Layered, Server-based Support for Object-Oriented Application Development

Guruduth Banavar  Douglas Orr  Gary Lindstrom
Department of Computer Science
University of Utah, Salt Lake City, UT 84112 USA
{banavar, dbo, lindstrom}@cs.utah.edu

Abstract

This paper advocates the idea that the physical modularity (file structure) of application components supported by conventional OS environments can be elevated to the level of logical modularity, which in turn can directly support application development in an object-oriented manner. We demonstrate this idea through a system-wide server that manages the manipulation of such components effectively. The server is designed to be a fundamental operating system service responsible for binding and mapping component instances into client address spaces.

We show how this model solves some longstanding problems with the management of application components in existing application development environments. We demonstrate that this model's effectiveness derives from its support for the cornerstones of OO programming: classes and their instances, encapsulation, and several forms of inheritance.

1 Introduction

In a traditional application development environment such as UNIX, application components ultimately take the form of files of various kinds — source, object, executable, and library files. Entire applications are typically built by putting together these components using inflexible, and sometimes ad-hoc, techniques such as preprocessor directives and external linkage, all managed via makefile directives.

It is also natural for application developers to generate components corresponding to incremental changes to already existing application components, especially if they subscribe to the software engineering principle known as “extension by addition.” This principle holds that it is better to extend software not by direct modification, but by disciplined addition of incremental units of software. Advantages of “extension by addition” include better tracking of changes and more reliable semantic conformance by software increments. Most importantly, the increments themselves have the potential to be reused in similar settings.

This perspective leads one to conclude that traditional OS environments inadequately support component manipulation and binding for modern application development. Object-oriented (OO) programming offers a potential solution to this inadequacy. In OO programming, inheritance is a mechanism that aids in the effective management of software units and incremental changes to them. Indeed, in advanced OO languages, increments as well as base components have independent standing (e.g., “mixins”). Other aspects of OO programming, notably encapsulation, have demonstrated benefits to large-scale software development via enhanced abstraction. There is much to gain from supporting these features within the infrastructure of an application development environment, beyond whatever support is provided by the languages in which application components are written.

In this paper, we demonstrate a principled, yet flexible, way in which to effectively construct applications from components. This facility is orthogonal to makefiles, and does not impose new techniques for building individual application components. Instead, it relies on the idea that the physical modularity of traditional application components (i.e., files) can be endowed the power and flexibility of logical modularity. Such logical modules can then be manipulated using the concepts of compositional modularity, where first-class modules (defined in Section 3.1) are viewed as building blocks that can be transformed and composed in various ways to construct entire application programs. Individual modules, or entire applications, can then be instantiated into the address spaces of particular client processes. Compositional modularity has a firm foundation [4], and has been shown to be flexible enough to support several important effects and styles of object-oriented programming [2].

This approach has other advantages besides making system building more principled and flexible. First, it