Iteration 2 [All phases]

Slide Series 6

Iteration 2: Requirement Elicitation

Hmmm... possible have missed something?

- If so update RE sections of RAD

Iteration 2: Analysis

If changes to RE probably changes to analysis parts

- If so update analysis sections of RAD

Iteration 2: Design and Implementation

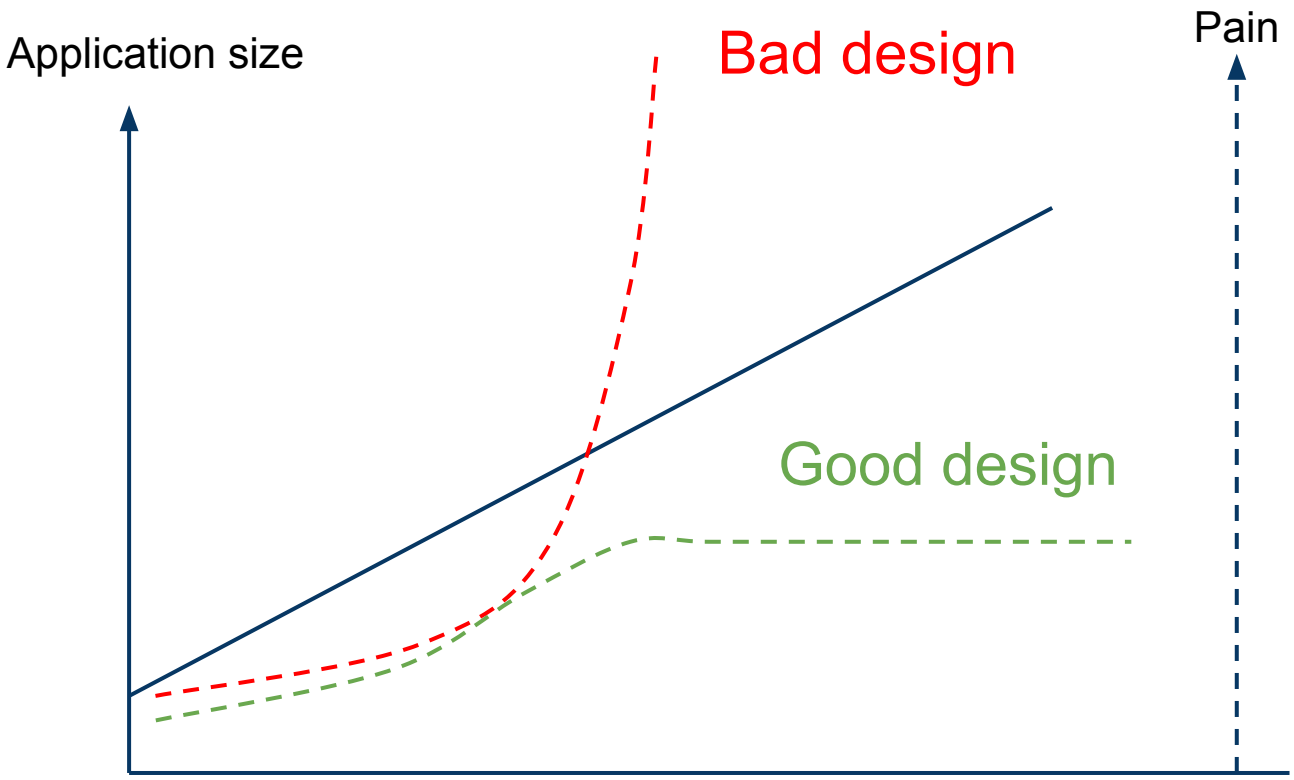
Still much to do...

- Model should be "initially" stable from previous iteration (at least parts of)

Goal is to get an "as stable as possible" design (the final infrastructure)

- Remainder: If design not stable we will fall back... (if model not stable, disaster...)

The Impact of Design



On which curve are we..?

Design Principles and More

Some considerations

- Information expert
- Single responsibility principle
- Open closed principle
- Programming to interface
- Low coupling/high cohesion
- Information hiding
- Law of Demeter
- Invariants
- Mutability
- Minimize State
- Canonical form for objects (equals, clone...)
- Threading? Remember Swing is single threaded!

During our work we must keep an eye on this, if violating, refactor (and test)! Run tools!

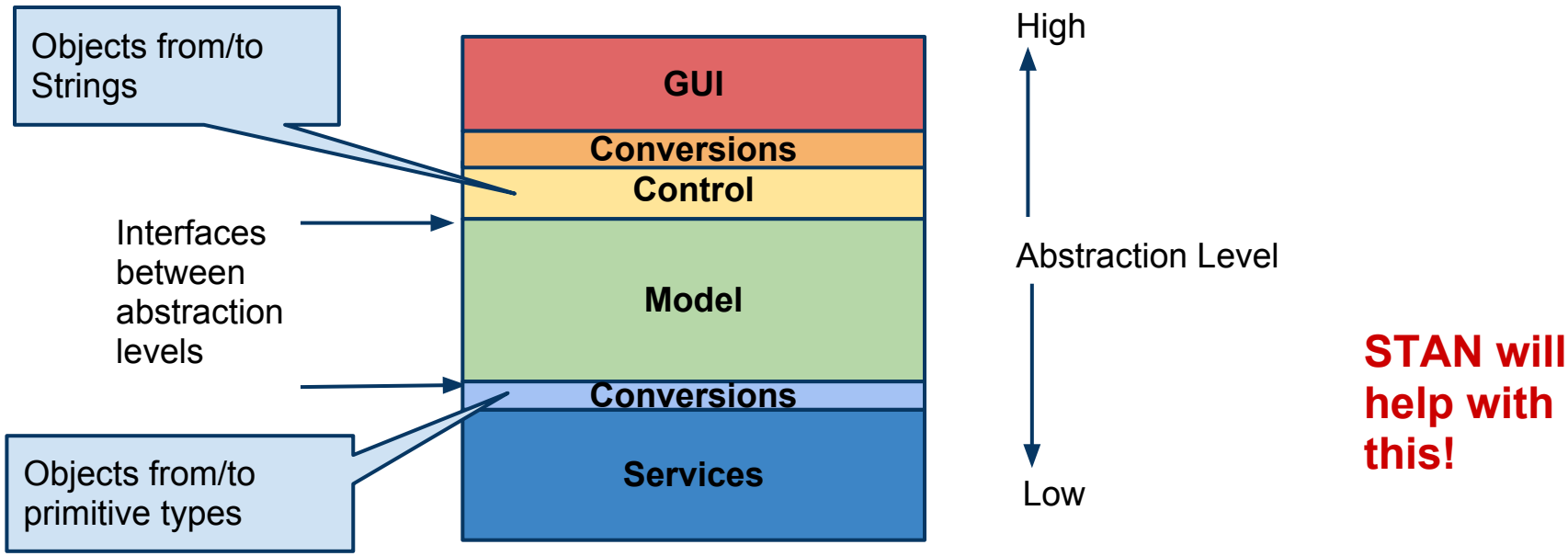
Design patterns possible can help ...

Partitioning and Layering

Have **partitioned** application into packages (modules)

Should also order by abstraction level, **layering**

- Note: Thickness of layers



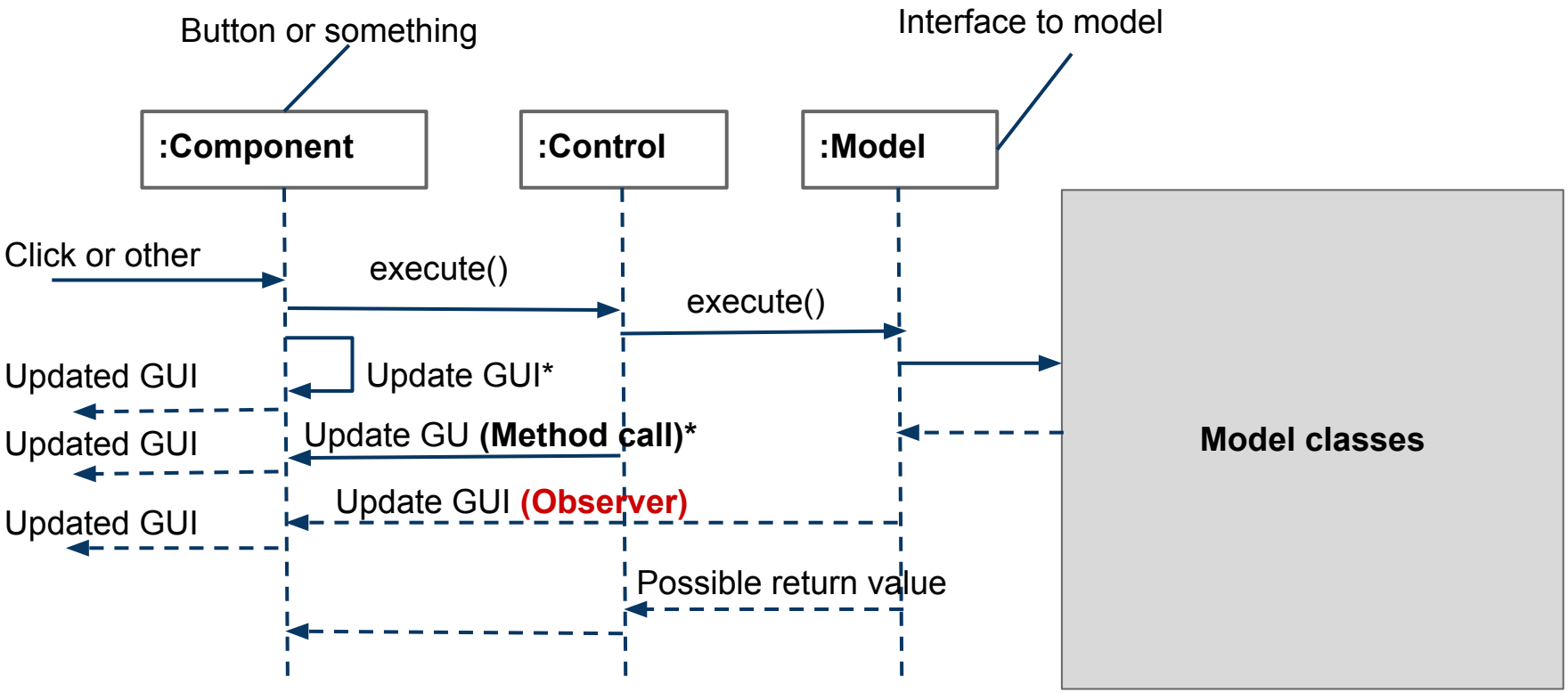
Fat model

So far we have put a lot of logic in the model, the model is fat

- Everything that doesn't need user interaction can be done inside the model
- If need interaction with user some logic will be in the thin "control"-part of MVC

There's also "anemic" models, no logic in model all logic in control parts (we don't use). Violates OO!

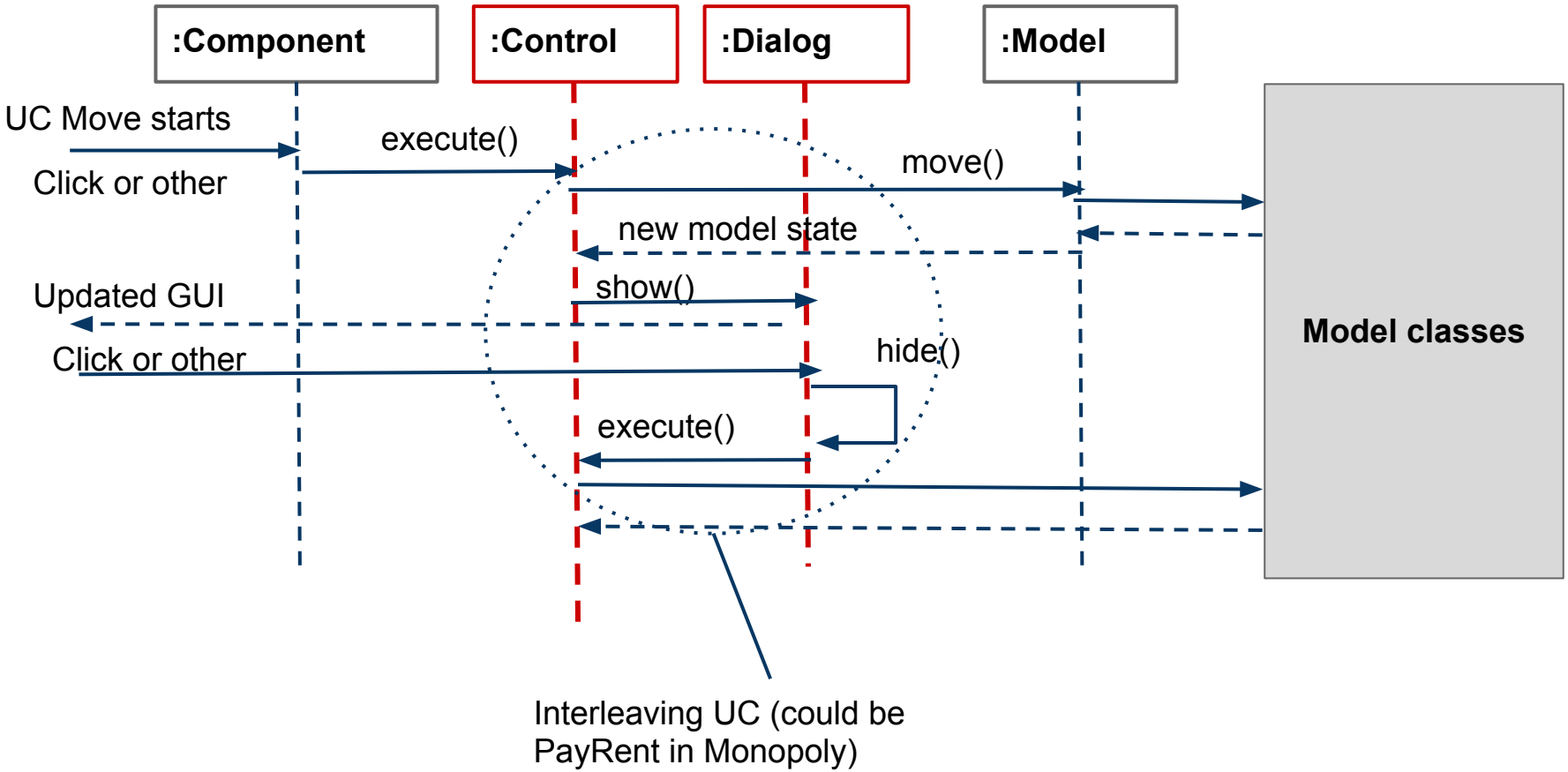
MVC: Simple Interaction



*) Possible alternate updates of GUI

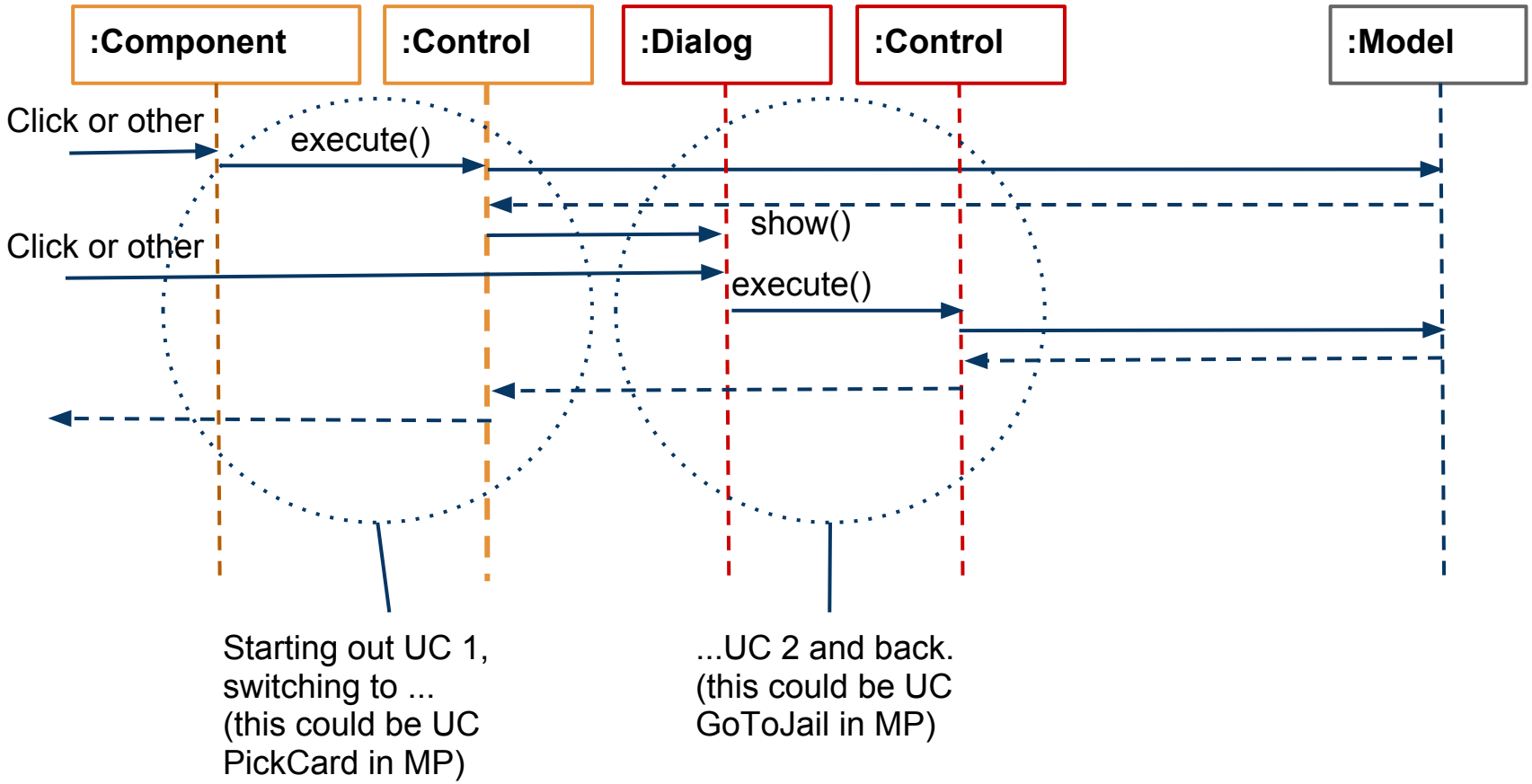
MVC: Complex Interaction, alt1

UC's handled by one control



MVC: Complex Interaction, alt2

More controls (diagram simplified)



Services

Model need services to run, some typical

- Persistence (store/retrieve)
- Printing
- Rule systems (business/game rules)
- Engines, simulation engine
- Graphics 2D, 3D...
- Processors (text formatter, spell checker)
- Security, authorization module
- Mappers, mapping between formats
- Communication, Network, ...
- ...

Remainder: Emulate

It's not the technique you need, you need the service!

- Create interface to service
- Emulate technique with something you can handle (if time, replace with "real thing" later)

Services: In House or ?

"In house" = we implement system

- ... possible many man-month needed for complex services!

Typically you don't implement subsystems for

- Graphics and sound, physics engines, .databases, data handling, XML, ...networking, ...

... find it somewhere! Look for high level abstractions

- Databases to Objects, very hard, use JPA, Hibernate, ...
- Network
  - Bad: Sockets, too low level.
  - Better: XML-RPC, RMI, ... other, ... much better (have protocol in place)

Implementing a Service

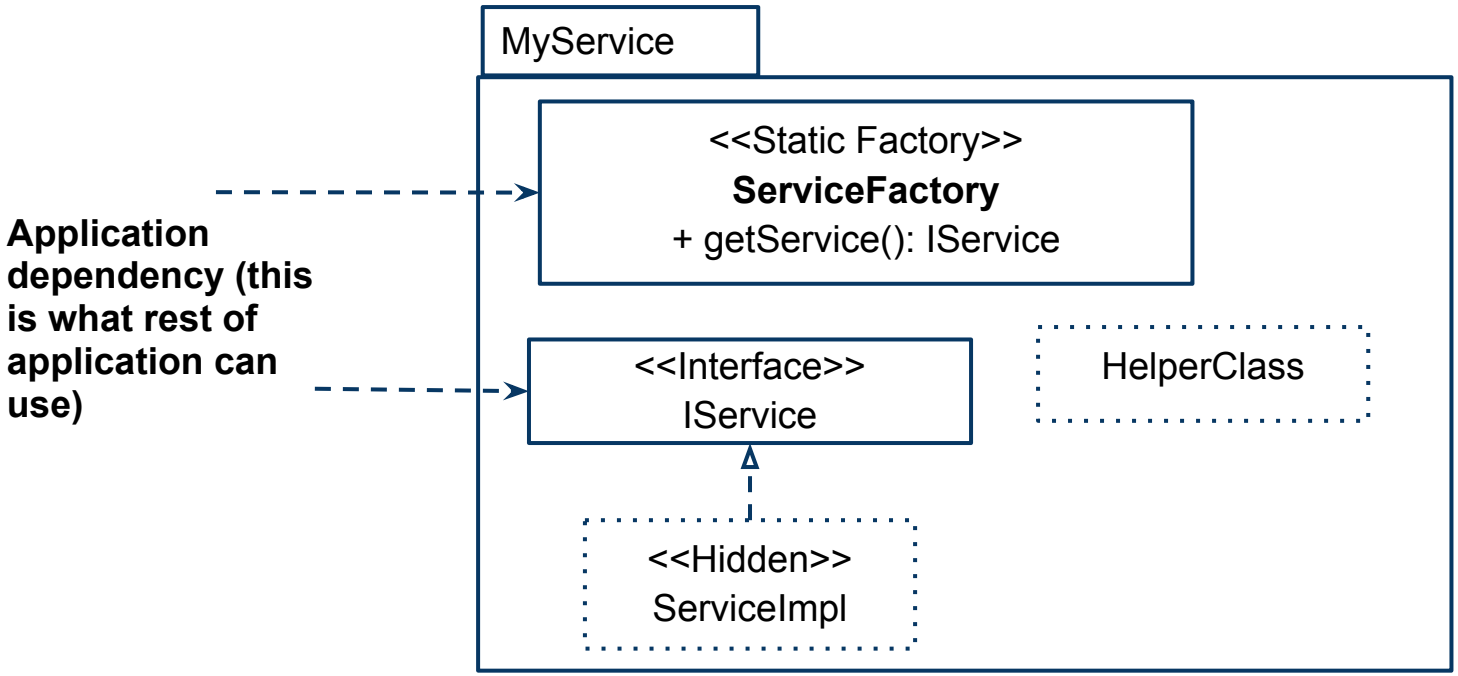
Often beneficial to split services into steps

- Basic step get/put primitive type, String (possible reusable)
- Conversion step; convert from/to objects (application specific)

Implementing a Service, cont

Complex services implemented using Facade pattern

- I.e. an interface used by model (or other) and a Factory to get some implementation



Services: TDD

Of course we write tests for the service

- Must know service is working before integrate into application
- Services are well suited for TDD, should have few dependencies on model, possible parameters/return values Should be possible to test in (close to) isolation
- If problems with dependencies use layering inside service
- Possible use of generics can remove dependencies

Parameterization of the Model

The model normally has quit a few parameters

- In MP; number of players, how much money for each player, number of spaces of board,...)
- Hardcoded for now

Don't want parameters spread all over the model

- Use dedicated class to keep all parameters (possible to change from GUI), "Options" class or similar (put in default values for everything)

Resources

Application need resources

- Config files, icons, sprites, images, localization, i18n, ...

Resource handling

- java.util.ResourceBundle, File automatically read and converted to Java "map-like" object, for texts in GUI
- For images use ClassLoader class getResource(s), findResource(), ...
- XML, use XStream or similar

Exception Handling

If possible to recover, do so at any location in application

If not, ... handle in control layer

- Close to GUI, possible to show dialogs etc.
- Avoid checked exceptions (possible by tunneling = wrap checked exception in runtime exception, rethrow)

Aside: Swing Issues

Swing is a complicated creature, sometimes it just doesn't work ...??!

- Force repainting using; validate(), repaint() or possible both
- Use setSize() or pack()
- Some components fire events when data changed! Possible have to disable when updating component models
- All drawing in Swing thread, possible have to hand over form other thread (SwignUtilites.invokeLater)

Iteration 2: MP

We don't do any RE or Analysis, focus on design and implementation

- Refactoring to full MVC
- Layering
- Adding a service (file handling) in-house (it's a simple service)
- Possible Rule-engine as a service (not implemented)
- No interfacing with different paradigms
- Resource handling, yes a lot... (not implemented)

Refactoring MP to Full MVC

Observer pattern, using an event based solution (EventBus)

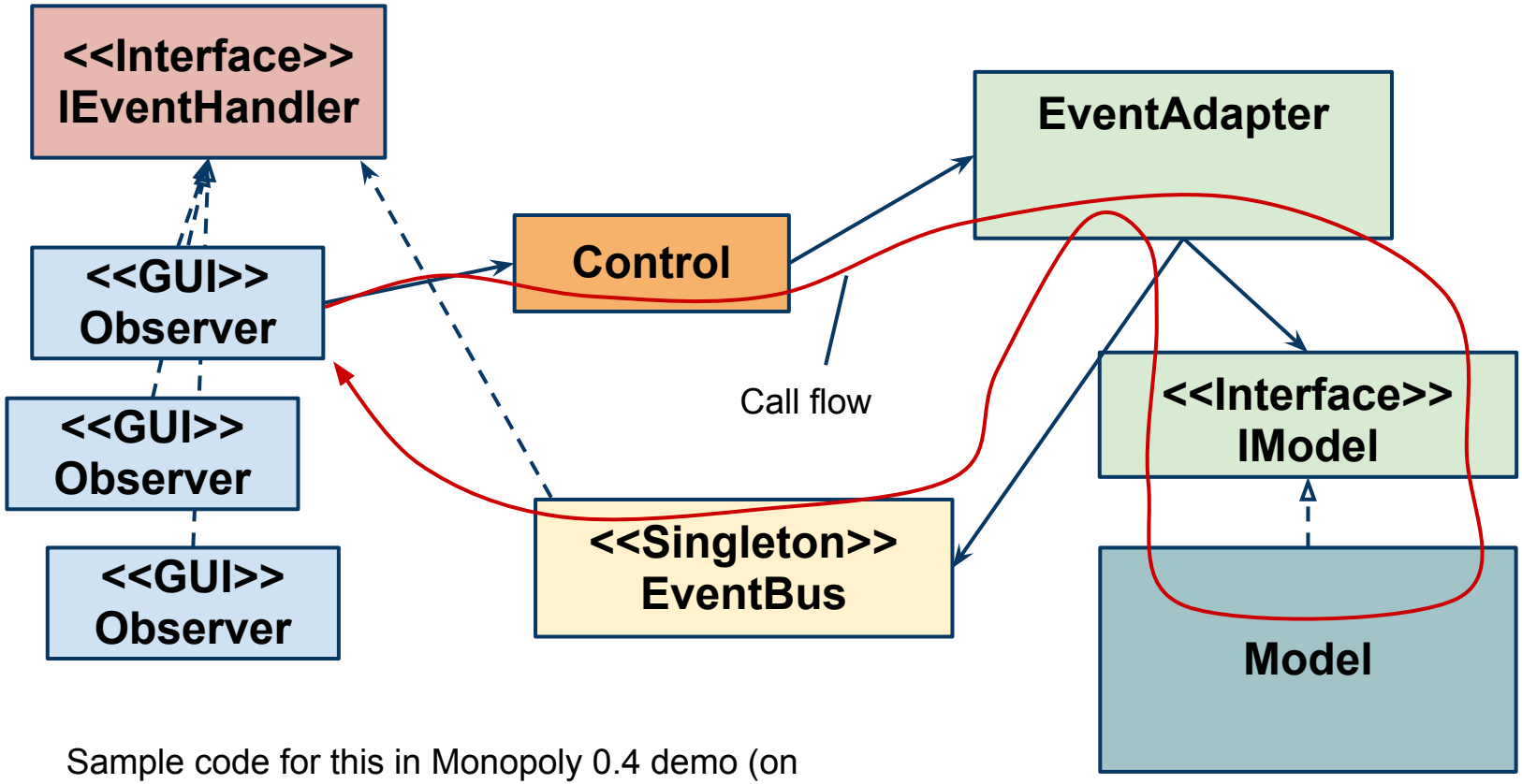
- An in-house solution
- There are others, ... Google Guice, Java Context and dependency Injection, ...

Using an EventAdapter between GUI and model (keep model clean, no event handling in model)

- Anemic control objects (most work in model) most updating of GUI using observer

Separating construction (even more), more factories

MP: MVC



Sample code for this in Monopoly 0.4 demo (on course page)

MP: Control Objects

Control objects are fuzzy

- Previously one control/use case (use case controllers)
- Also possible: Technical controllers, MenuControl

MP: Services

File handling for Spaces data

Basic step in MP

- Simple class to read rows (strings) from plain text file
- Later a Generic reader

Conversion step

- Using converter classes
- Later a Generic converter

The rule system

- ..tricky not implemented ...

Demo and Walkthrough for MP 0.4

Application has grown, not possible to grasp whole

Design Model for MP 0.4

After iteration 2, full MVC and a service

Interfacing with Other Paradigms

All the way we have been using the OO paradigm

All software is not OO...!

- 3d graphics is built on an rendering pipeline (have to adapt MVC model)
- Relational databases are not OO they are sets!
- The web is not OO it's just strings

If using any above have to incorporate different paradigms in one application

- ...hard... (often have to put adapting layers (interfaces) in between)
- Not covered in course

Documenting Iteration 2

Design model has changed a lot, need to update SDD

Check the code

- Self-documenting?
- Comments

Summary

We started with a running version of model with a basic GUI

Relaying on design principles, best practices, refactoring, tools, TDD we

- Implement a full MVC model
- Expand model with helper classes
- Added a service

Should, from here, be able to quickly (and controlled) expand the application until criteria from RAD fulfilled

Hmm...


Our Disaster Recovery Plan
Goes Something Like This...
HELP!
HELP!