System Family Engineering for Leader Election in Ring

by

Wei Shi

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Masters of Computer Science

Ottawa-Carleton Insititute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

10th October 2003

The undersigned hereby recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

Title **System Family Engineering for Leader Election in Ring**

submitted by

Wei Shi

Ottawa-Carleton Insititute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

10th October 2003

1 Chapter 1 Introduction

1.1 The context (1.5pages)

The problem of current software engineering is that:

- Software developers usually end up building one concrete software system.

- Sometimes programmers build the software without really know how they got there.

In the first case, each time software developers build a system, they start from the very beginning, even there might be a lot of similar products existing. This usually leads to great waste. In the second case, due to the lacking of proper design knowledge, either bad decision is made which might course the entire system disaster, or very limited modification and variability kill the possibility of reuse. And these make software maintenance and evolution very difficult and costly to perform.

When we compare software systems, we usually find 60-to-70 commonality from one software application to another. [3] This includes code, design, functional and architectural similarities. At all levels of development from requirements specifications to code, there are components that by the nature of implementing tasks and representing information on a computer must appear over and over again in software applications.In order to improve the current situation of Software Engineering, we bring ***Software reuse*** into the context.

Software reuse is the process of creating software systems from predefined software components [**?**]. Reuse can cut software development time and costs. The major reasons for practicing reuse are to: Increase software productivity; shorten software development time; improve software system interoperability; Develop software with fewer people; move personnel more easily from project to project; reduce software development and maintenance costs; produce more standardized software; produce better quality software and provide a powerful competitive advantage.

Generative modeling and programming is a recent software engineering paradigm based on modeling software system families such that, given a particular requirements specification, a highly customized and optimized intermediate or end-product can be automatically manufactured on demand from elementary, reusable implementation components by means of configuration knowledge.[2] Generative Programming is about bringing the benefits of automation and reuse to software development. On the one hand we think of automation and reuse which reduce all kinds of cost. On the other hand, the stakeholders are demanding specialized software that fits their specific needs. Software must therefore be easily customizable. This creates conflicting forces on software development: reusing existing software artifacts while creating customized high-quality software that meets the specific needs of its stakeholders. As the instantiation of Generative Programming and Modeling, software product-lines (SPL) and system-family engineering(SFE) well proved how to balance the tradeoff of between software automation, reuse and high-quality software customization.

Withey gave the difference between a product-line and a system family in his technical report. A product-line is "a group of products sharing a common, managed set of features" and a system family is "a group of products that can be built from a common set of assets"[1]. Product-line engineering applies to the reuse of components and architecture in a family of products. System-family engineering, on the other hand, applies to the development of specialized software that meets specific stakeholder requirements out of a family of existing softwares. This software system family is also called a specific Domain that we predefined. In this case, an analysis is first performed to determine the commonality and differences that exist between the required specialized software and the existing software system family. In order to facilitate the development of customized family members, a highly- flexible architecture and a set of reusable components should be designed ahead of time.

System Family Engineering and Product-line are all follow what Generative Programming focused on, namely modeling in domains (software system families) rather than one-of-a-kind systems. In this thesis, we will look at how System Family Engineering can be used to deal with a algorithm family in distributed computing – Leader Election in Ring.

1.2 The problem (1.5 pages)

In the previous section, we introduced the context of this thesis. As we pointed out that Generative Programming is about bringing the benefits of automation and reuse to software development. Before we go any further into the problem of how to reuse by using Generative approach, we will give a small case study to get to know Generative Programming and all the related techniques.

Case study – Car

In the car manufactories, there are several standard kinds of cars are being supplied to the market. For example:

1 a). Whopping 200 hp
b). 230 lb.-ft. of torque
c). 3800 Series III V6 engine
d). Closed bodywork
e). With a reconfigurable Driver Information Center
f). Power windows
g). entry and available tire inflation monitor
h). Included with anti-lock braking system (ABS)
i). 15" X 6.5" wheels
j). Two door
k). Air conditioning
l). Six colors available: silver, black, white, red, green, blue.
m). Automatic transmission
... ...

2 a). With 260 hp
b). 280 lb.-ft. of torque
c). 3.8 L supercharged 3800 Series III V6 engine
d). under the hood
e). No reconfigurable Driver Information Center
f). Manual windows
g). H-rated tires
h). Included with anti-lock braking system (ABS)
i). 17" X 6.5" wheels
j). Four door
k). No air conditioning
l). Three colors available: silver, black, white
m). Shift transmission
... ...

Those are the existing standards designed by the manufactories who produce the products that full fill most customers' requirements. Let's say that one car manufactory supply the customer design service. Here is the customer's requirement:

a). With 220 hp
b). 250 lb.-ft. of torque
c). 4.0 L supercharged 3800 Series III V8 engine
d). under the hood
e). reconfigurable Driver Information Center
f). Power windows
g). H-rated tires
h). Included with anti-lock braking system (ABS)
i). 17" X 6.5" silver-painted aluminum wheels
j). Two door
k). Air conditioning
l). Color: Orange
m). Automatic transmission
... ...
It is not very difficult to figure out the difference and similarity among those three set of car settings. For example: a car must has engine, to describe the kinds of engine we have: V4, V6, V8,.etc; a car must has either of the transmission: automatic transmission or shift one; a car must have a color; a car can have air conditioning or do not have air conditioning; a car can have two or four doors; a car must have four wheels with the same size that must be within a certain range; a car can have a normal key, or a remote key, or have both of them at the same time.....

After studied these car setting, and figured out the difference and similarity, we can get the following diagram

, It is important to note that variability is different from modifiability. Modifiability is a quality of service, that is, a possible objective of the system pertaining to of ; whereas variability is understood as the ability for a software piece to vary. For

example, a security component offers a variable encryption strategy if this component can use distinct encryption strategies. On the

We want to reuse yet need customized s/w SPL/SFE and generative programming address this tradeoff: why generative Basic idea: model variability (using feature diagrams) Distinguish modifiability from variability Disclaimers (2nd last paragraph of CASCON intro) Before going any further, we remark that this paper is not intended to address the issue of establishing and recognizing a family of systems. Insight into domain establishment and recognition is available elsewhere (e.g. [9]). Nor do we address the problem that exists in both anticipated and unanticipated software evolution, when a change produces a software system that lies outside of its intended domain. Finally, we note that one a domain has been established, it too may evolve, but addressing such evolution is beyond the scope of this paper. We wish to make it clear that although we do not specifically tackle any of the aforementioned issues in this paper, we do not wish to demean their importance.

References

[1]

[2] Ulrich W. Eisenecker K. Czarnrcki. *Generative Programming - Methods, Tools, and Applications*. Addison Wesley, 2002.

[3] W. ed. Tracz. Software reuse: Emerging technology. *IEEE Press*, 1988.