Trusted Components
Reuse, Contracts and Patterns

Prof. Dr. Bertrand Meyer
Dr. Karine Arnout
Lecture 7: Pattern Wizard, project presentation
Agenda for today

- Pattern Wizard
- Project presentation
Componentizability classification

1. Componentizable
   - 1.1 Built-in
     - Prototype
   - 1.2 Library-supported
     - Template
   - 1.3 Newly componentized
     - Flyweight
     - Observer
     - Mediator
     - Abstract Factory
     - Factory Method
     - Visitor
     - Command
     - Composite
     - Chain of Responsibility
   - 1.4 Possible component

2. Non-componentizable
   - 2.1 Possible skeleton
     - Singleton
     - Iterator
     - Facade
     - Interpreter
   - 2.2 Some library support
   - 2.3 Design idea

Design pattern

1.3 Componentizable but not comprehensive
   - 1.3.1 Fully componentizable
   - 1.3.2 Componentizable but not comprehensive
     - Builder
     - Proxy
     - State
   - 1.3.3 Componentizable but unfaithful
     - Strategy
   - 1.3.4 Componentizable but useless
     - Memento

2.1 Method
   - Decorator
   - Adapter
   - Memento

2.1.1 Method
   - Iterator

2.1.2 No method
   - Singleton
   - Proxy
   - Factory Method
   - Abstract Factory
   - Flyweight

2.2.2 Possible skeleton
   - Singleton
   - Proxy
   - Factory Method
   - Abstract Factory
   - Flyweight
Why an automatic code generation tool?

- Interesting for:
  - Non-componentizable patterns → Supported by the Pattern Wizard
  - Componentizable but not comprehensive
  - Fully componentizable patterns that cannot be applied to a given problem (e.g. performance) → Will be supported in the next version
Pattern Wizard

- Applicable to non-componentizable patterns
- Automatically generates skeleton classes

*Eiffel Pattern Wizard, version 1.0*

- **Non-componentizable**
  - **Skeleton**
    - With method
      - Decorator
    - Adapter
      - class adapter
      - object adapter
  - No method
    - Template method
      - original pattern
      - with agents
    - Bridge
      - with deferred classes (original pattern)
      - with effective classes (like in EiffelVision2)
      - with non-conforming inheritance
- Possible skeleton
  - Singleton

- Generate root class and Ace file
- Close the Eiffel Pattern Wizard after code generation

→ Generated code
Overall architecture

**Pattern Wizard**

- * PW_INITIAL_WINDOW_IMP
- + PW_INITIAL_WINDOW
- * PW_PATTERN_VERTICAL_BOX
- + PW_DECORATOR_VERTICAL_BOX

**Model**

- + PW_PATTERN_INFORMATION
- * PW_DECORATOR_INFORMATION
- * PW_PATTERN_CODE_GENERATOR
- + PW_DECORATOR_CODE_GENERATOR

**GUI**

- gui

**Support**

- PW_SKELETON_NAMES

**Generation**
Class `PW_DECORATOR_INFORMATION`
deferred class
  PW_INITIAL_WINDOW_IMP
...
feature {NONE} -- Implementation (Pattern information)
  decorator_info: PW_DECORATOR_INFORMATION is
    -- Selected information about the chosen pattern
    require
      decorator_pattern_vbox_not_void: decorator_pattern_vbox /= Void
    local
      frame: PW_DECORATORPROPERTY_SELECTOR
    do
      create Result
      frame := decorator_pattern_vbox.pattern_properties_frame
      Result.set_component_class_name (frame.component_class_name)
      Result.set_feature_name (...)
      ...
      if frame.is_component_with_additional_attribute_generation then
        Result.set_component_with_additional_attribute_generation (True)
      ...
      end
      if frame.is_component_with_additional_behavior_generation then
        ...
      end
    ensure
      decorator_info_not_void: Result /= Void
  end
end
deferred class
   PW_PATTERN_CODE_GENERATOR
feature -- Access
   pattern_info: PW_PATTERN_INFORMATION
      -- Pattern information needed for the code generation
      -- (name of classes, name of features, etc.)
   project_directory: STRING
      -- Path of the project directory (where the code will be generated)
feature -- Status report
   root_class_and_ace_file_generation: BOOLEAN
      -- Should a root class and an Ace file be generated?
feature -- Element change
   set_pattern_info (a_pattern_info: like pattern_info) is ...
   set_project_directory (a_project_directory: like project_directory) is ...
   set_root_class_and_ace_file_generation (a_value: like root_class_and_ace_file_generation) is ...
feature -- Generation
   generate
      -- Generate code for this pattern.
         require
            pattern_info_not_void: pattern_info /= Void
            pattern_info_complete: pattern_info.is_complete
         invariant
            project_directory_not_empty_and_exists_if_not_void: project_directory /= Void implies
               (not project_directory.is_empty and directory_exists (project_directory))
   end
deferred class

<DECORATOR_COMPONENT_CLASS_NAME>

feature -- Basic Operation

<DECORATOR_FEATURE_NAME> is

-- Do something.

defered

defered

end

deferred

end
deferred class
    PW_PATTERN_CODE_GENERATOR
...
feature -- Generation
    generate is
      -- Generate code for this pattern.
      require
        pattern_info_not_void: pattern_info /= Void
        pattern_info_complete: pattern_info.is_complete
      do
        if root_class_and_ace_file_generation then
          generate_ace_file
          generate_root_class
          end
        end
      end
    generate_pattern_code
...
end
deferred class interface
   PW_PATTERN_CODE_GENERATOR

... feature {NONE} -- Implementation (Code generation)
   generate_code (a_new_file_name, a_skeleton_file_name: STRING;
   some_changes: LINKED_LIST [TUPLE [STRING, STRING]]
   -- Generate new file with file name a_new_file_name from the skeleton
   -- corresponding to a_skeleton_file_name by reproducing the skeleton
   -- code into the new file after some_changes, replacing a value by
   -- another. (some_changes should be of the form:
   -- LINKED_LIST [[old_string, new_string], ...]
   require
   a_new_file_name_not_void: a_new_file_name /= Void
   a_new_file_name_not_empty: not a_new_file_name.is_empty
   a_skeleton_file_name_not_void: a_skeleton_file_name /= Void
   a_skeleton_file_name_not_empty: not a_skeleton_file_name.is_empty
   a_skeleton_file_exists: file_exists (a_skeleton_file_name)
   some_changes_not_void: some_changes /= Void
   no_void_change: not some_changes.has (Void)
   -- no_void_old_string: forall c in some_changes, c.item (1) /= Void
   -- no_void_new_string: forall c in some_changes, c.item (2) /= Void

... end
Related work

- Existing tool by Budinsky, Finnie, Yu, and Vlissides [Budinsky 1996]
- Goal: fast turn-around, flexibility
- 3-part architecture:
  - “Presenter”: HTML browser for user interaction; transmits input as Perl scripts
  - “Mapper”: Perl interpreter
  - COGENT (Code GENeration Template) interpreter: Code generator

- The Pattern Wizard brings a fully O-O design and implementation using Eiffel. (No such tool in Eiffel before.)
Future work

- Enable choosing the root class name and creation procedure name
- Enable using existing files and add to them the pattern code
- Support other patterns

- Possibly support other languages
Agenda for today

- Pattern Wizard
- Project presentation
Grading

- **Exam (2h):** 40%
  - 2 February 2005

- **Project:** 60%
  - Deadline: 4 February 2005
  - Extension of a “Pattern Wizard”
The project

- **Pattern Wizard**
  - Design + implementation in Eiffel
  - GUI + Business model

- **Documentation**
  - User guide: how to use the tool
  - Developer guide: description of the architecture, main classes, limitations, how to extend the tool

- **Test suite**
  - Thorough set of test cases
What you need to do

- Enable choosing the root class name and creation procedure name

- Enable using existing files and add to them the pattern code

- Support two componentizable patterns:
  - Abstract Factory
  - Visitor
Grading criteria

- **Design (30 points)**
  - Soundness (5 points)
  - Extendibility (5 points)
  - Ease of use (5 points)
  - Minimal requirements (15 points)
- **Quality of contracts (20 points)**
- **Documentation (20 points)**
  - User guide (10 points)
  - Developer guide (10 points)
- **Test (10 points)**
  - Quality of test suite (5 points)
  - Correctness of the tool (5 points)
- **Quality of code (10 points)**
  - Style guidelines (5 points)
  - Quality of code (5 points)
- **Effort devoted to the project (10 points)**
Complementary material

- From Patterns to Components:
  - Chapter 21: Pattern Wizard

- Further reading:


End of lecture 7