A Flexible Approach to Decentralized Software Evolution

Peyman Oreizy
444 Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
peymano@ics.uci.edu

1 INTRODUCTION

Reducing the costs and risks associated with changing complex software systems has been a principal concern of software engineering research and development. One facet of this effort concerns decentralized software evolution (DSE), which, simply stated, enables third-parties to evolve a software application independent of the organization that originally developed it. Popular approaches to DSE include application programming interfaces or APIs, software plug-ins, and scripting languages.

Application vendors employ DSE as a means of attracting additional users to their applications—and, consequentially, increasing their market share—since it opens up the possibility that a third-party modified version of the application would satisfy the needs of end-users unsatisfied with the original version. This benefits everyone involved: the original application vendor sells more product since customization constitutes use; third-party developers deliver a product in less time and with lower cost by reusing software as opposed to building it from scratch; and customers receive a higher quality product, customized to suit their needs, in less time and with lower cost. By increasing the opportunity for buying and customizing software instead of building it from scratch, DSE attacks Brook's "essential" difficulties of software development [1].

2 CURRENT APPROACHES

Existing software applications use a number of different approaches to DSE. The characteristic approaches are: APIs, software plug-ins, scripting languages, component-based architectures, event-based systems, and source code. Each of these is briefly summarize below.

APIs: An API consists of a collection of functions and data types that third-party add-ons use to augment the functionality of the host application.

Scripting languages: A scripting language combines a domain-specific language for specifying add-on behavior and a runtime environment for executing those behaviors. Scripting languages commonly leverage special properties of the application domain to facilitate add-on development, especially for non-programmers. Spreadsheets, macro systems, and programming by demonstration systems are essentially scripting languages optimized for particular purposes.

Plug-ins: A plug-in is a behavioral placeholder in the host application that third-party add-ons fill to customize or specialize its behavior. The host application expects plug-ins to adhere to a particular functional interface and behavior. Netscape's Communicator, for example, uses plug-ins as a means of allowing third-parties to support new media types.

Component-based architectures: In a component-based approach, the host application consists of an explicit collection of functional components that communicate with one another through pre-specified interfaces. Third-parties customize the host application by implementing new components that invoke functions exposed by existing components and/or replacing existing components with ones that adhere to the same functional interface.

Event-based systems: In an event-based approach, the host application and its add-ons communicate indirectly by passing messages through an event mechanism. The event mechanism acts as a communications mediator by localizing and encapsulating binding decisions among the host and its add-ons. This allows binding decisions to be changed independently of implementation, thereby reducing the coupling between the host and its add-ons.

Source code: In a source code-based approach, the host application vendor distributes the application's source code to third-party add-on developers, enabling them to modify any aspect of its functionality.

Combining approaches: A host application typically employs several DSE approaches as a means of combing their benefits and avoiding the limitations of any one approach. For example, Emacs combines: a scripting language for programming add-ons, an API for exposing internal functions to add-on developers, source code for a subset of Emacs implemented in the scripting language, a plug-in mechanism based on Lisp-style hook functions, and an integrated documentation mechanism.

3 PROBLEM STATEMENT

Although DSE has the potential to significantly reduce the costs and risks associated with software development, two problems curtail its effectiveness. One, the differences between current approaches are poorly understood, making it difficult for software practitioners to select the approaches most appropriate for their needs. Two, current approaches expose a limited subset of the types of change possible and thereby preclude potentially desirable changes. We have recently completed a survey of DSE approaches (see [2]) that, in part, addresses the first problem. Our thesis aims to address the second.

APIs and scripting languages enable third-parties to augment the functionality of the host application, but preclude replacement, removal, or interposition of functionality. Plug-ins enable third-parties to provide alternate implementations of particular host application modules, but preclude the interposition of functionality. Component architectures enable third-parties to augment, replace, interpose, and remove functionality, though the direct coupling of components places severe restrictions on these changes. Event-based systems reduce such coupling, but since the internal structure of the host application is not exposed, changes are confined by the host application's event interface. Source code approaches enable third-parties to change any aspect of the host application's functionality, but at a very high cost—the low level of abstraction at which changes are made confounds analysis and add-on composition.

A DSE approach that supported broader types of change would increase the likelihood that an existing application could be customized to suit user needs.

4 HYPOTHESIS AND APPROACH

Our examination of the capabilities and inherent limitations of current DSE approaches has lead us to the following insights and hypotheses.

1. The lack of flexibility exhibited by current approaches stems from two sources: (a) the internal structure of the host application is not exposed by API, plug-in, scripting language, and event-based approaches, which circumscribes large classes of change; and (b) the high coupling among the host and add-on components in a component-based approach severely restricts the types of change possible.

   Our hypothesis is that a new DSE approach that combines a component-based approach to expose the internal structure of an application with an event-based approach to reduce component dependencies will engender broader types of change than currently available.

2. Current DSE approaches are fragile and error-prone because interactions between third-party add-ons are not modeled and hence cannot be verified when changes are made.

   Our hypothesis is that deploying a system model with the application that (a) contains the semantics necessary to detect inconsistencies between add-ons, and (b) is used as the basis for making changes to the application can be used to verify particular system properties whenever changes are made.

Our approach to developing and validating a flexible approach to DSE based on our insights and hypotheses is described below.

1. Survey current approaches to DSE and develop a theory for evaluating and comparing the types of change afforded by each approach.
2. Based on our theory, develop a DSE approach that supports novel types of third-party change using an explicit, malleable system model.
3. Develop a reference architecture for a toolset that supports our proposed approach. In accordance with the reference architecture, implement a reusable toolset that supports our proposed approach.
4. Demonstrate and evaluate the utility of our proposed approach in three ways:

   4-1. Demonstrate that our approach affords broader types of change than that of current approaches by (a) enumerating the types of change supported by previous approaches, (b) emulating these types of change using our proposed approach, and (c) identifying at least one type of change that is supported by our proposed approach but not by others.

   4-2. Use a set of exemplar changes to a simple hypothetical application to conceptually motivate and demonstrate the increased flexibility of our proposed approach as compared to current approaches.

   4-3. Demonstrate and empirically evaluate our proposed approach by applying it to several small systems (approximately 10K source lines of code) and to a portion of large legacy system (approximately 1M source lines of code). In the context of these applications, discuss the practical benefits (i.e., in terms of flexibility) attained by our proposed approach.

5 EXPECTED CONTRIBUTIONS

Our thesis expects to make the following contributions:

- A theory for evaluating and comparing the flexibility of DSE approaches.
- A system model-based approach to DSE that supports a broader class of changes than previously possible.
- A reference architecture for implementing a system model-based approach to DSE.
- A policy neutral mechanism for governing independent changes to a deployed software system that uses system models.

REFERENCES

1. Fredrick P. Brooks Jr. *The Mythical Man Month.* Anniversary Edition. Addison-Wesley Publishing Company, Inc. 1995.
2. Peyman Oreizy. Decentralized Software Evolution. *UCI-ICS Technical Report 98-42*, Department of Information and Computer Science, University of California, Irvine, December 1998.