International Conference on Software Engineering 2004

Feature-Oriented Programming and the AHEAD Tool Suite

Don Batory
Department of Computer Sciences
University of Texas at Austin
Austin, Texas, 78712 U.S.A.
batory@cs.utexas.edu

Abstract[1]

Feature Oriented Programming (FOP) is an emerging paradigm for application synthesis, analysis, and optimization. A target application is specified declaratively as a set of features, like many consumer products (e.g., personal computers, automobiles). FOP technology translates such declarative specifications into efficient programs.

AHEAD is a model of FOP that is based on step-wise refinement, which advocates that complex programs can be synthesized from simple programs by incrementally adding features. The *AHEAD Tool Suite (ATS)* supports program development in AHEAD. AHEAD and ATS are among the most advanced models/tools for large-scale program synthesis.

1 Introduction

The future of software engineering lies in a paradigm shift towards automation. Critical to this vision are technologies that synthesize large-scale software systems — their designs, source code, and other related artifacts. Such technologies require advances in the following areas:

- *Domain-Specific Languages* — notations that simplify the specification of program designs;
- *Automatic Programming* — optimizing a program design from a declarative specification;
- *Generative Programming* — automatically mapping a program design to an executable;
- *Compositional Programming* — supporting tools and benchmarks that derive, verify, and analyze properties of generated programs.

A spectacular example of their integration was realized over twenty-five years ago: *relational query optimization (RQO)* (Figure 1). SQL is a prototypical declarative domain-specific language. The *design* of a query evaluation program is specified as a composition of relational algebra operators; relational algebra is a prototypical model of compositional programming. Automatic programming is achieved by query optimizers that rewrite an inefficient expression (or inefficient program design) to a semantically equivalent but more efficient expression (design). Mapping a relational algebra expression to an efficient executable is generative programming.

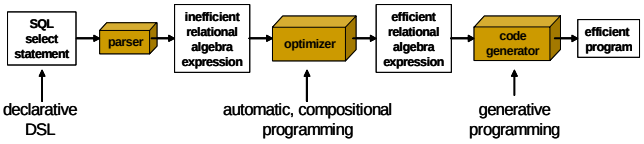

Figure 1: The Relational Query Optimization Paradigm

A "holy grail" of Software Engineering is to replicate the paradigm and success of RQO in other domains. *Feature Oriented Programming (FOP)* is an emerging paradigm that is a generalization of RQO. FOP uses features to describe and differentiate programs in a domain. Feature specifications of programs are typically declarative (e.g., GUIs with check boxes, web pages with menu selections). An FOP model of a software domain is an algebra, where each operator implements a feature. Program designs are represented by expressions — compositions of features/operators — that can be both optimized and translated to efficient executables.

2 AHEAD and ATS

AHEAD (Algebraic Hierarchical Equations for Application Design) is a model of FOP based on *step-wise refinement (SWR)*. SWR advocates that complex programs can be built from simple programs by incrementally adding features. The distinguishing characteristics of AHEAD are:

- features are the primary units of software modularity;
- features encapsulate hierarchically structured program representations (e.g., code, makefiles, documentation, design rules, regression tests);
- source code that is encapsulated by features corresponds to "cross-cuts" (i.e., fragments of multiple classes);
- features are operators. An application is defined by a composition of such operators;
- features are composed by merging corresponding program representations, thereby synthesizing consistent

1. This work was supported in part by the U.S. *Army Simulation and Training Command (STRICOM)* contract N61339-99-D-10.

representations of the target application.

The *AHEAD Tool Suite (ATS)* is a set of Java-based tools that support program development in AHEAD. ATS is being used in the construction of fire support simulators for the U.S. Army [3] and in the synthesis of ATS itself [2]. That is, ATS tools have been bootstrapped: each ATS tool has its own AHEAD expression and the tool itself is synthesized from this expression. A novelty of ATS is that it supports a superset of Java, i.e., Java with refinement declarations and state machines. The set of features that can be added to Java is expressible in AHEAD, so that ATS really is a product-line, where each member of this product line is a tool suite for a particular dialect of Java.

A key property of AHEAD is that it is *scalable*: there is no limitation on the size of the programs or the number of representations that it can synthesize. ATS, for example, currently exceeds the equivalent of 250K LOC in Java, yet its specifications are a few lines of FOP expressions. The set of program representations that ATS can compose include: XML documents, programs written in extended Java, state machines, language grammars, design rules, and expressions.

3 ATS Demonstration

ATS consists of a variety of artifact composition tools, including:

- **`jampack`** and **`mixin`** — code composition tools
- **`unmixin`** — a tool that propagates changes made to composed source code back to its original feature counterparts
- **`baliComposer`** — a BNF grammar composition tool
- **`xmlComposer`** — a tool that composes XML documents and their refinements
- **`jedi`** — a language-extensible version of javadoc
- **`modelexplorer`** — a graphical tool for composing, browsing, and building feature-based designs

New AHEAD tools are coming on line, including a composer for Java byte codes, and a tool to generate declarative GUI front-ends from feature models.

4 Relationship to Other Work

AHEAD refinements have a long history, originating in collaboration-based designs [8], mixins [4], and mixin-layers [6][4]. AHEAD is an example of the *Aspect-Oriented Programming (AOP)* paradigm [5] in that features are implemented by cross-cuts (i.e., updates to multiple classes). The novelty and power of AOP is in *quantification* — a predicate that defines where advice is to be inserted in a program. In contrast, the use of quantification in AHEAD is comparable to traditional OO frameworks. That is, adding a feature to a framework requires certain methods and classes to be extended. AHEAD takes this idea to its logical conclusion: instead of having two different levels of abstraction (e.g., the abstract and concrete classes), AHEAD allows arbitrary numbers of levels, where each level implements a particular feature.

AHEAD is also an example of the *Multi-Dimensional Separation of Concerns (MDSOC)* paradigm [7], where modularity can be understood as a multidimensional space of concerns. We have shown program specifications that could be $O(k^n)$ features long have short and easy to understand specifications of length $O(kn)$, where n is the number of dimensions and k is the number of features per dimension [2]. That is, systems with exponential complexity have quadratic-length specifications. AHEAD differs from MDSOC in that AHEAD defines an algebraic model of program specifications and compositions.

5 References

[1] D. Batory, J.N. Sarvela, and A. Rauschmayer, “Scaling Step-Wise Refinement”, *International Conference on Software Engineering*, 2003.

[2] D. Batory, J. Liu, J.N. Sarvela, “Refinements and Multi-Dimensional Separation of Concerns”, *ACM SIGSOFT 2003 (ESEC/FSE2003)*.

[3] D. Batory, D. Brant, M. Gibson, M. Nolen, “ExCIS: An Integration of Domain-Specific Languages and Feature-Oriented Programming”, *Workshop on New Visions for Software Design and Productivity: Research and Applications*, Vanderbilt University, Nashville, Tennessee, December 13-14, 2001. Also, http://www.cat.utexas.edu/

[4] M. Flatt, S. Krishnamurthi, and M. Felleisen, “Classes and Mixins”. *POPL 98*, San Diego, California. ACM, New York, NY, 171-183.

[5] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, “Aspect-Oriented Programming”, *ECOOP 1997*, 220-242.

[6] Y. Smaragdakis and D. Batory, “Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs”, *ACM Transactions on Software Engineering and Methodology*, March 2002.

[7] P. Tarr, H. Ossher, W. Harrison, and S.M. Sutton, Jr., “N Degrees of Separation: Multi-Dimensional Separation of Concerns”, *ICSE 1999*.

[8] M. VanHilst and D. Notkin, “Using role components to implement collaboration-based designs”. *OOPSLA 1996*, ACM Press, 359-369.