Issues in Component-Based Software Engineering

Kyo C. Kang
Department of Computer Science and Engineering
Pohang University of Science and Technology (POSTECH)
San 31 Hyoja-Dong, Pohang, 790-784, Korea

Introduction

Software reuse is generally considered as one of the most effective ways of increasing productivity and improving quality of software. To make software reuse happen, however, there should be a change in the way we develop software: software must be developed for reuse and with reuse, and there must be a paradigm shift from the notion of specific application "development" to that of "integration." Component-based software engineering (CBSE)[3] is an emerging software engineering paradigm in which applications are developed by integrating existing components. Here, components refer to any units of reuse or integration, including computational (i.e., functional) components, interface components, communication components, and architectures.

In order to maximize the productivity gain and cost reduction, CBSE must be based on domain-oriented components as well as generic components. In general, the productivity increase from the reuse of domain-oriented components is higher than the productivity increase from the reuse of generic components, although reusability of domain-oriented components might be lower than that of generic components. Therefore, CBSE should happen in the context of domain-orientation. SAP R/3 [7], Baan IV [1], and Oracle Applications [6] are good examples of domain-oriented CBSE.

Domain-oriented CBSE is not for any application domains. In immature and unstable domains, there may not be much domain-oriented components to reuse and CBSE may be limited to the reuse of generic components. Therefore, CBSE should be applied in mature and stable application domains, or in an organization where a family of closely related products is produced.

To develop an application by integrating components, there must be components that can solve the problems of the given application. Therefore, CBSE must address not only the issues of how to integrate components but also the issues of how to produce integratable components. CBSE can only be successful when the issues of both producer’s and consumer’s are resolved.

In section 2, a CBSE framework is presented in which major activities of both producing and using components are identified. Section 3 summarizes some of the important component engineering principles. Some of non-technical but important issues underlying CBSE are discussed in section 4. Section 5 concludes this position paper.

CBSE Framework

To develop software by integrating components, components must be developed for reuse. Therefore, CBSE must address both the development of reusable components and the development applications using the reusable components, as shown in Figure 1 [4].

Application-oriented reusable components must be developed for the common needs of the application domain not for a specific application to maximize reusablity. Domain engineering, therefore, consists of activities for identifying commonalities of the applications in the domain (i.e., domain analysis), developing alternative architectures, and developing reusable code components for the architectures. Once domain engineering is performed for a domain and reusable components are created, application engineering may proceed applying the reusable components. User requirements analysis may be performed using the domain commonality model and an architecture appropriate for the application may be selected. Based on the selected architecture, the application software can be created by integrating reusable code components.

Figure 1: CBSE Process (*figure appears on the next page*)

One of the most important elements for successful CBSE is the development of reusable (i.e., implementing common needs/functions, adaptable, maintainable) components. Some of the important component engineering principles are discussed in the following section.

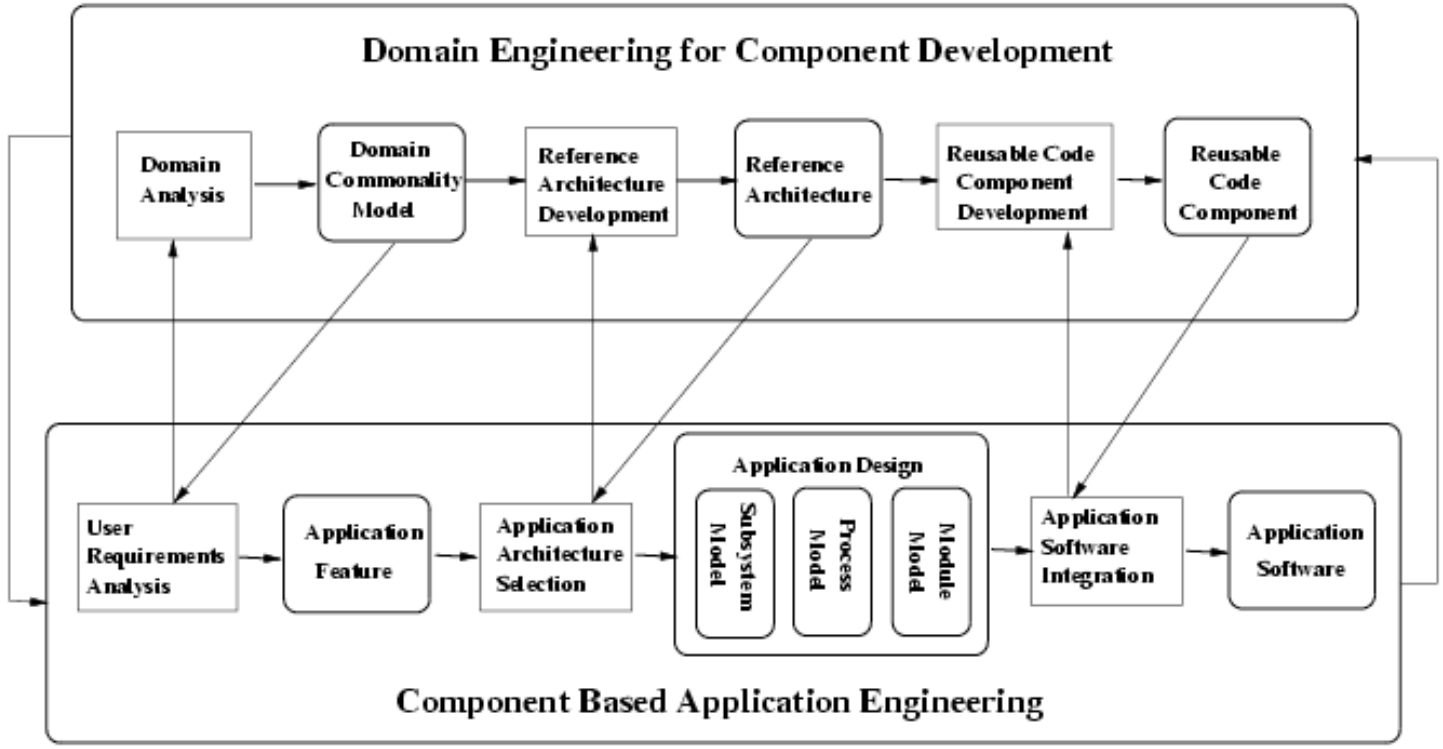
Engineering Principles for CBSE

There are principles for the development of components to support component integration. Some of important component-engineering principles are discussed below.

Domain Orientation: Software reuse may be the most effective ways of increasing productivity and reducing maintenance as well as development cost of software. To achieve successful software reuse, commonalities of related systems must be discovered and represented in a form that can be exploited in developing similar systems. Domain orientation is one such approach. It attempts to discover commonalities of systems in an application or a technical area (i.e., a domain) and then develop models or components that can be used in developing systems in the domain. This approach will help evolving an application as well as developing a family of applications.

Although domain orientation is believed to be the key element in achieving successful CBSE, most domain-oriented engineering technologies are still in their infant stage. There are many technical issues that must be resolved before we can mature these technologies.

Separation of Concerns: This engineering principle is one of the key engineering principles supporting CBSE. Components must be designed so that each performs a unique singular


Domain Engineering for Component Development
Domain Analysis
Domain Commonality Model
Reference Architecture Development
Reference Architecture
Reusable Code Component Development
Reusable Code Component
Application Design
Subsystem Model
Process Model
Module Model
User Requirements Analysis
Application Feature
Application Architecture Selection
Application Software Integration
Application Software
Component Based Application Engineering

function. Also, each functional component must be designed independent of the interface mechanisms it employs to communicate with other components. This principle implies that selection of a particular functional component or an interface method/mechanism does not impose any restriction on selection of other components.

Abstract Virtual Machine Interface: Interface of a component must be designed as a virtual machine. A component must provide a complete, non-redundant (i.e., minimal) interface. The interface must also hide internals (i.e., implementation decisions) of the components so that different implementations can be made for the component.

Postponement of Context Binding: In the design of components, we need to strive for the development of "context free" components focusing only on the core functionality. To the extent it is possible, binding with particular contextual parameters such as data type, storage size, implementation algorithms, communication methods, operating environment, etc. should be postponed until the component integration time when performance optimization is made.

Design Reuse: For component-based software integration to happen, design (i.e., component development context) must be shared and reused among the potential users. That is, design reuse must happen before component reuse. An architectural design shows the allocation of functionality to components and, for a component integration, reuse of the underlying architecture based on which components were developed must occur.

Hierarchically Layered Architecture: Architectures and code components must be designed for maximum flexibility in composing components. Figure 2 includes a layered architecture model [4] which seperates application task-oriented components, which do controlling and activity coordination, from functional components, which do mostly computations. Implementation techniques that are commonly used in the domain are separated from the functional components as implementation techniques can change for the same functions. Data communication models (e.g., message queue, task synchronization) are also separated from the technologies that implement various communication models. This architecture model allows postponement of the binding of particular implementation techniques until the component integration time when performance considerations are made.

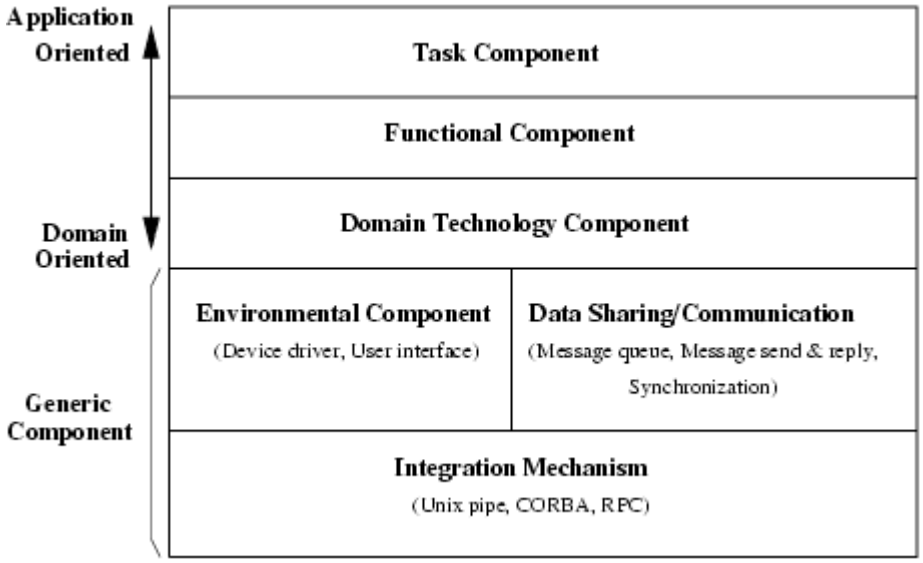


Figure 2: Layered Architecture

Non-technical Issues

One of the most serious problems that impede CBSE is the cost involved in producing reusable components. The cost of producing reusable components is substantially higher than the cost of producing a single application without reuse consideration, as high as five times the cost of producing a single application in some cases [2], few will make this large investment for others or for the future. This approach may be considered economically viable in the commercial software development where frequent customization is needed or in the environment where a family of systems are developed and maintained. It is likely that it will take some time before CBSE becomes the common practice in software engineering. Some of the economics issues are discussed in [5].

Developing software for future reuse or for others is not an easy matter in any corporate environment where software development is managed in terms of projects. It is difficult to see any development projects that were not under schedule pressures or not in shortage of resources. Giving incentives (e.g., award, promotion) to contributors or reusers doesn't usually make much difference unless those incentives are significant. One of the most effective ways to promote CBSE in a corporate environment might be to have a central support organization dedicated to: (1) identify common needs, (2) develop, maintain, and advertise reusable components, and (3) teach and support reuse.

This support organization should consists of highly skilled software engineers who are able to perform domain analyses for the application areas of the development organizations (projects) and develop reusable components applying techniques such as meta-programming, application generator, macro processing, and template. As they can oversee many projects, they should be able to identify common problems among projects and provide generic solutions. The cost of this organization could be amortized through reuse across development projects.

Conclusion

The systematic discovery and exploitation of commonality across related software systems is a fundamental requirement for achieving successful CBSE, and domain orientation is one of the most important elements of CBSE. Domain orientation aims at codifying the development knowledge in an application domain as models and components, and using these models and components in applications development. CBSE might be applied most effectively in mature and stable domains where domain-oriented reusable components can be identified.

For CBSE to become the common practice in software engineering, non-technical as well as technical issues for both producers and consumers of reusable components must be addressed. The development cost of reusable components is especially high and CBSE may not happen unless there is a social/organizational infrastructure supporting the production and exchange of components and amortize the development cost. In a corporate environment, a central support organization dedicated to the development, maintenance, and support of reusable components might serve as a support infrastructure.

References

1 Baan Co., http://www.baan.com

2 Basset, P., Netron Inc., Toronto, Canada (private conversation)

3 Brown, A.W., Editor, Component-Based Software Engineering, IEEE Computer Society, 1996.

4 Kang, K.C., et al., "FORM: A feature-oriented reuse method with domain-specific architectures", Annals of Software Engineering, Vol. 5, pp. 143-168, 1998.

5 Kang, K.C., Levy, L.S., "Software methodology in the harsh light of economics", The Economics or Information Systems and Software, (Veryard, R., editor), pp. 183-203, Butterworth-Heinemann, Ltd, 1991.

6 Oracle Corp., http://www.oracle.com

7 SAP-AG, http://www.sap.com

About this document ...

Issues in Component-Based Software Engineering

This document was generated using the **LaTeX**2HTML translator Version 96.1-h (September 30, 1996) Copyright © 1993, 1994, 1995, 1996, Nikos Drakos, Computer Based Learning Unit, University of Leeds.

The command line arguments were:
latex2html `cbs.tex`.

The translation was initiated by on Tue Apr 13 16:40:22 KST 1999

Tue Apr 13 16:40:22 KST 1999