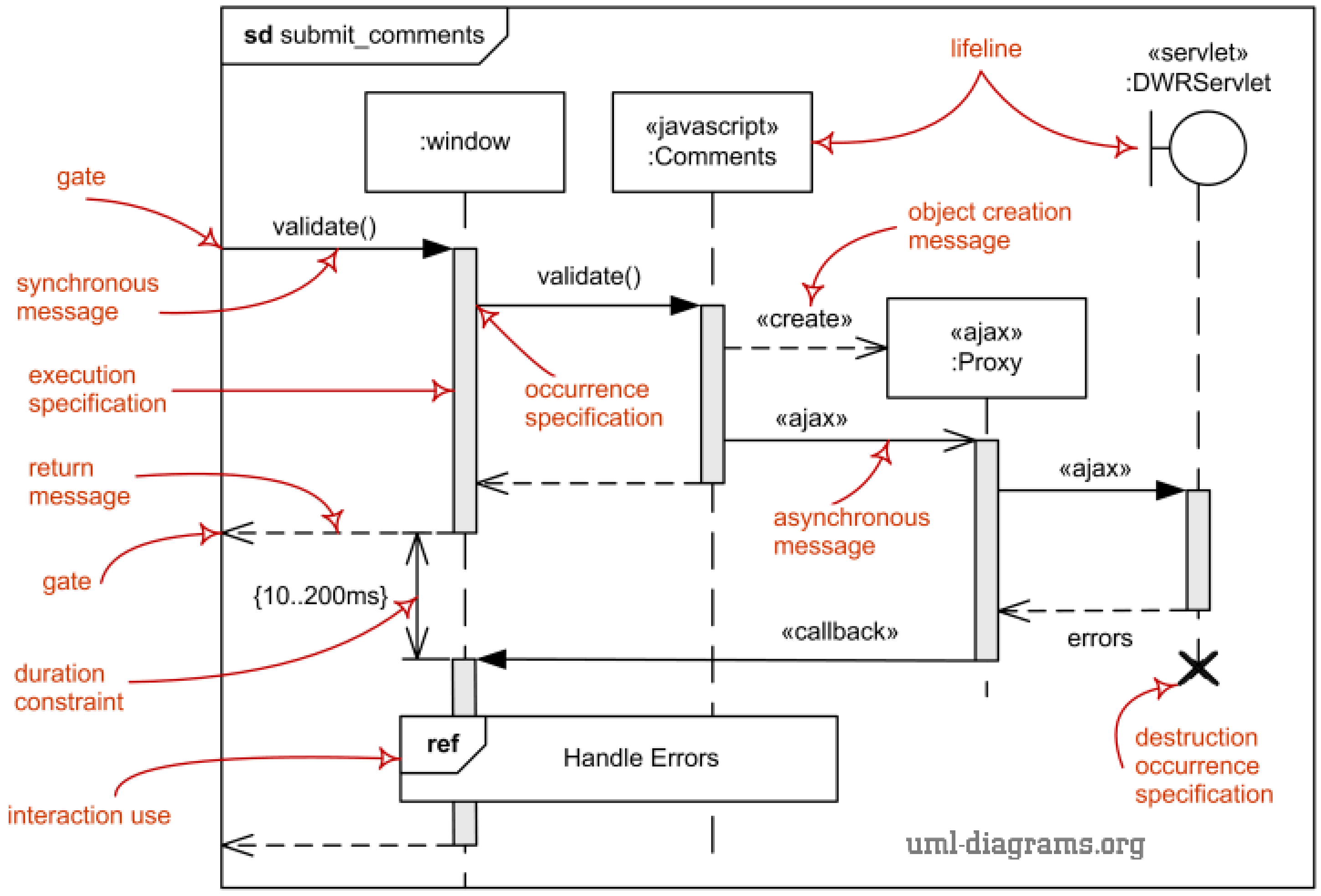
Sequence Diagram Example 3



Sequence Diagram Example 4



Example 5, illustrating what each of the notations mean



Component architecture

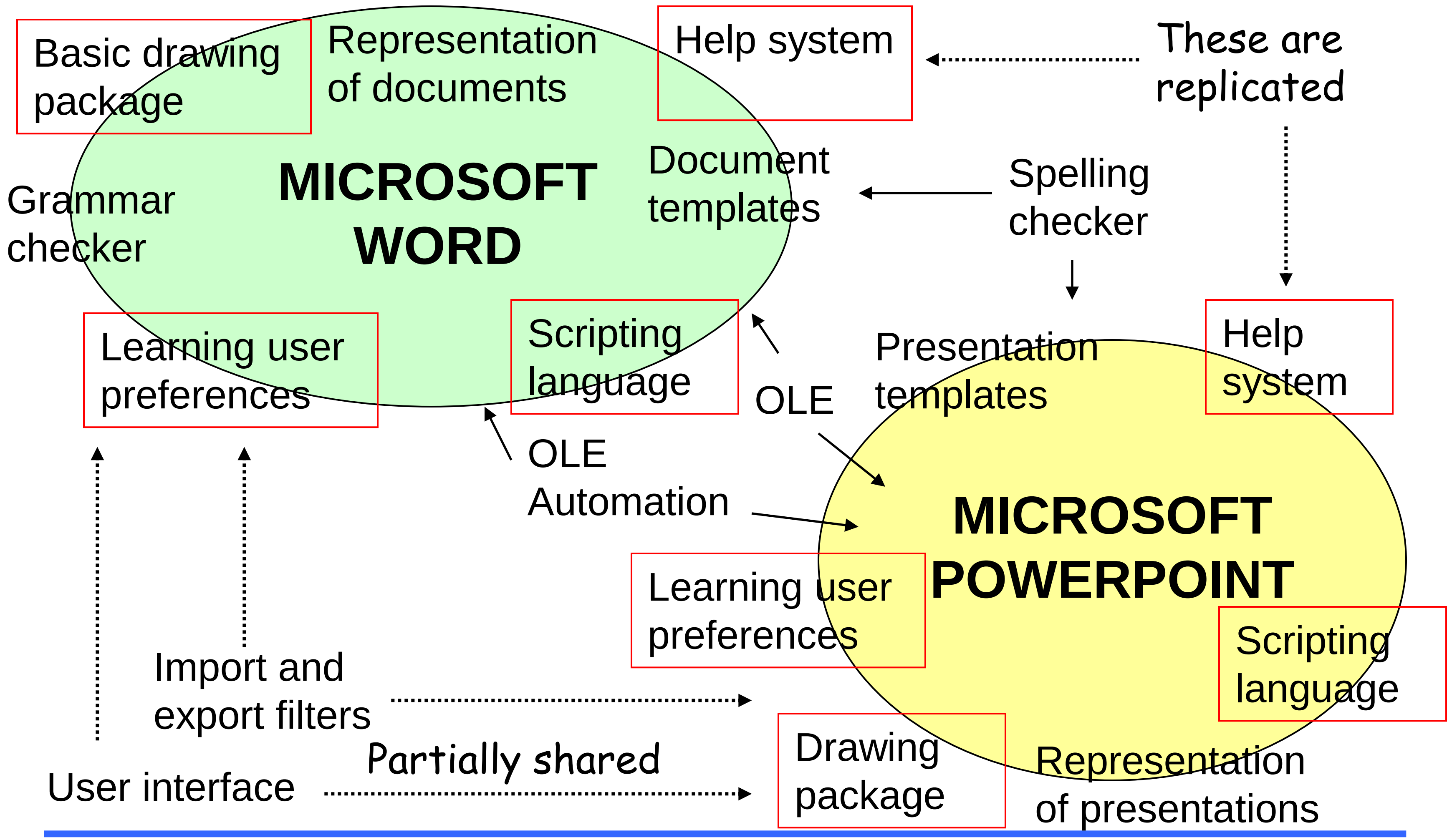
Build applications out of a set of identified but rather interchangeable pieces

- At compile time – easier configuration, no user changes
- At start-up time – configuration files
- On the fly – add and remove components while running

Implications

- A set of component families – layout components, import components, …
- Each member of the family must “look the same” – generally be substitutable in the sense of the LSP

Systems of applications



What’s happening

Re-factoring to share components

- A shared spell checker – write once and re-use
- Support different languages, loading the correct one at run-time

Not everything similar can be shared

- Help systems – may be desirable to unify them, may be too confusing
- Can the scripting languages be the same? Or are they just similar? Or can they be factored into a common part and several application-specific parts?

In a large suite like Microsoft Office, which parts are shared? Are these the right parts?

Summary – applications and systems

The same ideas in different guises

- Changeability – decouple and make cohesive
- Re-use of macro-architectures, factoring the changeable parts
- Re-use of components, possibly dynamically loaded
- Sharing functionality between applications

Application and system architectures

- Strive to make things easier to maintain
- Separate concerns
- Provide the same thing differently

Decisions here impact – and are impacted by – the largest contexts for the software: enterprise and global