Chapter 4
Component-Based Modeling for Information Systems Reengineering

Malleswara Talla
Concordia University, Canada

Raul Valverde
Concordia University, Canada

IGI GLOBAL PROOF

ABSTRACT

An Information System can be envisioned as a set of interdependent components that provide the intended services. The component based modeling serves as a tool for collecting requirements of an Information System in user perspective and business perspective at various stages of software development. The chapter presents a methodology for component based modeling and development of an Information System, starting from the requirements definition phase, arriving at candidate components and creation of final components and their interfaces. The methodology aims at clarifying the intricate details and usage of an Information System via business type models and use-case models. The chapter presents the interaction diagrams in order to describe interactions among objects in systems perspective, and context diagrams for reflecting upon the business domain. Finally, the chapter proposes component replacement as a methodology for system reengineering, and model-view-control framework for component refinement and evolution in order to achieve a reengineered information system that reflects upon current requirements in business domain. The reengineering techniques proposed in this chapter can be applied to legacy systems to turn them into a component-oriented reengineered system.

DOI: 10.4018/978-1-4666-0155-0.ch004

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

1. INTRODUCTION

Recent trend in systems engineering is component-based using the technologies such as Component Object Model (COM), Common Object Request Broker Architecture (CORBA), Easy Java Simulations (EJS), etc. A *component-based model* is an interconnected set of software components that collectively represent an information system. A component is a modular piece of software that provides a service. Components communicate via interfaces. Component-based software architecture allows system design in a pragmatic manner, either for developing a new system or reengineering an existing system via reverse engineering where needed. The *reengineering* of a system is actually examining and modifying the system that has been in-use for a long time, to improve it and turn it up-to-date in terms of business rules and system performance needs. Reverse engineering is proposed in (Landry Chouambe, Benjamin Klatt, and Klaus Krogmann, 2008) when the interfaces are not explicit or composite components are not supported in a system. An adaptive component model can be dynamic and evolve with the changing needs via adaptation to different execution contexts (Xin Peng, Yijian Wu, Wenyun Zhao, 2007). The component-based modeling offers an opportunity to speed up the systems development via Commercial Off-The-Shelf (COTS), Open Source, or In-House components readily available for inclusion during design stage (Pasquale Ardimento, Giovanni Bruno, Danilo Caivano, Giuseppe Visaggio, 2007). The component based modeling also promotes an incremental specification validation and runtime adaptability in a distributed component information systems (Nasreddin Aoumeur, Kamel Barkaoui, Gunter Saake, 2007). An *information system* is a computer application that efficiently processes, stores and relays information within an organization and across its customers, suppliers, and all other public relationship organizations. The component-based models are so flexible that these models can support dynamic decisions in concurrent via component selection and reuse at runtime (Biplav Srivastava, 2004). Likewise, component based modeling resulted in efficient system design, development, and reengineering.

This chapter presents a methodology for component-based modeling and development of an information system, starting from the requirements definition phase, identification of components and their interfaces. The methodology uses *business type* models and *use case* models for understanding the requirements and identifying the system components. A business type model is a conceptual map of all data and information of interest for the information system. A use Case model is a description of steps that reflect upon a usage scenario or a system feature with respect to a system user or actor. The component interaction diagrams describe the interactions among objects in the systems perspective, and context diagrams for reflecting upon the business domain. While design follows a well defined methodology, system reengineering is proposed via component replacement strategies, where complexity of components is analyzed for appropriate resizing of components. The same methodology can be used for reengineering legacy information systems where parts of legacy system features can be turned into new components or replaced with existing components.

2. COMPONENT BASED SYSTEM MODELING

A component is a reusable piece of software that offers a defined service as a part of a system in which it is used. The challenge in component based architecture and design is to reuse as many existing components and designing new ones that are not already available. Component-based modeling includes several activities to start with an objective of defining the requirements of an information system at the technical and user

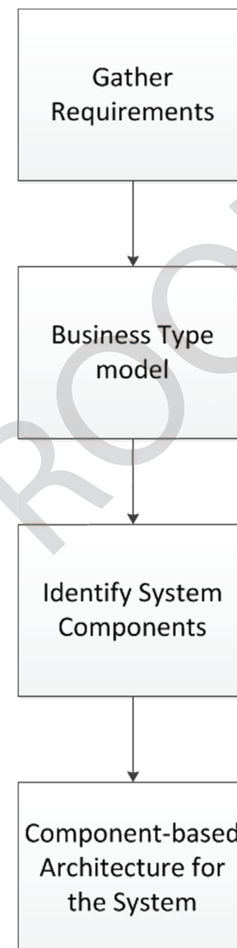
IGI GLOBAL PROOF

level. The first phase of the component-based modeling methodology is to capture the requirements and preparing an appropriate *business type models* which serves as a basis for a *component architecture* by identify component specifications, interfaces and dependencies. The Figure 1 presents a series of activities performed during requirements definition phase of component-based systems development.

The system requirements are captured via *use case* model, and appropriate system components are identified (Cheesman, J. & Daniels J. 2001). The business type models define the information in terms of software specifications, whereas the Use Case modelling provides the requirements in terms of system usage point of view. Based on these two types of models, a component architecture is constructed, and the component responsibilities and dependencies are defined. The components provide interfaces to the direct users, where a user can be another system component which may be processing intensive, or the one that may provide a graphical user interface (GUI) for seeking input or presenting the results. Therefore, system components are characterized by an interface model, which is a reflection of input or output to a component as a set of attributes or data structures, operations and invariants at the interface level. The Figure 2 starts with business type models and use case models for identifying the components that serve as input to systems analysis phase during which a set of components are identified. The Figure 2 also identifies a comprehensive architecture which is a set of interacting components that provide the required results.

In this methodology, an information system can be viewed as a set of collaborating components. Each component acts upon a set of input parameters and states, and provide an interface to other components. The component interfaces in the interface diagram have operations with data signatures and use the terminology defined in the business type model or use case model.

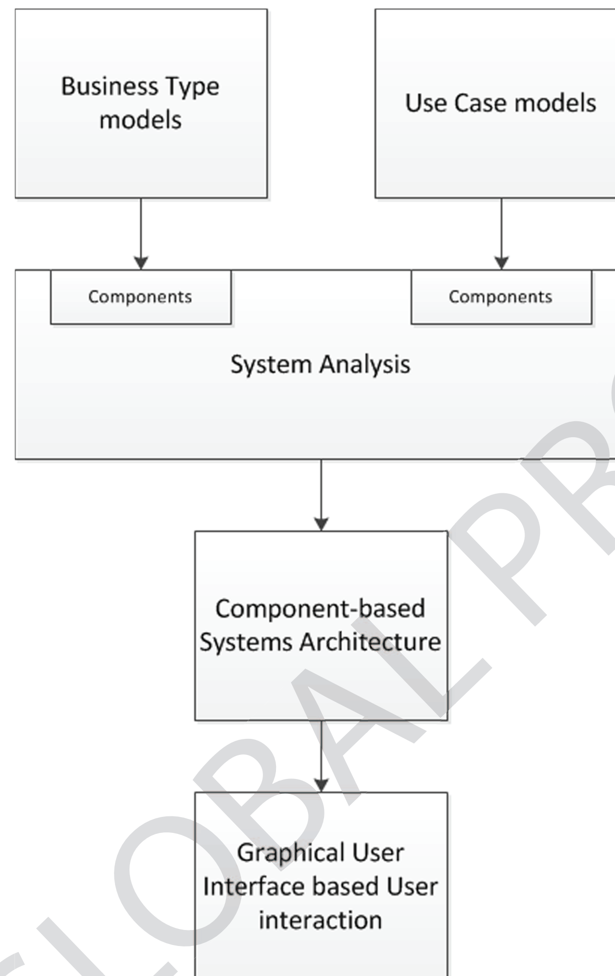
Figure 1. Requirements definition



The component interactions at the interfaces can be detailed in the *component interaction diagrams* which as the messages or data exchanged between the collaborating objects whereas the functionality resulted among these interactions can be characterized in *context diagrams* (Brown, A.W. 2000). The Figure 3 presents a comprehensive set of components interacting among one another that constitute a component-based systems architecture. The incoming arrows represent input to a component whereas an outgoing arrow present a result or a service offered by that component. The overall system provides the intended results.

IGI GLOBAL PROOF

Figure 2. Deriving component based systems architecture



IGI GLOBAL PROOF

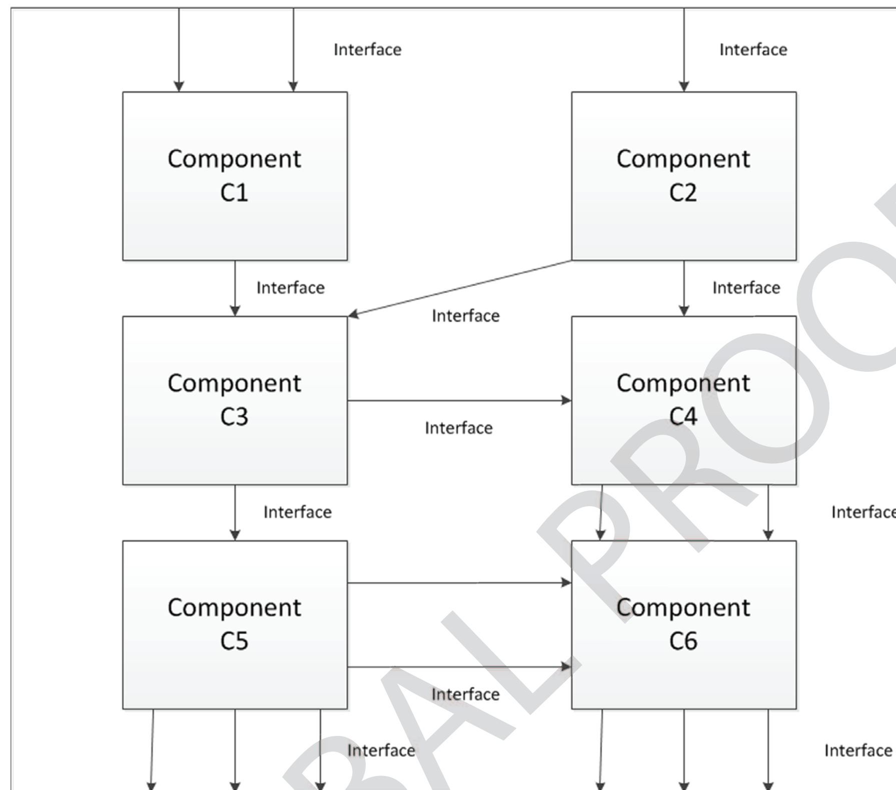
2.1 Business Type Models

The goal of *business type modeling* is to gather various information that helps the systems architect in conceiving a set of components each of which works with unique data structures, and provide interfaces to the users or other components. The requirements gathering via business type models can be performed in the following steps (Brown, A.W. 2000):

a. Collect information of interest, type of information, and limitations such as maximum and minimum values,
b. Conceptual map of information collected in step (a) while eliminating redundant information, and
c. Identifying the system components.

First, the information of interest and core types are identified and a *type model* depicting how the concepts of one entity relates to another entity. Next, the conceptual map is refined into a non-redundant *type model* that details various stages of data processing in the information system, which is just like the traditional Entity Relationship Diagrams (ERD); where entities are called *types* and relationships are called *associations*. Finally,

Figure 3. Component based system architecture



IGI GLOBAL PROOF

refine the *business type* model eliminating the redundant types for identifying the unique business elements (Brown, A.W. 2000) that specify structured data types. The resulting diagram can be called as business type model which provides the required information to the systems architect in modeling component-based architecture. The Figure 4 presents a set of information entities and their relationships with one another that serves as a source of systems architecture and design. The conceptual map is further detailed into actionable at runtime as business information in Figure 5.

2.2 Use Case Modeling

Use case modeling is the process of modeling a system functions in terms of usage scenarios and business events, that detail how the system responds to the events (Whitten, J. L., Bentley D. L. and Dittman K.V. 2000). A Use Case is a description of a user interaction with the information system (Brown, A.W. 2000). An *use case* contains the participating *actor*s. An *actor* could be anything (a user, a role, a person) internal or external that interacts with system to exchange information. An actor can be a user, a role, or a person, who can be inside or outside the system. (Rumbaugh, J. 1994). The Figure 6 presents a set of users, actually customers receiving service of a sales system.

2.3 Interaction Diagrams

An *interaction diagram* is a reflection of a *use case* detailing the interactions among the objects to provide the functionality specified in the *use case*. The Unified Modeling Language (UML) tools provide two types of diagrams: *sequence*

Figure 4. Conceptual map of information

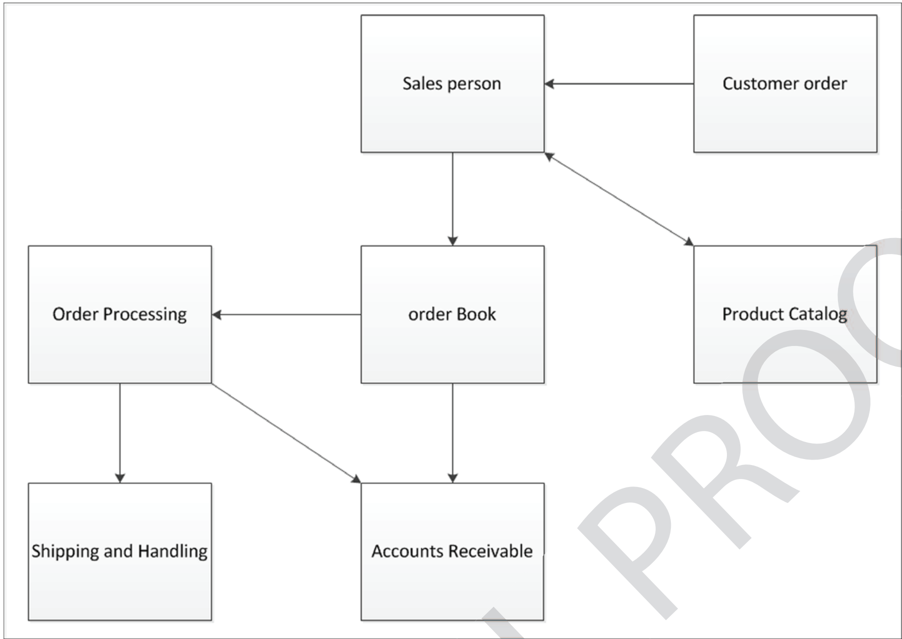
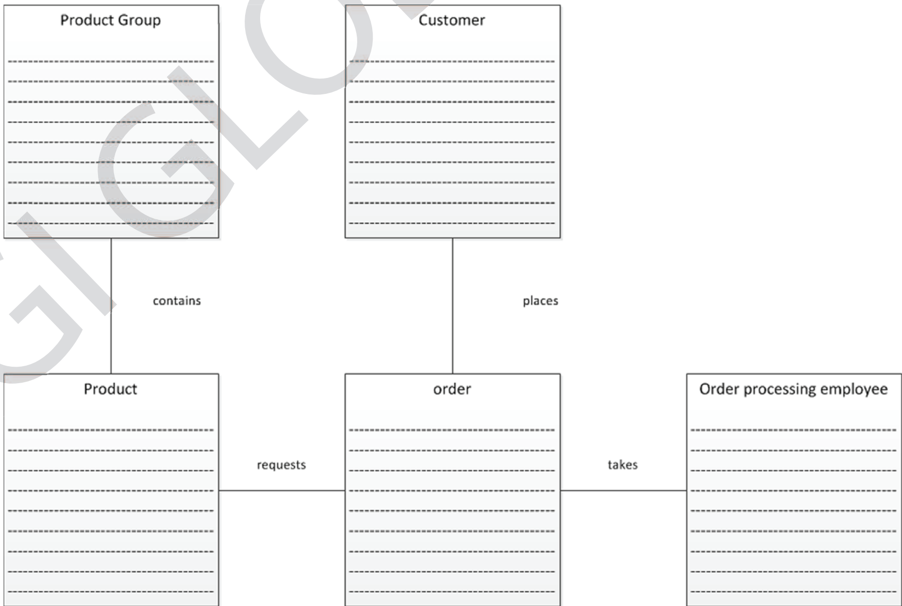


Figure 5. Business type model



IGI GLOBAL PROOF

Figure 6. Use case model

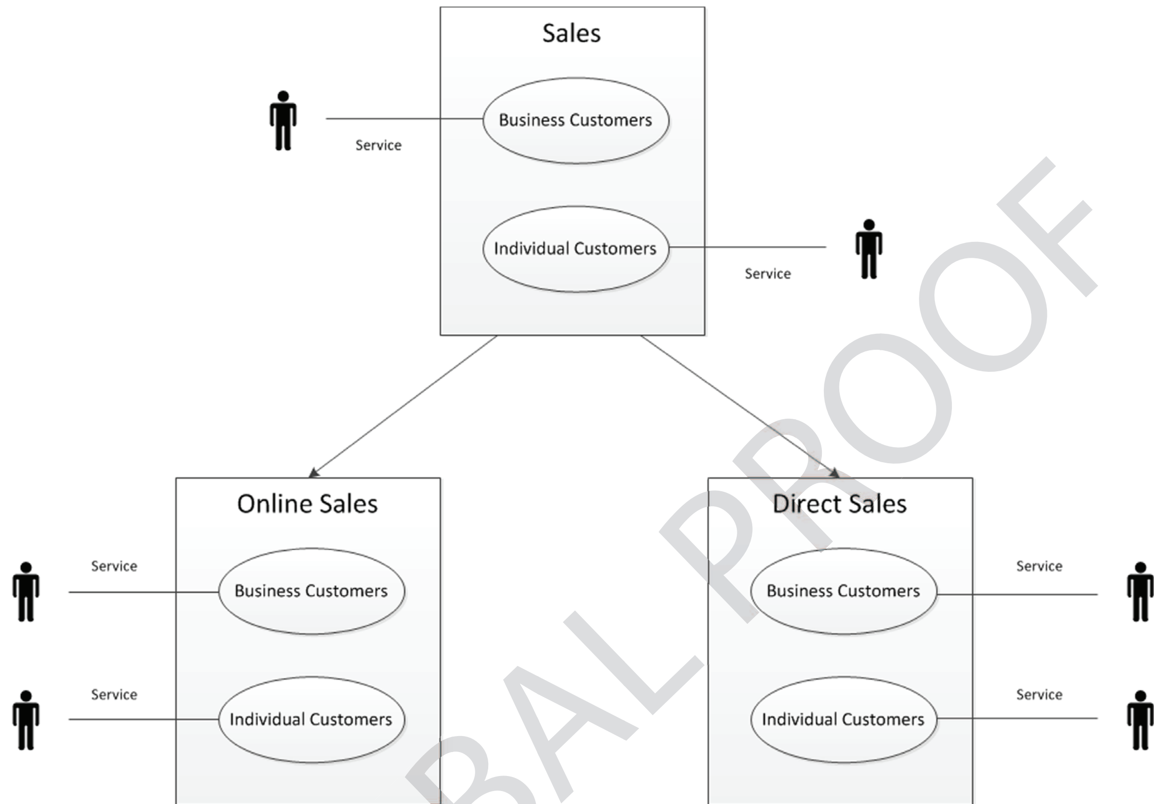
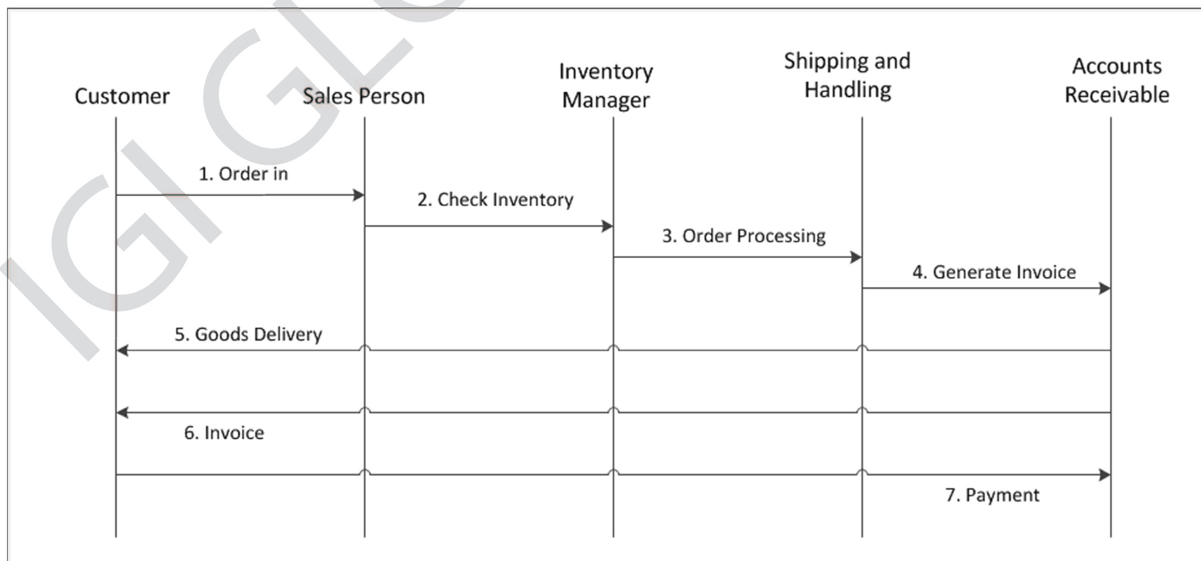
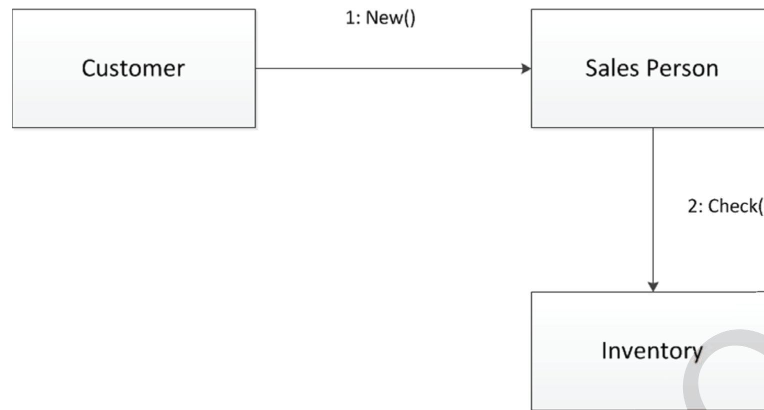


Figure 7. Sequence diagram



IGI GLOBAL PROOF

Figure 8. Collaboration diagram



diagram, and *collaboration diagram* to graphically depict these interactions. A *sequence diagram* provides interactions among objects that complete desired operation or result. (Brown, A.W. 2000). The Figure 7 presents a set of objects interacting one another via messages exchange, where each message triggers further actions in the system.

A *collaboration diagram* is to get a quick overview of all the objects that collaborate to support a given *use case* scenario. The Figure 8 represents a collaboration among objects for a specific use case.

In UML, both *sequence diagrams,* and *collaboration diagrams* are often used for eventually reflect upon the *use cases*.

2.4 Interface Modeling

An interface to a component consists of a set of parameters or data structures, messages, or procedure triggering a certain processing, which provide the required information either as an input for further processing, or for an output to present the results. The goal of interface modelling is to define a set of parameters and messages at the interfaces and describe their details. An interface is used for detailing the common behaviour of a set of objects (Brown, A.W. 2000). It specifies collaborations between components and illustrates how components will collaborate to support each atomic *use case*.

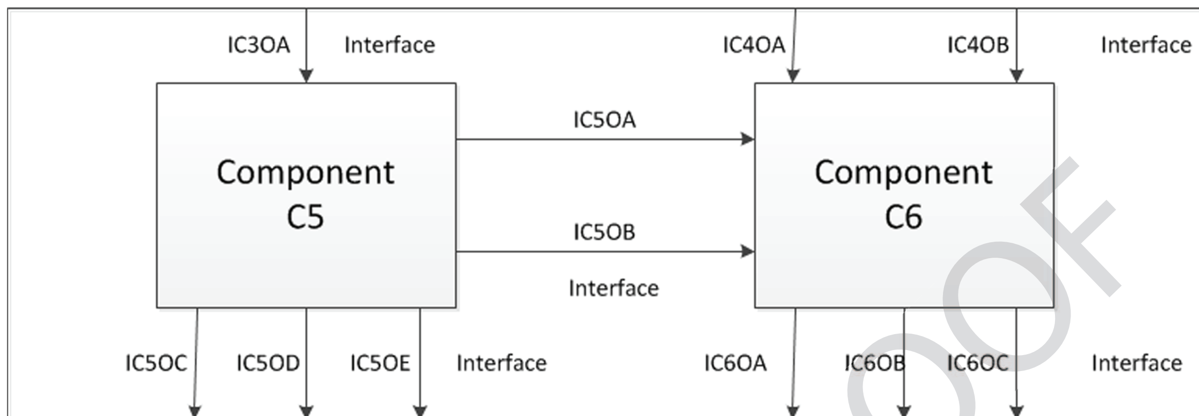
User components access them through their interfaces; these are usually described using three kinds of information:

- *Attributes* for recording information about the state of instances of the interface;
- *Operations* supported by that interface providing access that the interface offers; and
- *Invariants* constraining the allowable states of objects supporting that interface.

The Figure 9 presents various input and output details at the interface. The components receive input data at the interface and provides output data as a set of parameters. Once an understanding of the roles and responsibilities of each of the interfaces are established, the interface details can be completed. First, the *signature* and behaviour are considered. For each operation, its parameters are defined by making use of appropriate types. Two different types of conditions namely *pre-condition* and *post-condition* behaviours are defined (Brown, A.W. 2000).

IGI GLOBAL PROOF

Figure 9. Interface model



IGI GLOBAL PROOF

2.5 Context Modeling

Once a candidate component architecture has been created, the behaviour of each component and the resulting system can be analysed in more detail. At this stage, the expected system functionality is defined through interfaces. Context modelling focuses on the system being developed, and the responsibilities of all involved components. A context model is a high-level view of business context in terms of procedures, roles and responsibilities. A context model can be used to describe a complete system in the context of a business domain or can be scoped to describe a particular component and its context. An example of a context model is depicted below (Brown, A.W. 2000). The Figure 10 presents the details of processing a use case, as a set of sequential or parallel processing procedures.

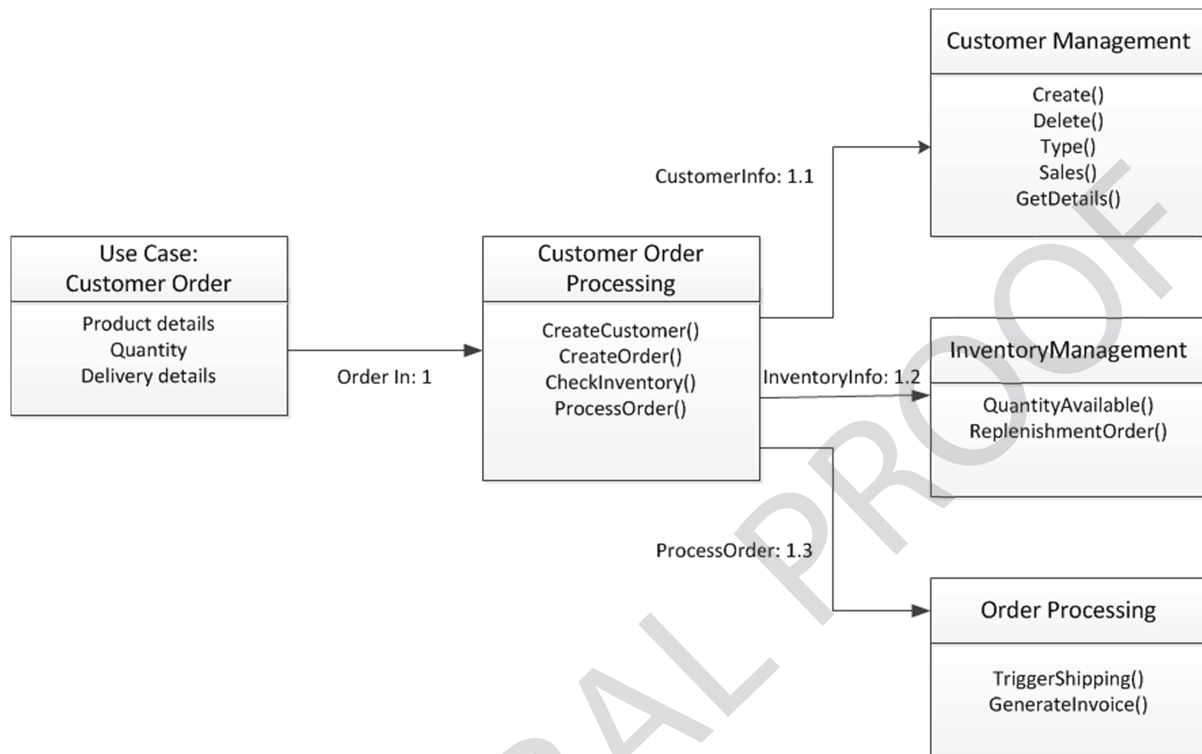
2.6 Component Architecture

The component architecture model should define all components, their responsibilities, specifications, interfaces and dependencies. The Figure 11 presents a set of components that collectively provide a service of the system. The very inputs are received via user interface which can be a graphical user interface (GUI) involving a real person, or another system that provides all input via systems interaction.

3. COMPONENT REPLACEMENT FOR REENGINEERING

The component-based reengineering involves either redesigning and revising an existing component for the required changes, or replacing an existing component with an equivalent new one that provides the same service. Redesigning an existing component requires reverse engineering to fully understand the existing software component, when software evolution was not fully documented. The overall effort for redesigning a component could be expensive, and a simple component replacement could be a better strategy for reengineering. It is hard to find another component that provides an exact interface; however it may be possible to take into consideration only the required services and ignore irrelevant services at the interface; on the other hand, a *glue code* can be triggered for any required trivial changes at the interface level. Component replacement strategy for system reengineering requires a careful evaluation for plug-and-playing a component since a component requires all relevant inputs in order to produce the required output. If an input parameter

Figure 10. Context model



IGI GLOBAL PROOF

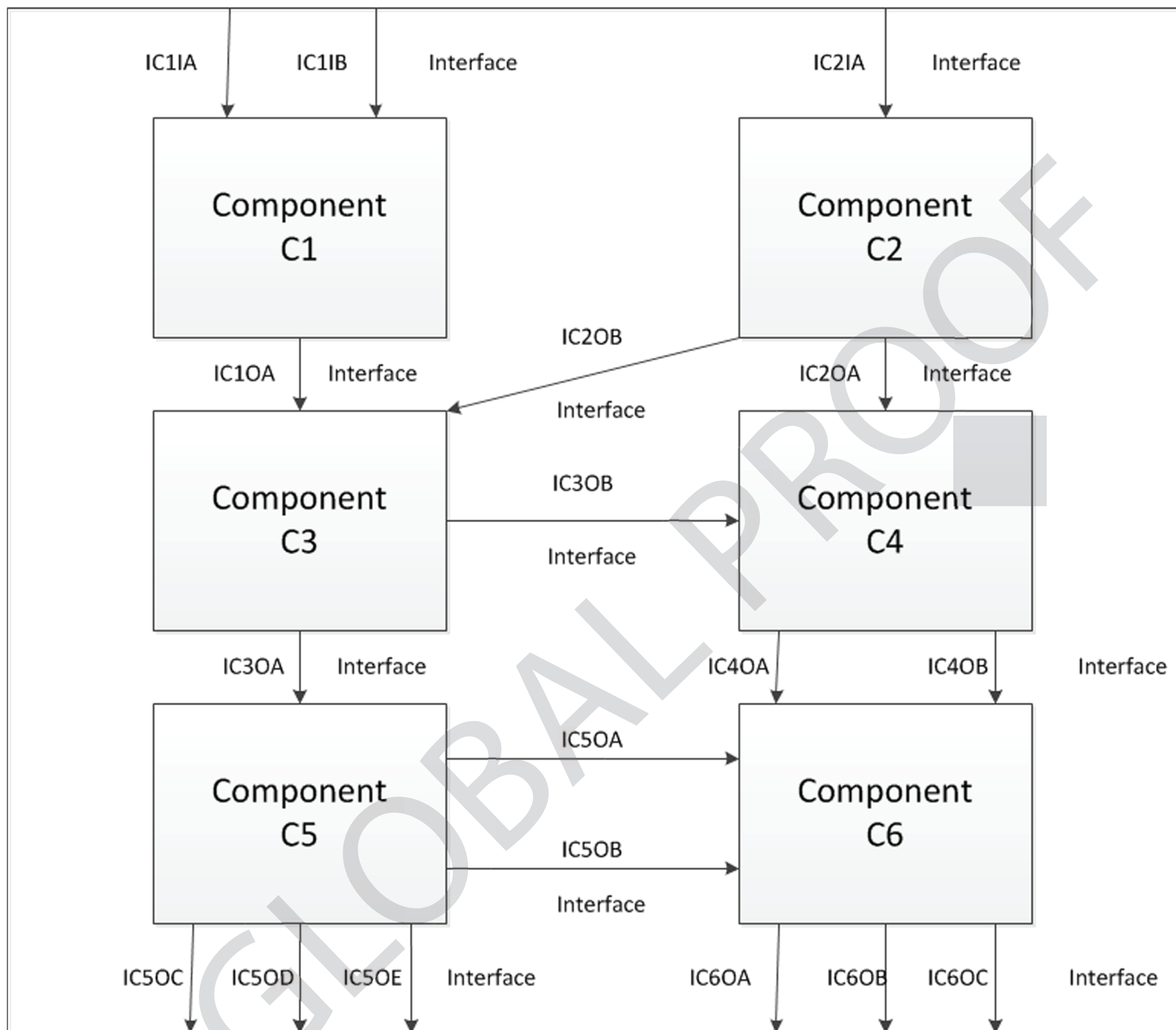
to a software component is not required, such input parameter can be ignored and be left to a default value at the interface level. While replacing a component with another that provides similar but not exactly the same service, then a trivial *glue code* may help to revise the service as required. The *glue code* framework is analogous to the control part of Model-view-Control (MVC) framework, which not only provides an opportunity for testing but also tuning the component services to meet the user expectations. A MVC framework involves a model as a set of classes reflecting upon a component, view as a GUI interface to the user, and control as a communication control providing the required interface. The Figure 12 presents a MVC framework where the glue code can be on the model side in (a) and on the view side in (b).

A carefully documented component provides all information regarding the service offered by a component and how it is accomplished; and such documentation provides a basis for evaluating the feasibility of using the component as a replacement for an existing component in a system reengineering. The component replacement strategy could be implemented in an appropriate manner based on complexity as follows:

a. One-to-One component replacement,
b. One-to-Many component replacement,
c. Many-to-One component replacement, and
d. Many-to-Many component replacement.

A component of moderate size and complexity can be replaced with an equivalent component that provides the same service. Consider a system consisting six components with well defined interfaces, the above component replacement strategies can be implemented as described below. In Figure 13, component C6 is replaced with a new component NC6. While conducting such replacement, it

Figure 11. Component architecture modeling



IGI GLOBAL PROOF

is important that the interface provided by the new component NC6 should be exactly same as that of C6. However, due to system evolution if any of the input interface elements (IC4OA, IC4OB, IC5OA, and IC5OB) or output elements (IC6OA, IC6OB, and IC6OC) become redundant, such elements can be omitted in the new component NC6. On the other hand, if any of these elements require some kind of transformation, the *glue code* can do the needful.

When a component is very complex and an equivalent component is not available, but a set of components collectively match the services offered by the existing one, it can be replaced with those new components by carefully reengineering the interfaces, and eventually accomplishing the same service. In Figure 14, component C6 is replaced with NC61 and NC62 which receive their required interfaces from other components and collectively provide a matching interface of C6. Such reengineering simplifies the architecture and enhances system maintenance and the scope for system evolution.

Figure 12. MVC framework

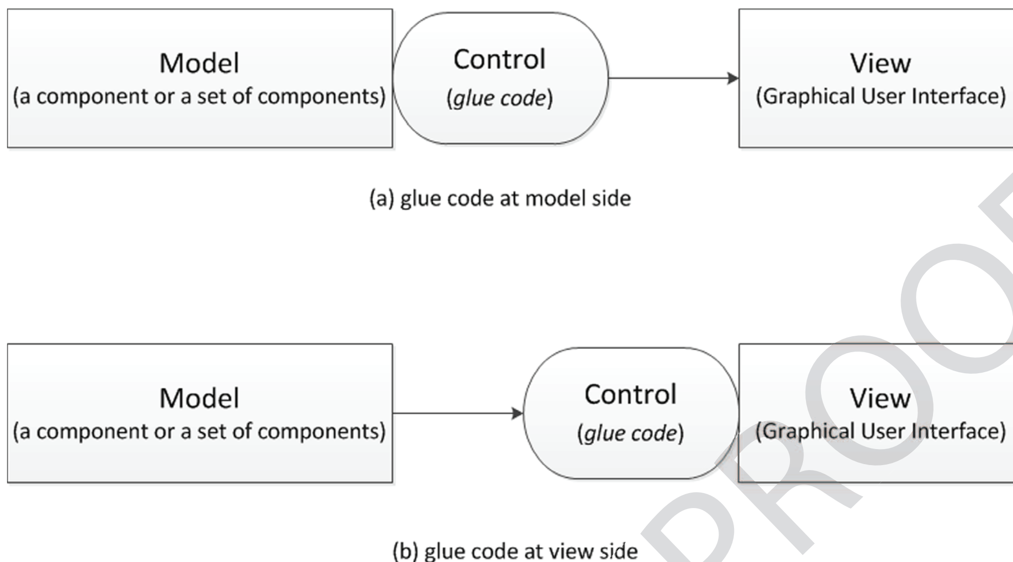
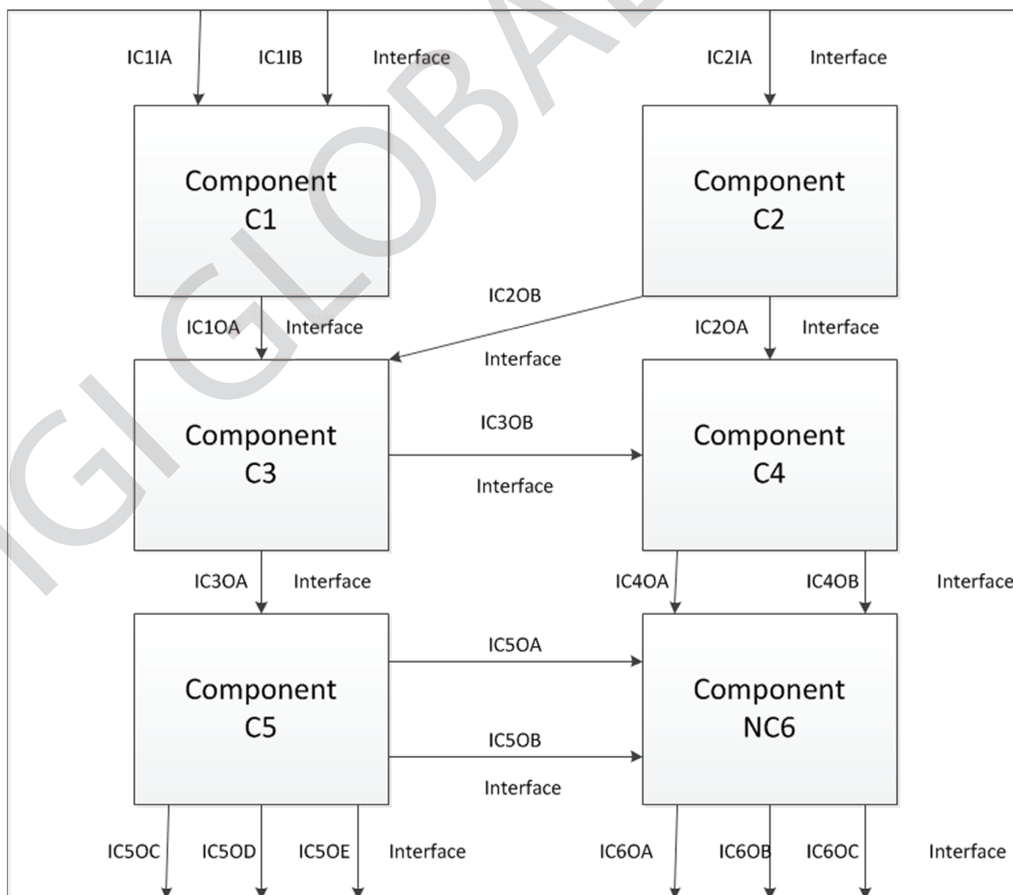
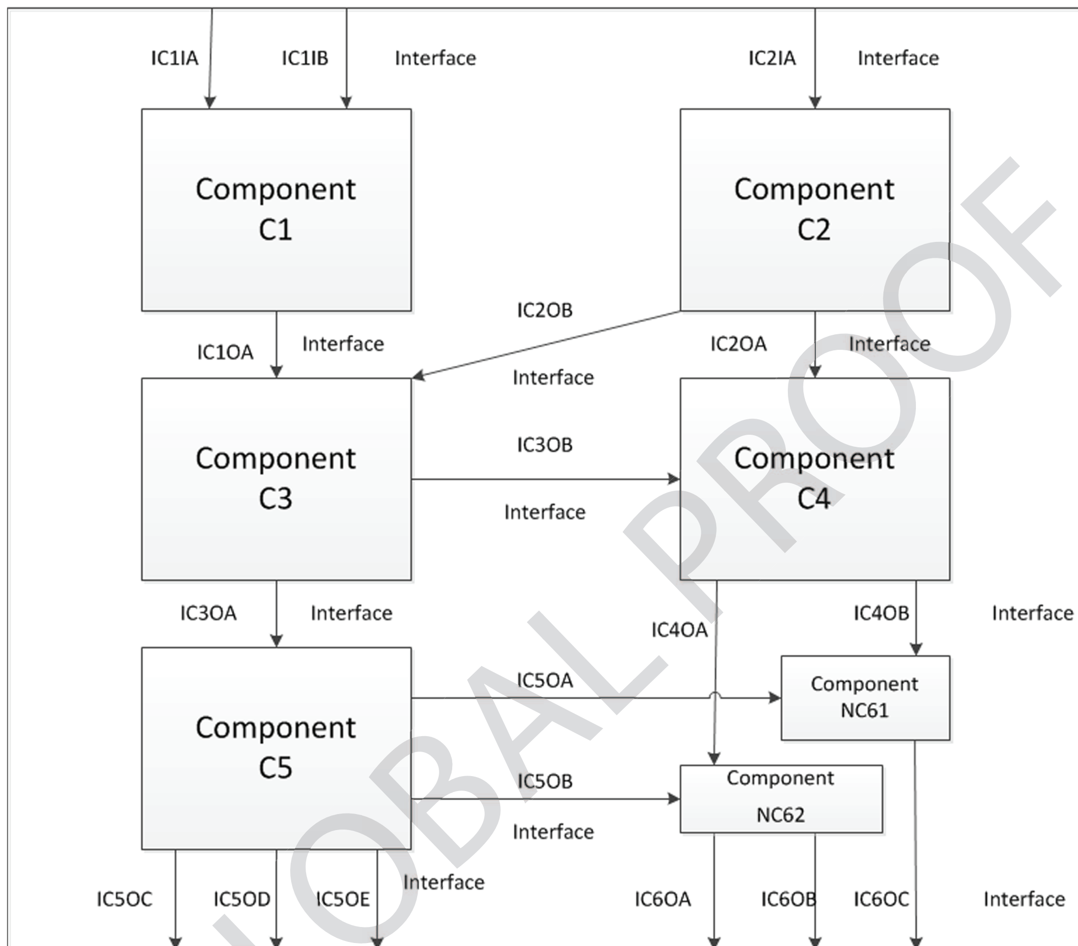


Figure 13. One-to-one component replacement for reengineering



IGI GLOBAL PROOF

Figure 14. One-to-many component replacement for reengineering



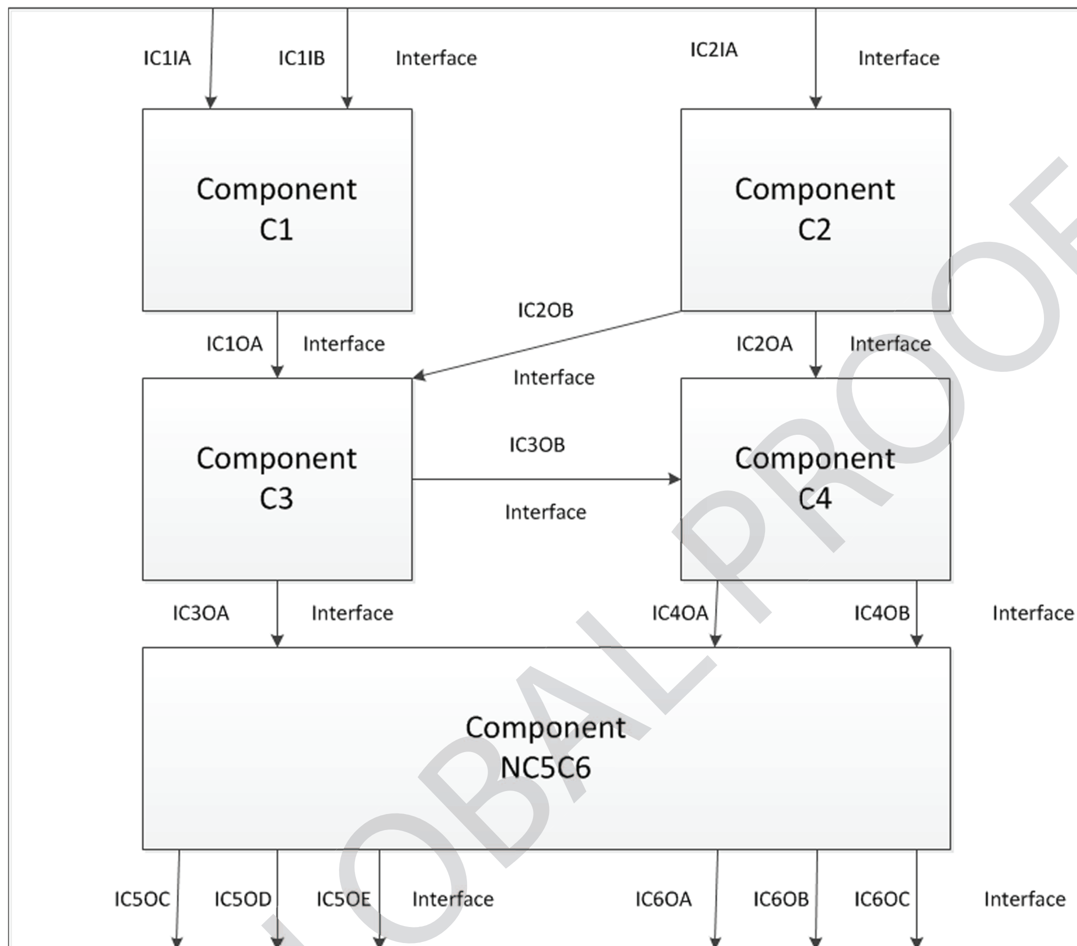
IGI GLOBAL PROOF

When several trivial components collectively can be replaced with a moderate component that provides all services, then it is worthwhile to use the new set of components to replace the existing complex component and reengineer the system to accomplish the same service. In Figure 15, the components C5 and C6 are replaced with a new component NC5C6 while reflecting upon the same interface of both C5 and C6.

When the above strategies are not viable and a group of components that define a service can be collectively replaced with a new modular group of components to accomplish the same service. The resulting reengineered system will look well adapted at a multitude of aspects of the system. In Figure 16, the components C4, C5 and C6 are replaced with a new component NC41, NC42, and NC5NC6 while reflecting upon the same final interface of both C5 and C6 provided by NC5NC6.

A careful reengineering of a system involves a keen evaluation of all above strategies in order to choose the one which is the most economical. While testing an existing component to understand its functionality, or a new component to check the results at the component level, interface simulators can be used which enhance the speed of testing, and any required changes can be quickly performed.

Figure 15. Many-to-one component replacement for reengineering

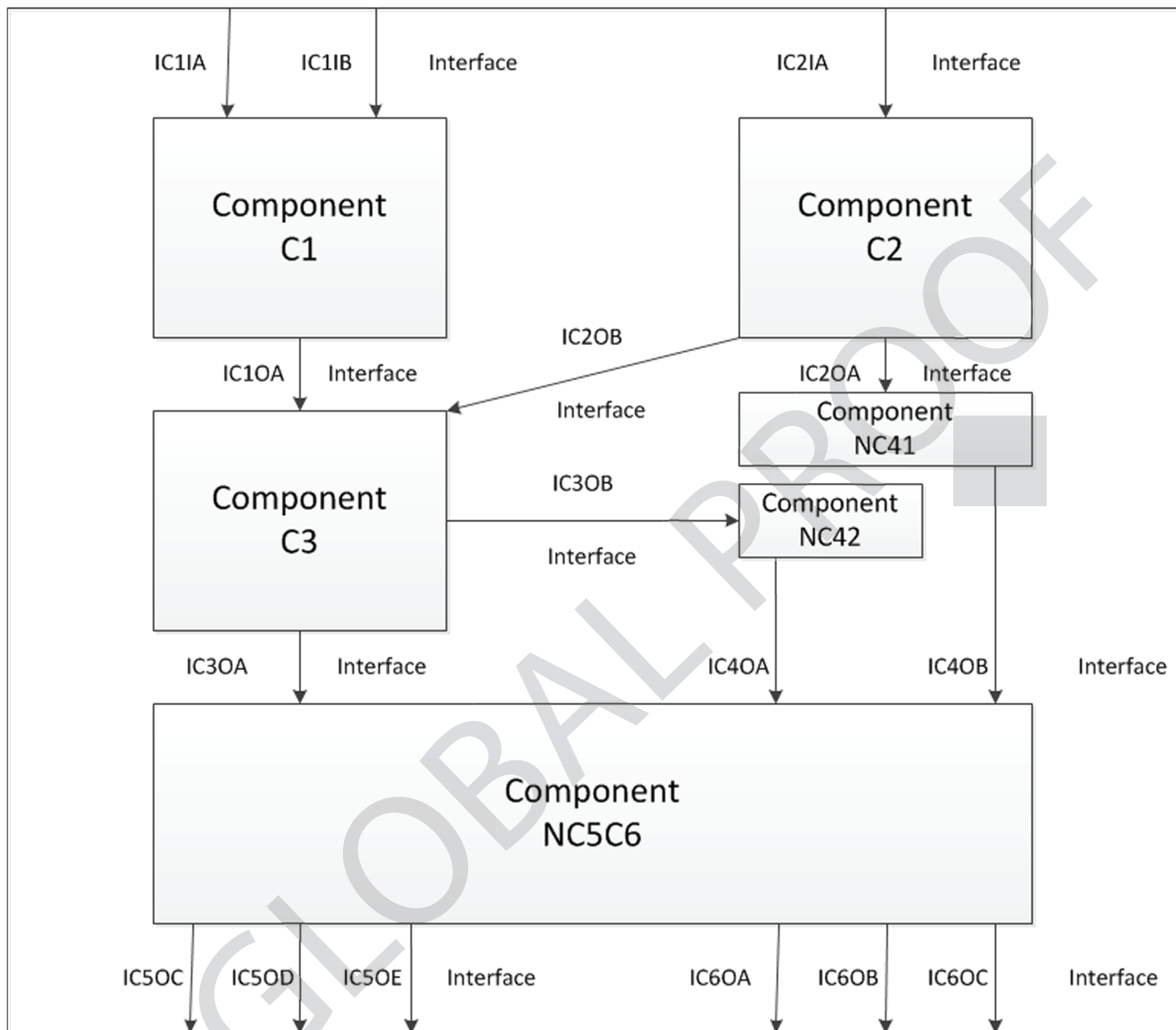


IGI GLOBAL PROOF

4. CONCLUSION

The component based modelling methodology presented in this chapter can be used for both new systems as well as reengineering an existing legacy system. If the existing system is already component-based, it can be replaced with new components that use the latest business rule set; however if the existing system is a legacy system, component-based modelling allows reengineering of such systems via plugging equivalent components and a gradual system reengineering and development could be possible. The contribution of this chapter is in reengineering via component replacement where resizing of components is proposed that balances the complexity among the components. Such reengineered system enhances system maintenance and evolution in an efficient manner. One way to look at the complexity is the size of software component via the amount of processing. When a component is processing intensive, it can be divided into multiple components all of which can exist with in the system, or distributed over a network. The glue code proposed in this chapter allows a client/server architecture in systems reengineering. A legacy system can be reengineered to adopt a new client/server architecture. Further work involves in evaluating component-based modelling to exploit network oriented client/server architecture where

Figure 16. Many-to-many component replacement for reengineering



IGI GLOBAL PROOF

most common components can be centralized at server end whereas the user interface components can be located at client end. Likewise, component-based modelling opens a gateway for numerous applications reengineering.

REFERENCES

Aoumeur, N., Barkaoui, K., & Saake, G. (2007). Incremental specification validation and runtime adaptativity of distributed component information systems. *Proceedings of IEEE Conference on Software Maintenance and Reengineering.* 0-7695-2802-3/07

Ardimento, P., Bruno, G., Caivano, D., & Visaggio, G. (2007). A maintenance oriented framework for software components characterization. *Proceedings of IEEE Conference on Software Maintenance and Reengineering.* 0-7695-2802-3/07

Brown, A. W. (2000). *Large-scale, component-based development*. New Jersey: Prentice Hall.

Cheesman, J., & Daniels, J. (2001). *UML components: Simple process for specifying component-based software*. Addison Wesley.

Chouambe, L., Klatt, B., & Krogmann, K. (2008). Reverse engineering software-models of component-based models. *Proceedings of IEEE Conference on Software Maintenance and Reengineering*, (pp. 93-102). 978-1-4244-2157-2

Peng, S., Wu, Y., & Zhao, W. (2007). A feature oriented adaptive component model for dynamic evolution. *Proceedings of IEEE Conference on Software Maintenance and Reengineering.* 0-7695-2802-3/07

Rumbaugh, J. (1994). Getting started: Using use cases to capture requirements. *Journal of Object-Oriented Programming*, *7*(5), 8–23.

Srivastava, B. (2004). A feature oriented adaptive component model for dynamic evolution. *Proceedings of IEEE Conference on Software Maintenance and Reengineering.* 0-7695-2802-3/07

Whitten, J. L., Bentley, D. L., & Dittman, K. V. (2000). *Systems analysis and design methods*. New York, NY: McGraw-Hill.

KEY TERMS AND DEFINITIONS

Business Type Model: A conceptual map of all data and information of interest for the information system.

Component-Based Model: An interconnected set of components that collectively represent an information system. A component is a modular piece of software that provides a service. Components communicate via interfaces.

Context Model: A high-level view of business context in terms of procedures, roles and responsibilities.

Information System: A computer application system that efficiently processes, stores and relays information within an organization and across its customers, suppliers, and all other public relationship organizations.

Interface Model: A reflection of input or output to a component as a set of attributes, operations and invariants at the interface level.

Legacy System: An application that uses an outdated hardware or software platform.

MVC Framework: Model-View-Control where model is a set of classes reflecting upon a component, View is a GUI interface to the user, and Control is communication control providing the required interface.

Reengineering: Examining and modifying a system that has been in-use for a long time, in order to improve it and make it up-to-date in terms of business rules and system performance needs.

Use Case: A description of steps that reflect upon a usage scenario or a system feature with respect to a system user (or actor).

IGI GLOBAL PROOF