ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

IJESRT
INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY

Analysis of Component Integration in Component-based Development Paradigm

Manbir Kaur
Student, Department of ECE, Global Institute of Management and Technology, Amritsar, India
manbirkhurana90@gmail.com

Abstract

Component-based software development is achieving more popularity in today's software development community due to the increasing complexity of software systems, increasing costs of maintenance, and decreasing costs of underlying hardware. Business organizations therefore prefer to build their systems from previously built components which means they should concentrate more on selecting and composing components than manually adopt software systems. As the popularity of such approaches grows, commercial software vendors tend to develop more commercial software components and connectors. In this paper, an effort has been made to analyze various component integration techniques with respect to component-based software development.

Keywords: Component, Component-based Software Development, Component Integration, Risk Analysis.

Introduction

Modern software systems become more and more large-scale, complex and uneasily controlled, resulting in high development cost, low productivity, unmanageable software quality and high risk to move to new technology . Consequently, there is a growing demand of searching for a new, efficient, and cost-effective software development paradigm. One of the most promising solutions today is the component-based software development approach. This approach is based on the idea that software systems can be developed by selecting appropriate off-the-shelf components and then assembling them with well-defined software architecture. This new software development approach is very different from the traditional approach in which software systems can only be implemented from scratch. These commercial off-the-shelf (COTS) components can be developed by different developers using different languages and different platforms. This can be shown in Figure 1, where COTS components can be checked out from a component repository, and assembled into a target software system. Component-based software development (CBSD) can significantly reduce development cost and time-to-market, and improve maintainability, reliability and overall quality of software systems. This approach has raised a tremendous amount of interests both in the research community and in the software industry. The life cycle and software engineering model of CBSD is much different from that of the traditional ones. This is what the Component-Based Software Engineering (CBSE) is focused.

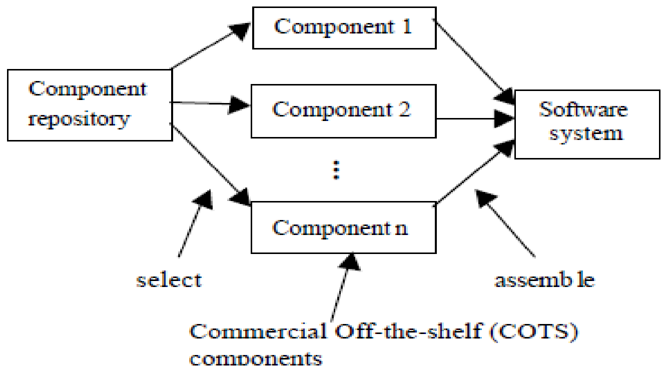

Fig. 1 Overview of Component-based Software Development Process

The term component-based software development (CBD) can be referred as a process for building a system which consists of the following activities: searching and identifying components based on preliminary assessment; selecting components based on stakeholder requirements; integrating and assembling the selected components; and updating the system as components evolve over time with newer versions.

Related work

Component-based Software Development (CBSD) is used to develop high quality and reliable products in less time domain and low cost. Component Integration are one of the big challenges in CBSD and different authors have presented different techniques but lacking in desired performance, accuracy and user

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

friendliness. The CBSD initiates with the identification and selection of components.

The success of CBSD depends upon the ability to integrate the suitable components Fox, Lantner & Marcom (1997) in their paper "A Software Development Process for COTS-based Information System Infrastructure" Proposed IIDA (The Infrastructure Incremental Development Approach). This process is based on the combination of two models; waterfall model and spiral model. It has two phases: Analysis and Design Phase.

Tran & Liu (1998) proposed a model known as CISD (COTS-based Integrated Systems Development) in their paper "A Procurement-centric Model for Engineering Component-based Software Systems". Which is used to select multiple homogeneous COTS products. The two phases of CISD model are: Identification and Evaluation.

George T. Heineman (1998) Adaptation and Software Architecture, This evaluation survey provides an interesting overview of the state-of-the-art in component adaptation and provides a good starting point for discussions on the nature of component adaptation mechanisms.

S. Mahmood, R. Lai and Y.S. Kim (2007) in their paper," State-of-the-art", research in the area of CBD. The areas surveyed were techniques for component identification and selection, integration, deployment and evolution.

Amandeep Kaur Johar & Shivani Goel (2011) in this paper," Risk identification approach for component-based development" is discussed, a number of risks in various component-based development stages and these risks arise due to the widespread belief that it is a low risk development strategy.

Nicolas Pessemier, Olivier Barais, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien (2002) 'A Three Level Framework for Adapting Component-Based Systems', France,2002. This paper deals with the issue of software adaptation. This also focus on Component-Based Software Development including Architecture Description Languages, and clearly identify three levels of adaptation.

Component integration approaches

Integration is the composition of implemented and selected components to constitute the software system. The *integration process* is based on system architecture and deployment standards defined by component framework and by communication standard for component collaboration [21]. The important task during the integration of components is to deal with the mismatches that may occur when putting together pieces developed by different parties, usually unaware of each other [22]. Several other aspects need to be taken into consideration like component adaptation, validation and testing of selected components, reconfigurations of assemblies and emerging properties of assemblies integrated into the system.

The primary function of adaptation is to adapt the behavior of a component C to integrate it within a target application app. Consider figure 2. The target application, App, has an interface it expects C to support. The identified component may provide most of the expected behavior, but not enough; in the figure, there is glue code written which, in conjunction with C, provides the necessary adapted behavior.

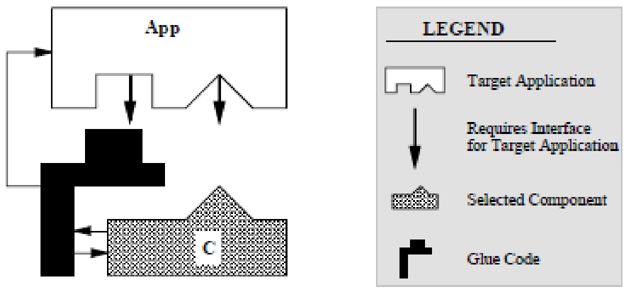

Fig. 2 Adaptation Context

A. *Active Interfaces*

An active interface for a component can be programmed to take action when a method is invoked. There are two phases to a method request: the before-phase occurs before the component performs any steps towards executing the request; the after-phase occurs when the component has completed all execution steps for the request. An active interface allows user-defined callback methods to be invoked at each phase for a method and thus may augment, replace, or even deny a method request. Briefly, each component has an associated component arbitrator that maintains the callback methods installed for the active interface. The arbitrator and the component communicate through a special Adaptable interface. An adaptation to a component is specified at an architectural level and is translated into lower level adaptations.

B. *Subclassing*

Inheritance is a mechanism that allows an object to acquire characteristics from one or more objects. Essential inheritance relates to the inheritance of behavior and other externally visible characteristics of an object while incidental inheritance emphasizes the inheritance of part or all of the underlying implementation of a general object. Essential

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

inheritance is a way of mapping real-world relationships into classes and is used mostly during the analysis and design phase of an object-oriented project. Incidental inheritance often is a vehicle for simply reusing or sharing code that already exists within another class. Inheritance is both an adaptation technique and mechanism. It is automatically built-in to any component written using an object-oriented language like Java or C++. Inheritance has the benefit that newly created subclasses are separate from the original component being adapted.

C. Open Source

Open Source modification occurs when the application builder applies the necessary changes directly to the source code for a component. Naturally, such an approach is possible only if the source code is available and if the application builder is capable of understanding the component's code well enough to make the desired changes. Increasingly, however, more software systems are being developed and deployed using this basic technique.

D. Wrapping

As an adaptation technique, wrapping can be used to alter the behavior of an existing component C. A wrapper is a container object that wholly encapsulates C and provides an interface that can augment or extend C's functionality. The Adapter and Decorator patterns from are useful ways in which to coordinate the controlled extension of classes, but it is typically very hard to impose a design pattern onto an existing class hierarchy. The Wrapping technique typically has no supporting adaptation mechanism [23].

E.Glue code

Glue code is the code used to provide the functionality to integrate different components. It deals with control flow, Component Bridge and exception handling.
Glue code can be used to transfer information between computer languages, it is not required to do so. Generally, it allows one piece of code to call functions in the other, or allows small data values to be passed between code blocks. Generated glue code, particularly when it connects distinct computer languages, often contains code pieces specific for each connected code module.

Conclusions

The idea behind the Component-Based Development (CBD) is to develop software systems not from scratch but by assembling prefabricated parts. It gathers requirements from the customer and selects the appropriate architectural style to meet the objectives of the system to be built. It then selects the components for reuse and qualifies those in order to check that whether they properly fit in the architecture for the system. Component-based software development mainly involves COTS (Off-the-Shelf Components) identification, COTS selection and COTS integration activities. Component Integration gradually puts the pieces together – COTS, glueware and traditionally developed software – to assemble the final application. The important issue when integrating components is to deal with the mismatches that may occur when putting together pieces developed by different parties, usually unaware of each other. Thus, it is extremely important that component services are provided through a standard, published interface to ensure interoperability.

References

1. G.Pour, “Software Component Technologies: JavaBeans and ActiveX”, Proceedings of Technology of Object-Oriented Languages and systems, 1999, pp. 398-402.
2. Szyperski, C., Gruntz, D. and Murer, S. (2008), *Component Software: Beyond Object-Oriented Programming*, Addison-Wesley Professional, Boston, First Edition 1997. ISBN 0-201-17888-5.
3. Mahmood, S., Lai, R. and Kim, Y.S. (2007), “Survey of Component-based Software Development”, IET Software, Volume 1, No. 2, pp. 57-66.
4. Alves, C. and Finkelstein, A. (2002), “Challenges in COTS decision-making: a goal-driven requirements engineering perspective”, In *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering* (Ischia, Italy, July 15 - 19, 2002). SEKE '02, vol. 27. ACM, New York, NY, pp 789-794. DOI= http://doi.acm.org/10.1145/568760.568894
5. Alves, C. and Finkelstein, A. (2003), “Investigating conflicts in COTS decision-making”, *International Journal of Software Engineering***,** 13 (5). pp. 1-21. ISSN 02181940.
6. Maiden, N.A., and Ncube, C. (1998), “Acquiring COTS software selection requirements”, IEEE Software, 15, (2), pp. 46–56.
7. Kontio, J., Chen, S.-F., Limperos, K., Tesoriero, R., Caldiera, G., and Deutsch, M. (1995), “A COTS selection method and experience of its use”, *Proc. 20th Annual Software Engineering Workshop*, Greenbelt, Maryland, 1995.
8. Tran, V., Liu, D.-B., and Hummel, B. (1997), “Component based systems development:

ISSN: 2277-9655
Scientific Journal Impact Factor: 3.449
(ISRA), Impact Factor: 1.852

challenges and lessons learned", *Proc. Eighth IEEE Int. Workshop Software Technol. Eng. Practice*, pp. 452–462

9. Lee, S.D., Yang, Y.J., Cho, F.S., Kim, S.D., and Rhew, S.Y. (1999), "COMO: a UML-based component development methodology", In *Proc. Sixth Asia Pacific Software Eng. Conf.*, pp. 54–61.
10. Lee, J.K., Jung, S.J., Kim, S.D., Jang, W.H., and Ham, D.H. (2001), "Component identification method with coupling and cohesion", In *Proc. Eighth Asia-Pacific Software Eng. Conf.*, pp. 79–86.
11. Jain, H., Chalimeda, N., Ivaturi, N., and Reddy, B. (2001), "Business component identification – a formal approach", *Proc. Fifth IEEE Int. Enterprise Distributed Object Computing Conf.*, EDOC '01, pp. 183–187.
12. Chung, L. and Cooper, K. (2002), "Knowledge based COTS aware requirements engineering approach", *Proc. 14th Int. Conf. Software Eng. Knowledge Eng., (ACM Press)*, pp. 175–182.
13. Carney, D.J., Morris, E.J. and Place, P.R.H. (2003), "Identifying commercial off-the-shelf (COTS) product risks: the COTS usage risk evaluation", Carnegie Mellon Software Engineering Institute (SEI), CMU/ SEI-2003-TR-023, Sept. 2003.
14. Lee, S.C., and Shirani, A.I. (2004), "A component based methodology for web application development", Journal of System Software, 71, pp. 177 - 187.
15. Dietrich, S.W., Patil, R., Sundermier, A. and Urban, S.D. (2006), "Component adaptation for event-based application integration using active rules", J. Syst. Softw., 79, (12), pp. 1725–1734.
16. Rogerson, D. (1997), *Inside COM*, Microsoft Press, ISBN 1-57231-349-8.
17. [COR] OMG, CORBA, http://www.omg.org/corba
18. [JAVA] Sun Microsystems, "JavaBeans 1.01 Specification", http://java.sun.com/beans
19. [NET] Microsoft .NET Framework (2009), http://en.wikipedia.org/wiki/NET_Framework, preview release 19-10-2009.
20. Zhuge, H.: 'A Problem oriented and rule based component repository', J. Syst. Softw., 2000, pp. 201–208
21. Pressman, R.S. (2000), *Software Engineering-A Practitioner's Approach*, McGraw-Hill International Ltd., New York.
22. Cechich, A., Piattini, M. and Vallecillo, A. (2003), "Assessing component based systems, Component based software quality", LNCS 2693, pp. 1–20.
23. Boehm, B., Abts, C., "COTS Integration: Plug and Pray", *IEEE Computer*, 32(1), Jan. 1999, pp. 135-138.