Chapter 3
Components and Frameworks in the Cloud Era

Dino Konstantopoulos
The MITRE Corporation, USA

Mike Pinkerton
Northrop Grumman Corporation, USA

Eric Braude
Boston University, USA

ABSTRACT

The emergence and popularity of Cloud computing could not have happened were it not for the massive amount of underlying software reuse, as reuse of services and applications is what saves development dollars and allows IT infrastructure to adapt more nimbly to the changing environment. Software fosters two reuse syntaxes: Sharing and reuse of human-readable source code (open source), or sharing and reuse of machine-readable binary code (proprietary). Software also caters to two reuse semantics: where the software is either prepackaged as components or as frameworks. In this chapter, the authors examine software reuse in the Cloud. By categorizing it in these dual categories, they glimpse the practices that prime Cloud-based software for massive reuse.

INTRODUCTION

Source code, binary code, components, and frameworks are the currencies and exchange mechanisms of software reuse. This reuse occurs at different points on the production line of new software. For components, it’s early in the production line, and for frameworks it is late in the production line of new software: Before a software program can be executed, it needs be converted from source code to binary code, and so sharing and reuse can occur upstream of the software production line (as source code), when the software may not even be complete, or downstream, when it is packaged and ready for download and execution (as binary code). Components are mature products on their own right but they’re most often leveraged early in the software production line of other software applications: The application is built around them. Frameworks on the other
hand host software products as add-ins, and these add-ins extend their host’s functionality. And so frameworks represent the end of the production line for those software products. Moreover, much like there are two currencies, or syntaxes, of software reuse (source code and binary code), and two exchange mechanisms, or semantics (components and frameworks), there are also two communities of interest that promote the reuse: The free software and open source movements promote “no-cost” sharing of source code, while the proprietary world promotes market mechanisms through exclusive rights of ownership to binary code, protected by licenses and copy protection mechanisms. Currencies, exchange mechanisms, and communities of interest are well represented in today’s public Clouds. Let’s examine the exchange mechanisms a bit closer. It is important to note that when we talk about sharing and reusing code, it is in the context of (third party) developers sharing and reusing code originally written by other developers, in order to incorporate original functionality into a new program or extend the original functionality with new features. Software users don’t really reuse or share software: They “use” it in the sense that they execute it. And so the exchange mechanism is between developers. When source code is exchanged between developers, it is packaged in the form of a “source tree” that can be loaded into a development environment and compiled into a binary executable. Source code functionality can be easily uncovered by reading the source code. It is easily modifiable, and so there need not be a standard architecture for the source tree. Third-party developers extend it with new functionality in various and multifaceted ways. In contrast, when binary code is reused, it is packaged in standard architectures, which are either software components or software frameworks, abstractions in which common binary code providing generic functionality can be most easily leveraged, extended, and specialized by third-party developers. Let’s take a closer look at the technical differences between components and frameworks.

When developers reuse software components, they write code that calls the components’ code. Software components are often called component “libraries”, and software that uses component libraries is often known as “component-oriented architectures”. In contrast, when third-party developers reuse software frameworks, they write “callback” code that is called by framework code. In other words, third-party developers implement interfaces dictated by the architecture of the software framework. When the framework is executed, it calls the new third-party implementations which it discovers by a registration mechanism specified by the framework’s developers. And so, framework reuse consists of both implementation and design reuse, while component reuse is usually just an implementation reuse. Perhaps the most notable example of this distinction consists in how software programs are hosted today: One can either write programs that are self-hosted and run as standalone components on an operating system like Microsoft Windows, Linux, or Apple iOS, or one can write programs that are hosted by browsers like Microsoft Explorer, Firefox, or Apple Safari. In the latter case, the programs are reusing very popular frameworks that consist of implementation (internet browsers) and architecture (for writing add-ins and enhancing the browser’s functionality). We first examine the origins of these two types of software abstractions, and then we concentrate on their best principles, in the sense that they facilitate software reuse on the Cloud and guide the design of the building blocks of Cloud computing.

**A SHORT HISTORY OF COMPONENTS**

The original vision of software reuse was based on components: Units of software re-use across independent parties. Component-oriented soft-