Design Patterns

Massimo Felici

IF 3.46 0131 650 5899

mfelici@inf.ed.ac.uk

Reuse in Software Engineering

- Software Engineering is concerned with processes, techniques and tools which enable us to build "good" systems
- Object-Orientation is a methodology, technique, process, suite of design and programming languages and tools with which we may build good systems
- Components are units of reuse and replacement

Examples of Types of reuse

- **Application system reuse**: the whole of an application system may be reused by incorporating it without change into other systems
- **Component reuse**: components of an application ranging in size from sub-systems to single objects may be reused
- **Object and function reuse**: Software components that implement a single function, such as a mathematical function or an object class, may be reused

Problems with reuse

- Increased maintenance costs
- Lack of tool support
- Not-invented-here syndrome
- Creating and maintaining a component library
- Finding, understanding and adapting reusable components

Planning Reuse: Key Factors

- The development schedule for the software
- The expected software lifetime
- The background, skills and experience of the development team
- The criticality of the software and its non-functional requirements
- The application domain
- The platform on which the system will run

Types of Reuse

- Knowledge reuse
  - Artificial reuse
  - Pattern reuse
- Software reuse
  - Code reuse
  - Inheritance reuse
  - Template reuse
  - Components
  - Framework reuse

Reuse of Knowledge: Artifact Reuse

- Reuse of use cases, standards, design guidelines, domain-specific knowledge
- **Pluses**: consistency between projects, reduced management burden, global comparators of quality and knowledge
- **Minuses**: overheads, constraints on innovation (coder versus manager)

Reuse of Knowledge: Patterns

- A design pattern is a solution to a common problem in the design of computer systems
- Reuse of publicly documented approaches to solving problems (e.g., class diagrams)
- **Plusses**: long life-span, applicable beyond current programming languages, applicable beyond Object Orientation?
- **Minuses**: no immediate solution, no actual code, knowledge hard to capture/reuse.

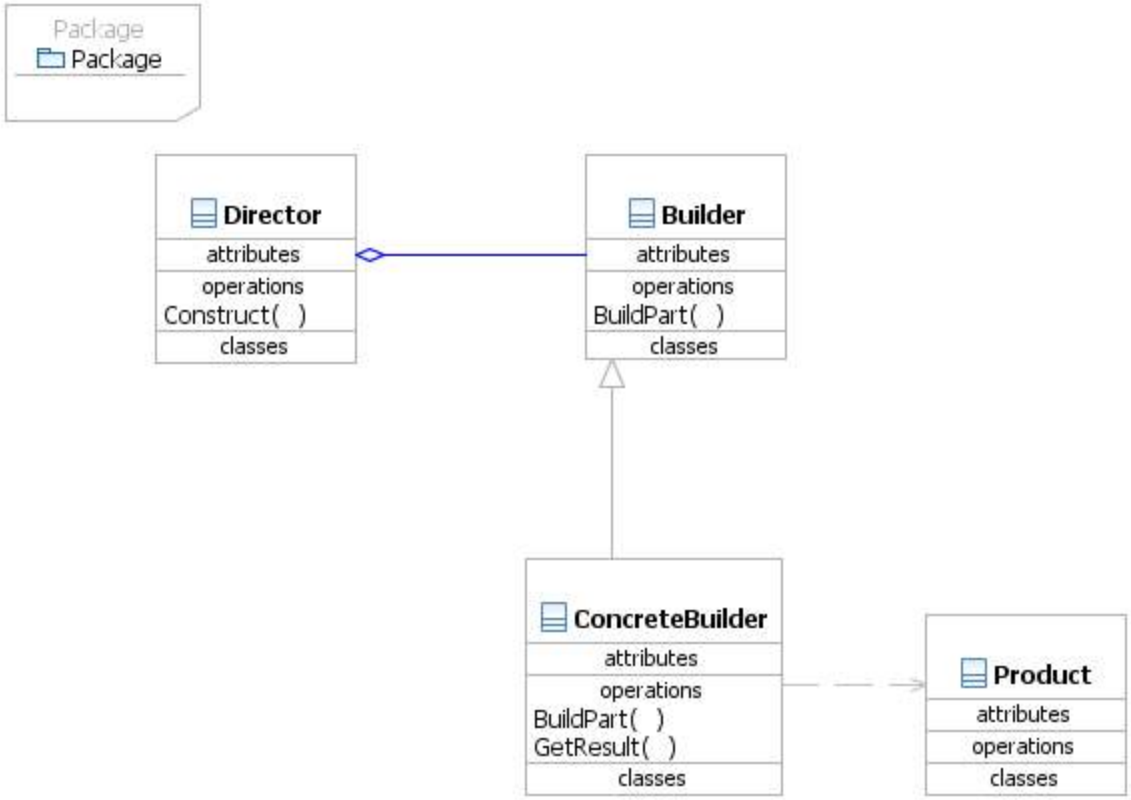
Documenting Patters

- Name
- Problem
- Context
- Forces
- Solution
- Sketch
- Resulting context
- Rationale

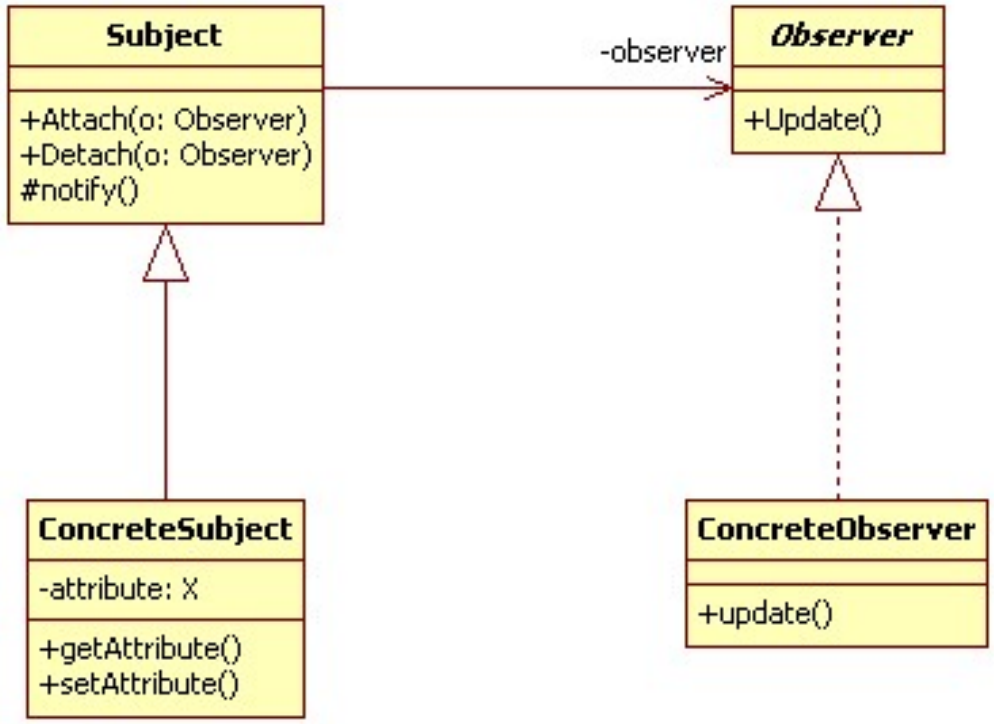
Classification of UML Patterns

- Creational
- Structural
- Behavioural

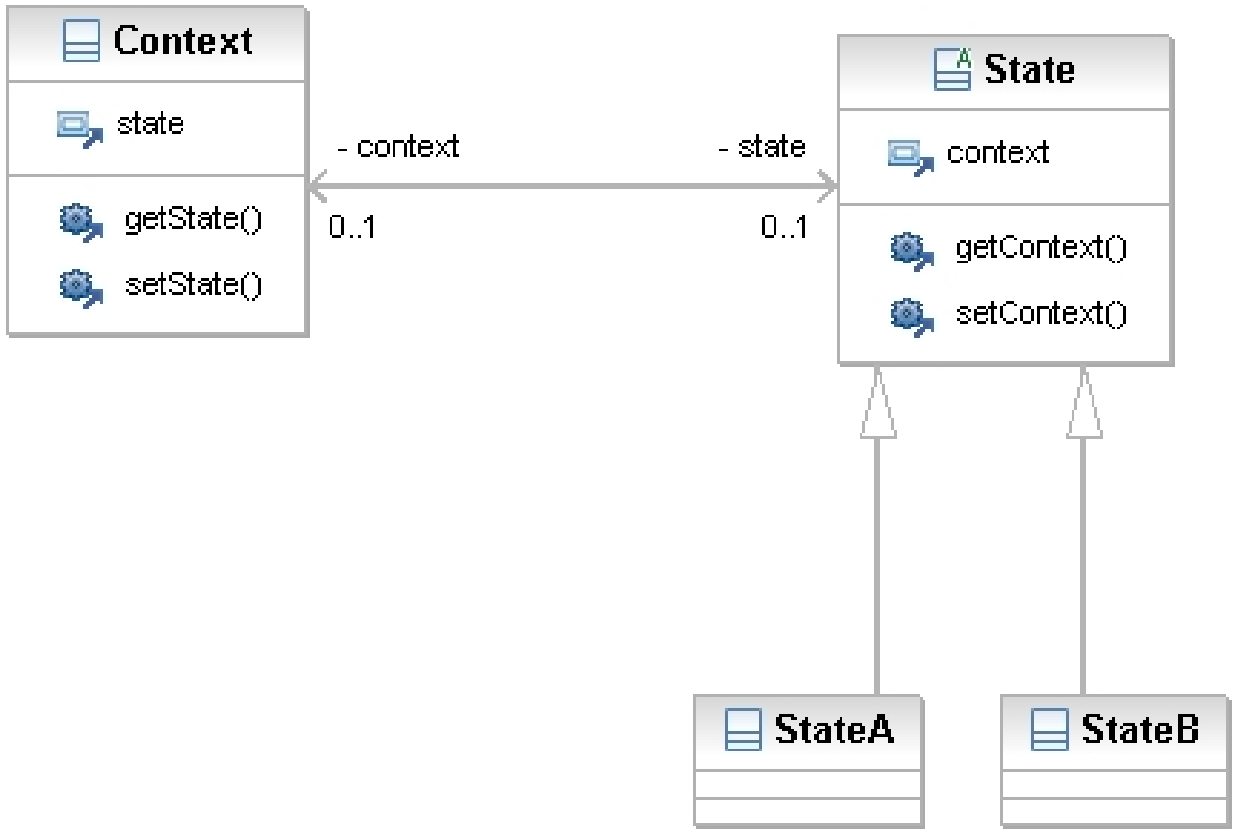
Example: Builder Pattern



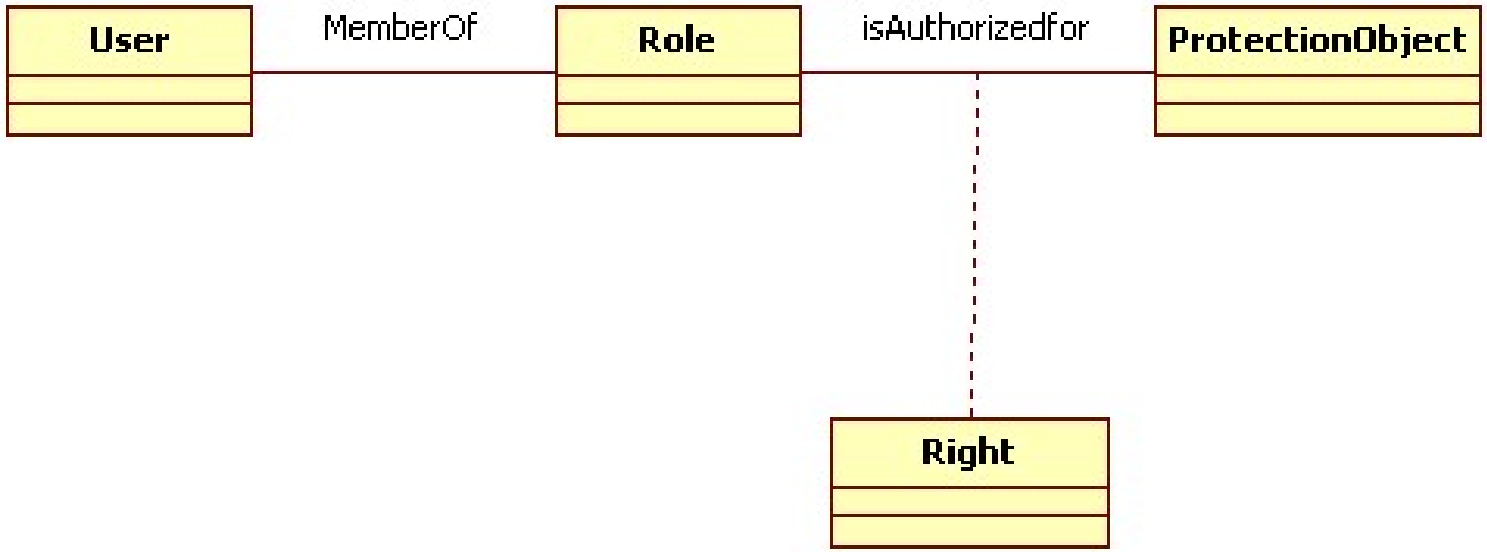
Example: Observer



Example: State



Example: Role-Based Access Control



How to use a pattern

- Does a patter exist that address the considered problem?
- Does the pattern's documentation suggest alternative solutions?
- Is there a simple solution?
- Is the context of the pattern consistent with the context of the problem?
- Are the results of using the pattern acceptable?
- Are there constraints?

Types of Software Reuse: Code Reuse

- Reuse of (visible) source code – code reuse versus code salvage
- **Pluses**: reduces written code, reduces development and maintenance costs
- **Minuses**: can increase coupling, substantial initial investment

Types of Software Reuse: Inheritance

- Using inheritance to reuse code behaviour
- **Pluses**: takes advantage of existing behaviour, decrease development time and cost
- **Minuses**: can conflict with component reuse, can lead to fragile class hierarchy – difficult to maintain and enhance

Types of Software Reuse: Template Reuse

- Reuse of common data format/layout (e.g., document templates, web-page templates, etc.)
- **Pluses**: increase consistency and quality, decrease data entry time
- **Minuses**: needs to be simple, easy to use, consistent among groups

Types of Software Reuse: Component

- Analogy to electronic circuits: software "plug-ins"
- Reuse of prebuilt, fully encapsulated "components"; typically self-sufficient and provide only one concept (high cohesion)
- **Pluses**: greater scope for reuse, common platforms (e.g., JVM) more widespread, third party component development
- **Minuses**: development time, genericity, need large libraries to be useful

Types of Software Reuse: Framework

- Collection of basic functionality of common technical or business domain (generic "circuit boards") for components
- **Pluses**: can account for 80% of code
- **Minuses**: substantial complexity, leading to long learning process, platform specific, framework compatibility issues leading to vendor specificity, implement easy 80%

Reuse Pitfalls

- **Underestimating** the difficulty of reuse
- Having or setting **unrealistic** expectations
- **Not investing** in reuse
- Being **too focused** on code reuse
- Generalising after the fact
- Allowing too many connections

Difficulties with Component Development

- **Economic**
  - Small business do not have long term stability and freedom required
- Where is the third party component **market**?
  - Effort in (re)using components
  - Cross-platform and cross-vendor compatibility
  - Many common concepts, few common components
  - Some success: user interfaces, data management, thread management, data sharing between applications
  - Most successful: GUIs and data handling (e.g., Abstract Data Types)

Readings

- **UML course textbook**
  - Chapter 17 on Design Patterns
- T. Winn, P. Calder, Is This a Pattern?. In IEEE Software, January/February 2002.

Summary

- Many types of reuse – of both knowledge and software
  - Each has pluses and minuses
- Component reuse is a form of software reuse
  - Encapsulation, high cohesion, specified interfaces explicit context dependencies
  - Component development requires significant time and effort
  - As does component reuse
  - Component reuse has been successful for interfaces and data handling
- Employing reuse requires management