International Journal of Information Technology and Knowledge Management
January-June 2011, Volume 4, No. 1, pp. 11-14

TWIST OF ASPECT ORIENTED AND COMPONENT ORIENTED

Shailendra Narayan Singh[1] & Manu Pratap Singh[2]

Object–oriented is a set of tools and methods that enable software engineers to build reliable, user friendly, and maintainable, well documented, reusable software systems that fulfills the requirements of its users. It is claimed that object-orientation provides software developers with new mind tools to use in solving a wide variety of problems. Object-orientation provides a new view of computation. Aspect Oriented Programming allows programmers to express in a separate form the different aspects that intervene in an application which are composed adequately at a later stage. This paper analyses the problem of crosscutting which is produced during component development, and a component based development extension using Aspect Oriented techniques is proposed. This Component based Software Engineering extension has been named Aspect Component Software Engineering. Component Based Software Engineering and Aspect Oriented Programming are two disciplines of software engineering, which have been generating a great deal of interest in recent years. From the Component point of view, the building of applications becomes a process of assembling independent and reusable software modules called components. However, the necessary dependencies description among components and its latter implementation causes the appearance of crosscutting, a problem that Aspect Orientation resolves effectively. A software system is seen as a community of objects that cooperates with each other by passing messages in solving a problem.

Keywords: Implementation Object, Component, Software Engineering, Reusability Adaptability, Scalability

1. Introduction

Component-Based Development is gaining recognition as the key technology for the construction of high-quality, evolvable, large software systems in timely and affordable manners. Constructing an application under this new setting involves the assembly/composition of prefabricated, reusable and independent pieces of software called components. A component should be able to be developed, acquired and incorporated into the system and composed with other components independently in time and space [1]. The ultimate goal, once again, is to be able to reduce developing costs and efforts, while improving the flexibility, reliability, and reusability of the final application due to the (re)use of software components already tested and validated. Component Oriented Programming aims at producing software components for a component market and for later composition (composers are third parties). This requires standards to allow independently created components to interoperate, and specifications that put the composer into the position to decide what can be composed under which conditions. This approach moves organizations from application development to application assembly.

However, most of the publicity surrounding these component models and platforms is oriented towards gaining the race way under between middleware architects and vendors to establish their products as standards for developing open distributed systems. Thus, whilst companies are focused on highlighting the benefits of software developing using the plug and play mechanism of their products, there is little or no discussion in the media of how to really design reusable, flexible and adaptable components. In this sense, there are reasons in Component Based System, which cause a lack of reusability and adaptability: This imposes a structure on the programs that makes it difficult to have different concerns well-modularized: code-tangling is inherent to Component programs [2]. The uses statement during component specification may be considered harmful. The purpose of these statements is on the specification of receptacles (i.e., a component reference in order to use the operations it provides). However, these references express an aggregation relation between components, thus establishing strong.

Dependencies among components which make them difficult to reuse, adapt and evolve. Aspect-Oriented Software Development is an emerging technology that provides direct support for separating and weaving concerns that crosscut the functional components in a typical software system[3]. Aspect Oriented Programming has been created with the objective of allowing programmers to express separately the different concerns of an application, in order to be composed adequately at a later stage. The main characteristics of software developed with Aspect Oriented Programming are flexibility, adaptability and reusability of the elements used to compose the system [4].

This paper focuses on the study of the current problems of Component Based Systems. On the one hand, each phase

[1]Department of CSE, AFSET, Dhauj, Faridabad, Haryana

[2]Department of CSE, Dr. B.R Ambedkar, University Khandri Agra

E-mail:[1]shailendranarayansingh@yahoo.co.in

of Component based development; that is specification, implementation, package, assembly and deployment, is revised. On the other hand, an Aspect Oriented Programming methodology to develop Component Based Systems business rules is proposed. The final software systems are composed using Aspect Oriented techniques as a "glue" among components, giving that the main advantages provided by aspect orientation to component-based systems. The rest of the paper is as follows: in section 2, the problems arising during Component-based development will are identified. In section 3, our proposal is presented. Finally, we feature a set of conclusions.

2. CURRENT COMPONENT BASED SYSTEM PROBLEMS

Currently, common component platforms like CORBA Component Model (CCM)[5] or Enterprise Java Beans (EJB)[6] are based in the idea of D'Souza in Catalysis[7]. This idea is simple: to build software systems using modules (components) like a builder builds a house, using independent modules. Each module has a specification and an implementation, and then, each is composed to build the final software. For this objective, the interfaces which a component provides and requires are used like the connectors in a "lego piece". In this context, building applications are based on a process to compose/assemble plug&play components. Therefore, building an application requires the following phases: on the one hand, the description and implementation of plug&play components is needed. On the other hand, a process to interconnect and deploy components is required. Initially, component base system methodology increases the quality of software by providing flexibility, adaptability and reusability through the assembly/composition of independent software components. However, individual components are not as reusable and adaptable as may appear in a first place because the crosscutting phenomenon arises in a actual way, as we explain in the following paragraphs.

2.1. Component Based System Adaptability and Reusability

From the adaptability point of view, a system must be adapted when new requirements appear. This means that changes in the business rules can be applied to systems already built with minimal changes. Let us proceed to analyze how we can adapt the functionality of component based system to new requirements. First, a business rule is a process in a software system; therefore, business rule changes introduce software systems changes. These system changes can include ones in functional or non-functional properties.

Updating Non-functional Properties: Common component platforms offer a container to manipulate the non functional system properties like security, persistence, distribution, etc. The container properties facilitate the development of components, because the containers offer common services for all components. The container configuration can be changed during the package phase. During this phase the developer can specify various kind of policies for each component. For example, if we are developing a system using CCM, the security, transactions, and persistence for each component can be configured in Package Phase.

Updating Functional Properties: The container can not be used to change a functional system behavior. CBS functional behaviors are specified by business rules which describe the interconnection among components. This interconnection among components will form the final system. Besides, business rules change for each system, for each domain, etc. Business rules respond to the\ specific problem to be solved, and they establish the specification and interconnection components in the design phase [8]. When a component A declares that uses the services offered by other component B, that declaration affects both the specification and implementation of A. This is due to the fact that the uses statement expresses an aggregation relation between A and B. Furthermore, in the implementation of A, direct calls to B methods appear and they are hard-coded. Consequently, changes in business rules involve updating both the specification and implementation of software components. In addition, specification changes affect to package, assembly and deployment phases of component based system.

3. CONSTRUCTING CBS USING AOP

In this section a new component based development methodology is presented. This methodology combines the principles of Component Based Software Engineering and the flexibility, adaptability and reusability characteristic are provided by Aspect Orientation. This methodology is called Aspect Component Based Software Engineering. In the following paragraphs we are going to express the changes necessary to apply aspect oriented programming in each component based system development phase (design and specification, implementation, package, assembly and deployment).

3.1. System Design and Components Specification Phases

During the component-based system specification, the interfaces that a component provides and requires must be described. For example, in the specification of a CCM[5] component, the interfaces that it provides (facets) and those that it requires (receptacles) are described. We will focus our attention on the dependencies introduced by the uses clause, that is to say, the interfaces it requires from other components. There are two alternatives to define the

dependencies between components: doing it during the specification phase, or leaving it for subsequent phases.

Both approaches have their advantages and disadvantages:

- If the dependencies of a component are described during the specification, they belong to that component and, therefore, they must be maintained by all implementations of that component specification. With this alternative, the component provides a clear and concise idea of its behavior. However, the use of this component is quite limited. For example it is possible that in a specific framework the handling of some of the dependencies is unnecessary or, even worse, the introduction of new dependencies becomes a difficult task.
- If the dependencies between components are not represented during the specification phase. As an advantage, the components can be easily adapted to the requirements of each context. However, CBD phases concerns system architecture (package, assembly and assembly) should be reviewed to apply the component dependencies.

3.2. System Implementation Phase

The implementation phase allows for implementing the component functionality. The component developer should use only the intrinsic dependencies, and in the component implementation there are many calls to methods to intrinsic interfaces. This means that each component only implements the basic business rules. Throughout the implementation phase of the components, each component implements the interfaces it provides, as well as all the methods needed to carry out its functionality. During the implementation of these methods, dependencies can be used in the component implementation but it can only use intrinsic component dependencies. However, all those dependencies defined as non intrinsic dependencies are applied throughout the package phase of software components. Therefore, crosscutting is not being introduced in the implementation of the component due to non-intrinsic dependencies.

3.3. System Package Phase

During the Package Phase, XML descriptors (for example, Component Descriptor in CCM) are used to describe the component properties which form a part of the component system. In this XML description each component identifies the interconnections with other components; that is to say, the system architecture is described through the connection among interfaces which are provided or required by components. The Package Phase allows us to apply the non intrinsic dependencies on the new component based system. The steps are the following:

- First, the non intrinsic dependencies must be defined during System Design Phase using a graphic representation like UML.
- Second, the dependencies which have been described in UML are translated to XML Component Descriptor Specification.
- Third, this XML Component Descriptor Specification describes the non-intrinsic dependencies for each component. This means identifying the new business rules or new dependencies in new contexts or new domains.
- Finally, the information that a specific component describes in its XML Component descriptor is pre-processed in order to recompile the component code, and add the restrictions and dependencies that are specified in this XML Component Descriptor. These dependencies are expressed as aspect implementations through a generic aspect-oriented programming language, such as AspectJ, AspectC++ etc.

3.4. Aspect Component Based Software Engineering Advantages

- Reusability: The component is not recoded when the components are used in other domains or contexts, because the component implementation can be adapted to new business rules by changing the non intrinsic dependencies. Then these components can be coupled with others components.
- Adaptability: Programmers are offered the possibility of modifying the component descriptor by altering the final component functionality.
- Scalability: The system can be easily scalable because we obtain new component implementations and new component specifications. Then these new components with their intrinsic and non intrinsic dependencies can be used to compose new systems.
- Compressibility: Developing a new system is based on following a set of structured phases (Design and Specification, Implementation, Package, Assembly and Deployment). All information about the system is stored by using the common schemas (UML or XML). Besides, the interconnection code is generated by XML translation.

4. CONCLUSIONS

In this paper we have presented a joined component based software engineering and aspect oriented programming proposal in which two of the recent tendencies in software system development are united. We have expanded the life cycle of a component-based system through techniques of aspect-oriented programming with the aim of making good use of the advantages of both tendencies and obtaining more flexible, adaptable and reusable software systems.

In a component based system, the business rules establish and determine the components specification and their relations. However, these relations or dependencies provoke the appearance of crosscutting as we have seen in this paper.

Therefore we have detached every one of the stages in the component based development. Every one of these stages is expanded so that a new description model of dependencies between components, which are materialized during the system composition phase, is implanted. These interconnection descriptions in XML permit us to save time and cost, due to the fact that almost the entire code necessary is generated automatically. Finally, it should be emphasized that currently the interconnection between components is totally transparent to the programmer. This Aspect Component Based Software Engineering methodology has been developed with success in the CORBA Component Model domain.

REFERENCES

[1] Bzyperski, Component Software: Beyond Object- Oriented Programming, Addison-Wesley(1998).

[2] Huclos J, Mstublier and Horat P., "Describing and Using Non Functional Aspects in Component Based Applications," Proceedings of the 1st International Conference on Aspect-oriented Software Development, Netherlands(2002).

[3] Kieberherr K, and Lezini M, Programming with Aspectual Components, Northeastern University(2003).

[4] Piczales, G. Aspect-Oriented Programming, Verang (2004).

[5] M'Souza, M. Objects, Components and Frameworks with UML, 2004.

[6] Dhessman J. and Maniels J., UML Components: A Simple Process for Specifying Component-Based Software, Addison-Wesley, (2001).

[7] C. Tudinsky, M. Ainnie, J. Glissides, "Automatic Code Generation from Design Patterns", Object Technology, (2003).

[8] F. Nolland, "The Design and Representation of Object-Oriented Components", Computer Science Department, Northeastern University (2001).