Component-Based Development Environment: An Integrated Model of Object-Oriented Techniques and Other Technologies

Oh-Cheon Kwon, Seok-Jin Yoon and Gyu-Sang Shin

Computer & Software Technology Laboratory

ETRI(Electronics and Telecommunications Research Institute)

Taejon, Korea

{ockwon, sjyoon, gsshin}@etri.re.kr

Abstract

Object-Oriented Programming(OOP) has some weaknesses in that it does not always produce reusable software and is not suitable for a large project and does not support the complete encapsulation of classes due to the inheritance of subclasses. As a evolutionary method of OOP, Component-Based Software Engineering(CBSE) or Component-Based Development(CBD) has recently been hot issues for the Object-Oriented community and reuse community, and the component market is also growing rapidly. Thus, in order to overcome the limitations of OOP and maximize the benefits from reuse, the authors propose an integrated model that links the OOP paradigm with the emerging CBD paradigm. In addition, the authors review most of the technologies related to an integrated CBD environment and describe our current research on re-engineering that will be extended to support a whole CBD environment.

1. Introduction

Object-Oriented Techniques have been considered a powerful means of solving software crisis through their high reusability and maintainability. However, Object-Oriented Programming(OOP) has not brought many benefits since they have not provided interoperability of components at the binary/runtime level. The following article from Byte Magazine shows that the paradigm for a software development method has shifted from OOP to Component-Based Development(CBD).

"Object technology failed to deliver on the promise of reuse. Visual Basic's custom controls succeeded. What role will Object-Oriented Programming play in the component-software revolution that's now finally under way?" [Byte94]

A software component is defined as a unit of software that implements some known functions and hides the implementation of these functions behind the interfaces that it exposes to its environment. The term componentware is also described as software assembled from a set of components [Ovum98]. Ovum’s definition of a component is limited to software, but in order to maximize the benefits from reuse we should include all information generated in the course of development, ranging software products to requirement specifications, design specifications, project plans, test plans, quality plans, user manuals, including ideas, methods, experiences, etc.

Component-Based Software Engineering(CBSE) or CBD is considered a new paradigm of a software development method that consists of component production, selection, evaluation and integration. In order for CBSE to be successful and effective, many problems related to this new technology should be solved. This paper reviews several technologies that need to be investigated in order to build a Component-Based Development environment that will work as a total solution. Since CBSE requires the linkages between many related technologies, a tool supporting CBSE/CBD may not work as a feasible tool if these technologies are all not implemented in the tool. There are some hopeful component infrastructure tools or models such as OMG’s CORBA 2 specification [Orfali97], Microsoft’s ActiveX/OCX and COM/DCOM[Chappell96], and Sun’s Java Beans[Sun97]. However, since these component tools are not complete and sophisticated they should include the technologies described in the next section in order to be used for a total solution.

2. Trend of CBD Related Technologies and our Position

As shown in Figure 1, we propose a CBD environment that consists of several technologies related to CBSE. These technologies are categorized into component building technologies and component use technologies. Component building technologies contain ’Design/Development for Reuse’ based on Object-Oriented Programming(OOP), Design Patterns and Frameworks, Re-engineering, Component Description, Domain Engineering and Component Certification. Component use technologies include ’Design/Development with Reuse’ based on a Reuse Repository, Domain Engineering and Reuse Metrics. Domain Engineering is relevant to both component building technologies and component use technologies. As shown in Figure 1, the CBD environment has been built as an integrated model in order to merge the OOP paradigm with the emerging CBD paradigm, thereby leading to maximization of reuse effects through overcoming the limitations of OOP that is described in the next section. The authors review most of the technologies supporting CBD and describe our current research on re-engineering that will be extended to support a CBD methodology.

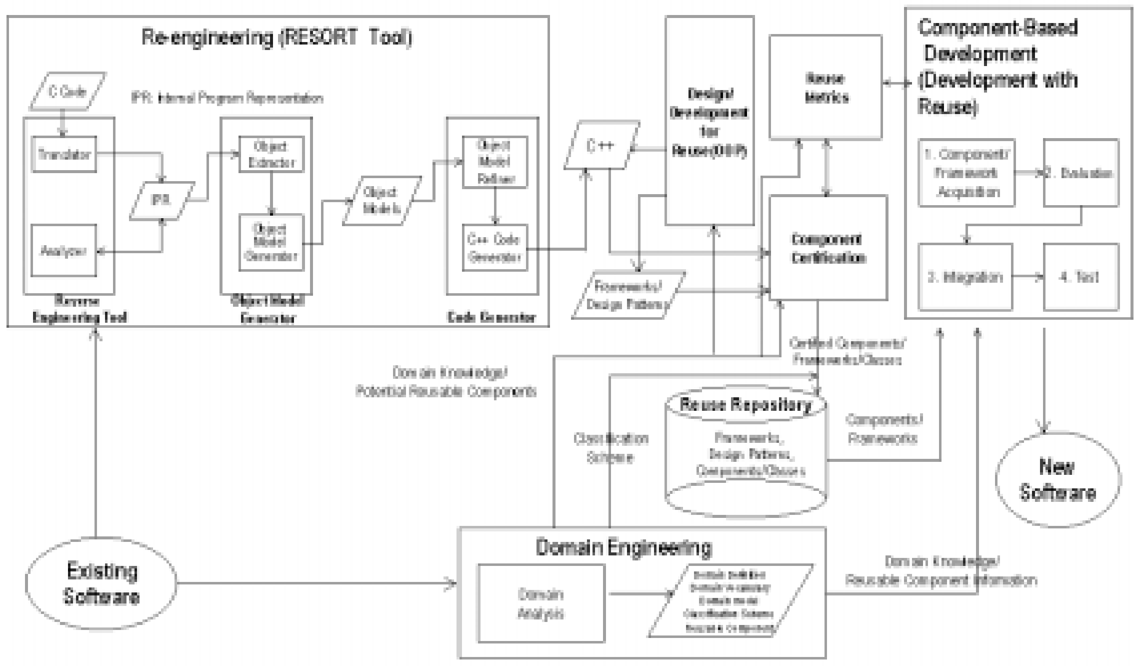
Re-engineering (RESORT Tool)
C Code
IPR: Internal Program Representation
Translator
IPR
Analyzer
Reverse Engineering Tool
Object Extractor
Object Model Generator
Object Model Generator
Object Models
Object Model Refiner
C++ Code Generator
Code Generator
C++
Design/ Development for Reuse(OOP)
Reuse Metrics
Component Certification
Frameworks/ Design Patterns
Component-Based Development (Development with Reuse)
1. Component/ Framework Acquisition
2. Evaluation
3. Integration
4. Test
Certified Components/ Frameworks/Classes
Domain Knowledge/ Potential Reusable Components
Classification Scheme
Reuse Repository
Frameworks, Design Patterns, Components/Classes
Components/ Frameworks
New Software
Existing Software
Domain Engineering
Domain Analysis
Domain Knowledge/ Reusable Component Information

Figure 1. A Component-Based Development Environment Integrating OOP with Related Technologies

2.1 'Design/Development for Reuse' Based on Object-Oriented Programming

The major benefits of CBD can accrue from the inherent reusability of software components. Therefore, 'Design/Development for Reuse' technology based on Object-Oriented Programming(OOP) should be introduced to an IT organization so that components with good quality and reusability can be created, classified, stored and managed for future reuse. However, it is well known that designing reusable components is hard and supported by too few tools, so it is only a job for the best skilled designers and programmers[Short97]. OOP has some weaknesses in that it does not always produce reusable software and is not suitable for a large project and does not support the complete encapsulation of classes due to the inheritance of subclasses. We should agree that the component technology owes some of its characteristics to OOP but the classes built by OOP are not always reusable components.

For the above reasons, some people argue that object-oriented development has failed to draw strong attentions from software developers. Nevertheless, others argue that components are just encapsulated objects/classes, and large scale reuse will be also finally achieved if software developers are motivated to reuse reusable components and provided with class libraries and frameworks described in the following section. From our points of view, OOP should be used to create encapsulated components that consist of some objects and enable users to plug-and-play easily with the components in order to build a new system. Thus, since CBD is not a revolutionary but evolutionary method, the CBD method needs to be cooperated with OOP within a CBD environment.

2.2 Design Patterns and Frameworks

Design patterns can be defined as an abstract solution for common problems in Object-Oriented Programming and consist of classes, objects and interactions between them. Experienced software engineers can find these patterns from their knowledge and experiences and reuse them in developing applications with the same pattern. Gamma[Gamma95] defines design patterns as "descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context". Many design patterns have been found in software architectures[Coad95], [Gamma95].

In OOP, the concept of a framework has proven to be a very useful way to reuse skeletal implementations of architectural designs that can be customized by application developers. Some commercialized frameworks such as IBM’s SanFrancisco, Microsoft’s MFC, Inprise’s OWL and Icon Computing’s Catalysis[Icon98] have been released to the market. Since Microsoft’s Visual Basic, ActiveX/OCX and COM/DCOM, Sun’s Java Beans and OMG’s CORBA were announced in the software component market, OOP enthusiasts have argued with component enthusiasts about which way to go to enhance software development productivity and to reduce development costs. In particular, the birth of Visual Basic in 1991 has led to the creation of the large component market.

2.3 Re-engineering

A narrow definition of re-engineering focuses on code restructuring whereas a broad definition of it includes forward engineering as well as reverse engineering. Reverse engineering does not change the system but provides alternate views of the system at various levels of abstraction. Forward engineering is the process of system-building, starting with an existing system structure.

The ultimate objective of our research into re-engineering of the broad context is to implement RESORT system (REsearch on object-oriented SOftware Re-engineering Technology) that supports an integrated software re-engineering environment through transforming an existing system written in procedural languages into a system written in Object-Oriented languages, thereby leading to modernize outdated software and to elevate reusability and maintainability of existing software. As shown in Figure 1, the RESORT system consists of 5 subsystems: a program comprehension tool, a re-documentation tool, a domain object modeling tool, an object extraction tool and a C++ code generator.

The RESORT system provides users with a process that validates and refines the extracted object models after the object generator extracts object models automatically through analyzing Internal Program Representations(IPRs) translated from procedural C programs by a reverse engineering tool.

2.4 Component Description(Specification)

In the case of ’black box reuse’, a component consists of an executable component itself that will be used to meet user’s intentions or requirements, and a specification that describes the model and service/interface of a component and how to use it. A specification of the component can be specified by an interface description language or a formal description language.

When application developers implement software using a CBD method, they do not need to know about implementation details. They can use components easily by calling the operations described in the interface of a specification separated from implementation. For example, an Interface Definition Language(IDL) is part of the CORBA specification for defining interfaces and operations[CORBA95]. In the ’design for reuse’ activity, if the specification of a component is described formally using a formal specification language, it can be transformed into target source code.

In Architecture Specification Language(ASL), the functionality of a component is defined by its interfaces that are classified into provided interfaces and required interfaces[CORBA95]. In addition, description techniques such as UML[UML97] and Catalysis[D’Souza97] have been well known to a CBSE area.

2.5 Domain Engineering

Domain engineering consists of domain analysis and domain modeling activities. A domain is a specialized body of knowledge and an area of particular business. Domain analysis is an activity of identifying objects and operations in a set of related systems, and identifying common objects across a set of existing or future systems through commonality analysis. Domain analysis is a key to successful reuse since it enables us to find reusable components from a legacy system and to decide which reusable components should be created for a new system in collaboration with ’design/development for reuse’. As shown in Figure 1, most information acquired from domain analysis is also used for building a new system using reusable components(i.e., ’design/development with reuse’). A domain modeling activity is associated with understanding component requirements and translating them into interface descriptions of a component specification.

Domain analysis is carried out for a set or family of related systems, whereas system analysis is performed for a single system. Domain models can be used to check the completeness and consistency of system requirements[McClure96]. After performing domain analysis and modeling, we can determine domain vocabularies, domain models, possibly reusable components and classification schemes, as shown in Figure 1.

2.6 Component Certification

After components have been created or possibly reusable components have been extracted, these components must be evaluated and certified through a software metrics system. Component certification enables reusers to eliminate ’Not Invented Here(NIH)’ syndrome that is one of barriers to obstacle the success of reuse.

2.7 ’Design/Development with Reuse’

If a reuser is not satisfied with a component retrieved from the reuse repository, he/she needs to modify or customize the component. The component to be modified is called a ’white box’ component. An example of ’white box reuse’ in OOP is a process of developing programs through writing child classes after understanding and subclassing of the parent classes. ’Black box reuse’ or ’gray box reuse’(reuse by instantiation/parameterization) is easier to use than ’white box reuse’ since the internals of the related classes do not have to be understood. Class users of ’black box reuse’ are only required to grasp the interfaces of classes.

2.8 Reuse Metrics

Reuse metrics are used to identify the components that are highly reusable and the business areas and systems in which reuse has the high potential to provide the greatest benefits to an organization. They can identify these components that recur most frequently across the systems through domain analysis. McClure[McClure95] describes 10 factors for reuse metrics as follows: commonality of a component, reuse threshold of a component, reuse merit of a component, reuse creation cost of a component, reuse usage costs of a component, reuse maintenance cost of a component, degree of commonality of a system, degree of reusability of a system, reuse target level of a system, reuse merit of a system. Reuse metrics are the key technology to elevating benefits from reuse together with component certification.

3. Related Work

In this paper, the authors describe related research work in terms of two aspects: a re-engineering method and a CBD method since we currently focus on a re-engineering area among the technologies for supporting a CBD environment.

A typical re-engineering method called COREM(Capsule Oriented Reverse Engineering Method) [Gall95] that carry out design recovery from a legacy system through a reverse engineering process, create object models by manual intervention and finally build Object-Oriented software. CORET(Capsule Oriented Reverse Engineering Technique) project[Gall97] that started in order to enhance the COREM prototype, has recently been implementing a process that extracts objects from procedural C code and produces C++ code using extracted objects. However, CORET project has used commercialized tools in order to reverse engineer C code and model the objects of a target system, whereas our RESORT project has developed these tools for ourselves.

There have been two representative researches into a CBD environment. Firstly, a CBD methodology called Catalysis[D’Souza97] has been built in order to support object modeling, specification activities and design models by composition. Catalysis is a UML[UML97] and OMG[CORBA95] compliant method for component and framework based development. The Catalysis method provides users with frameworks that can be applied to various areas ranging from business models and many common design patterns to very fundamental definitions. Secondly, Andersen Consulting[Ning98] has carried out a research project for building a Component-Based Development environment that enables an architecture-driven and component plug-and-play style of software system development. Andersen Consulting’s CBD environment includes many advanced research ideas and results from the academia and industry in the field of software components and architectures.

4. Conclusions

Component-Based Development(CBD) can be regarded as the best way of developing applications that are cheaper and faster. These applications can match a growing distributed object technology. The authors proposed an integrated CBD model of the component technologies and OOP that produces reuse effects by synergy. The technologies related to the CBD model described in this paper are very central to the success of CBD. In particular, our research is currently focusing on re-engineering that extracts object models from a legacy system written in C and then generates C++ code. After completing our re-engineering project this year, we will extend our research to ’design/development for reuse’ that creates a new component and framework, including other technologies related to CBD.

5. References

[Byte94] J. Udell, "Cover Story: Componentware", Byte Magazine, May 1994.

[Chappell96] D. Chappell, "Understanding ActiveX and OLE", Microsoft Press, 1996.

[Coad95] P. Coad, D. North, et al., "Object Models: Patterns, Strategies, and Applications", Yourdon Press, Prentice Hall, 1995.

[Coleman94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, et al., "Object-Oriented Development: The Fusion Method", Prentice-Hall, 1994.

[CORBA95] Object Management Group, Inc., CORBA, "The Common Object Request Broker: Architecture and Specification", July 1995.

[D’Souza97] D. D’Souza and A. Wills, "Composing Modeling Frameworks in Catalysis", TechnicalReport, Icon Computing Inc., 1997.

[Gall95] H. Gall, R. Klosch and R. Mittermeir, "Object-Oriented Re-Architecturing", Proc. of European Software Engineering Conference, September 1995.

[Gall97] H. Gall and J. Weidl, Object-Model Driven Abstraction-to-Code Mapping, Technical Report, Technical University of Vienna, December 1997.

[Gamma95] E. Gamma, R. Helm, et al., "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley, 1995.

[McClure95] C. McCLure, "Model-Driven Software Reuse", Extended Intelligence, Inc., 1995.

[McClure96] C. McCLure, "Software Reuse Techniques: A Guide to Adding Reuse to the Software Process", Extended Intelligence, Inc., 1996.

[Ning98] J. Q. Ning, "CBSE Research at Andersen Consulting", Proceedings of the 1st International Workshop on Component-Based Software Engineering, April 1998.

[Orfali97] R. Orfali and D. Harkey, "Client/Server Programming with Java and CORBA", John Wiley and Sons, 1997.

[Ovum98] K. Ring and N. Ward-Dutton, "Componentware-Building it, Buying it, Selling it", Ovum Ltd., 1998.

[Short97] K. Short, "Component-Based Development and Object Modeling", Sterling Software, Technical Handbook Version 1.0, February 1998.

[SUN97] Sun Microsystems, Java Beans 1.01, Sun Microsystems, 1997.

[UML97] UML Group, Unified Modeling Language 1.1, Rational, 1997.