

# Package-Oriented Programming of Engineering Tools

Kevin J. Sullivan, Jake Cockrell, Shengtong Zhang, David Coppit  
Computer Science Department  
Thornton Hall  
University of Virginia  
Charlottesville, VA 22903 USA  
{sullivan, emc5a, sz2n, dwc3q}@virginia.edu  
+1 804 982 2202

## Keywords

COTS, integration, package-oriented programming, reuse, wrappers, mediators, architecture, software engineering

## ABSTRACT

We present an innovative fault-tree analysis tool that we developed using a novel software architectural style that we call *package-oriented programming* (POP). The tool is largely implemented using multiple, tightly integrated, shrink-wrapped software packages as components. These packages are integrated using custom-coded *integration mediators* into a cohesive superstructure that provides much of the functioning required of many sophisticated modeling and analysis tools. In addition to serving as a proof of concept, the tool provides a test-bed for further research on the approach.

## PACKAGE-ORIENTED PROGRAMMING

Sophisticated, specialized software tools are needed in many engineering activities and disciplines. Unfortunately, it is hard and expensive to build such tools quickly and inexpensively. One result is that when the market is small for a tool supporting a new technique, it is either prohibitively expensive to develop the tool or it ends up functionally inadequate. Lack of tool support then inhibits the use and development of the technique. We need ways to build sophisticated tools quickly and inexpensively.

In our research program, we are exploring a new technique for building high-value, industrially viable tools quickly and at low cost using a novel architectural style that we call *package-oriented programming* (POP). The POP style is based on the use (or reuse, if you prefer) of massive, architecturally coherent, interactive components as basic building blocks [3]. In current work, we are using modern, commercial off-the-shelf (COTS), shrink-wrapped software packages as these basic building blocks.

Permission to make digital/hard copies of all or part of this material for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copyright is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires specific permission and/or fee

ICSE 97 Boston MA USA

Copyright 1997 ACM 0-89791-914-9/97/05 ..\$3.50

The idea of reusing package software is hardly new. However, innovations appearing in recent packages, such as those of Microsoft Office, have created promising new architectural possibilities. We have demonstrated that we can implement powerful, cohesive software tools using tightly integrated systems of such packages to provide the bulk of the function needed in such tools.

This approach to software construction appears to be novel and interesting enough that it suggests new thinking on a range of software engineering issues. How does the use of massive, executable packages as components affect modularity, reuse, the software lifecycle, the construction of documentation, testing, and evolution? The POP style also touches on issues of increasing concern to a growing number of people, especially the trend toward system development through the "wrapping" and integration of only partially known and independently developed COTS.

## POP TOOLS

We chose software tools as a domain in which to explore POP architecture because there seemed to be a good match between the requirements of many software tools and the availability of particular large-scale components. In particular, many tools seem to split into small specialized modeling and analysis kernels that are surrounded by vast superstructures supporting such functions as technical drawing, text editing, data entry and management, report generation, and so on. Attacking the essence of the tool construction problem thus demands an effective attack on the problem of building tool superstructures.

In earlier work, Garlan et al. [1] described the difficulties that they faced when attacking a similar problem through an approach based on the reuse of large-scale components. The problem was that the selected components turned out to be architecturally incompatible, even though they were ostensibly designed to be reusable.

We noted claims of not only architectural compatibility but also ease of specialization and integration made by the vendors of a range of shrink-wrapped software packages.

We hypothesized that these components supported large-scale reuse and that they would shed light on how to design components for such reuse. In particular, we thought we might build tool superstructures by specializing and integrating just a few of these massive components in simple and straightforward ways. We believe that we have demonstrated the potential of the POP architectural style.

#### A FAULT-TREE ANALYSIS TOOL

Our demonstration application is a tool for fault-tree analysis [7] that support novel modeling and analysis techniques developed by Dugan and her students [2]. We built our tool superstructure by specializing and integrating packages that support technical drawing, text editing, and data management. In particular, we used Visio Corporation's Visio drawing program and Microsoft's Word and Access programs.

We specialized these packages using their built-in specialization mechanisms. We integrated them with each other and with the computational core (which we ported from Dugan's tool) by writing a program in Microsoft's Visual C++ that essentially functions as a large-scale integration mediator [4,5,6]. This mediator presents an interface to the user for controlling the tool. We use information hiding, integrity-maintaining interfaces, or "wrappers," implemented as C++ classes, to abstract the packages. These wrappers use OLE Automation (a Microsoft-defined, late-bound, remote procedure call mechanism now supported by many software packages) to manipulate the underlying packages.

#### EVALUATION

Our experience has been positive in that we have been able to build a sophisticated tool with a significant level of useful functioning quickly and at low cost. We believe in particular that we have achieved a new level of integration of COTS shrink-wrapped packages employed as reusable components.

However, some important issues remain unresolved. One problem is that the packages we have used do not support wrapping, or encapsulation, adequately. The result is that users can execute package functions that violate integrity constraints imposed by the tool of which the packages are components. In essence, users have undue access to the "implementation details" of the tool. We need mechanisms that will permit designers who want to use packages as components to selectively hide and restrict package functions whose execution would violate such constraints. Of course, we want to be able to do this while leaving the desired, powerful functions of the packages available to the tool user.

Other problems remain as well, of course; but the demonstrated promise of package-oriented programming

appears to be sufficient to warrant ongoing investigation. We can see emerging, vaguely, powerful new concepts of modularity that just might provide a basis for profitable software construction through large-scale, systematic reuse in the not-too-distant future.

#### ACKNOWLEDGEMENTS

This work was supported by the National Science Foundation under grants CCR-9502029, CCR-9506779, and MIP-95-29258; by the Defense Advanced Research Projects Agency (DARPA) through Rome Labs AFB under grant F30603-96-1-0314; and with a software grant from the Visio Corporation.

#### REFERENCES

- [1] Garlan, D., R. Allen and J. Ockerbloom, "Architectural Mismatch: Why Reuse is so Hard," *IEEE Software*, vol. 12 No. 6, Nov. 1995, pp. 17-26.
- [2] Gulati, R. and J.B. Dugan, "A Modular Approach for Analyzing Static and Dynamic Fault Trees," 1997 Proceedings of the Annual Reliability and Maintainability Symposium, Philadelphia, Pennsylvania, Jan. 1997, pp. 57-63.
- [3] Sullivan, K.J., and J.C. Knight, "Experience Assessing an Architectural Approach to Large-Scale Systematic Reuse," Proceedings ICSE 18: Eighteenth International Conference on Software Engineering, Berlin, Germany (March 1996), pp. 220-229.
- [4] Sullivan, K.J., *Mediators: Easing the Design and Evolution of Integrated Systems*, Ph.D. Dissertation, University of Washington Department of Computer Science and Engineering, Seattle, WA, 1994. Also available as University of Washington Department of Computer Science and Engineering Technical Report 94-08-01.
- [5] Sullivan, K.J. and D. Notkin, "Reconciling Environment Integration and Evolution," *ACM Transactions on Software Engineering and Methodology* vol. 1, no. 3, July, 1992.
- [6] Sullivan, K.J., I.J. Kalet and D. Notkin, "Mediators in a Radiation Treatment Planning System," *IEEE Transactions on Software Engineering*, Vol. 22, No. 6, August, 1996, pp. 563-579.
- [7] Veseley, W.E., F.F. Goldberg, N.H. Roberts, and D.F. Haasl, *Fault Tree Handbook*, U.S. Nuclear Regulatory Commission, NUREG-0492, Washington DC (1981).