Knowledge Transfer Schema: Software Patterns

1 Plan-driven design approaches

Plan driven approaches to software development are a well-established means for transferring knowledge about how to structure and design systems

- Tend to put main emphasis upon following procedures, and making use of notations to create models of the system
- Work well for those classes of problem that lend themselves to being modelled using only a few viewpoints

Hence, plan driven forms usually work best with architectural styles such as call-and-return. As the variety of architectural styles increase, and so did their complexity, so there was a need to find other ways of formulating a design

2 Knowledge Schema

A schema might relate to organisation of system structure, behaviour, sequencing etc

Design patterns are one way in which solution-related knowledge schema can be codified.

3 Why are patterns useful?

Patterns create a shared language for communicating insight and experience about problems and their solutions. This lets us capture the body of knowledge which defines out understanding of good design solutions that meet the needs of users.

The primary focus is on creating a culture to document and support sound engineering architecture and design

4 Adoption of patterns for software

Can be used for detailed design within the object-oriented community. Reasons for this include:

- The difficulty in devising effective OO analysis and design methods, as the complexity of the OO model creates major complications for a plan-driven approach
- Since information hiding helps to make it possible for objects to be loosely coupled, objects offer a good source of reusable components, and hence an OO architectural style might be expected to encourage reuse of elements and structures
- Objects have attributed, and so lend themselves to being catalogued and included in libraries

5 Useful patterns

Useful patterns describe:

- A single kind of problem
- The context in which the problem occurs
- The solution in terms of software constructs
- Design steps or rules for constructing the solution
- The forces leading to the solution
- Evidence that the solution optimally resolves forces
- Details that are not allowed to vary, and those that are not
- At least one actual instance of use
- Evidence of generality across different instances

Useful patterns describe or refer to:

- Variants and subpatterns
- Other patterns that it replies upon
- Other problems that rely upon this pattern

Useful patterns also relates to other problems with similar contexts, problems or solutions

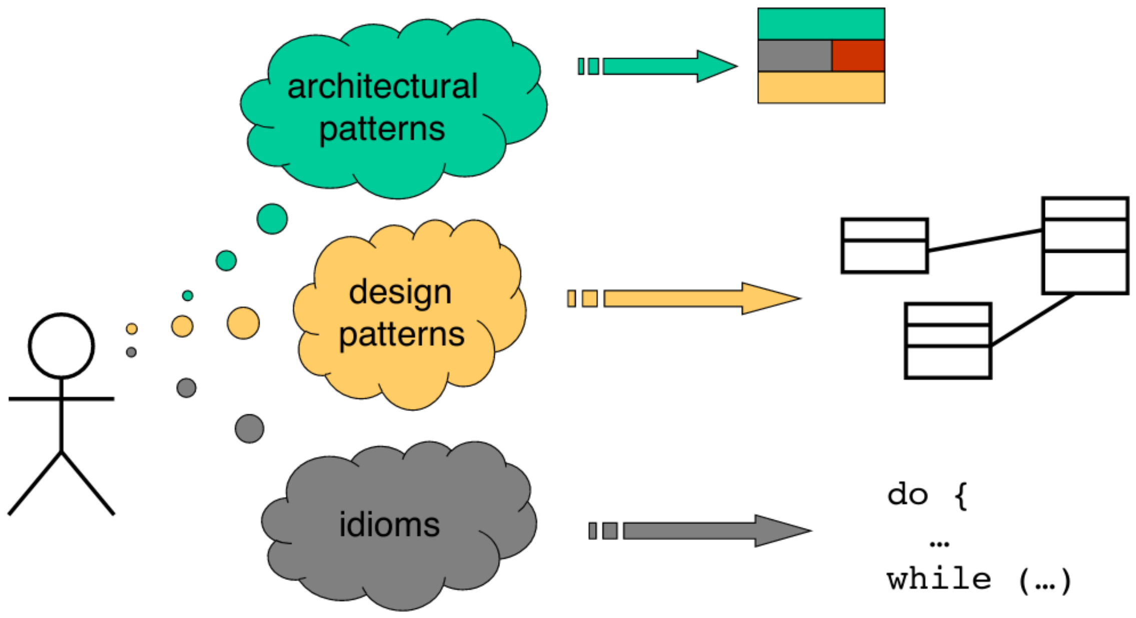
6 What is a pattern

Focus on reuse, but of concepts (schema) rather than code, in that it describes how part or all of a design solution is organised

- More concerned with composition than inheritance
- Represents the collective experiences of experienced designers
- Classified in terms of an abstract and recurring problem and the form of solution to adopt

7 Levels of abstraction for patterns

- **Architectural patterns** - provide templates that describe architectural forms. Some overlap with architectural style
- **Design patterns** - Concerned with the organisation of classes and objects
- **Idioms** - Language specific forms used in object oriented programming, and provide a low-level type of pattern



7.1 Architectural patterns

Concerned with the overall structural organisation of a software system, how any subsystems should be organised and what the relationships between these should be.

Some overlap with architectural style, but style refers to the form of the elements and their interactions, while a pattern is concerned with the way that they are organised to provide a ”solution”

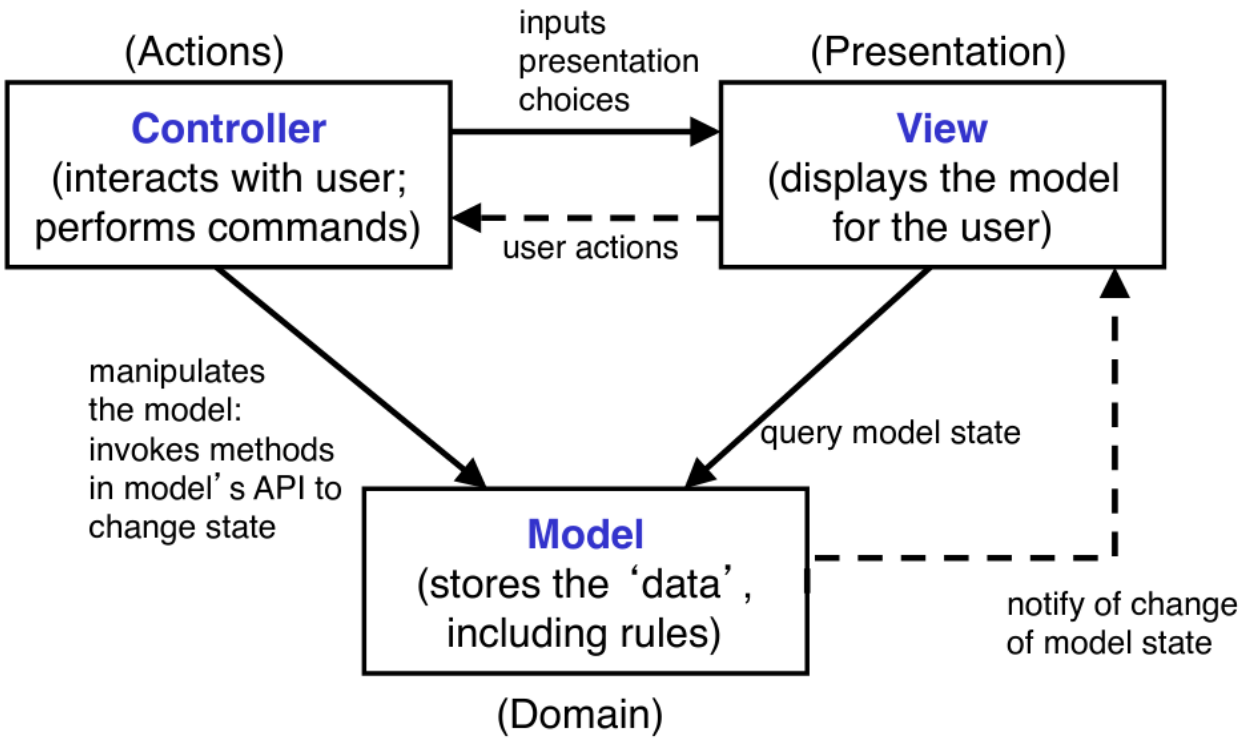
7.1.1 Model-View-Controller

This divides an interactive application into three elements:

- **Model** - contains the core functionality and data
- **Views** - display information to the user
- **Controllers** - Handle user input. Each view has an associated controller that also handles related inputs

The user interface then consists of views and controllers together, and is independent of the model. In turn, the model needs to propagate information about changes to controllers.

Structuring a system in this way makes it easy to change the interface


(Actions)
Controller
(interacts with user; performs commands)
inputs presentation choices
user actions
(Presentation)
View
(displays the model for the user)
manipulates the model: invokes methods in model's API to change state
query model state
Model
(stores the 'data', including rules)
notify of change of model state
(Domain)

8 Patterns vs Styles

Architectural patterns are concerned with:

- Defining the basic structure for an application
- Addressing a specific problem as well as the solution

Architectural styles are concerned with:

- Families of software systems in terms of their structural organisation
- The forms of, and the interactions between, components, connectors and the context within which these exist