2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

UVM Verification Environment Based on Software Design Patterns

Darko M. Tomušilović Hagai Arbel

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Today in your local cinema... Design Patterns

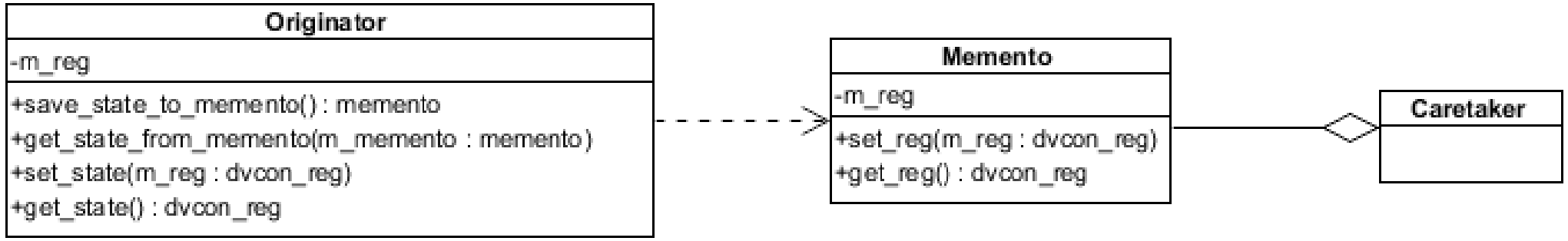
- What are design patterns?
- What types of design patterns are there?
- What are their benefits?
- Can they help verification engineers?
- What is UML?

Suggestion 1: Memento

Memento

- Capture the state of an object and restore the object into its previous state
- Promotes data hiding, not violating encapsulation
- “Undo” application
- Example from the Verification world: Multiple power domains modelling – save configuration register content upon LPM entry and restore it upon LPM exit

Memento – UML diagram

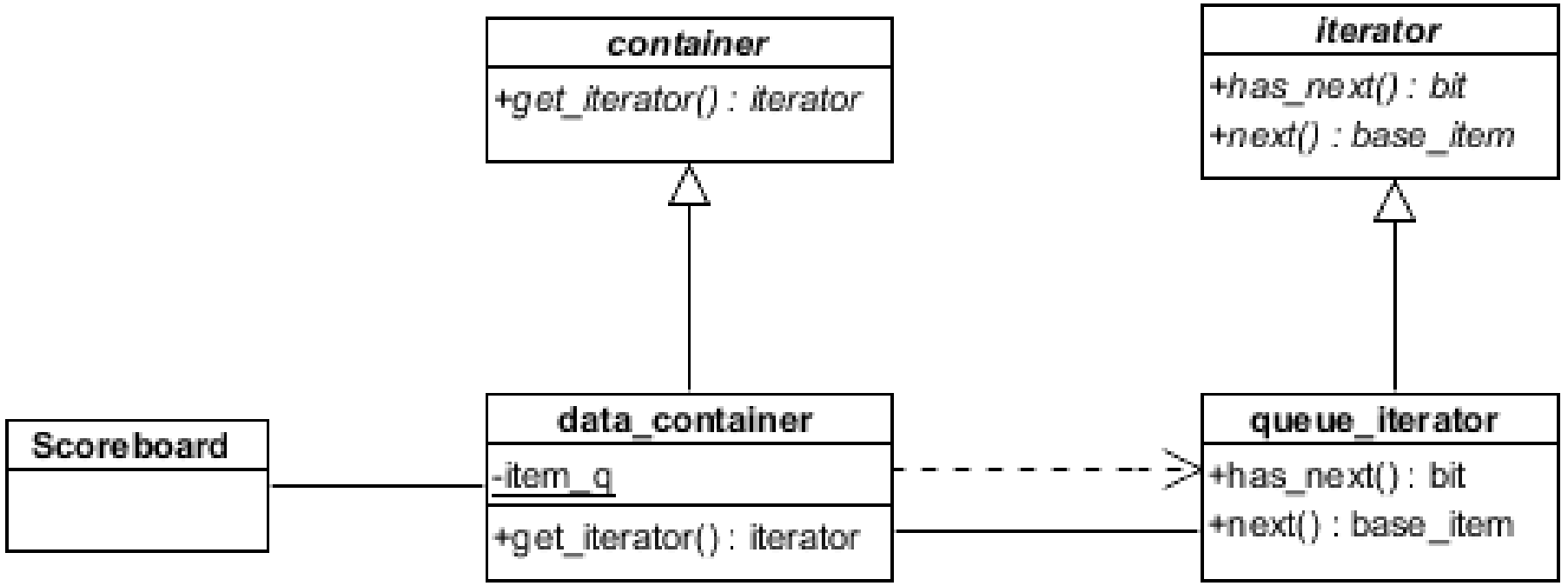


Suggestion 2: Iterator

Iterator

- Traverse a collection of objects, regardless of its internal structure
- The internal structure storing the data is not exposed and can therefore be modified without affecting the rest of the environment
- Improves flexibility
- Example from the Verification world: Add data items to the container in the reference model and upon ECC enable and disable, iterate the container and perform updates to achieve correct prediction

Iterator – UML diagram



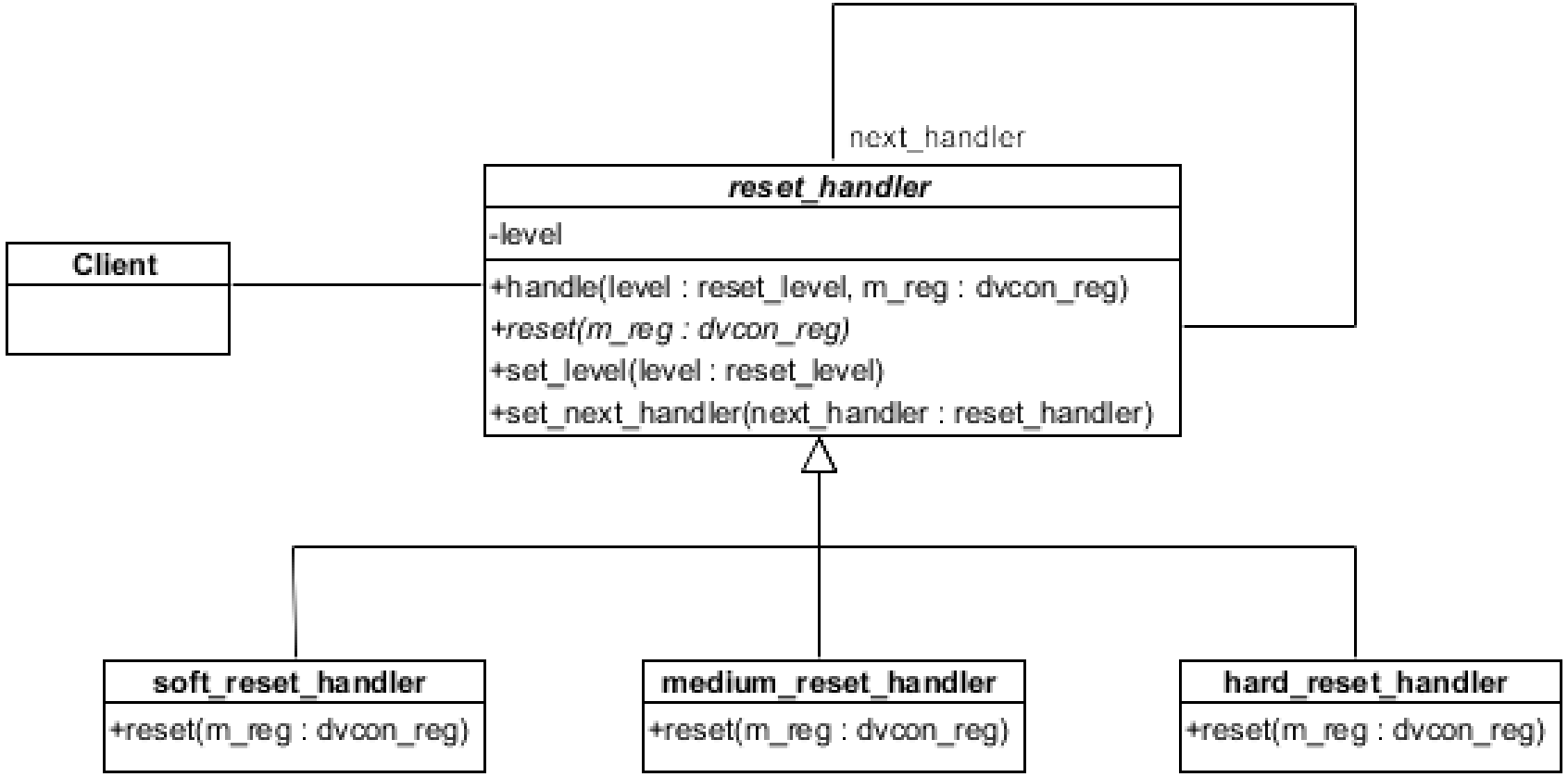
Suggestion 3: Chain of Responsibility

2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

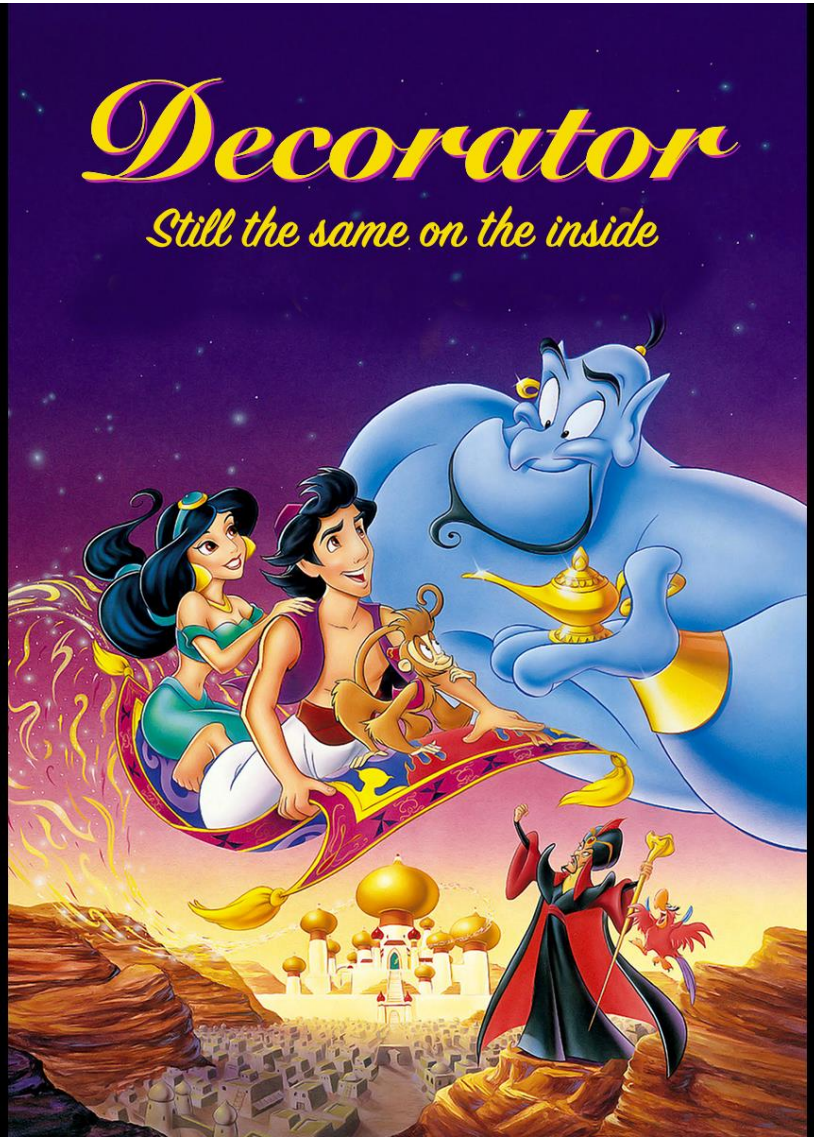
Chain of Responsibility

- Handle an action or command by multiple receivers
- The request is successively propagated from one receiver to another, giving more receivers the chance to handle the request
- Promotes decoupling between the handlers
- Example from the Verification world: Multiple reset levels modelling – update register model taking into the consideration which register fields are controlled by which reset level

Chain of Responsibility – UML diagram



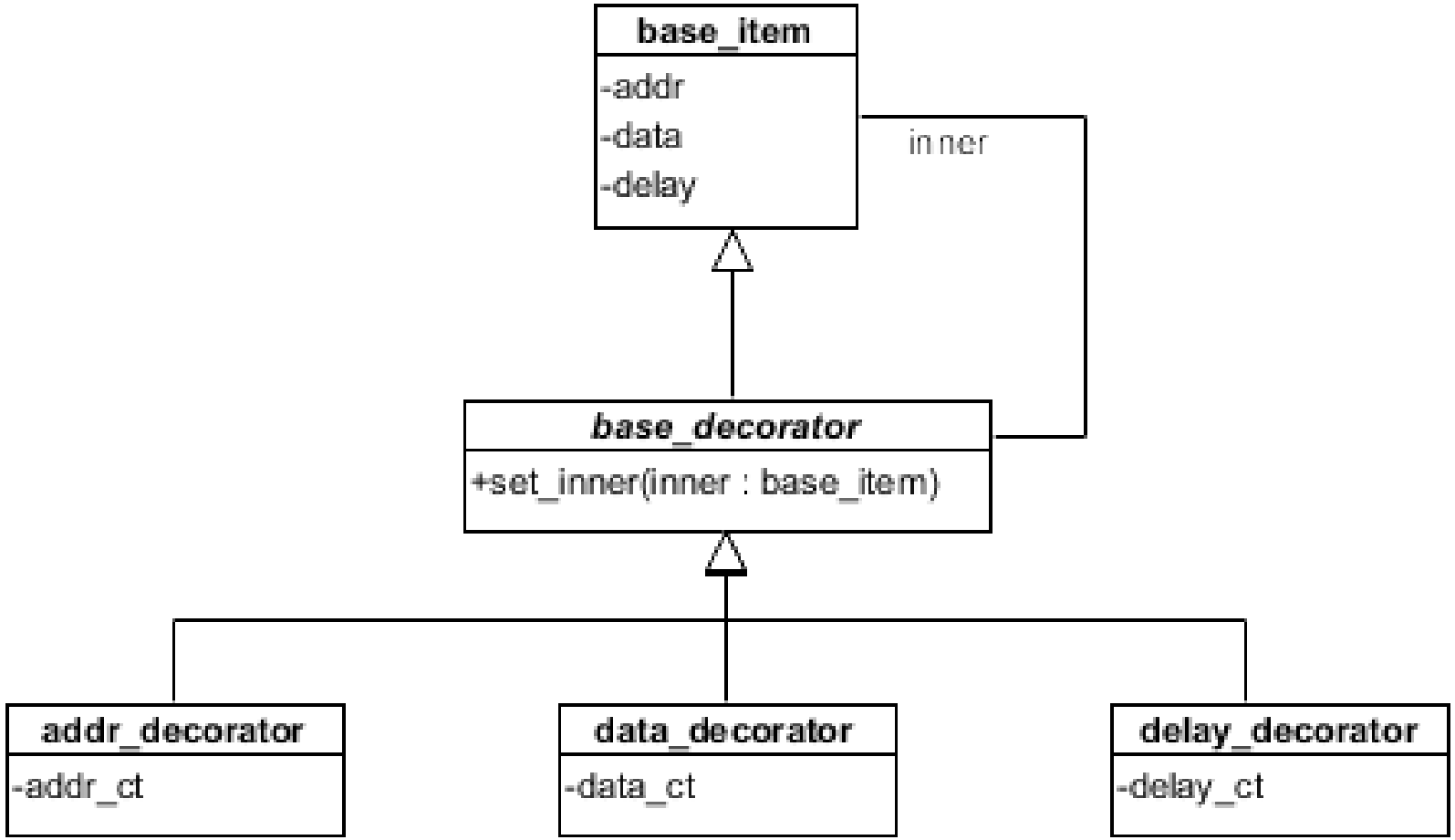
Suggestion 4: Decorator



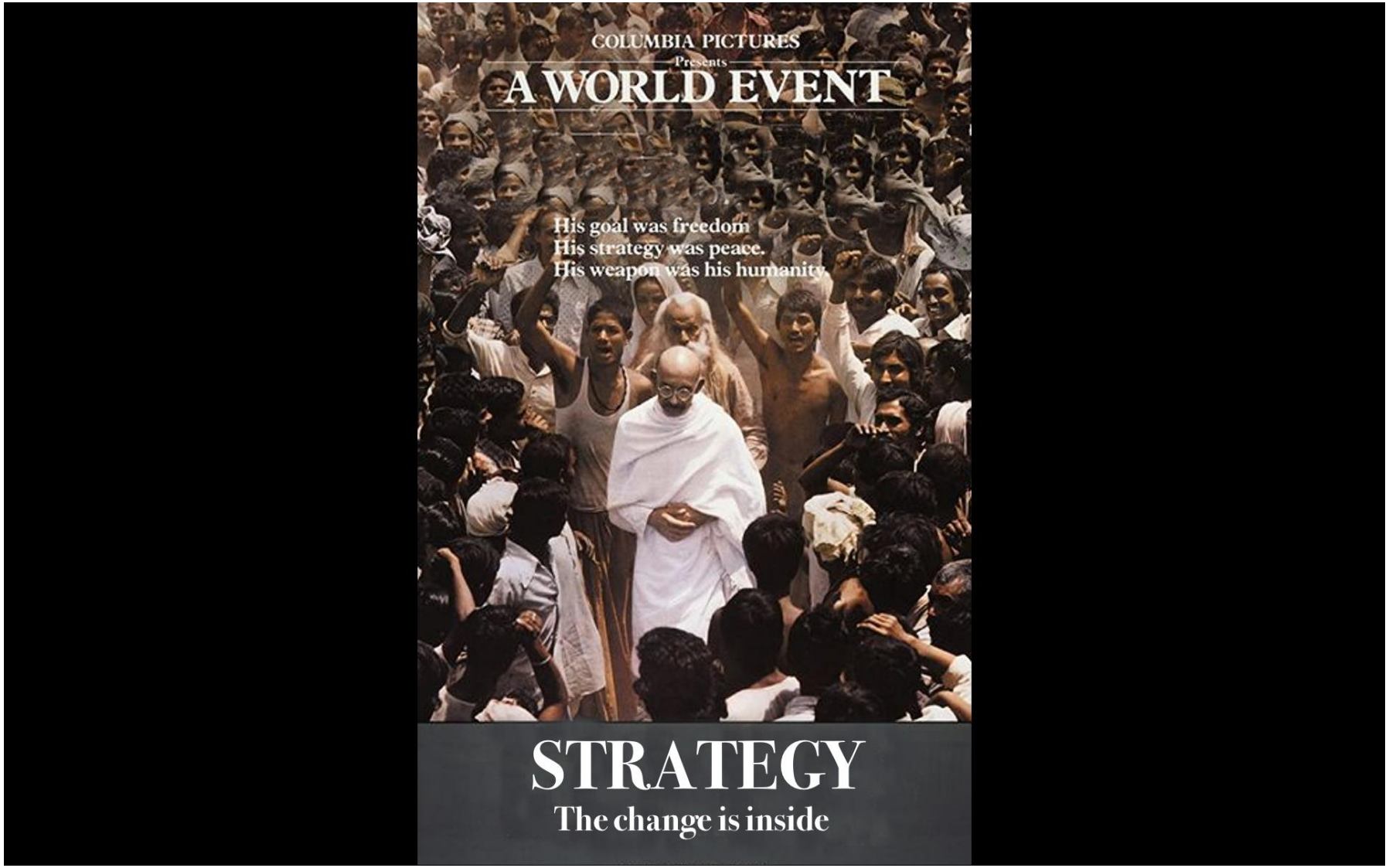
Decorator

- Modify a class object, by adding behaviour to it – without affecting other objects of the same class
- Base class code future-proof for unforeseen changes
- Dynamic addition of behaviour to objects is achievable
- Multiple decorators can be added simultaneously
- Example from the Verification world: Modelling complex data items by applying additional set of constraints to them

Decorator – UML diagram



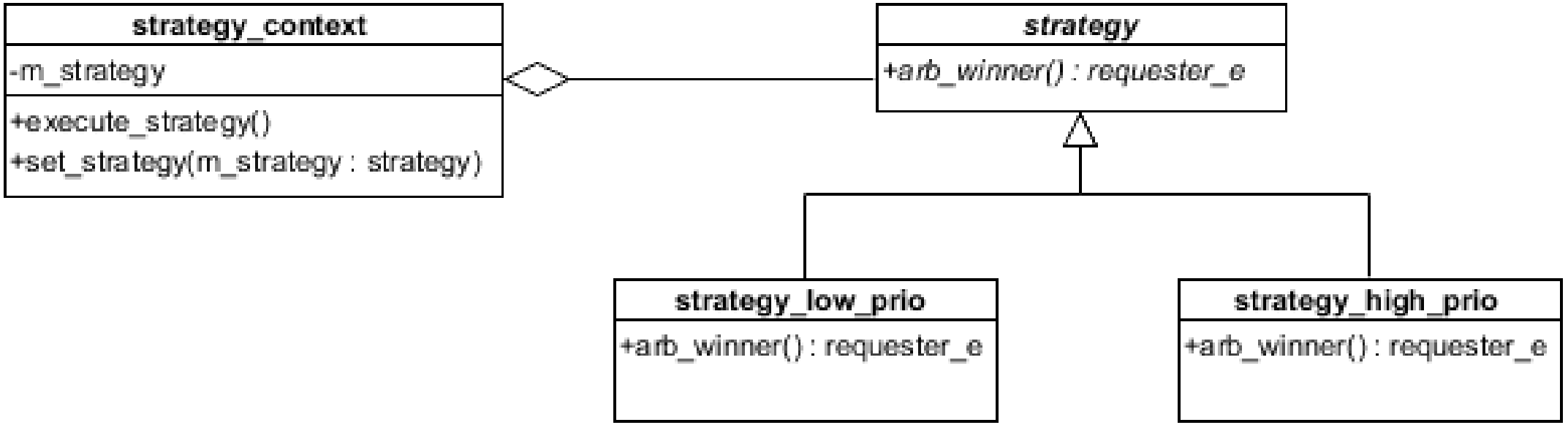
Suggestion 5: Strategy



Strategy

- Apply a certain algorithm at run-time out of family of provided algorithms
- Targets the change in the core functionality
- Wrapping each algorithm into a separate class improves code readability and extensibility
- Straightforward to add a new algorithm
- Example from the Verification world: Modelling dynamically configurable arbitration logic upon the memory access, in which the priority is determined using several algorithms (round robin, fixed priority, …)

Strategy – UML diagram

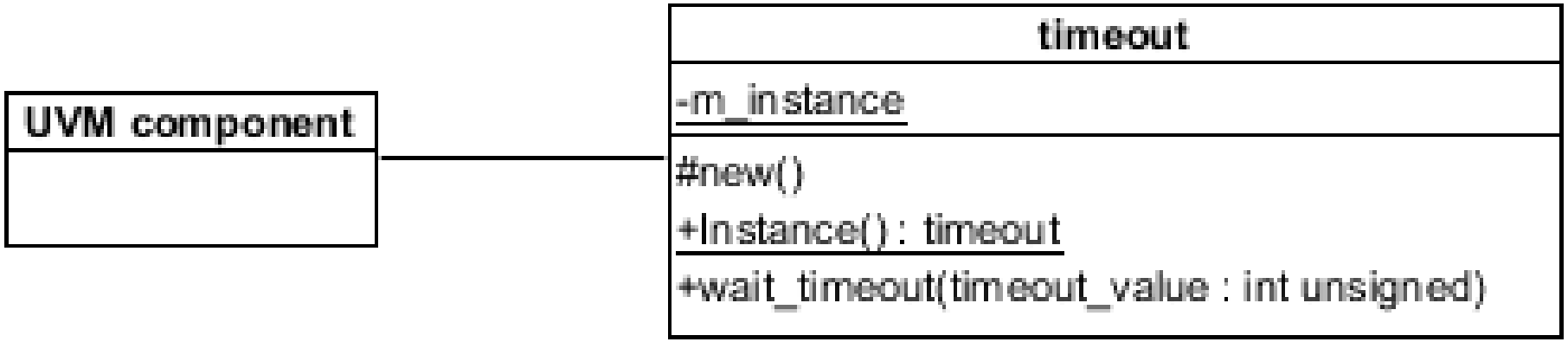


Suggestion 6: Singleton

Singleton

- Restrict the number of class objects that can be instantiated and provide global access to them
- Facilitates debugging
- Example from the Verification world: Timeout logic handling class – assure that all components in the environment that monitor for a timeout event access the same object to detect the timeout expiration

Singleton – UML diagram

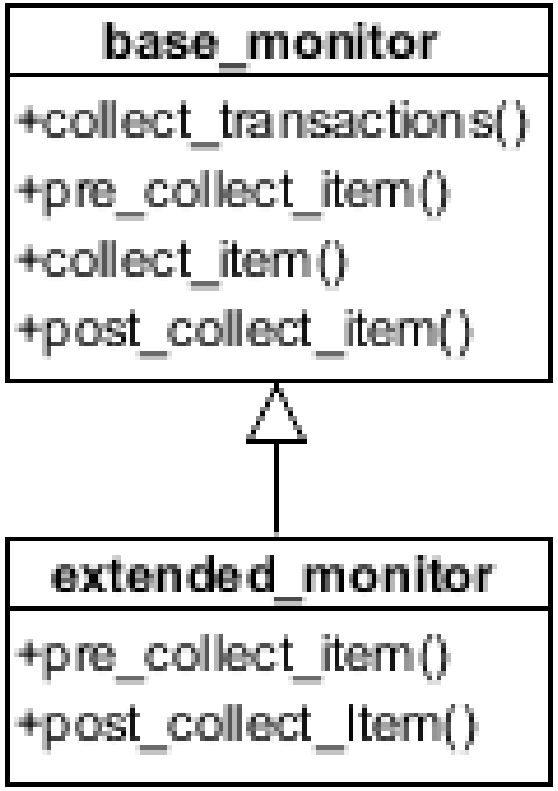


Suggestion 7: Template method

Template method

- Define a set of operations to be performed in order
- Leave the implementation of some steps to the derived classes while maintaining the overall algorithm structure
- Provide pre-processing and post-processing hooks to a main operation
- Example from the Verification world: Extend main monitor operation, utilizing hooks to perform project-specific checking, without changing the existing codebase

Template method – UML diagram



2018 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Other suggestions

- Factory
- Observer
- State and Mediator
- Visitor

Questions?

Thanks!