The UML Profile for Framework Architectures

Marcus Fontoura
CS Department
Princeton Univ.
U.S.A.
mfontoura@acm.org

Wolfgang Pree
Software Research Lab
Univ. of Constance
Germany
pree@acm.org

Bernhard Rumpe
Software Engineering
Munich Tech. University
Germany
rumpe@acm.org

© 2000, M. Fontoura, W. Pree, B. Rumpe

Context (I)

- **UML-F Foundations**
  - **Terminology: frameworks & components**
  - **Essential framework patterns**
  - **Goals of UML-F**
- **The UML-F Profile**
  - **Presentation tags** — ***How to present a model?***
  - **Basic modeling tags** — ***Basic modeling concepts***
  - **Essential pattern tags** — ***Annotating 5 essential patterns***
  - **Catalog & domain-specific pattern tags**
  - **UML-F & adaptation cookbooks**

© 2000, M. Fontoura, W. Pree, B. Rumpe

Context (II)

- **Case Study: The JUnit testing framework**
- **Tool support for UML-F**

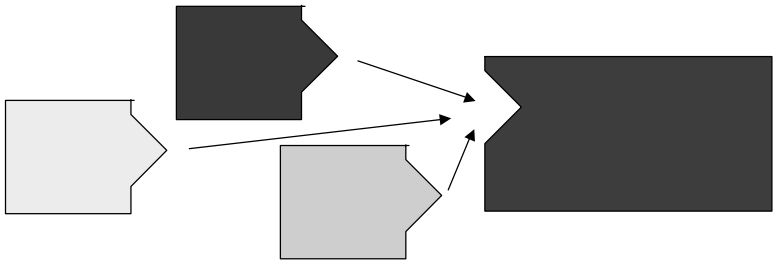
© 2000, M. Fontoura, W. Pree, B. Rumpe

Terminology: frameworks & components

© 2000, M. Fontoura, W. Pree, B. Rumpe

Vision

Construction of complex software systems out of reusable software components:

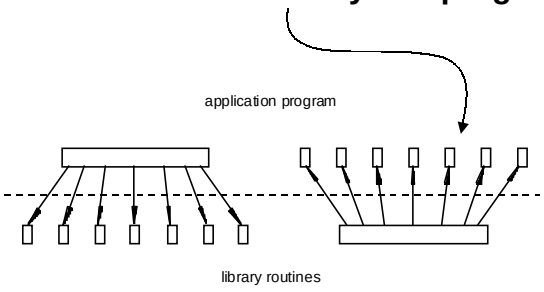


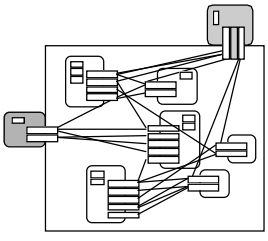
© 2000, M. Fontoura, W. Pree, B. Rumpe

Frameworks

Framework :=

A piece of software that is extensible through the callback style of programming





© 2000, M. Fontoura, W. Pree, B. Rumpe

Components

Component :=

A piece of software with a programming interface

Consequences:

» frameworks that offer a programming interface are components

» module-oriented languages (Modula, Oberon, Ada) and component standards (CORBA, COM, JavaBeans) just offer different ways of defining such programming interfaces

© 2000, M. Fontoura, W. Pree, B. Rumpe

Essential framework patterns

© 2000, M. Fontoura, W. Pree, B. Rumpe

Template & hook methods (I)

Templates and hooks provide insights in the few construction principles of frameworks, i.e., how to keep socalled hot spots/ **variation points** flexible.

Terminology, Part I:

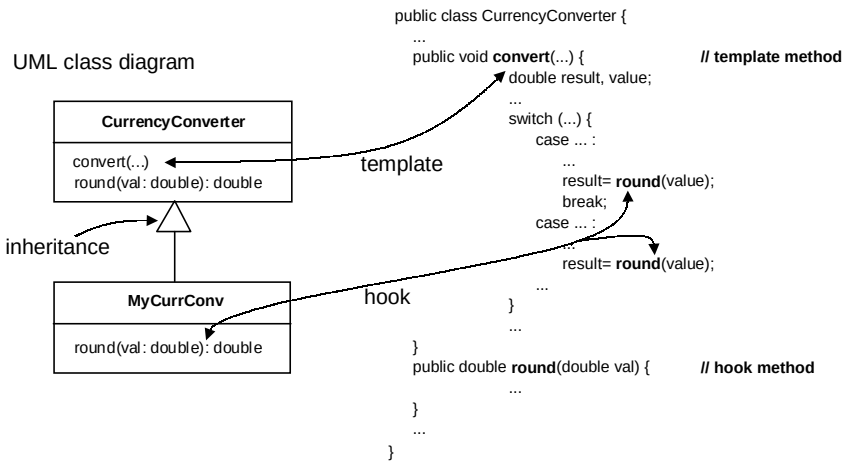
Basic framework building blocks provided by OO language concepts:

- » **template method** ⇔ **frozen spot**
- » **hook method** ⇔ **hot spot/variation point**

Template methods define a **complex default behavior** which is **adjustable by hook methods.**

© 2000, M. Fontoura, W. Pree, B. Rumpe

Template & hook methods (II)



© 2000, M. Fontoura, W. Pree, B. Rumpe

Template & hook classes

Terminology, Part II:

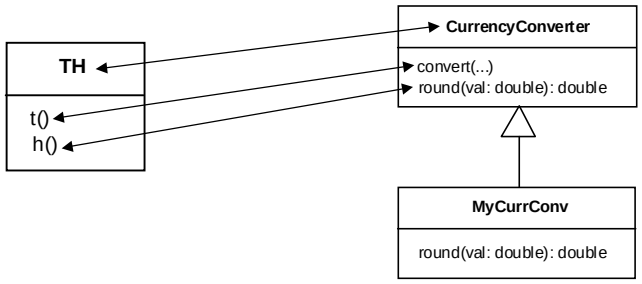
template class T := contains the template method
hook class H := contains the hook method(s)

⇒ **only a few combinations** are possible
these combinations form the **essential patterns**

In the CurrencyConverter example, template and hook method are in the same class, i.e., T and H are unified.

Unification pattern

Unification: T and H class are unified

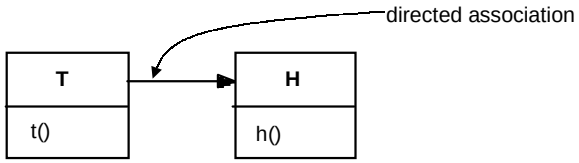


⇒ adaptations can only be done by overriding the hook method

⇒ adaptations require an **application restart**

Separation pattern (I)

Separation: T and H classes are separated

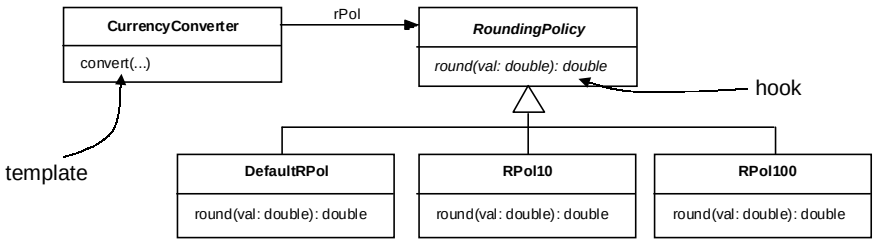


⇒ the behavior of a T object can be modified by composition, i.e., **by plugging in specific H objects**

⇒ **adaptations at run time become feasible**

© 2000, M. Fontoura, W. Pree, B. Rumpe

Separation pattern (II)




```
public void defineRoundingPolicy (RoundingPolicy rp) {
        rPol= rp;
}

public void convert(...) {
        double result, value;
        ...
        result= rPol.round(value);
        ...
}
```

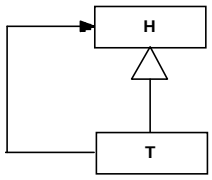
© 2000, M. Fontoura, W. Pree, B. Rumpe

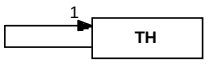
GoF pattern catalog entries based on the Separation pattern

Catalog entry	*hook class*	*hook method*
Abstract Factory	AbstractFactory	CreateProduct()
Builder	Builder	BuildPart()
Command	Command	Execute()
Interpreter	AbstractExpression	Interpret()
Observer	Observer	Update()
Prototype	Prototype	Clone()
State	State	Handle()
Strategy	Strategy	AlgorithmInterface()

© 2000, M. Fontoura, W. Pree, B. Rumpe

Recursive combinations





These patterns allow

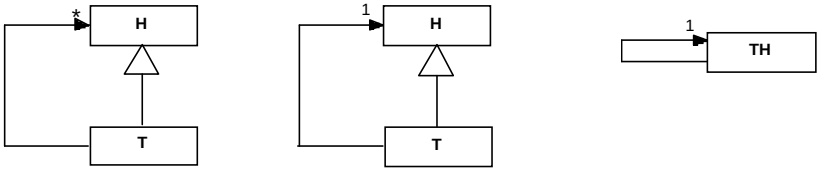
- » **building of directed graphs** of interconnected objects
- » **message forwarding** in the object graphs due to a certain structure of the template method

⇒ The playground of adaptations through composition is enlarged

© 2000, M. Fontoura, W. Pree, B. Rumpe

GoF catalog entries based on recursive compositions

Composite, **Decorator**, and **Chain of Responsibility**:



(and Unification, Separation are the **5 essential patterns**)

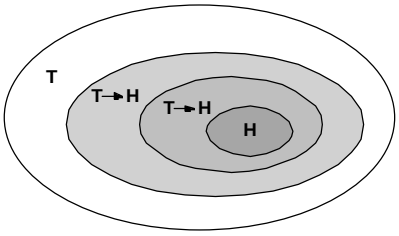
© 2000, M. Fontoura, W. Pree, B. Rumpe

Essential patterns scale up

What is a template method in one context may become a hook method in another context

⇒ **hooks scale up**

⇒ the **essential framework construction patterns scale up**



© 2000, M. Fontoura, W. Pree, B. Rumpe

Pattern documentation in UML-F

- Framework-centered GoF pattern **catalog entries** apply essential construction principles
  - » entries mainly **differ in the semantics of their hooks**.
  - » denoting this semantics in the model helps to understand
    - – the design & code
    - – how to adapt a framework
- Modeling standard UML 1.3 does not sufficiently support framework/pattern documentation.

© 2000, M. Fontoura, W. Pree, B. Rumpe

Pattern semantics

- Any framework part that applies a Unification or Separation pattern attaches a specific semantics (purpose) to the hook.
- Often the pattern is named after the hook's purpose.
- The structure of a framework-centered pattern relies on one of the essential patterns.

© 2000, M. Fontoura, W. Pree, B. Rumpe

Goals of UML-F

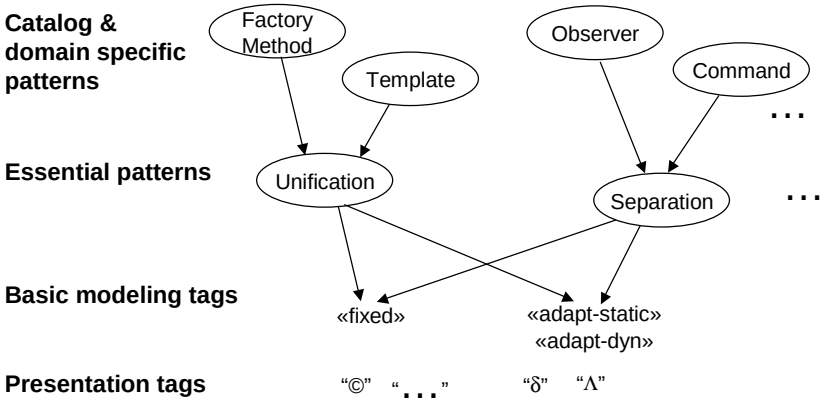
© 2000, M. Fontoura, W. Pree, B. Rumpe

Goals of UML-F

- UML-F provides the **notational elements** to **precisely annotate and document design patterns.**
- UML-F can be used to **concisely communicate** design.
- UML-F is itself structured in the spirit of frameworks, i.e., an **extensible profile.**
- UML-F comprises a **lean, memonic** set of notational elements.
- UML-F **relies on the UML** standard, that is, UML-F extends UML using existing UML extension mechanisms.
- The UML-F notational elements should be adequate for being integrated **in UML tool environments**.

© 2000, M. Fontoura, W. Pree, B. Rumpe

UML-F extension structure



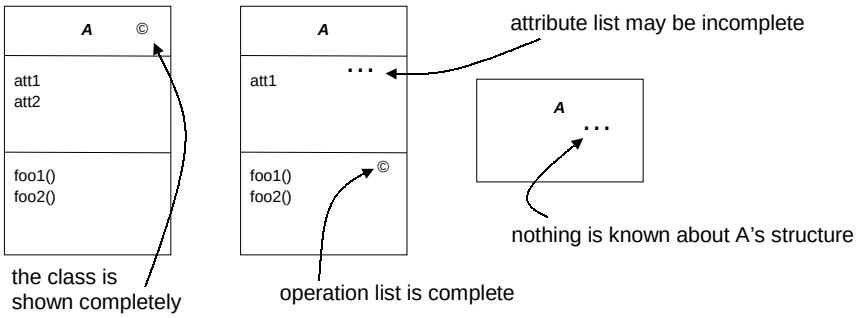
UML-F tag mechanism

- UML 1.3 provides **«stereotypes»** and **{tag=value}** pairs
  - » stereotypes are restricted: no value attached, only one per model element
  - » Tagged value pairs are more flexible and combinable
- UML-F unifies both in the "tag-mechanism", by writing
  - » **«tag:5»** and {tag=5} interchangeably
  - » **«tag»** is short for «tag:True»
  - » **«tag1,tag2»** is OK
  - » **Special forms** of tags, e.g. "©" instead of «complete:True»
  - » Tags values have **types** (e.g. Boolean,String,Integer)

Presentation tags

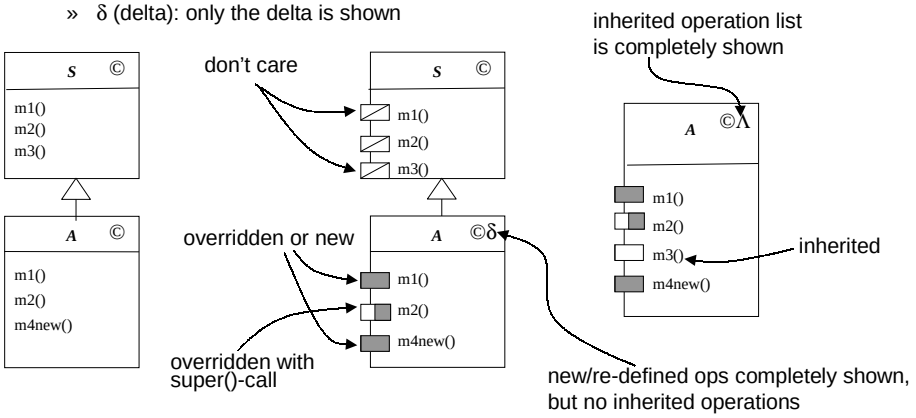
Presentation tags (I)

- Model ≠ diagrammatic representation on the paper
- Are all elements of the model presented in the diagram?
  - » "©" indicates: yes, all are shown
  - » "…" indicates: no, there may be more present in the model



Presentation tags (II)

- Flat and hierarchical representation of the design
  - » Λ (hierarchy-symbol): all elements are shown (expanded or flat version)
  - » δ (delta): only the delta is shown



Presentation tags (III)

- **white rectangle:** the method is inherited and not redefined
- **gray rectangle:** the method is either newly defined, or it is inherited, but completely redefined
- **half gray / half white rectangle:** the method is redefined, but uses the inherited method through a super() call
- **don't care rectangle:** the diagram does not reveal any information (can actually be either of the above three)

Basic modeling tags

© 2000, M. Fontoura, W. Pree, B. Rumpe

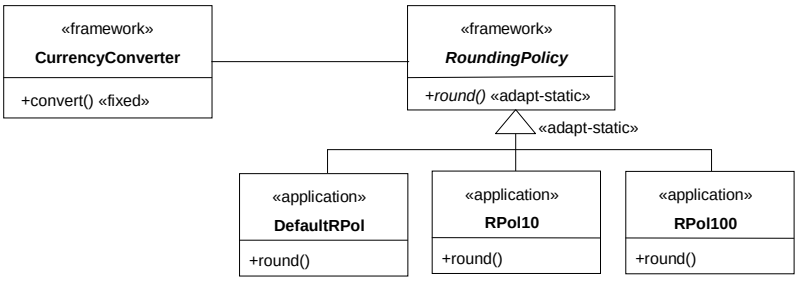
Basic modeling tags (I)

Tag name	*Applies to*	*Value*	*Description*
«fixed»	Class, Method, Generalization	Bool	The element is fixed.
«adapt-static»	Interface, Class, Method, Generalization	Bool	The element can be adapted during design-time through sub-classing.
«adapt-dyn»	Interface, Class, Method, Generalization	Bool	The interface, class, method can be changed through dynamic loading of new subclasses during runtime.
«application»	Class, Package, Interface	Bool	The element belongs to the application.
«framework»	Class, Package, Interface	Bool	The element belongs to the framework.

© 2000, M. Fontoura, W. Pree, B. Rumpe

Basic modeling tags (II)

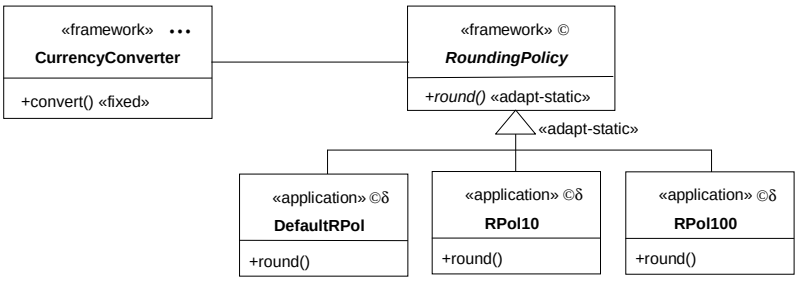
Modeling the Rounding example with the basic UML-F modeling tags



© 2000, M. Fontoura, W. Pree, B. Rumpe

Basic modeling tags (III)

Combining basic modeling and presentation tags



© 2000, M. Fontoura, W. Pree, B. Rumpe

Essential patterns tags

© 2000, M. Fontoura, W. Pree, B. Rumpe

Raising the level of abstraction

- The basic modeling tags are rather flexible but are also quite low level
  - » Use them as building blocks for introducing new tags
  - » Patterns usually combine several tags
- Each new layer of tags is more specific and less flexible
  - » the tags contain more semantic that describes purpose and usage

© 2000, M. Fontoura, W. Pree, B. Rumpe

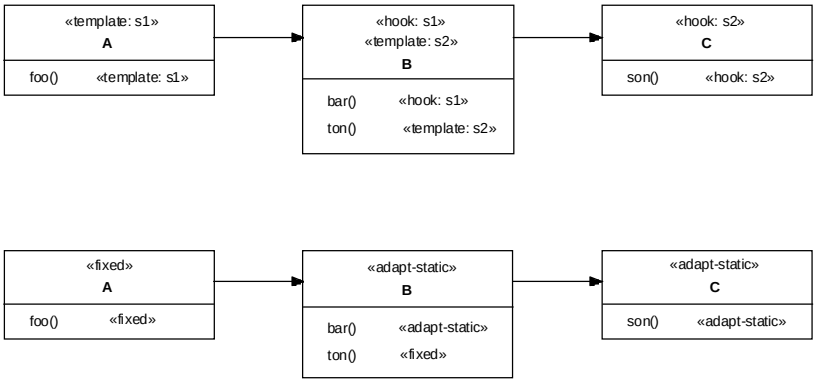
Template and hook tags definition

- The «fixed» method becomes «template»
- The «adapt-dyn» and «adapt-static» methods become «hook»
- The class with the template method «template»
- The class with the hook method «hook»

- A class can be both «template» and «hook»
  - » e.g in the Unification pattern,
  - » or through participation in several pattern
    - – identification through pattern names

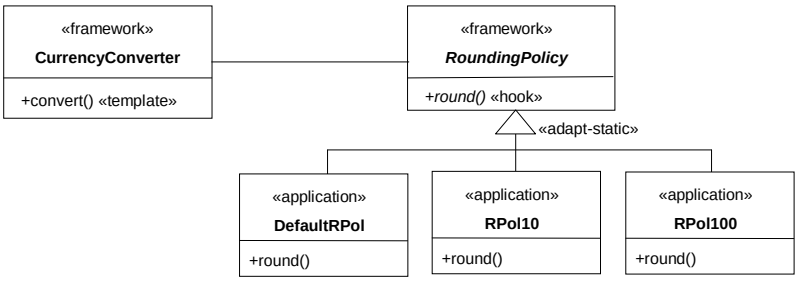
Template and hook tags example (I)

Two equivalent representations of the same design - using basic tags and using «template» and «hook» tags



Template and hook example (II)

The Rounding example with «template» and «hook» tags



© 2000, M. Fontoura, W. Pree, B. Rumpe

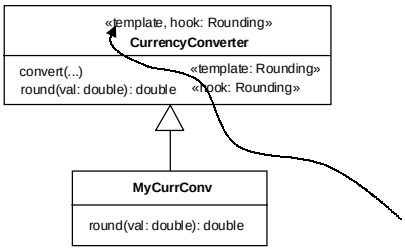
Template and hook tags as building blocks

- Define a tags to:
  - » Make the use of the basic framework patterns explicit
  - » Higher level of system documentation
    - – gives more compact, yet concise documentation
    - – easier to understand
- The «template» and «hook» tags are the building blocks for the basic framework patterns tags

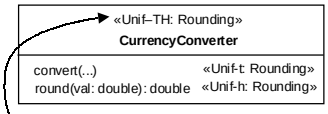
© 2000, M. Fontoura, W. Pree, B. Rumpe

Unification pattern tags

Design based on the «template» and «hook» tags



Design based on the Unification pattern tags

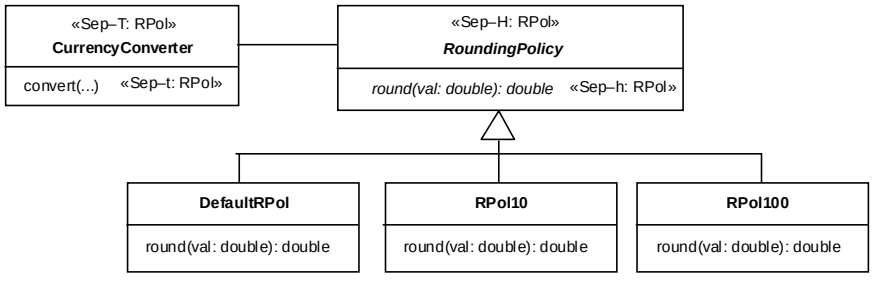


pattern specific tag as a combination of basic tags

© 2000, M. Fontoura, W. Pree, B. Rumpe

Separation pattern tags

Design based on the Separation pattern tags

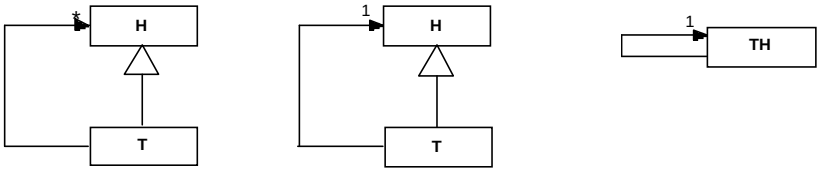


© 2000, M. Fontoura, W. Pree, B. Rumpe

Recursive framework patterns

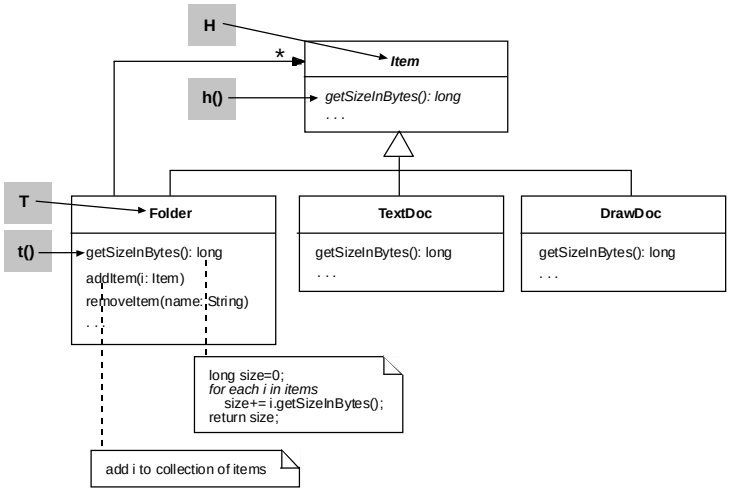
- The «template» and «hook» tags are also the building blocks for the recursive patterns tag set

Composite, **Decorator**, and **Chain of Responsibility**:



© 2000, M. Fontoura, W. Pree, B. Rumpe

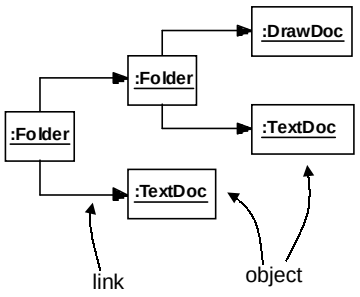
Composite pattern example (I)



© 2000, M. Fontoura, W. Pree, B. Rumpe

Composite pattern example (II)

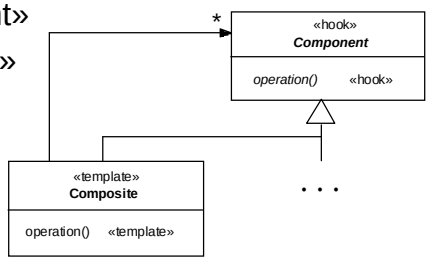
object diagram



© 2000, M. Fontoura, W. Pree, B. Rumpe

Composite pattern tags (I)

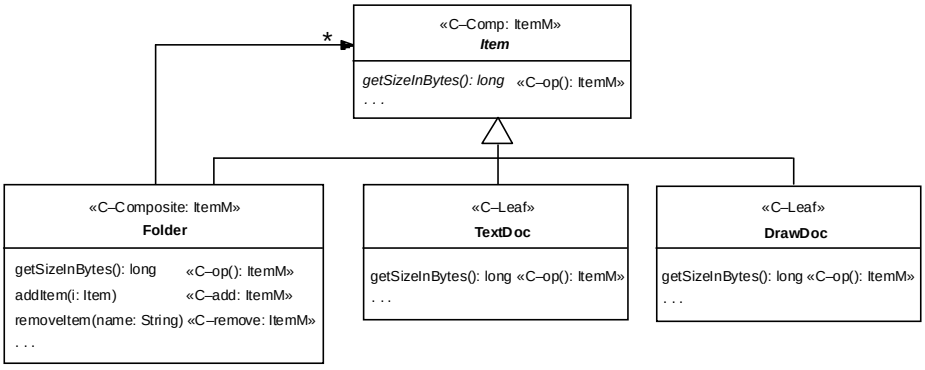
- Class tags
  - «Composite–Component»
  - «Composite–Composite»
  - «Composite–Leaf»
- Method tags
  - «Composite–op()»
  - «Composite–add()»
  - «Composite–remove»



© 2000, M. Fontoura, W. Pree, B. Rumpe

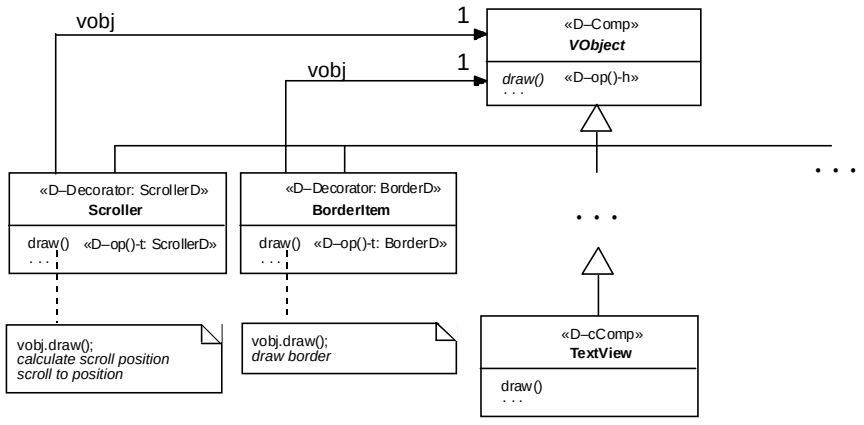
Composite pattern tags (II)

Modeling the Item-Folder framework with the Composite pattern tags:



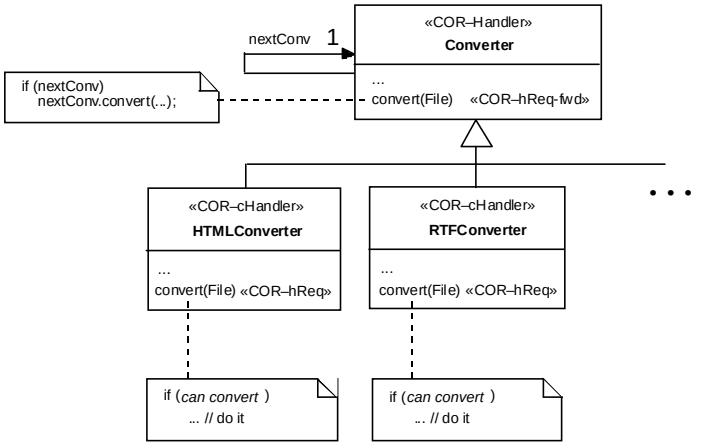
© 2000, M. Fontoura, W. Pree, B. Rumpe

Decorator pattern tags example



© 2000, M. Fontoura, W. Pree, B. Rumpe

COR pattern tags example



© 2000, M. Fontoura, W. Pree, B. Rumpe

Catalog & domain-specific pattern tags

© 2000, M. Fontoura, W. Pree, B. Rumpe

Tags for higher-level patterns

- The UML-F tags for the essential patterns already allow the annotation of frameworks on a pattern level
- In order to express more semantics, additional tags that resemble directly the framework-related patterns in the GoF catalog are helpful
  Recipe: take the GoF pattern structure
  => define UML-F tags
- Domain-specific pattern tags rely on the tags for the essential framework patterns

© 2000, M. Fontoura, W. Pree, B. Rumpe

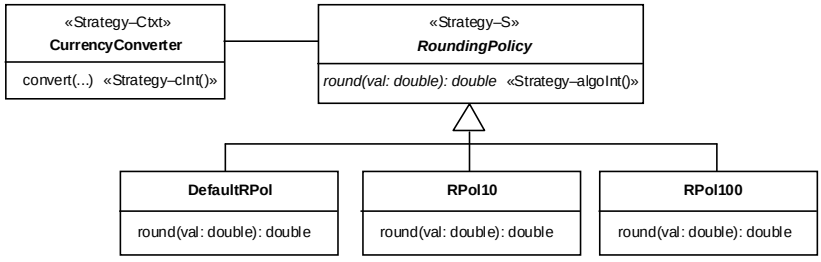
Catalog pattern example: Strategy tags

The Strategy pattern tags are defined in terms of the Separation pattern:

«Sep-t» = «Strategy-Ctxt»

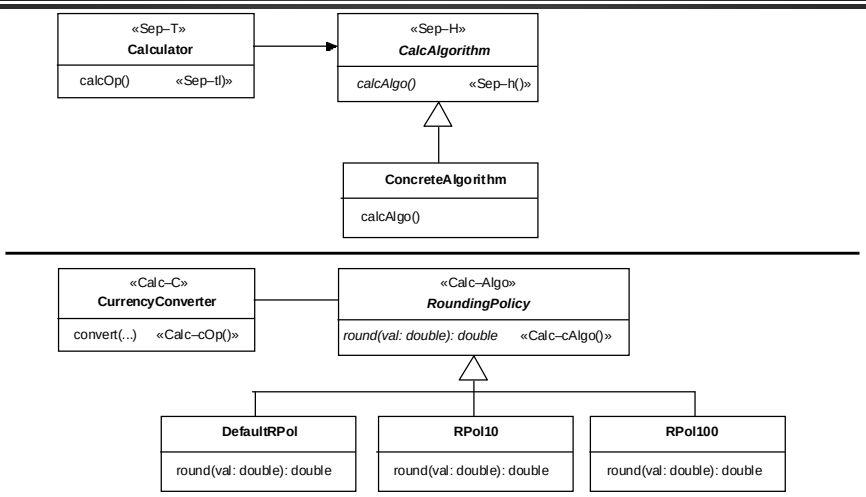
«Sep-h» = «Strategy-S»

…



© 2000, M. Fontoura, W. Pree, B. Rumpe

Domain-specific pattern example



UML-F & adaptation cookbooks

Adaptation cookbooks

- A cookbook is framework specific
- It is a textual description of the
  - » framework architecture and design, and
  - » how to apply the framework through its adaptation (collection of “Howto”-recipes)
- UML-F diagrams complement cookbooks describing both, the architecture, and giving suggestions, where and which code has to be written by the application developer

© 2000, M. Fontoura, W. Pree, B. Rumpe

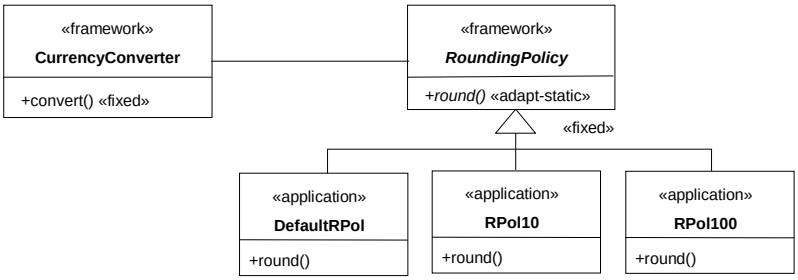
Cookbook recipe example

- How to adapt currency converter?
  - » If all the required rounding policies have been defined the adaptation is by composition only
  - » If a rounding policy still has to be defined:
    - – Subclass RoundingPolicy
    - – Override round()

© 2000, M. Fontoura, W. Pree, B. Rumpe

UML-F description of the adaptation (I)

The «fixed» generalization indicates the adaptation by composition principle—no new classes are added

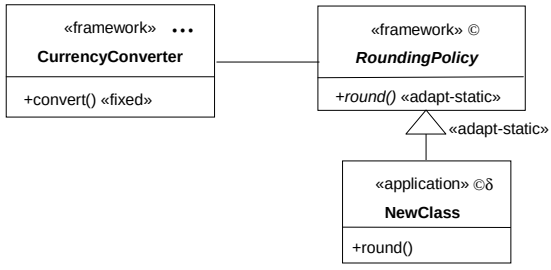


© 2000, M. Fontoura, W. Pree, B. Rumpe

UML-F description of the adaptation (II)

The «application» and «adapt-static» tags indicate where the code should be added

The © symbol indicates that the only method that has to be defined in the application classes is round()



© 2000, M. Fontoura, W. Pree, B. Rumpe

Reusing a cookbook

- Use the cookbook recipes for the basic framework patterns to define new ones
- The UML-F tags provide the mapping

- How to adapt currency converter **(Separation)**?
  - » If all the required **(concrete Sep-h classes)** rounding policies have been defined the adaptation is black-box
  - » If a rounding policy **(concrete Sep-h class)** still has to be defined:
    - – Subclass RoundingPolicy **(Sep-h)**
    - – Override round() **(Sep-h())**

© 2000, M. Fontoura, W. Pree, B. Rumpe

The JUnit testing framework

© 2000, M. Fontoura, W. Pree, B. Rumpe

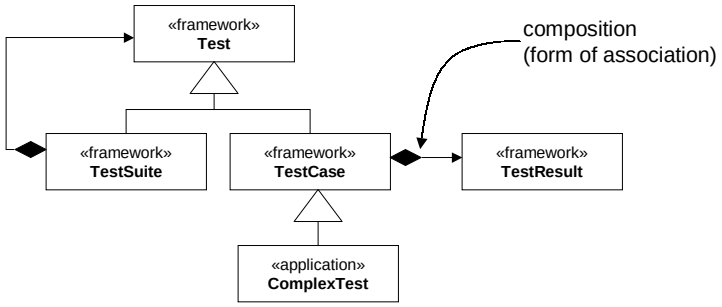
The JUnit components (I)

- Adding new test cases: JUnit provides a standard interface for defining test cases and allows the reuse of common code among related test cases.
- Tests suites: Framework users can group test cases in test suites.
- Reporting test results: the framework keeps flexible how test results are reported. The possibilities include storing the results of the tests in a database for project control purposes, creating HTML files that report the test activities.

© 2000, M. Fontoura, W. Pree, B. Rumpe

The JUnit components (II)

Overview of the JUnit design - Class ComplexTest defines test cases for complex numbers



© 2000, M. Fontoura, W. Pree, B. Rumpe

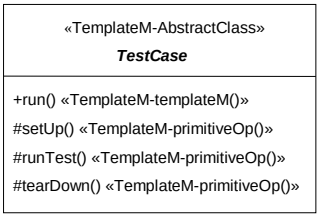
The TestCase variation point (I)

- The initialization part is responsible for creating the text fixture.
- The test itself uses the objects created by the initialization part and performs the actions required for the test.
- Finally, the third part cleans up a test.

© 2000, M. Fontoura, W. Pree, B. Rumpe

The TestCase variation point (II)

The TestCase design is based on the Template Method design pattern - method run() controls the test execution



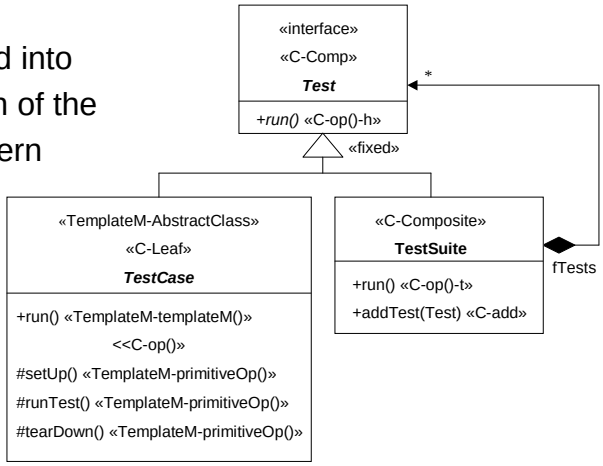

```
public void run() {
      setUp();
      runTest();
      tearDown();
}
```

© 2000, M. Fontoura, W. Pree, B. Rumpe

The TestSuite variation point

TestCases are grouped into TestSuites—a variation of the Composite design pattern

Black-box adaptation



© 2000, M. Fontoura, W. Pree, B. Rumpe

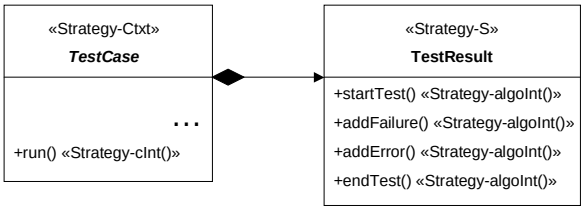
The TestResult variation point (I)

- Failures are situations where the assert() method does not yield the expected result.
- Errors are unexpected bugs in the code being tested or in the test cases themselves.
- The TestResult class is responsible for reporting the failures and errors in different ways.

© 2000, M. Fontoura, W. Pree, B. Rumpe

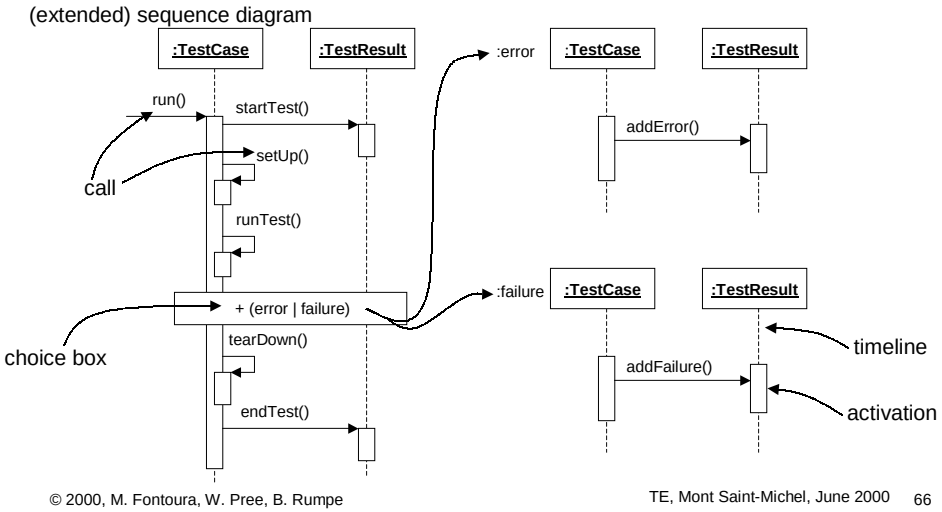
The TestResult variation point (II)

- TestResult must provide four methods:
  - startTest() - initialization code
  - addFailure() - reports a failure
  - addError() - reports an error
  - endTest() - clean-up code



© 2000, M. Fontoura, W. Pree, B. Rumpe

The TestResult variation point (III)



© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting JUnit

- Cookbook recipes and UML-F diagrams for each of the JUnit variation points
  - » Create a test case (ComplexTest)
  - » Create a test suite (for the ComplexTest methods)
  - » Create an HTML reporting mechanism

© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestCase (I)

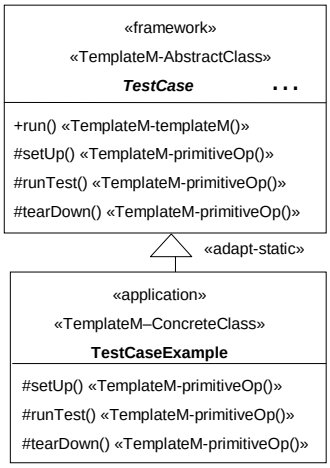
- TestCase adaptation recipe:
  - » Subclass TestCase
  - » Override setUp() (optional). The default implementation is empty
  - » Override runTest()
  - » Override tearDown() (optional). The default implementation is empty

© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestCase (II)

TestCaseExample exemplifies the code that has to be added by the application developer

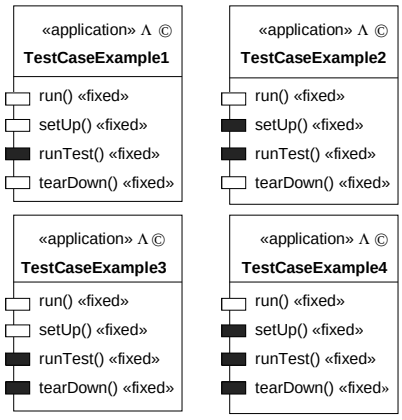
White-box adaptation



© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestCase (III)

For possible adaptation examples, considering the optional hook methods



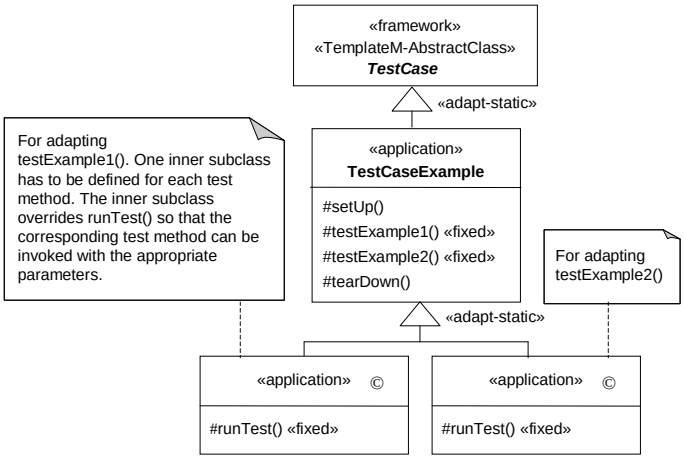
© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestCase (IV)

- One aspect in the TestCase class cannot be captured in UML-F design diagrams
  - » Method runTest() takes no parameters as input
  - » Different test cases require different input parameters.
  - » The interface for theses test methods has to be adapted to match runTest().

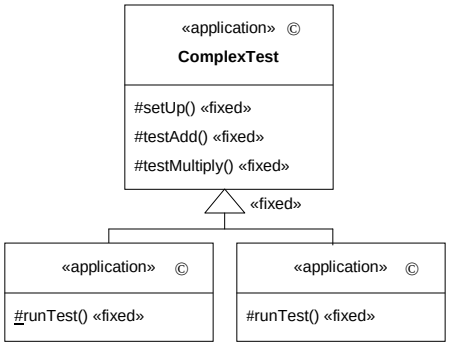
© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestCase (V)



© 2000, M. Fontoura, W. Pree, B. Rumpe

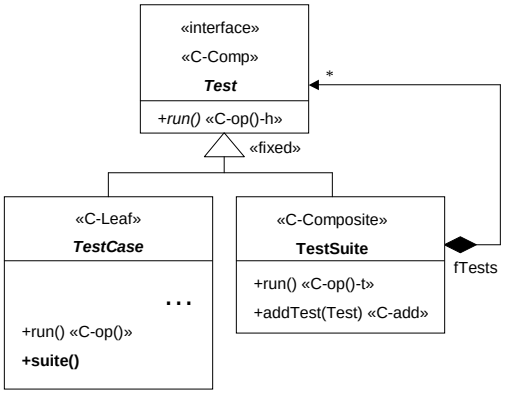
Adapting TestCase (VI)



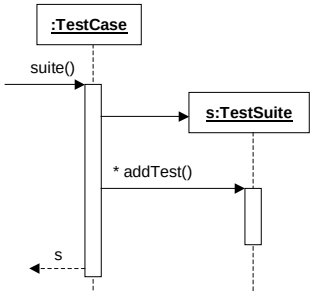

```
public class ComplexTest extends TestCase {
        private ComplexNumber fOneZero;
        private ComplexNumber fZeroOne;
        private ComplexNumber fMinusOneZero;
        private ComplexNumber fOneOne;

protected void setUp() {
        fOneZero = new ComplexNumber(1, 0);
        fZeroOne = new ComplexNumber(0, 1);
        fMinusOneZero = new ComplexNumber(-1, 0);
        fOneOne = new ComplexNumber(1, 1);
}

public void testAdd() {
        //This test will fail !!!
        ComplexNumber result = fOneOne.add(fZeroOne);
        assert(fOneOne.equals(result));
}

public void testMultiply() {
        ComplexNumber result = fZeroOne.multiply(fZeroOne)
        assert(fMinusOneZero.equals(result));
}
```

© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestSuite (I)



Adaptation by overriding the **suite()** method



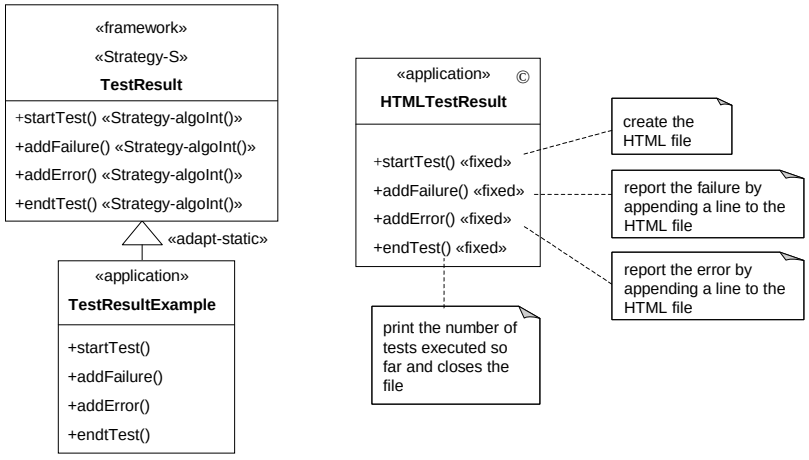
© 2000, M. Fontoura, W. Pree, B. Rumpe

Adapting TestSuite (II)

TestCase and TestSuite are related variation points

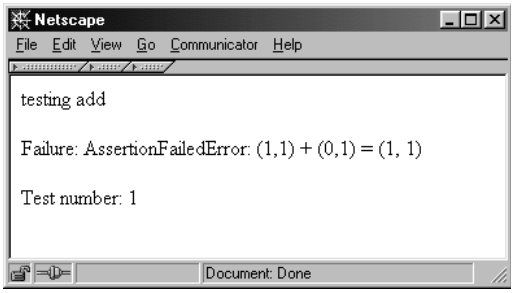
```
public static Test suite() {
        TestSuite suite = new TestSuite();
        suite.addTest(new ComplexTest("testing add") {
                        protected void runTest() { this.testAdd(); }
        } );
        suite.addTest(new ComplexTest("testing multiply") {
                        protected void runTest() { this.testMultiply(); }
        } );
        return suite;
}
```


Adapting TestResult (I)



Adapting TestResult (II)

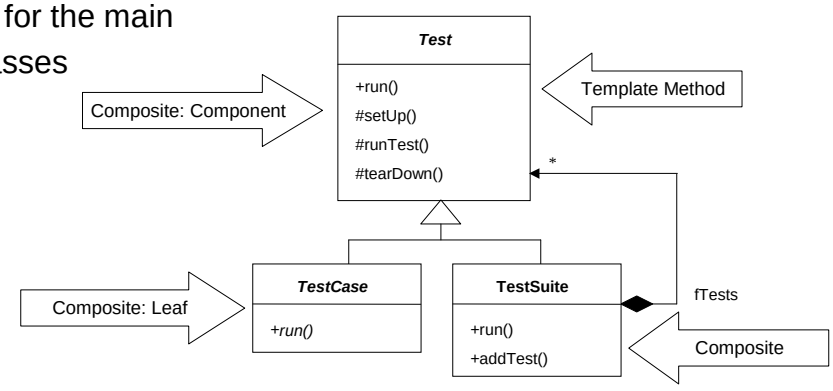
Display of a sample HTML file that reports a failure.



© 2000, M. Fontoura, W. Pree, B. Rumpe

Pattern-annotated diagrams

Pattern-annotated diagram for the main JUnit classes

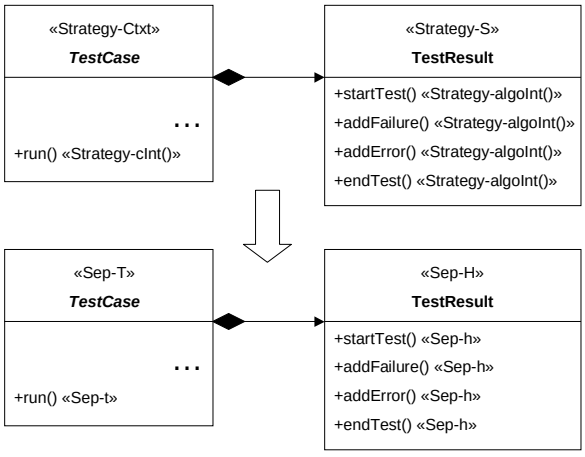


© 2000, M. Fontoura, W. Pree, B. Rumpe

Tool support for UML-F

© 2000, M. Fontoura, W. Pree, B. Rumpe

Moving between layers

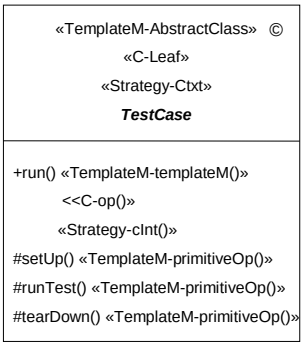


© 2000, M. Fontoura, W. Pree, B. Rumpe

Tags as hyperlinks (I)

- A single class participates in several design patterns

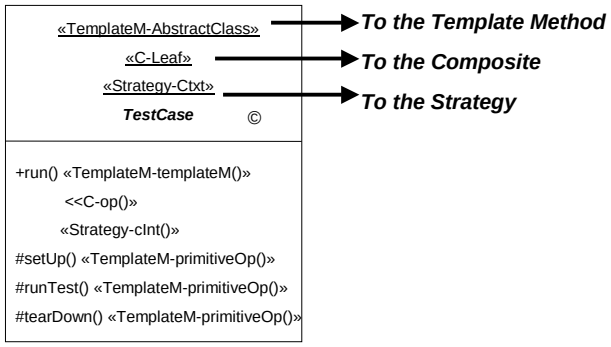
  This is a common situation that may lead to a polluted design



© 2000, M. Fontoura, W. Pree, B. Rumpe

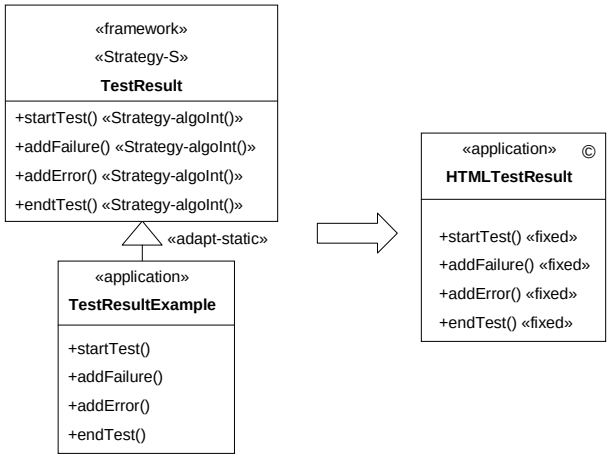
Tags as hyperlinks (II)

Use of design incompleteness presentation tags for "slicing the design"



© 2000, M. Fontoura, W. Pree, B. Rumpe

“Executing” adaptation diagrams



© 2000, M. Fontoura, W. Pree, B. Rumpe

Bibliography

© 2000, M. Fontoura, W. Pree, B. Rumpe

Bibliography (I)

UML-F Web-site: http://UML-F.net

Fontoura M., Pree W., Rumpe B. (2000) The UML Profile for Framework Architectures, Addison-Wesley.

Beck K. (1999). eXtreme Programming explained, Addison-Wesley

Beck K. and Gamma E. (1998a). JUnit: A Cook's Tour" (ftp://www.armaties.com/D/home/armaties/ftp/TestingFramework/Junit/)

Beck K. and Gamma E. (1998b). Test Infected: Programmers Love Writing Tests, Java Report, 3(7)

Boehm B. (1994). Megaprogramming. Video tape by University Video Communications (http://www.uvc.com), Stanford, California

Booch G. (1994). Object-Oriented Analysis and Design with Applications. Redwood City, CA: Benjamin/Cummings

Booch G., Rumbaugh J., Jacobson I. (1998) *The Unified Modeling Language User Guide.* Reading, Massachusetts: Addison-Wesley

Coad P. (1992). Object-oriented patterns. *Communications of the ACM*, **33**(9)

D'Souza D., Wills A. (1998) *Objects, Components, and Frameworks with UML*. Reading, Massachusetts: Addison-Wesley

D'Souza D., Sane A., Birchenough A. (1999) *First-Class Extensibility for UML – Packaging of Profiles, Stereotypes, Patterns.* In: «UML»'99 – The Unified Modeling Language. Conference Proceedings. Eds: R. France, B. Rumpe. Springer Verlag. LNCS 1723.

© 2000, M. Fontoura, W. Pree, B. Rumpe

Bibliography (II)

Fayad M., Schmidt D., Johnson R. (1999) Building Application Frameworks: Object-Oriented Foundations of Framework Design, Wiley

Fayad M., Schmidt D., Johnson R. (1999) Implementing Application Frameworks: Object-Oriented Frameworks at Work, Wiley

Fayad M., Schmidt D., Johnson R. (1999) Domain-Specific Application Frameworks: Manufacturing, Networking, Distributed Systems, and Software Development, Wiley

Gamma E., Helm R., Johnson R. and Vlissides J. (1995) Design Patterns—Elements of Reusable OO Software. Reading, MA: Addison-Wesley (also available as CD)

Jacobson I., Booch G., Rumbaugh J., (1999) *The Unified Software Development Process.* Reading, Massachusetts: Addison-Wesley

Johnson R. (1992). Documenting Frameworks Using Patterns, *ACM Conference on Object-Oriented Programming, Systems, Languages and Applications 1992*

Krasner G. and Pope S. (1998). A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, **1**(3)

Pree W. (1995) Design Patterns for Object-Oriented Software Development. Reading, Massachusetts: Addison-Wesley/ACM Press

Rumbaugh J., Jacobson I., Booch G., (1998) *The Unified Modeling Language Reference Manual.* Reading, Massachusetts: Addison-Wesley

Szyperski C. (1998) Component Software—Beyond Object-Oriented Programming, Addison-Wesley.

© 2000, M. Fontoura, W. Pree, B. Rumpe