Automatically Discovering Design Patterns and Assessing Concern Separations for Applications

Giuseppe Pappalardo, **Emiliano Tramontana**

Dipartimento di Matematica e Informatica

Università di Catania

Italy

Motivations

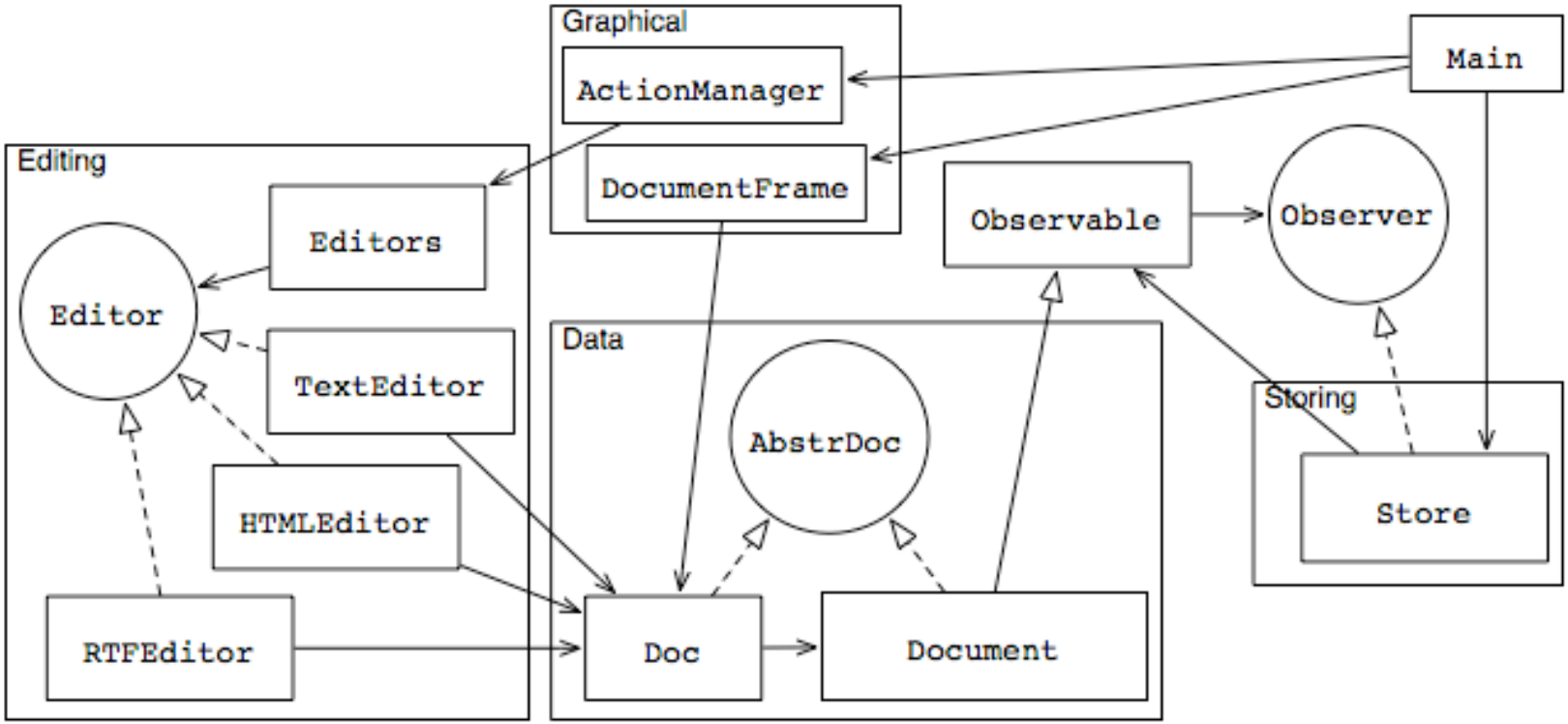
- Supporting the development of systems featuring a high separation of concerns
- By providing developers feedback about the overall design they are dealing with
  - Suggesting changes for the prototypical code to achieve better modularity
  - Helping to reduce the effort needed to recognise the structure of legacy systems and introduce change

Analysing Applications

- The realised toolset, called PaCo (Pattern Concern analyser)
  - Examines application code and determines the class structure
  - Recognises used design patterns
    - Matching from an existing catalogue
  - Highlights similarities between application classes and design patterns roles
  - Reveals concerns, i.e. set of classes addressing a common goal
    - Concerns are defined by the user
  - Detects dependencies between concerns

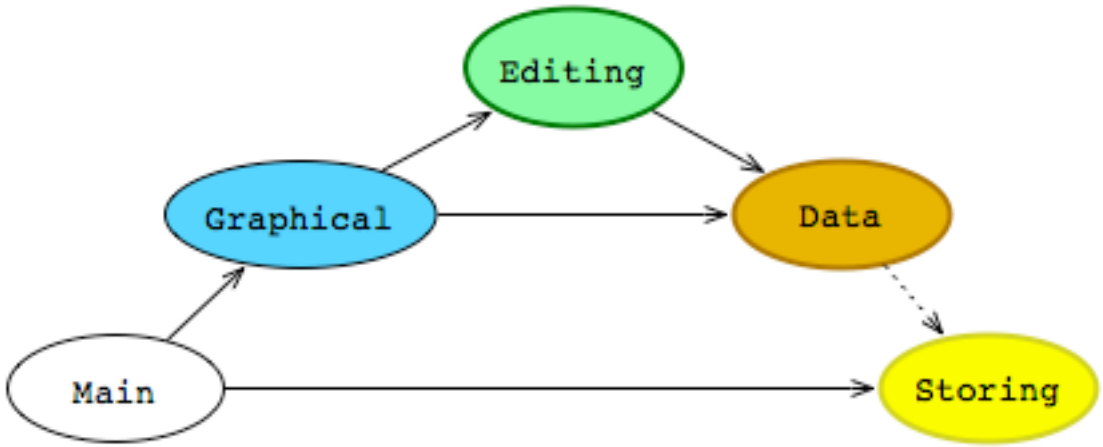
PaCo Class View

- PaCo provides the *class view*, presenting the application class diagram following the UML standard
  - Concern boundaries among classes are marked



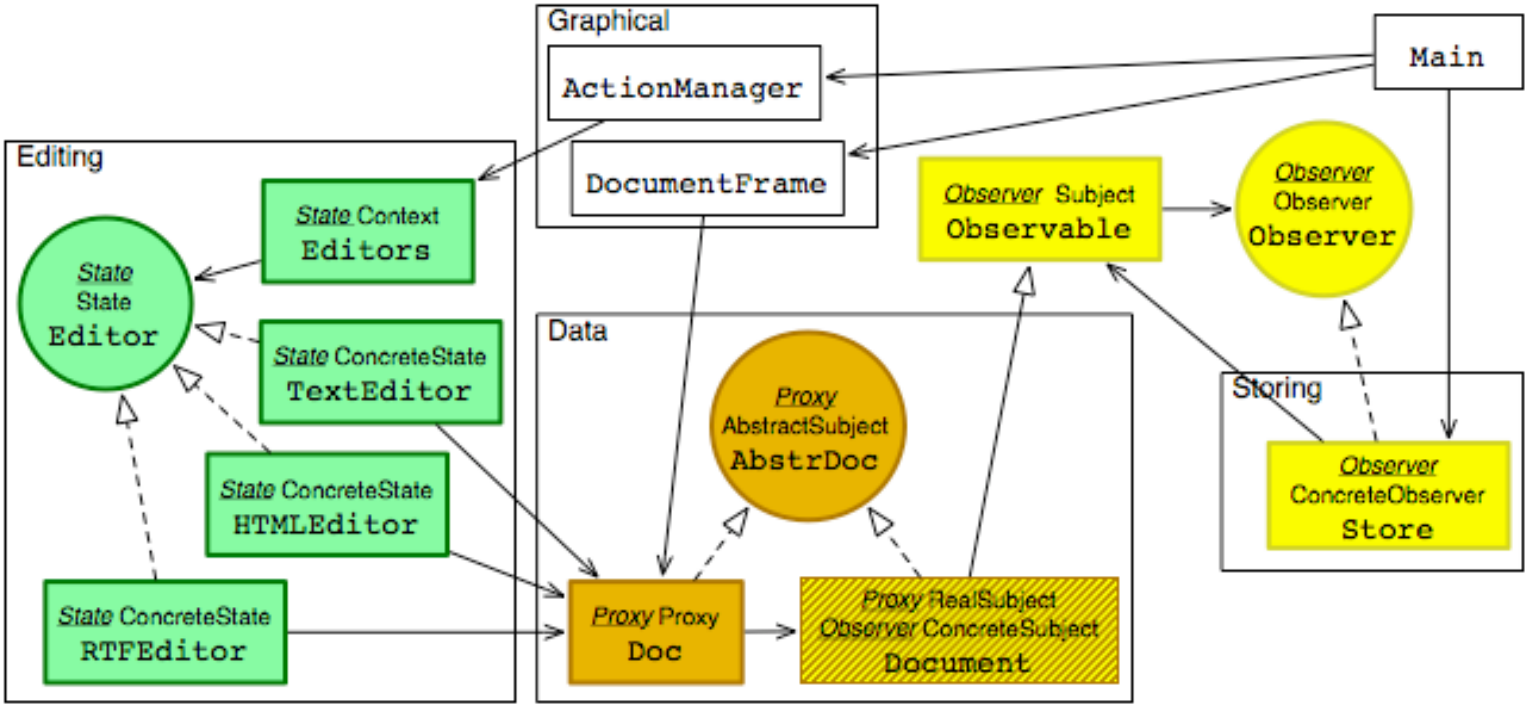
PaCo Concern View

- The *concern view* presents concerns as nodes of a graph and dependencies as straight or dotted edges
  - Straight edges indicate strong dependencies
    - Due to an invocation between classes of the concerns
  - Dotted edges indicate weak dependencies
    - Occur when classes are coupled according to their role within a pattern
    - E.g. `ConcreteSubject` and `ConcreteObserver` with the Observer pattern



PaCo Pattern Concern View

- As the class-view enriched with identified patterns
  - Each application class is mapped to a known pattern, if possible
  - Every class box in the UML representation, appears with the pattern name and the pattern role played
- Helps in identifying each class role and responsibility
- Supports the characterisation of dependencies between concerns



Patterns-Concerns Interplay

- Patterns are about schema of interaction among entities
- Concerns have to do with the semantics of those entities
- Hence, it should not be expected that patterns and concerns correspond 1:1 with each other
  - The same class may play several roles simultaneously for different design patterns
  - The classes may lie outside defined concerns
  - A set of classes interacting according to a design pattern do not necessarily belong to the same concern
  - Interaction within a concern does not necessarily adhere to a pattern
  - Concerns do not necessarily interact through a design pattern

Types of Investigations

- Mapping application classes to different pattern catalogues can give hints on
  - Modularity [Gamma]
  - Performance bottlenecks [Bruce]
  - Concurrency schemes [Lea]
  - Real-time issues [Douglass]
  - Potentially harmful solutions, i.e. antipatterns [Bruce, Smith]

Mining Data from Java Applications

- PaCo inspects Java bytecode to extract for each class
  - Methods names
  - Invoked methods and classes these belong to
  - Superclass
  - Implemented interfaces
- Inspection capabilities are made available through the Java and Javassist reflective libraries
- These extracted data are sufficient to automatically build the UML class diagram of an application

Design Patterns Repository

- Each design pattern is identified by the list of names of its constituting classes (or roles), and for each of these
  - A list of its methods' names
  - A list of the classes whose methods are called
  - Its superclass
  - Implemented interfaces
- These data give a labelled directed graph
- Data extracted from applications and data representing a design pattern have the same structure
  - The pattern repository is populated by inspecting an implementation of a design pattern
  - Revealing design patterns within an application is a matter of detecting given subgraphs within a larger graph

Revealing Design Patterns

- Comparing two graphs is known as *graph matching* (or querying)
- GraphGrep is the querying tool we have used
  - Only undirected graphs are dealt with
    - Source and sink nodes cannot be distinguished
    - I.e. whether `A` is subclass of `B` or vice versa is not considered
  - Edges are not labelled
    - Whether a relationship between classes is invocation, implementation or subclassing cannot be discriminated
  - Node labels cannot be partially matched
    - Unable to distinguish between an interface or a class

Post-Processing

- As a consequence of the said limitations
  - Matching subgraphs can return false positives
  - The false positive have the same structure, we were looking for, except for edge direction
- Simple post-processing is performed to filter out the false positives

Conclusions

- The proposed PaCo toolset provides
  - Useful assistance in identifying application concerns and mutual relationships
  - Support in understanding code of complex systems
    - Classes are labelled with their role
    - The concern view gives a high level representation of an application
  - Hints on how the code can be improved, for the sake of performance tuning, modularity or bug fixing