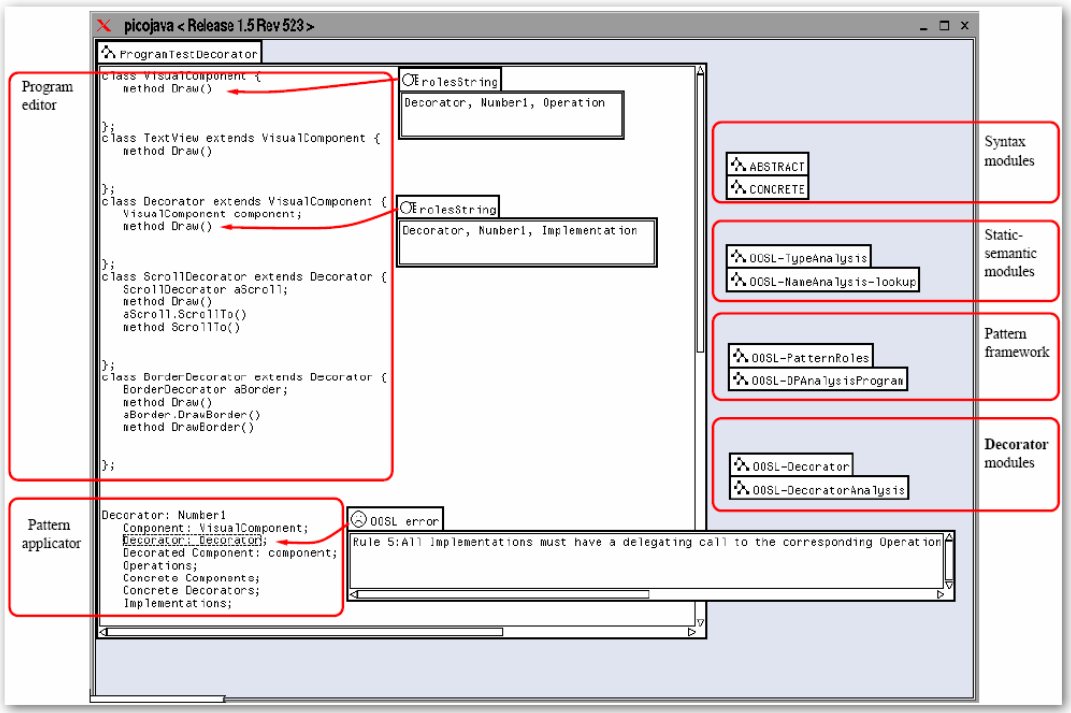
Statically Checked Documentation with Design Patterns



報告者: 林偉民

2006/12/06

Reference

- Aino Cornils、G¨orel Hedin :”**Statically Checked Documentation with Design Patterns”,** 2000 IEEE
- Aino Cornils’s web site http://www.daimi.au.dk/~apaipi/index.html
- APPALB tool web site http://www.cs.lth.se/Research/ProgEnv/Applab.html
- Marek Vok´a∨”An efficient tool for recovering Design Patterns from C++ Code”, Vol. 5, No. 1, January–February 2006 by JOT

Outline

- **Introduction**
  - **-What is the tool will to solve problem**
- **Overview of the approach**
  - **Introduction the DPDOC**
- **DPDOC inside Overview**
- **DPDOC Current status and future extensions**
- **Conclusion and compare to DPUT.**

Introduction-
what is the tool will to solve problem

Design patterns的起源

- Design patterns are descriptions of abstract solutions to often recurring problems.
- The most well-known catalogue of design patterns can be found in [Gamma95], hereafter called the GoF book.
- 四人幫之後，許多的pattern應用一一出現，也因此在design pattern的使用上有一些問題。
  - How to find the design pattern in program
  - How to validated the design pattern in program
  - How to refactoring code to design pattern
  - …

Related Work

- Design pattern tool classic
  - Only pattern detection or validated tool
    - An efficient tool for recovering Design Patterns from C++ Code
      - 專注在發現大量source code中的pattern。
  - Design pattern code generator
    - Template、refactoring
  - Check design pattern rule to help write program

What is the tool will to solve problem

- *some problems with the use of design patterns have been identified.*
  - *The so-called tracing problem describes the difficulty in documenting software systems using design patterns.*
- *First method*
  - *認真的寫一份記錄文件*
    - *問題----隨著軟體的開發，文件的內容會失去和軟體的一致性。*

The paper approach overview

- *The paper propose* approach overview
  - *checks the design pattern rules of a software system*
  - *semi-automatically documents of a software system*
    - *maintains the documentation of a software system*

- In some ways we want to be able to treat design patterns like language constructs. ?

- Both in the sense that they are visible in the code and that we want to be warned when we misuse them.

Overview of the approach

What is the user of DPDOC

- *DPDOC* helps designers use patterns in a safe and efficient way, almost as if they were language constructs. If a user misuses a construct in the programming language, the compiler answers with appropriate warnings.
- User feedback---a group student at the university of Aarhus.
  - 不能用在真實的開發環境中
  - Driver role是方便、有用的功能, 使他們可以想起design pattern的使用方式細節。
  - GUI is very frustrating

The tool base on APPLAB and some fundamentals introduction

?

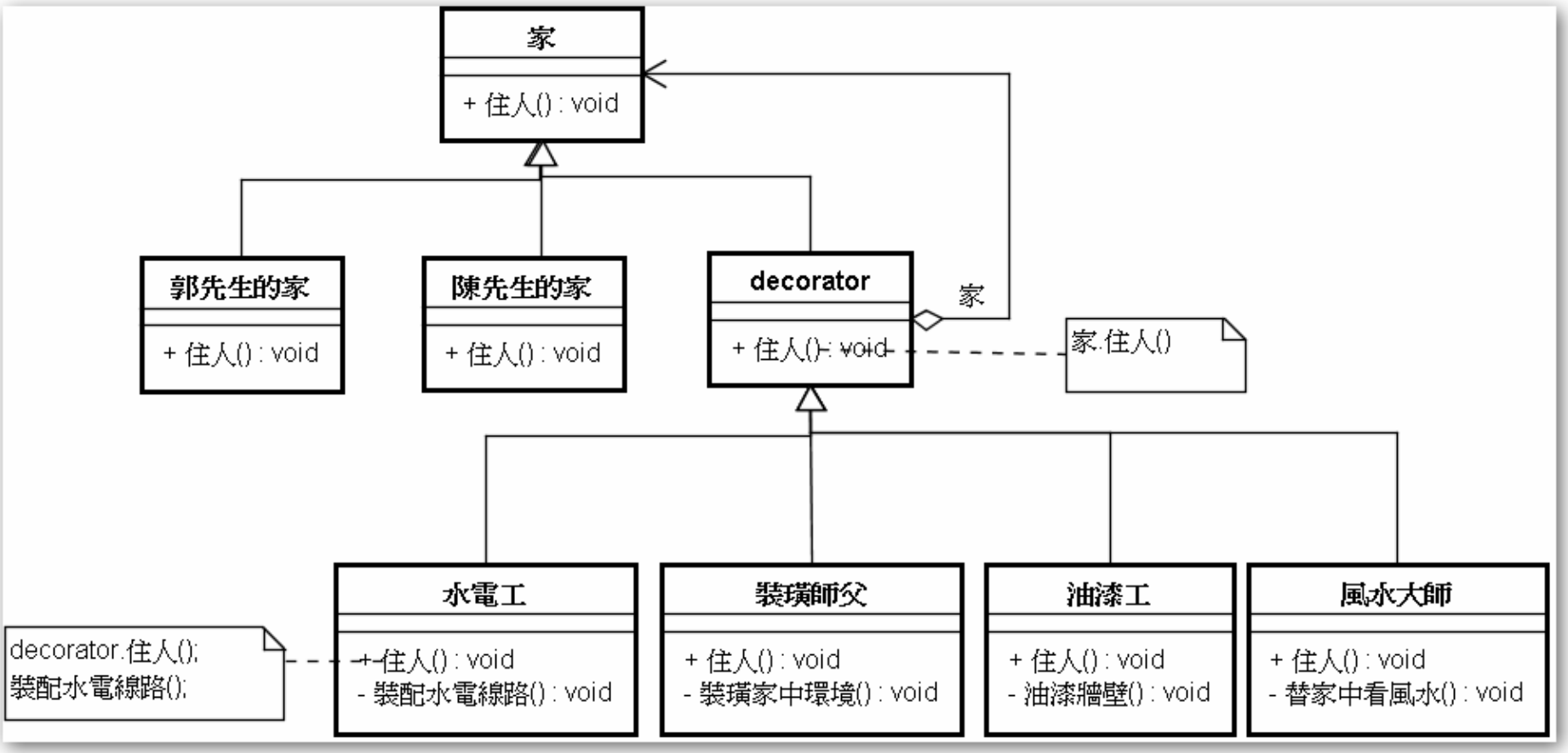
- The version of APPLAB that *DPDOC* is built on uses a subset of the grammar for Java called PicoJava
- In a program, language constructs are easily visible due to keywords and the structure of the code.
  - 作者認爲design pattern不易在程式中看出來。
- *DPDOC* proves particularly useful in relation to object-oriented *frameworks*.
  - user has to follow some rules for applying the framework. These rules can be expressed as patterns that *DPDOC* can check

Program element and *defining*、*derived roles*.

- In source code
  - A program element that participates in a pattern is said to play a certain *role* in that pattern.
    - program element can be used for a role: classes, methods, variables, etc
- In DPDOC framework
  - In our approach we distinguish between two kinds of roles: *defining roles* and *derived roles*.
    - For example, for class roles, a superclass will typically play the defining role and its subclasses will play derived roles.

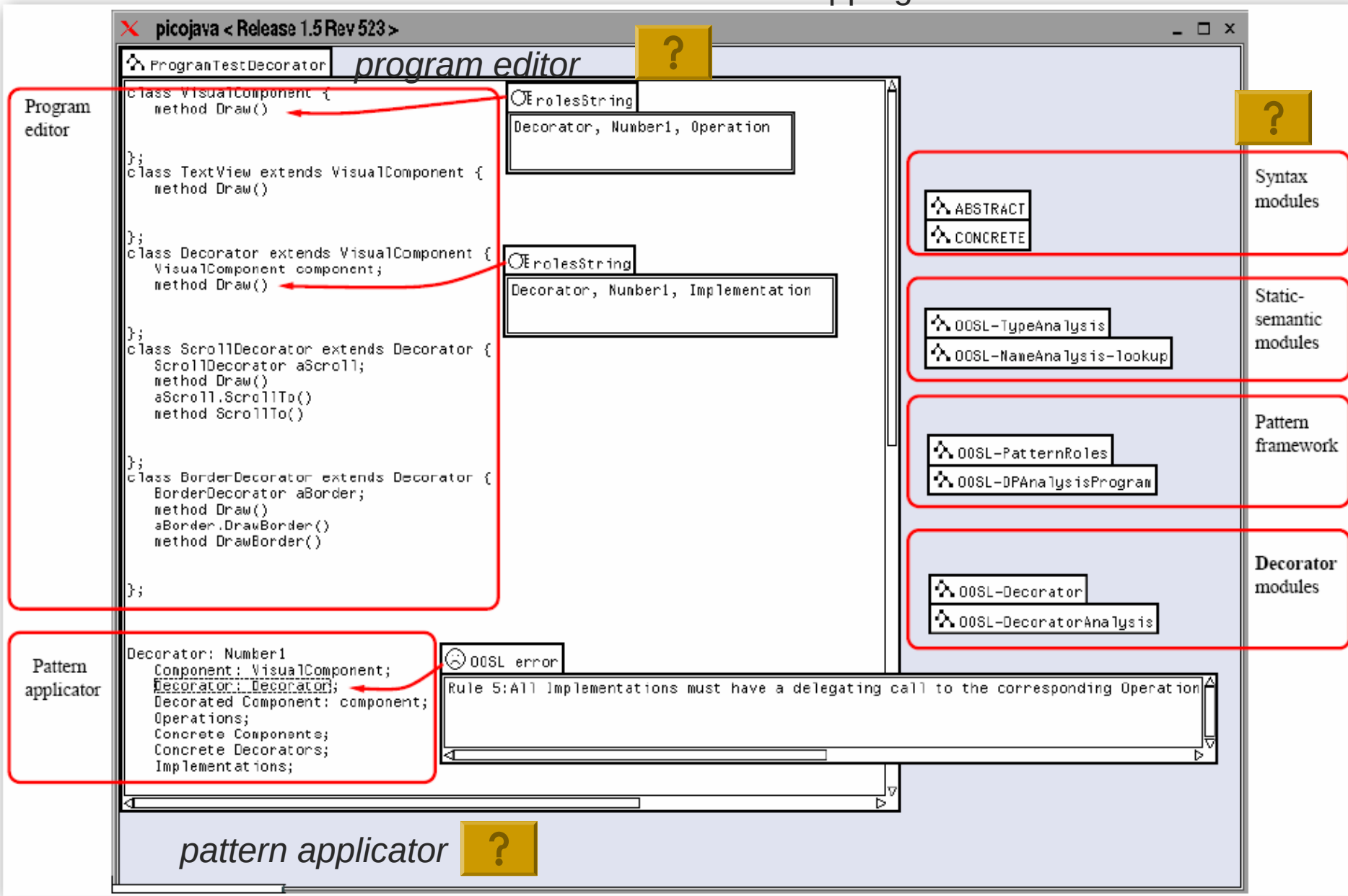
A example of DPDOC

A Decorator example



The user interface

The tool only support 1by 1 mapping



DPDOC inside Overview

Framework of DPDOC

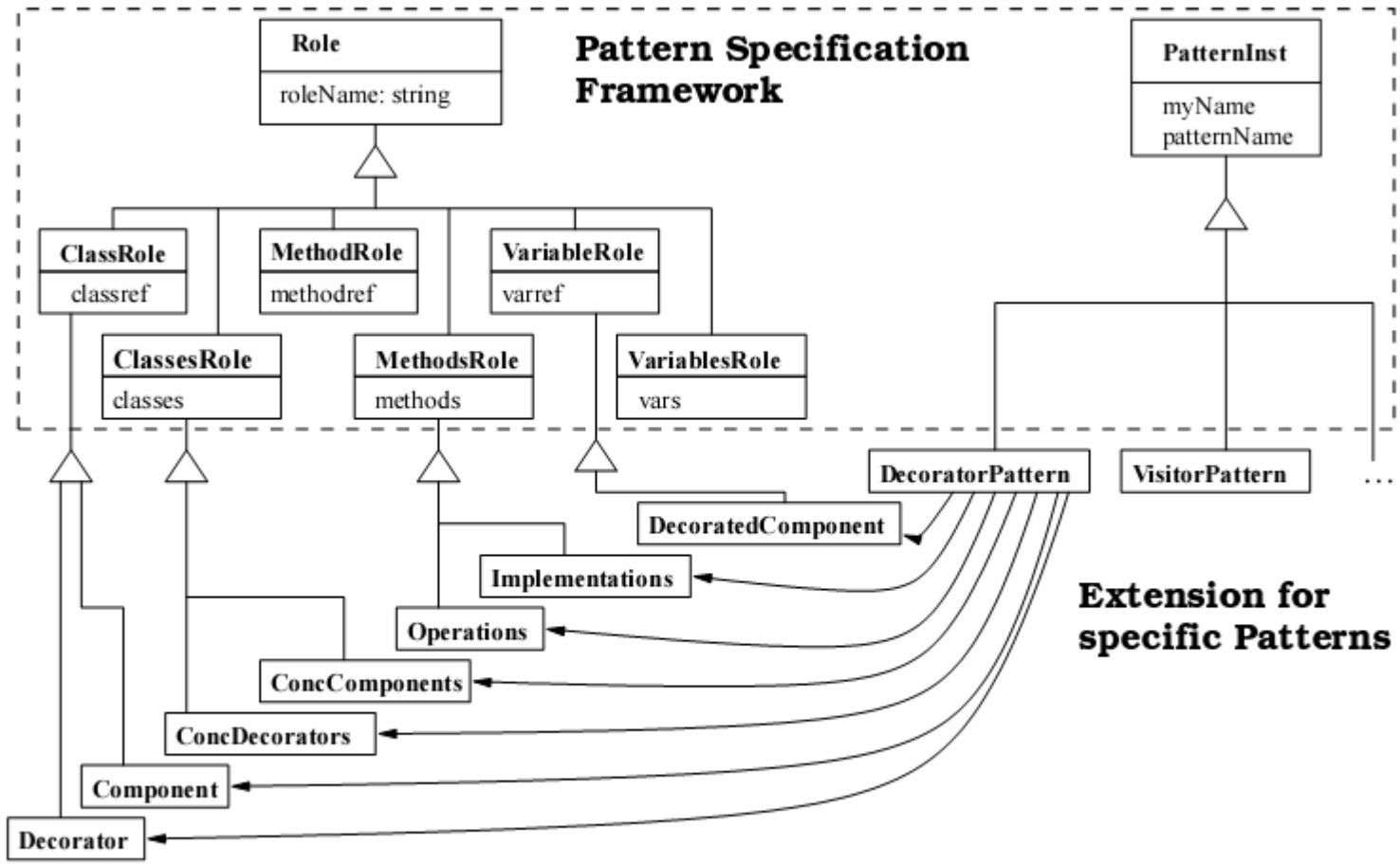
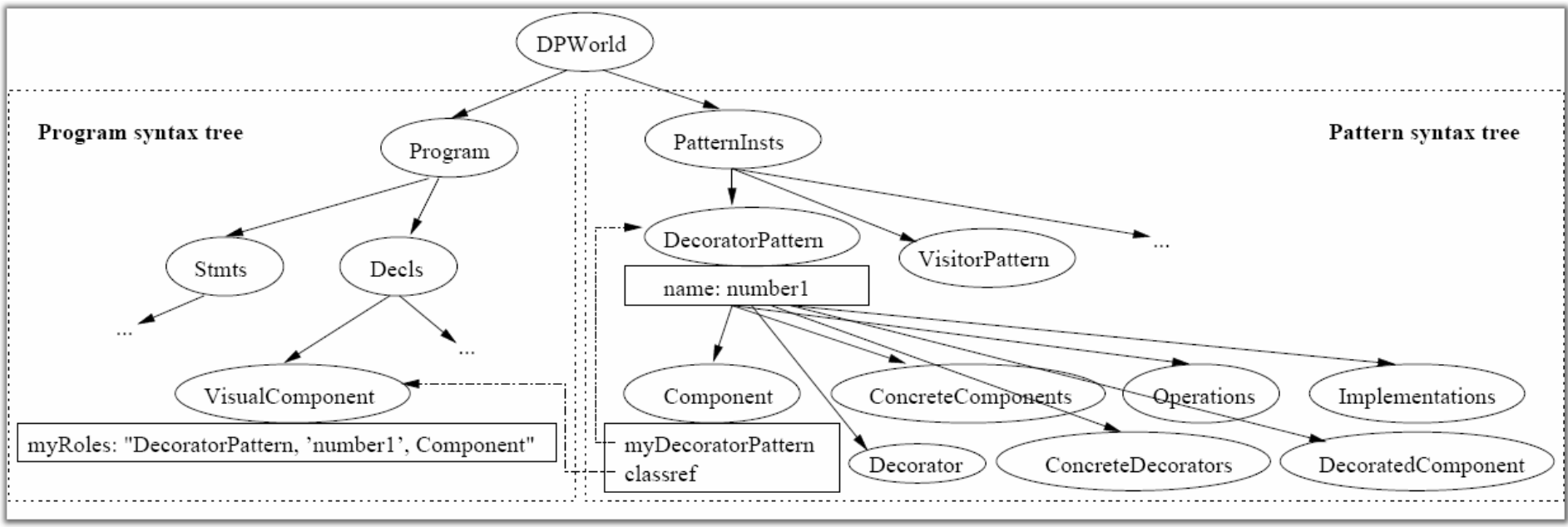


Figure 9.5: The pattern specification framework

Abstract syntax tree for Program code and Design Pattern



The pattern rule check is depended on Abstract syntax tree

The rule is specify by every pattern

DPDOC Current status and future extensions

Current status of the DPDOC

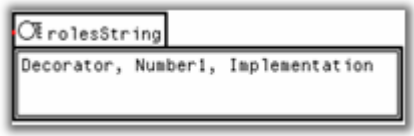
- *DPDOC* is a prototype tool. It was (and is being) developed to study how a tool can support the use of patterns and documentation in the best way.

- Now support PicoJava and change it is easy
  - PicoJava includes key object-oriented constructs of Java like classes, subclassing, and methods, but currently lacks support for, e.g., interfaces and exceptions.
  - change it is easy
    - You just changing the specification of the pattern application language.

- Currently *DPDOC* supports the use of 7 patterns from the GoF book: **Decorator**, **Observer**, **Mediator**, **Composite**, **Adapter**, **Factory Method** and **Visitor**.

future extensions of DPDOC

- Extend *DPDOC* with the remaining patterns from the GoF book
- Provide a more intuitive and GUI.

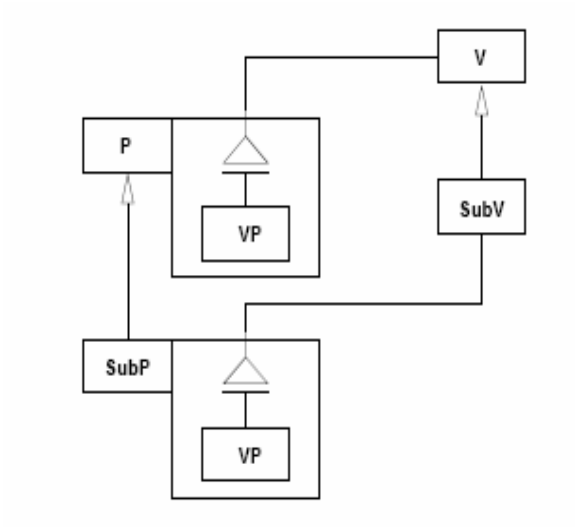


  - 在method、class上按一下，可以設定user-defined reference attributes
  - support navigation in terms of the pattern applications and roles.
    - EX: for example to navigate from the program code to the pattern application code or vice versa, or between different points in the program code that belong to the same pattern.
  - UML-Like Class diagram
- Generating design documentation from program
  - Like javadoc record the class API
  - DPDOC can provide some similar document.

Conclusion and compare to DPUT

DPDOC之後的工作

- 資料出自作者在2001的論文
  - Pattern have 9
    - Add two pattern: bridge、chain of responsibility
  - UML like in beta



作者對DPDOC的結論

- *DPDOC* is a language-based prototype tool supporting documentation of program code with patterns.
- The tool supports pattern visibility, rule checking, and automatic role derivation.
- By implementing seven of the most challenging of the GoF patterns in *DPDOC*, we have proven the viability of the technique, and we think the approach constitutes a sound basis for pattern tool support.

DPDOC and MAUT Manager compare

- DPDOC
  - 一個用來幫忙prototype picojava program, 寫design pattern文件的工具
  - Extendibility for new program language
    - Base on APPLAB support language, but APPLAB has not been updated for a long period of time.
  - *Role derivation*
    - Class ~ variable.
- MAUT Manager
  - The tool Using in java source code to write design pattern document and other system document.
  - Extendibility for new program language
    - Base on eclipse support program language.
  - *Role derivation*
    - Not support.

DPDOC Framework and DPUT Framework compare

- DPDOC framework
  - Support editing of the example program in APPLAB
  - Check rule is base on APPLAB's Attributed Grammars
  - Extendibility for new design pattern
    - Add new role class according the new pattern participants and participants member.
- DPUT framework
  - Support editing of the java program in Java sdk1.5
  - Check rule is base on the java reflection and annotation.
  - Extendibility for new design pattern
    - Add new pattern constraint build class and annotation define class(most two), a simple AST factory and a pattern identify type class (total 4~5)

Q & A

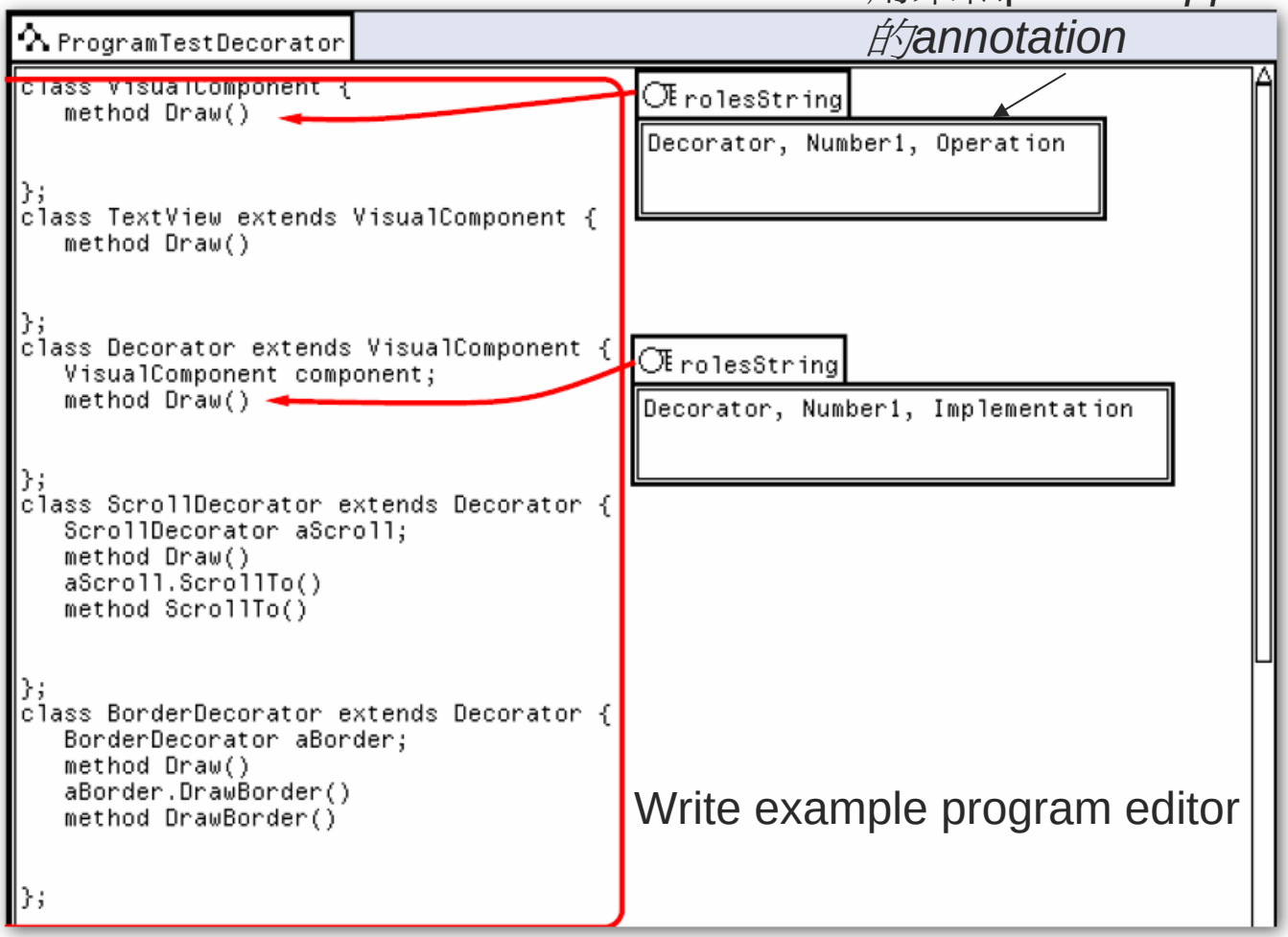
What is language constructs

ProgramTestDecorator

```
class VisualComponent {
   method Draw()


};
class TextView extends VisualComponent {
   method Draw()


};
class Decorator extends VisualComponent {
   VisualComponent component;
   method Draw()


};
class ScrollDecorator extends Decorator {
   ScrollDecorator aScroll;
   method Draw()
   aScroll.ScrollTo()
   method ScrollTo()


};
class BorderDecorator extends Decorator {
   BorderDecorator aBorder;
   method Draw()
   aBorder.DrawBorder()
   method DrawBorder()


};
```

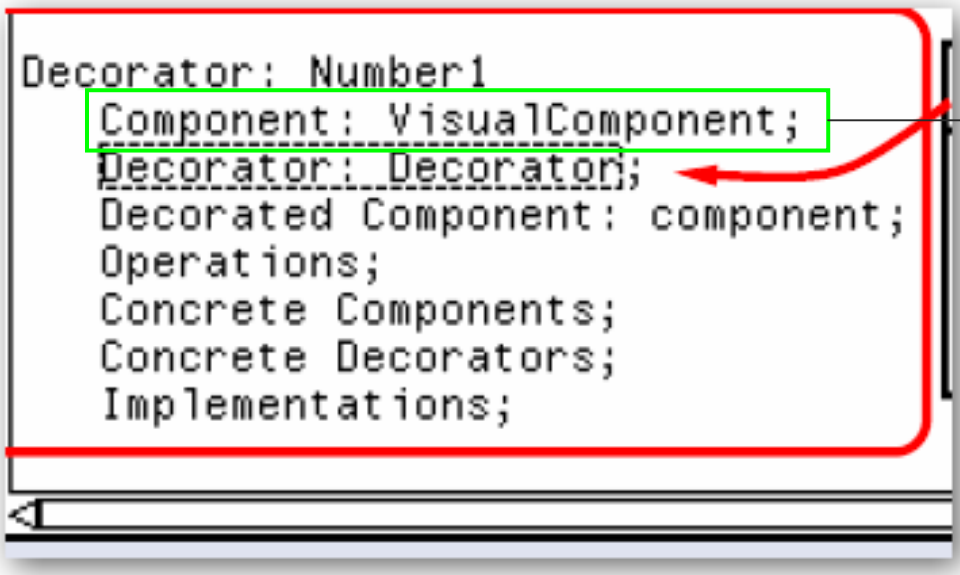
What is program editor

用來和pattern *applicator*合作的annotation



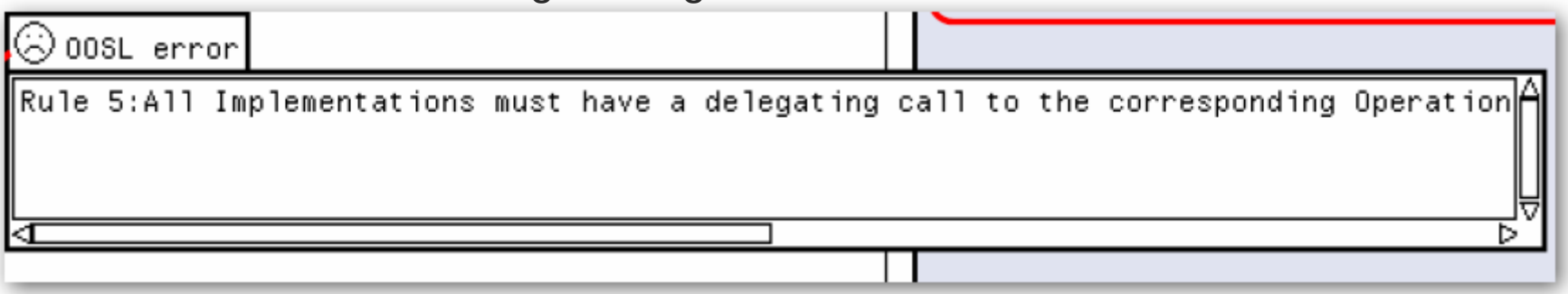
Write example program editor

pattern applicator

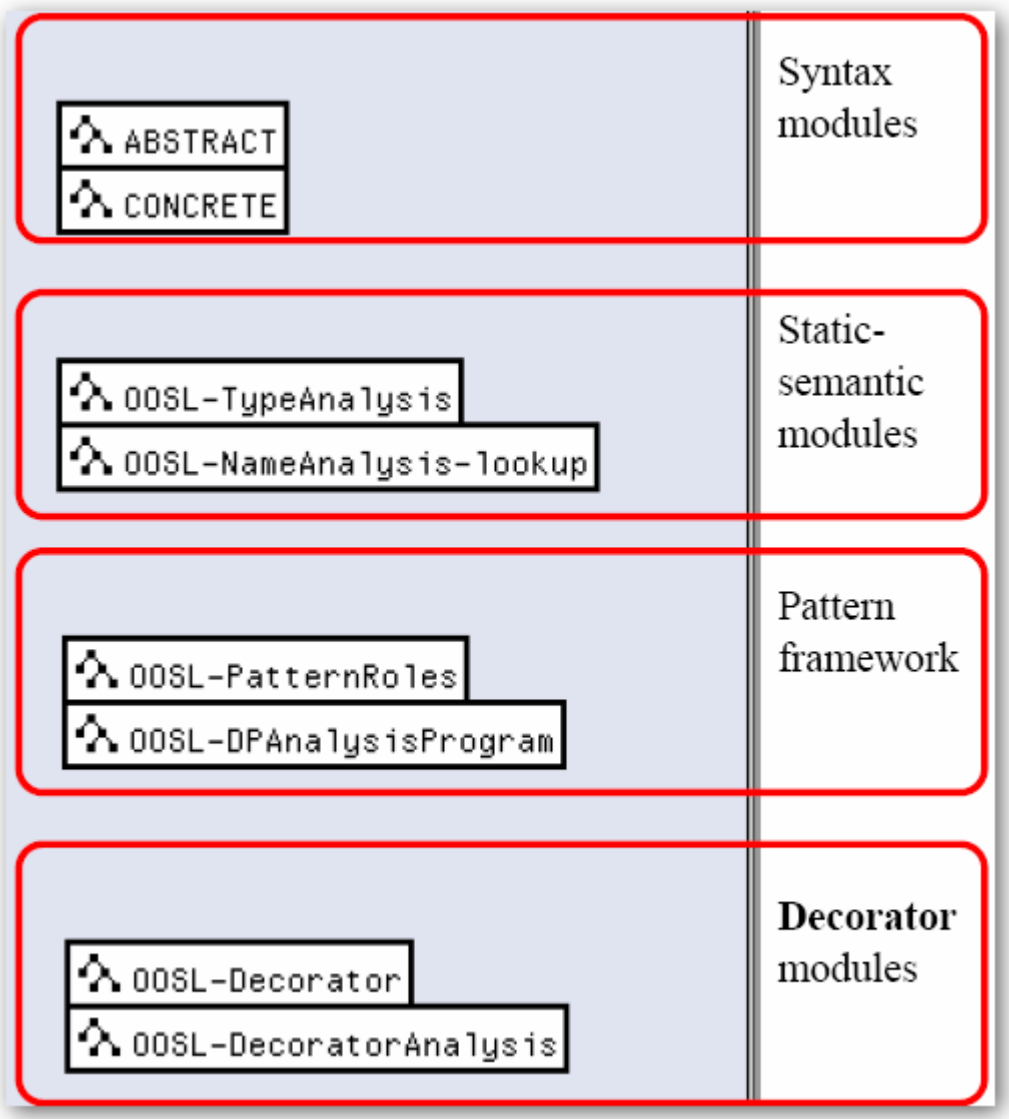


Mapping to program element

Error message dialog, talk about the error reason



DPDOC function module list



一個圖表示一個module，每個module都有數條rule，用來表示不同的rule set

What is APPLAB

- A language-based editor supports the editing of the example program, including syntax-directed editing, text editing with incremental parsing, static-semantic error checking, and semantics-directed editing.

- 最新版是2000年release，且已經無法下載
  - The system is implemented in Simula and runs on SUN Solaris. The system is available for experimental use.
  - DPDOC也沒有tool 可以下載

What is picoJava and language

- picoJava
  - 是一種低成本(大約$10-$50美金)且不需透過JIT編譯器來解譯，便可直接執行Java程式的RISC架構微處理器。這種新的晶片主要用途在於網路電腦，行動電話及呼叫器，掌上型電腦，以及企業週邊。由於 picoJava可直接執行Java Virtual Machine指令集，所以使用picoJava後，Java應用程式的程式碼將較以往小3倍左右，而執行速度則比以往快約5 倍，除了節省記憶體外，也比以往標準 CPU須使用Java解譯器的效果快上 20倍左右。
- **picoJava language**
  - 適用在picoJava處理器的語言，其本上還不能支援介面、例外等等功能
  - includes the major features of an object-oriented programming language: classes, inheritance, variables, qualified access, and reference assignment. For brevity, methods are omitted but the language allows nested class definitions [22, 26] and global variables, in order to show the combination of block structure and inheritance.