Using Extensible Languages and Plug-ins to Provide Versatility to Software

Roberto Silveira Silva Filho
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3430 USA
rsilvafi@ics.uci.edu

Abstract

This paper surveys the current use of extensible languages to design software that is versatile, able to be extended with new functionality in order to cope with requirements from different application domains. Our experience with YANCEES, a versatile publish/subscribe system is presented to illustrate the use of this extensibility technique.

1 Introduction

Software system should evolve to accommodate new requirements demanded by the applications it supports. According to Lientz and Swanson [1], the software maintenance phase (which includes software adaptation, fault repair and functionality addition and modification) represent 50 percent of the total software cost, having 65% of this cost directly related to new implementation requirements. Hence, the use of techniques during software design and development, that improves software maintainability and evolution, have a great impact in reducing the total cost of software [2].

In spite of this fact, most software today are not designed to cope with software evolution. This is not a surprise. According to Parnas [3], the majority of software systems are not designed for change. They are built to solve specific and well defined problems, which ends up hindering their ability to evolve due to its high costs of maintainability. Publish/subscribe infrastructures are not an exception to this observation.

On the light of this problem, Parnas proposes that, in order to support evolution and variability, software must be designed and implemented not as a single program, but as a family of programs that can be extended and contracted according to different application needs. This approach is motivated by his observation that software change is usually driven by the need to support: (1) Extensions motivated by social, organizational or technological evolution; (2) New and different hardware configurations; (3) Differences in its input and output data, while its function is preserved; (4) Different data structures and implementations due to differences in the available resources; (5) Differences in the size of data input and output; (6) And the need of some users of only a subset of features provided by the software.

Hence, according to Parnas, software can be considered **general** if it can be used, without change in a variety of situations; whereas it is considered **flexible** if it is easily changed to be used in a variety of situations [3],. Our notion of versatility is based on this original definition of flexibility, and incorporates additional design properties that are important to current pub/sub infrastructures. Parnas observations, even though still current and valid did not explicitly mention nor predict other kinds of concerns such as runtime (dynamic) change, module (or component) distribution and usability. The first two issues are central to distributed systems and publish/subscribe middleware, whereas the latter is essential for the acceptability and usefulness of the proposed approaches. Based on this motivation, we proceed to present our concept of versatility.

2 Software Versatility

According to the Cambridge Advanced Learner's Dictionary, versatility is the ability "to change easily from one activity to another" or to be "able to be used for many different purposes." In the context of software engineering, versatility can be defined as the ability of a computational system to serve multiple purposes or to accommodate the requirements of different use situations.

In light of the above discussion, we proceeded to research ways of providing and maintaining good software engineering qualities that allows the customization, expansion and contraction of software in a usable way. We adopted the term *versatility* in order to embrace an extensive set of qualities. Moreover, we sought a new term that could imply that that these qualities applied not only to technical needs but to the varying needs of human stakeholders and application workplace settings. Hence, from a software engineering perspective, and more specifically in the context of middleware and publish/subscribe architectures, versatility comprises the following requirements.

2.1 Examples

YANCEES brings versatility to middleware by using extensible languages and plug-ins [4]

The Aspect Oriented Markup Language and its support for plug-ins [5]

Monarch employs extensible languages and programmable components, providing a toolkit for building different application monitoring systems.

XADL and Archstudio provide a set of tools for supporting architecture-driven software engineering. XADL is extensible.

In all of those systems, we see the same motivation, the need for extensibility in the language driving the use of extensible languages. This extensibility motivates a software architecture that supports expansion and configuration of its sets of features, supporting the implementation of those extensions. This paper proposea a general design model to support the extensibility demanded by those applications. It combines extensible languages with smart parsers and plug-ins.

Acknowledgements

Here comes the acknowledgement to the grant and others who have helped and/or sparked ideas.

References

[1] B. P. Lientz and E. B. Swanson, *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations (Ch. 27)*. Rading, MA: Addison-Wesley, 1980.

[2] I. Sommerville, *Software Engineering (6th Edition)*, 2001.

[3] D. L. Parnas, "Designing software for ease of extension and contraction," presented at 3rd international conference on Software engineering, Atlanta, Georgia, USA, 1978.

[4] R. S. Silva-Filho, C. R. B. d. Souza, and D. F. Redmiles, "Design and Experiments with YANCEES, a Versatile Publish-Subscirbe Service," Institute for Software Research, Irvine, CA UCI-ISR-04-1, April 2004 2004.

[5] C. V. Lopes and T. C. Ngo, "The Aspect Oriented Markup Language and its Support of Aspect Plugins - UCI-ISR-04-8," UC, Irvine, Irvine October 2004 2004.