Available online at www.sciencedirect.com
ScienceDirect
Information and Software Technology 50 (2008) 259–279
INFORMATION AND SOFTWARE TECHNOLOGY
www.elsevier.com/locate/infsof

Micro and macro workflow variability design techniques of component

Chul Jin Kim [a], Hyun Sook Chung [b], Eun Sook Cho [c,*]

[a] Architecture Group, Software Engineering Part, Digital Solution Center, Samsung Electronics Co., Ltd., 12th Floor, Union Steel Building 890, Daechi4-dong, Gangnam-gu, Seoul 135-524, Republic of Korea
[b] Chosun University, Department of Computer Engineering, 375 Seosuk-dong, Dong-gu, Gwangju 501-759, Republic of Korea
[c] Seoil College, Dept. of Software #49-3, Myeonmok-8 Dong, Jungnang-Gu, Seoul 131-702, Republic of Korea

Received 4 June 2006; received in revised form 8 January 2007; accepted 24 January 2007
Available online 31 January 2007

Abstract

Components should provide variability in satisfying a variety of domains [C. Szyperski, Component Software: Beyond Object-Oriented Programming, Addison-Wesley, 2002.], but it is not easy to develop components which can be applied to all domains. Although components are developed by analyzing many different requirements, developing components that satisfy all requirements is difficult since unexpected requirements occur during the utilization of components. Hence, providing the variability of components becomes an important prerequisite for a successful component-based application development.

In this paper, we propose a variability design technique that can satisfy the business workflow requirements of many different kinds of domains. The technique addresses a method for designing the variability of the workflow in a more detailed method and uses an object-oriented mechanism and design patterns. One of the most important goals of this technique is to provide a practical process can be effectively applied in component-based application development.

Keywords: Component variability; Micro/macro workflow; Reusability

1. Introduction

Component technology is widely accepted in both academia and industry. Since object-oriented technology did not improve software development progressively [2], component technology came to its rescue. One of the main reasons for this was that objects have limited coverage in a typical development process. Components significantly decrease the burden of development because they provide workflow and are offered as functional units composed of objects. The concept of components regards software development as an assembly process which can rapidly and easily satisfy the requirements of different domains [3]. Components provide diversity to fulfill various domain requirements using component interfaces [4]. However, it is not easy to design variability to provide diversity. Furthermore, procedural techniques for variability design have not been exhaustively studied. Accordingly, we propose a variability design technique that can accommodate a variety of domain requirements.

In this paper, we propose process for designing the workflow variability of a component. The design techniques for workflow variability consist of techniques for designing variable message flows between classes within a component and between components in the complex component. We propose the mediator technique of design pattern for variable message flows within a component and the connector technique, which plays the role of coordinator, for variable message flows in the complex component.

The paper is organized as follows: Chapter 2 introduces how to design variability in component based development (CBD) methodology. In chapter 3, as the core of this paper, we propose the design technique for workflow variability. Chapter 4 assesses the practicality of our techniques through case studies. Chapter 5, conclusion, summarizes the proposed variability design techniques and describes the underlying goal of this paper.

* Corresponding author. Tel.: +822 490 7562; fax: +822 490 7398.
E-mail address: escho@seoil.ac.kr (E.S. Cho).

0950-5849/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2007.01.001

1.1. Variability

Variability is the difference between members of a product family [5] such as the difference between components, and the difference between component frameworks. The target of variability may be the classes of product families and their operations [6,7]. In this paper, the classes of product families are operations of a component interface, operations of the classes in a component, and components within a component framework. Types of variability include attributes, behaviors, and workflows.

1.1.1. Attribute variability

Attribute variability is the difference between attributes that have the same role among members of a product family. Elements of the attribute variability are the same attributes existing in the different classes of the same product family such as attributes in a component interface and attributes of classes within the same component.

1.1.2. Behavior variability

Behavior variability is the logic difference between behaviors that process the same function among members of the same product family. Elements of the behavior variability are operations among classes in the same product family [8]. In this paper, they are operations of a component interface and operations of classes within the same component.

1.1.3. Workflow variability

Workflow variability is the difference between message flows that process the same function among members of the same product family. Elements of the workflow variability are message flows of the operations among classes in the same product family. In this paper, they are message flows of a component interface within the same component framework or message flows of the class operation within the component.

1.2. Variability realizations

Variability can be realized using the following techniques.

1.2.1. Parameterization

Parameterization provides the configuration of parameters for customizing the behaviors of components [9]. The variability of components can be designed with parameters that can customize the variation points of components. If the variability technique provides the variety of selections through parameters, the reusable range of components will be extended. However, the disadvantage of this method is the increasing size of the components as the variation points are extended.

1.2.2. Inheritance

Inheritance is a technique primarily used in object-oriented frameworks. The hot spots of object-oriented frameworks can be customized to inherit from a variable class [10]. The variability design using inheritance defines an abstract class for variation points and also defines the sub-classes corresponding to the variants.

1.2.3. Plug-In

Plug-In is a technique that can plug components into a component framework to customize the behavior [11]. A component framework resembles object-oriented frameworks, but object-oriented frameworks rely on inheritance while component frameworks rely on the interfaces between plug-ins and a component framework [12].

1.2.4. Connector

Using connectors is a technique to design the interactions among components that mediate the communication and coordination tasks through different forms of interactions such as pipes, procedure calls, and event broadcast. Connectors have interfaces that define the roles of components participating in the interactions between components. They have roles such as the caller and callee roles of an RPC connector, the reading and writing roles of a pipe, and the sender and receiver roles of a message passing connector. Thus, connectors to design the interactions between components can provide a technique for designing variability in a component framework [13].

2. Related work

2.1. KobrA method

The KobrA method was developed as part of a project funded by the German Federal Ministry of Education and Research (BMBF) known as the *Komponentenbasierte Anwendungsentwicklung* or KobrA project [14,15].

Variabilities are characteristics that may vary from application to application. In general, all variabilities can be described in terms of alternatives. At a coarse-grained level, one artifact can be seen as an alternative to another artifact. Then during application engineering, the artifact that best matches the context of the system under development is selected. Although simple in theory, providing an effective representation of the variabilities in a product family is not only critical but also problematic factors in the success of a product line engineering project.

While KobrA method addressed the variability extraction, the variability design techniques did not address. The process for variability extraction requires the detail guidance. The notation for variability in KobrA proposes to use the stereotype "⟨⟨variant⟩⟩" in models that include the variability. This notation is insufficient to express a variety of variability design techniques.

2.2. FAST

FAST (family-oriented abstraction, specification, and translation) has been introduced to AT&T by David Weiss and further developed at Lucent Technologies Bell