Limits of Traditional SE in the design and development of a pub/sub product line -- Why modularization is hard?

Roberto Silveira Silva Filho, David F. Redmiles

Donald Bren School of Information and Computer Sciences
University of California, Irvine
Irvine, CA 92697-3430 USA

{rsilvafi, redmiles}@ics.uci.edu

Abstract. According to Baldwin and clark, the modularization of a system is a process that requires a long interactive process of re-design of a system based on the understanding of the domain it supports. The publish/subscribe domain is not different. Over the past thee years, we have been working on the support of an extensible and configurable publish/subscribe infrastructure which original goal was the support of different applications, more specifically, groupware tools such as peer-to-peer file sharing, application monitoring, awareness and mobile computing. This paper reveals some of the challenges faced, and the limitations of current OO technology in the design of such systems. It also discusses the trade-offs involved in using more recent programming language and software engineering strategies such as AOP and its impact on the design of such systems.

Initial motivation

This paper focus on software design and how the understanding of the relation between the variability dimensions can lead to better design and the proper choice of software engineering technologies that will realize this design.

We build upon Robsemblum and Wolf's design framework, and go deep in analyzing the inter-dependencies that exist between those dimensions and their implications on the implementation of extensible pub/sub infrastructures. Our goal is to provide a generalized model that can lead practitioners in choosing which software techniques employ, understanding their impact in the variability of the system.

Our hypothesis is that a proper design that accounts for the interdependencies existing in the problem at hand, can better inform designers and engineers on the design of product line architectures, allowing them to better select the appropriate software engineering technology.

As noted by Brooks, software techniques have focused too much on the addressing of the accidental aspects of its development, paying and not enough attention to the essence of the software engineering. Our analysis focus on the initial design of product line middleware systems and how the analysis of existing systems can help us understand and generalize the problem before the further implementation.

A model helps us reason about the problem without going into its accidental aspects, helping designers to focus on addressing the accidental aspects of software. Our model is constructed based on the analysis of three open source projects in different areas: Siena, JMS and an extensible framework. For such, we use well-known design analysis tools such as DSMs.

Motivate with the analysis of existing systems. Talk about experiences using YANCEES, an infrastructure that uses traditional OO techniques to partially achieve this goal.

I may need a species of design pattern catalog that describes why and when to use some software engineering technique, in the context of publish/subscribe, based on the variability dimensions that one needs to address in our model.

Our initial design

It is important to separate the design model from the implementation or target programming paradigm. A programming paradigm such as OO, or Functional programming may require different mappings from this high-level model.

Micro-kernel style, learned from previous experiences from OS. Extensible languages and plug-ins from compilers.

Hidden dependencies

What we missed: the hidden interdependencies of the publish/subscribe design dimensions. What do I mean by hidden inter-dependencies?

Desing and specification are important to understand the system to be built before engaging in the implementation. The value of design is to prevent pitfalls that may exist in the implementation phase, to understand the fundaments of the system and to allow the reasoning about the problem beforehand. A good design model allow the

restructure of the solution in order to prevent future issues, to cope with variability and evolution, in order to minimize the maintenance costs.

Insights: in the design of a generalized solutions:

A previous analysis of the variability dimensions can identify dependency circles which represent the main challenges in the generalization and modularization of a solution.

Based on these cycles one can work on selecting the appropriate software engineering technique to address the problem.

This selection must be based on trade-offs and limitations of each technique.

Mapping from design to existing software technologies

Once the problem is understood with its circular dependencies, modules and variability points, one can think of mapping the system to existing techniques (including variability realization techniques).

The problem with OO – why modularization (separation of concern) is hard

A *functional concern* is a piece of functionality provided for a client. The client may be the user, another program, or even another component of the same program. In an ideal decomposition, a functional concern could be isolated to a single module. However, standard object-oriented decomposition can prevent the programmer from encapsulating a piece of functionality in a single module. In some cases, a functional concern may be spread among several objects. In other cases, a large object may deal with several different functional concerns, which are grouped into one object because they all operate on a single set of data. Because mainstream object-oriented languages require a module to be an object, they cannot modularize many functional concerns well.

However, not everything is doomed to failure. OO can be used with success, allowing users to take advantage of its main capabilities (Gamma Design Patterns, Extension, Inheritance and so on) if the problem at hand is suitable for modularization.

Now, I need to contextualize this to the case of publish/subscribe systems.

Future work

References