

# Architecture-Oriented Programming Using FRED

Markku Hakala<sup>1</sup>, Juha Hautamäki<sup>1</sup>, Kai Koskimies<sup>1</sup>  
Jukka Paakki<sup>2</sup>, Antti Viljamaa<sup>2</sup>, Jukka Viljamaa<sup>2</sup>

<sup>1</sup>*Tampere University of Technology*

*{markku.hakala, csjuha, kk}@cs.tut.fi*

<sup>2</sup>*University of Helsinki*

*{jukka.paakki, antti.viljamaa, jukka.viljamaa}@cs.helsinki.fi*

## Abstract

*Implementing application-specific code conforming to architectural rules and conventions can be tedious. FRED is a tool prototype for architecture-oriented programming that takes an architectural description as a set of programming patterns and provides an interactive task-based programming environment for the architecture. Incorporating adaptive code generation and documentation, the tool provides a convenient way to adopt as well as effectively reuse a framework or architectural standard such as Java Beans.*

## 1. Introduction

Today, software development rarely starts from scratch. Instead, it is largely based on standards, conventions, underlying systems and architectures, often represented by object-oriented frameworks. This paper introduces our notion of architecture-oriented programming, where software development is guided by the underlying architecture. Our research prototype (FRED) allows a system architect (e.g. framework developer) to define architectural rules (e.g. specialization interface of a framework) so that the tool can provide interactive programming assistance in the form of programming tasks.

Typical programming tasks discussed herein include creation of a new class, method or field, refactoring a program element to adhere to some semantic constraint, or adding arbitrary piece of code. As a simple example, creation of a subclass can be seen as a task. When an operation needs to be overridden in that subclass, this can be defined as another task, which occurs only after the creation of the subclass. When dealing with complicated structures, which are typical for software, this task list cannot be adequately expressed by a linear step-by-step list. The mechanism should allow the list of tasks to

evolve during the development process, based on the earlier choices. This brings us to patterns as instruments of piecemeal growth [1].

## 2. Programming Patterns

In order to provide task-based tool support, FRED adopts a model that builds on the notion of patterns as generative descriptions that can be used systematically to produce a number of similar structures. These patterns are essentially a tool-supported formalization of pattern languages [1] for program construction. We shall call them *programming patterns*, or simply patterns.

This fits a more traditional view of a pattern as a description of a recurring problem along with a reusable solution to that problem within a certain context. However, programming patterns should not be confused with design patterns [4], often associated with requirements on the generality of the solution. With the goal of providing programming assistance using patterns in mind, there is no need to make a distinction between a generic solution and a specific solution. For example, specializations of a framework share a similar structure. Thus, even if the framework contained a structure that occurs nowhere else, it still defines a recurring solution that can be presented as a pattern.

Programming patterns are essentially a static concept. A pattern defines a recurring program structure. It can be seen as an algorithm that can be applied in several environments to build a similar structure. Thus, patterns can be used as building blocks in describing framework specialization interfaces as well as other architectural rules and conventions. Application of a pattern results in a pattern instance, essentially a record of bindings that relate fragments of code to the elements of the pattern. This bridge between an abstract solution and source code effectively enables architecture-specific typing checks.

The approach relates to specialization instructions provided by cookbooks [7], and even more to hooks [3].

However, based on the notion of generative programming patterns, we can provide tool support that gradually adapts to the current instantiation context making use of the bindings made so far. This adaptation includes textual documentation. For the specializer point of view, the mechanism reveals itself as informal tasks speaking with application-specific terms as they become defined during the specialization process.

Tasks also provide basis for code generation, and to some extent, verification. However, programming patterns are not templates of code, as we believe that pattern instantiation is not an atomic action but a process. Through the programming process, the tool is able to generate new pieces of code based on existing code and design decisions. Programming patterns provide architecture-oriented programming support that evolves with the development process, rather than relying on single-shot code generation and after-the-fact validation.

### 3. FRED Development Environment

FRED is a prototypical development environment for Java (see Figure 1). It integrates a pattern editor and an incremental programming environment where the patterns can be instantiated incrementally following the tasks generated by the tool.

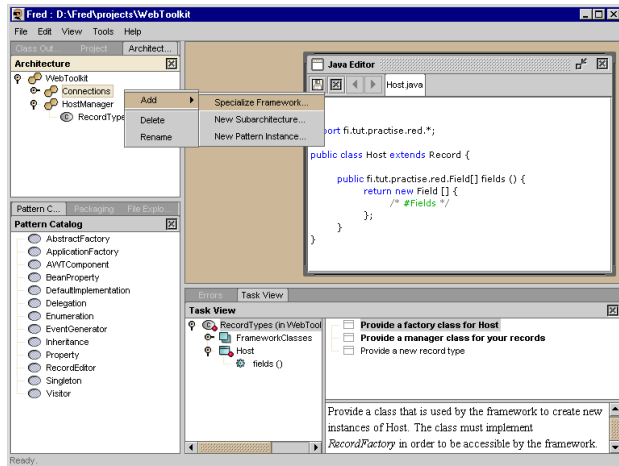


Figure 1. FRED User Interface

Most tasks relate directly to the creation of some program element such as a class or method. The user has the choice of generating the default implementation for the task, implementing the requested functionality by herself, or pointing out an existing implementation. The provided code is checked for any violations against the

pattern definition, such violations reported as tasks as well.

Completing a task may lead to generation of new tasks. Following the ever-changing task list, a point is reached where no more mandatory tasks exists. As a result, an abstract structure defined by the pattern is specialized in a user-defined context. In the context of framework specialization this means that part of the specialization has been finished. Completing all the patterns describing the framework specialization interface this eventually results in a complete specialization.

The tool has been evaluated in the Finnish industry in the context of framework specialization. It can be downloaded from the project web site at <http://practise.cs.tut.fi/fred>. At the time of writing, release 1.1 of the tool is publicly available.

**Acknowledgements.** FRED methodology and programming environment is developed jointly between the University of Tampere, Tampere University of Technology, and the University of Helsinki. The project is funded by the National Technology Agency of Finland (TEKES) and by several software companies. Preliminary results are reported e.g. at [5] and [6].

### References

- [1] C. Alexander, M. Silverstein, S. Angel, S. Ishikawa and D. Abrams, *The Oregon Experiment*, Oxford University Press, 1975.
- [2] C. Alexander, *The Timeless Way of Building*, Oxford University Press, 1979.
- [3] G. Froelich, H. Hoover, L. Liu and P. Sorenson, "Hooking into Object-Oriented Application Frameworks", *Proc. of ICSE '97*, 1997, pp. 491-501.
- [4] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns – Elements of Reusable Object-Oriented Architecture*, Addison-Wesley, 1995.
- [5] M. Hakala, J. Hautamäki, J. Tuomi, A. Viljamaa, J. Viljamaa, K. Koskimies and J. Paakki, "Managing Object-Oriented Frameworks with Specialization Templates", *ECOOP'99 Workshop on Object Technology for Product-line Architectures*, European Software Institute, Spain, 1999, pp.87-98.
- [6] M. Hakala, "Task-Based Tool Support for Framework Specialization", *Proc. of OOPSLA'00 Workshop on Methods and Tools for Framework Development and Specialization*, Tampere University of Technology, Software Systems Laboratory, Report 21, October 2000.
- [7] W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, 1995.