IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com

ITP4272

A New Approach to Components

Zoran Stojanovic and Ajantha Dahanayake
Faculty of Information Technology and Systems, Delft University of Technology, The Netherlands
Tel: +31 15 278 6328, Fax: +31 15 278 6632, {Z.Stojanovic, A.N.W.Dahanayake}@is.twi.tudelft.nl

ABSTRACT

Component-Based Development (CBD) represents an advanced paradigm for building complex enterprise systems of nowadays. While the necessary technology has been already settled down in practice, new processes, strategies and techniques for component-based modeling, analysis and design are still lacking. This paper presents a new approach to components providing a consistent and traceable component-based process from business services to implementation artifacts.

INTRODUCTION

During the last years, a new paradigm of Component-Based Development (CBD) has been introduced. It represents a realization of a decades-old dream about software systems built from components. CBD provides a solution for developing faster, cheaper, more adaptable and reliable Internet-based, e-business systems. As in many cases before, first the new technology solutions have been initially adopted by developers, and then the analysis, specification and design activities have been modified to reflect the new concepts. While the component-based platforms and technologies, COM+ [COM], Enterprise Java Beans [Sun Microsystems] and CORBA Components [Siegel, 2000] are now de facto standards, there are strong requirements for methods, approaches and techniques for developing business-driven component-based systems targeting available technology [Stojanovic et al., 2001a]. Therefore, a set of consistent component concepts, completely independent of technology solutions, must be provided.

The aim of this paper is to provide a more comprehensive and integrated view on components and component-based development. By using the technology-independent set of component concepts, the paper provides a step toward a consistent and traceable component-based process from business services to implementation artifacts. The main idea of this approach is that the same component way of thinking can be applied across different aspects of enterprise systems development. In this way, a component, as a service provider with hidden interior and contract-based interfaces, represents a point of convergence of business and technical concerns, equally well understood by both sides.

STATE-OF-THE-ART OF COMPONENTS

The ideas of modularization and software reusability are almost as old as the idea of software. The NATO Conference in 1968 recognized that development of complex software systems should be treated as an engineering discipline, providing the system made from components [McIlroy, 1969]. Since that, the principles of separation of concerns, encapsulation, and pluggable parts have been distinguishing good, flexible software system designs from bad through functions, subroutines, modules, units, packages, subsystems, objects and finally components.

Different component definitions show that components may come in different forms and granularity and different participants in the development process see components differently. However, the physical perspective on components as binary packages of software is still predominant. The standard Unified Modeling Language (UML) treats components as packages of binary code and uses them in describing system implementation through component and deployment diagrams [Booch, 1999]. Catalysis, a component-oriented approach with its origins in object-oriented analysis and design methods, defines a component as a package of not only software code than also other software artifacts [D'Souza and Wills, 1999]. By [Szyperski, 1998], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. Gartner Group defines a runtime software component as a dynamically bindable package of one or more programs managed as a unit and accessed through documented interfaces that can be discovered at runtime [Gartner Group, 1997].

When introducing components, the question about similarities and differences between objects and components naturally arises. By [Udell, 1994] components represent a new silver bullet for system development in the Internet age, while objects have failed to provide higher level of reusability. For others, components are nothing else than larger-grained objects deployed on the network nodes [Booch, 1999]. By [Szyperski, 1998], a component comes to life through objects and therefore would normally contain one or more classes. In other debates [Henderson-Sellers et al., 1999] granularity has been seen as the main issue in distinguishing objects and components. By Catalysis, components are often larger-grained than traditional objects, and can be implemented as multiple objects of different classes.

In the sequel, we will present a consistent and comprehensive component-based approach that is technology-independent and can be equally applicable in business modeling, as well as system analysis and design.

AN INTEGRATED COMPONENT-BASED APPROACH

For years system developers and integrators have struggled to translate and interpret business requirements into systems implementation that at the same time fulfill business goals, delineate business structure, and provide efficient development of adaptable solutions. Component-based development, in our opinion, can represent a link between different perspectives on the system, making a common ground for business and technical concerns. Component-based thinking is meaningful and equally applicable at the levels of business modeling and system modeling, as it is in system implementation. Regardless of the context of usage, the essence of the component-based approach is the explicit separation between the outside and the inside of the concepts being addressed. This means that only the question WHAT is considered (what useful services are provided by the particular building block to the outside world) not the HOW (how these services are actually implemented).

Regarding the context in which components are identified, specified and used, components can be of different forms, granularity, and nature. Thus, any attempt to provide a single, general definition of a component covering all its possible aspects may be insufficient. Examining essential properties of a component can be more appropriate:

- Component is as independent as possible from other components, following "low coupling – high cohesion" principle.
- The interior of a component is completely hidden behind explicit, well-defined interface through which a component communicates with other components in the form of providing to and using services from them. Interfaces are responsible for specifying what is the role of the component in the wider context, what the component is, what it does, and under what conditions, but not how that behavior is actually implemented.
- Component is represented through detailed and precise specification of its provided and required interface(s). Fully specified interfaces by

This paper appears in *Issues and Trends of Information Technology Management in Contemporary Organizations,* the proceedings of the Information Resources Management Association International Conference. Copyright © 2002, Idea Group Inc.

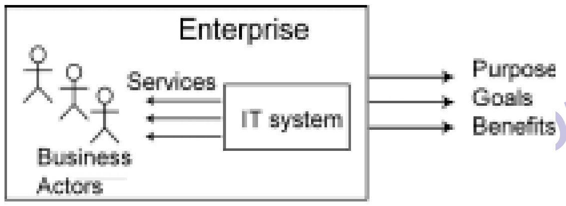
pre-conditions, post-conditions, invariants and guarantees actually form a contract between components - consumers of services and components - suppliers of services.

- Component denotes composition. It does not exist in isolation; it takes part in collaborations with other components in order to fulfill a goal. Composition is an inherent characteristic of a component. Every component can be represented as a composition of smaller-grained components, but also can form a larger-grained component in composition with other components.

At the highest level of abstraction, the whole enterprise can be considered as a collaboration of its components, i.e. business actors (people, systems, departments, companies, etc.) that have responsibilities, fulfill roles, manage necessary information, perform actions, take part in interactions, and work together to achieve some business goals. All these actors can be represented as components of the enterprise; their interior is hidden and not important as long as they provide correspondent services to fulfill their roles in the enterprise; they collaborate in the contractual manner in order to fulfill the common goal.

In the modern enterprise world some roles can be performed by the information technology (IT) systems. The behavior of the system, in the context of the business for which it is implemented, has to be specified. Functional and non-functional requirements on the system must be defined in the form of business concepts and services, which must be supported and realized by the system. At this level, the whole system can be considered as a component of a given enterprise that performs particular roles and provides services for automated business processes. These roles of the system identify the parts of the business processes for which the system is responsible and the artifacts that are involved. Such artifacts and resources represent the information held and acted upon by the system, usually stored in the form of database.

Figure 1: IT system as a component of an enterprise



Considering cohesive set of services with minimal interfaces provided by the system to automate business processes for which the system is responsible represents a good foundation for identification and specification of business components as providers of those services. At this level it is not important how these services of the system are implemented as long as they, in collaboration, provide expected behavior of the system in the enterprise. In fact the service seems to be a pivotal concept that links the business and system perspectives, providing traceability from business processes, to business components and then to software components. Business component inherits all essential component characteristics and provides business and/or business-oriented technical services for the business processes in which the system participates, i.e. it adds certain value to the business actors interacting with the system.

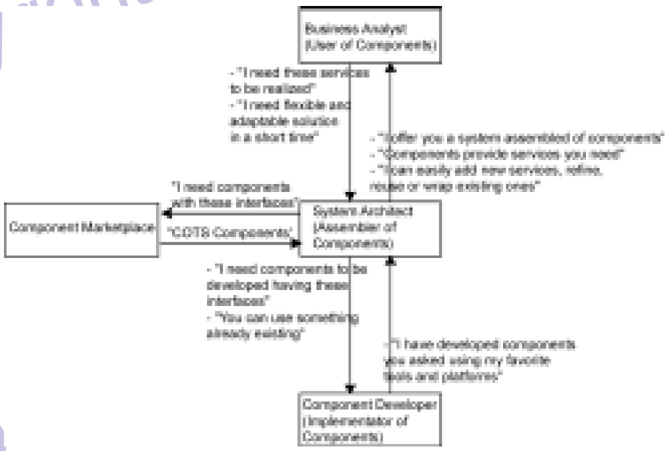
Assigning service requirements that should be fulfilled by the system to autonomous business components provides decomposing both requirements and system function into well-partitioned areas, managing complexity through the implicit divide-and-conquer strategy.

When encapsulated, contractually specified, highly cohesive and low-coupled providers of services, i.e. business components are defined, they should be put in the context of the system being realized in terms of system components. At this level system architecture should be defined. It gives a logical view of the system structure in terms of system components and dependencies between them. Each previously defined business component can be represented as one or more system components. System components as opposite to business components, provide some lower grained services, which have more technical than business meaning and therefore do not add value to the actors around the system. After the system architecture is defined a development team can assembly a component-based solution by deciding to:

- build business components and/or system components from scratch using advanced component-based technology,
- wrap existing assets with proper interfaces to form component-like structures
- buy components as Commercial-off-the-shelf (COTS) assets
- combine these three strategies

According to traceable usage of component thinking presented above, three human roles in the system development process using components can be defined: business analyst (user of components), system architect (assembler of components) and component developer (implementator of components). For a business analyst a component represents a clear, rigorous, implementation-independent specification of an encapsulated business service. For a system architect, a component is an encapsulated building block of the system architecture with fully specified interfaces to the rest of the system. For proper assembling of the components into an overall system solution implementation details are not important as long as exposed interfaces are guaranteed. Component developer "opens" a black-box component and provides component implementation according to previously defined business and system requirements, and interfaces that should be realized. Dialogs between these three roles in the development process, as well as their views on components are presented in Figure 2. In this way the same set of component concepts and way of thinking are used throughout the development process, providing traceability from business concepts and processes to implementation artifacts.

Figure 2: Dialogs between actors in the development process


Business Analyst (User of Components)
- "I need these services to be realized"
- "I need flexible and adaptable solution in a short time"
- "I offer you a system assembled of components"
- "Components provide services you need"
- "I can easily add new services, refine, reuse or wrap existing ones"
"I need components with these interfaces"
Component Marketplace
"COTS Components"
System Architect (Assembler of Components)
- "I need components to be developed having these interfaces"
- "You can use something already existing"
- "I have developed components you asked using my favorite tools and platforms"
Component Developer (Implementator of Components)

CONCLUSION

There has been a rapid rise of interest in component-based development paradigm, followed by the huge marketing hype. By some, components make objects obsolete; by others components are larger objects. Because of different points of view on components, variety of proposed component definitions and loose usage of the term, there has been much confusion about how to use component concepts efficiently. Physical perspective on components as packages of binary code has been predominant, significantly reducing the potential benefits from component paradigm.

The aim of this paper is to provide a more comprehensive and integrated view on components and component-based development. The paper defines the set of essential component properties, and an integrated component approach consisting of different perspectives (business, system, and software) on the component usage. The same way of thinking can now be applied from business to software, providing traceability and consistency along the way. The concept of component, as an encapsulated service provider, represents a common ground for business and technology, well understood by both sides.

At the moment when IT and business have started to fully exploit components, a new system development paradigm, called web services, has emerged [Microsoft, 2001][IBM, 2001]. Web services are self-contained, self-describing, modular units providing a location independent business or technical service that can be published, located and invoked across the Internet. They represent a natural extension of component thinking and further convergence of business and technology. From a technical perspective the web service is essentially an extended and enhanced component interface constructs, but now instead of maintaining a necessary component in-house, required services are invoked across the Internet, without taking care what artifacts are responsible for providing them.

REFERENCES

Allen P., Frost S. (1998), "Component-Based Development for Enterprise Systems: Applying the Select Perspective", Cambridge University Press.

Booch, G., Rumbaugh, J., Jacobson, I. (1999), "The unified modeling language user guide", Addisson-Wesley.

COM, Microsoft Component Object Model Technologies, information available at http://www.microsoft.com/com/

D'Souza D.F., Wills A.C. (1999), "Objects, Components, and Frameworks with UML: the Catalysis Approach", Addison-Wesley.

Sun Microsystems, Enterprise JavaBeans, The source for Java^{TM} technology at http://java.sun.com

Gartner Group (1997): "Componentware: Categorization and Cataloging," Applications Development and Management Strategies Research Note, by K. Loureiro and M. Blechar, December 5, 1997, http://www.gartnergroup.com.

Henderson-Sellers, B., Szyperski, C., Taivalsaari, A., Wills, A., (1999)"Are Components Objects?" In OOPSLA'99, Panel Discussion.

IBM (2001), IBM Web Services, available at http://www.ibm/com/webservices

McIlroy, M.D. (1969), "Mass Produced Software Components", Software Engineering: Concepts and Techniques, Edited by P. Naur et al., Mason/Charter Publishers Inc., New York, pp. 138-150, 1969.

Microsoft (2001), Microsoft .NET, available at http://www.microsoft.com/net/

Siegel, J. (2000), "CORBA 3: Fundamentals and Programming" OMG Press, John Wiley & Sons, Inc.

Stojanovic, Z., Dahanayake, A., Sol, H., (2001a) "A Methodology Framework for Component-Based System Development Support", the sixth EMMSAD'01, Interlaken, Switzerland, June 4-5 2001, pp. XIX-1 – XIX-14.

Stojanovic, Z., Dahanayake, A., Sol, H., (2001b) "An Integrated Component-Based Approach to Enterprise System Specification and Development", the 3rd ICEIS 2001, Setubal, Portugal, July 7-10 2001, pp. 667-672.

Szyperski C. (1998), "Component Software: Beyond Object-Oriented Programming", ACM Press, Addison-Wesley.

Udell J. (1994), "Cover Story: Componentware", Byte Magazine, May 1994.