ASPECT-ORIENTED COMPONENT ENGINEERING

GUOLIANG DING

Aspect-Oriented Component Engineering

By: Guoliang Ding

Supervisor: Professor John Grundy

A thesis submitted in the fulfilment of the requirements
for the degree of Master of Science in Computer Science
The University of Auckland

February 2002

Abstract

Software development methodologies have evolved quite fast. Recently, the Component-Based Development technology has become more and more popular for large-scale, complex, well structured, plug-and-play, and potentially reusable systems. A number of component-based development approaches and technologies have emerged. However, most of current component-based development techniques are only focused on component provided services i.e. the low-level functional decompositions of component interfaces. They normally vertically slice system and group components. As a result, the components lack systemic-level and non-functional properties, are difficult to understand and reconfigure, and making too many assumptions about other components to be composed with.

We develop a new Aspect-Oriented Component Engineering (AOCE) methodology to address the above concerns and component crosscutting issues, and to characterize component high-level perspective information by specifying either functional or non-functional aspect services. The aspect services are "horizontal slices" of an overall system; they are factored into component provide/required services that can be used to reason about components.

The concept of component aspects is illustrated using an example system and represented by using UML notation set through requirements and design. Furthermore, we implement the example system by using EJB, which are well-mapped aspect service. Finally, we design and implement component aspects testing agents, and test the example system's components with aspect information.

Acknowledgements

I would like to thank my supervisor, Professor John Grundy for his remarkable advice and guidance during my research throughout the year. Without his enthusiasm, ideas, experience and knowledge, my thesis would not have reached so far. What I've learnt from my supervisor is far beyond this thesis and will benefit my future career.

Secondly, I would like to thank Aaron Waddington and Paul Chilton – directors of TMNetworks Limited. They give me great support and encouragement. Their helps almost extend through my whole research. I really appreciate it.

Lastly, I would like to thank my wife – Liping Wang for all the encouragement and support while I was studying and doing the research. She gives me more than a wife can give. I would like to dedicate this thesis to my wife and my lovely son Simon.

Contents

List of Figures

Chapter 1 Introduction

Modern software systems have become much larger and more complex than a decade ago. Software engineers continually seek new, more efficient, and cost-effective software development paradigms to deal with large-scale software systems. The Component-Based Development (CBD) methodology is the most successful one used to compose and dynamically reconfigure system applications [8]. Using component-based approaches develop software from relatively dependent individual “building blocks” – components by reconfiguring reusable components and assembling them together.

A number of component-based technologies and development methodologies have emerged to help component developers and software engineers. For example, Java Beans, Enterprise JavaBeans™, COM, CORBA, and JViews [3, 27, 28] are component-based technologies, and The Catalysis™, The Select Perspective™ and COMO [2, 12, 13] are examples of component-based development methodologies. However, most of these techniques focus on low-level functional identification and specification, vertically decomposing overall system functionalities into functional service-based groups. These lead to components with a lack of high-level systemic characteristics. They especially lack non-functional, cross-cutting perspective information, such as user interfaces, transaction management, persistency, collaboration and security aspects of the systems and components.

To enrich components with the above aspect properties, we have developed a new component-based software development methodology, and called it Aspect-Oriented Component Engineering (AOCE) [24].

1.1 Motivation and Objectives

When software engineers and component developers develop sophisticated large-scale software systems or well-characterized components, some complex engineering issues arise [22]:

- In order to reuse components in a particular situation, components must be configured properly.
- To be reused in some situations, a component's user interface, middleware capabilities and configuration capabilities have to be appropriately adaptable.
- Both end users and other components sometimes need to access information about component capabilities and reconfigurations at run-time to utilize plug and play.
- To well-structure components, the inter-component relationships have to be reasoned about, and its capabilities have to be identified and described clearly and completely. The services need to be identified and described including the component provides to and requires from other components.
- Engineers need to be very careful making assumptions about related components because those potential reuse situations of components are usually difficult to know.
- Developers don't have source code to access and control, when they intend to reuse 3rd party or commercial off-the-shelf (COTS) components.
- Engineers and developers need to go through the whole software engineering lifecycle to refine component requirements, designs, implementations, testing and deployment. It is better to use a consistent methodology to capture component capabilities, reason about inter-component relationships and between components interaction.

These issues haven't been effectively addressed and supported by most current component-based development methodologies, for example, The Select Perspective™, The Catalysis™ and COMO. These methods pay more attention to component functional services/interfaces decomposition and specifications that vertically group domain specific components. They do not sufficiently characterize high-level systemic perspectives on component behaviour and constraints, and do not fully support reasoning about component capabilities. Therefore, we developed a new Aspect-Oriented Component Engineering (AOCE) technology to overcome these shortcomings.

The concept of component aspects has been utilized in our method to help developers better categorise component properties and capabilities. It does this by analysing functional services and non-functional constraints, especially those of "cross-cutting systemic aspects" service that contribute to or use, i.e. user interfaces, transaction, persistency and security etc. These system-level aspects are "horizontal-slices" of overall system functionalities and non-functional constraints. Aspects of components can further be decomposed into "aspect-details" and detailed "aspect properties" to characterize component services more explicitly at different software engineering levels of abstraction.

The main feature of our AOCE technique is that it not only identifies component provided aspect services for others, but also specifies component-required aspect services for both end users and other components. Furthermore, the provided/required aspect information can be used to reason about requirements, inter-component relationships and component interactions during requirements engineering and system design. At the run-time component aspect information can be used to introspect component's capabilities and support dynamic component reconfiguration. Testing components with aspect services are also important, as during the testing phase, developers can actually verify and validate whether the components they have designed match in aspect services of the requirements.

The Unified Modelling Language (UML) has become a standard for software modelling and development, although it mainly focuses on object-oriented software development and lacks component support [12, 29]. The AOCE technology uses an extended UML and its stereotypes to annotate component aspect services and provide developers more richness of component characteristics.

The one thing the reader needs to be aware of is that aspect-oriented component engineering is different from "aspect-oriented programming", which uses a technique of "code weaving" and run-time reflection to solve systemic crosscutting issues. AOCE successfully factors those systemic crosscutting concerns into component interfaces and carefully documents these issues in requirements, design and implementation to give developers different views of components. AOCE uses component provided/required services to successfully avoid “code weaving”.

The main objectives of this thesis are to illustrate how to apply the AOCE methodology to component-based systems through requirement engineering, design, implementation and testing by using an e-commerce example system. Firstly, in the requirement and design stages, we are trying to use standard CASE tools such as Rational Rose™ to illustrate and document our aspect-oriented component services. Secondly, we use Enterprise JavaBeans™ technology to implement the example system with aspect services and deploy it to a J2EE server. Finally, we designed and implemented testing agents for components with aspects, and tested the example system’s components by automatically extracting their aspect information.

1.2 The Thesis Overview

The following is the structure of the thesis:

- Chapter 1 briefly introduces the new methodology of Aspect-Oriented Component Engineering and objectives of this thesis.
- Chapter 2 introduces and discusses several current component-based software development technologies.
- Chapter 3 introduces the concept of aspect and aspect-oriented technologies used in the software field, mainly focusing on AOP and AOCE, and brief comparisons of them.
- Chapter 4 illustrates an e-commerce systems component-based requirements and design.
- Chapter 5 uses the example system to demonstrate aspect-oriented analysis for requirement and component design by using AOCE.
- Chapter 6 illustrates how systemic aspect services are annotated in the standard UML based CASE tool using Rational Rose™ as an example.
- Chapter 7 maps components with aspects into EJB frameworks and illustrates the implementation of the example system.

- Chapter 8 describes the design and implementation of component aspects testing agents and shows testing results analysis.
- Chapter 9 summarizes the contributions of the thesis and describes future work in this AOCE research field.

Chapter 2 Component-Based Development and EJB Architecture

Components are well established in all other engineering disciplines, and in recent years they have been successfully applied in the world of software. Modern software systems have become larger, more complex and harder to control, resulting in high development cost, low productivity, unmanageable software quality and high risk to move to new technology [7]. Consequently, there is an increasing demand for a new efficient and cost-effective software development paradigm. One of most successful solutions today is the component-based software development approach [8]. This approach is based on the idea that software system can be developed by selecting appropriate commercial off-the-shelf (COTS) components and assembling them with a well-designed software architecture [9]. Therefore, several component-based development methodologies have been proposed in recent years. This chapter will briefly discuss three Component-Based Development (CBD) approaches and their weakness.

2.1 Component-Based Software Development Methodology

Component-based software development has emerged to increase the reusability and portability of software pieces. Component-based development aims at constructing software products by assembling components. Generally components have three main features: 1) a component is an independent and replaceable part of a system that fulfills a clear function; 2) a component works in the context of a well-defined architecture; 3) a component communicates with other components through its interfaces [11]. Recently, other component-based technologies have emerged such as EJB, COM+, and .Net etc. and also new approaches to component-based development have been proposed. Here, we'll mainly focus on three of them; they are The Catalysis™ approach, The Select Perspective™, and COMO approach.

2.1.1 The Catalysis Approach™

Catalysis is based on, and has helped shape, standards in the object modelling and component world. It has also borrowed many ideas from other technologies, and reorganized the ways in which software is produced. The techniques and methodology of Catalysis provide the following [12]:

- For Component-based development: How to precisely define interfaces independent of implementation, how to construct the component kit architecture and component connectors, and how to ensure that a component conforms to connectors.
- For high-integrity design: Precise abstract specifications and unambiguous trace ability from business goals to program code.
- For reengineering: Techniques for understanding exist software and designing new software from it.

The Catalysis approach utilizes Unified Modelling Language (UML) to model software systems. It includes static models (object attributes and invariants modelling, behaviour models), object type and operation modelling, and interaction models (use case, actions, and collaborations modelling). Furthermore, it also includes composition models and specifications, frameworks, and template packages modelling. These models are presented a number of features in the Catalysis approach [12]:

- Actions are described in terms of their effects on objects. They can be defined with post conditions and illustrated with snapshots.
- Abstract specifications can be made very precise, avoiding ambiguities.
- Actions and objects can be abstracted and refined – that is, described at different levels of detail. The relationship can be traced, or retrieved, all the way from business goals to program code.
- Collaboration-schemes of interaction; these are first-class units of design.
- Component and objects are designed similarly, although with different emphasis on the way they are chosen and have responsibilities assigned.
- Components with different views and representations of a business concept can be related to the common business model with retrievals.

- Components can be designed to plug in to each other and in to frameworks. The plug-in-points are defined with action specifications.

Using the Catalysis CBD approach, we find that the approach is mainly focused on component lower-level functional decomposition and component functional interfaces. Catalysis models basic actions and collaborations, and further vertically slices system to model components. It's a very good CBD technology, however, it still lacks some of the systemic high-level component characteristics such as user interfaces, transaction management, persistency and security etc. aspect information.

2.1.2 The Select Perspective™ Approach

The Select Perspective™ is a component-based approach for developing enterprise systems. An enterprise system is one that meets the needs of a large business with complex business processes. A fundamental characteristic of such systems is that they need to be delivered rapidly in tight time frame, while facilitating reuse though component technology. The Select Perspective technology incorporates a component-based architecture and includes the latest developments in UML. The technology architecture is aimed at harnessing a service-based approach with effective object-oriented modelling in order to capitalize on the increasing power of the fast-developing technology [2].

There are six core-modelling techniques based on UML in the Select Perspective approach, and UML notation has been streamlined and minimally enhanced to meet the practical needs of enterprise systems. The six models of The Select Perspective are: Business Process Model, Use Case Model, Class Model, Object Interaction Model, State Model, Component Model, Deployment Model and Logical Data Model [2]. In these six, the Business Process Modelling (BPM) and Logical Data Modelling are not found in the general software modelling techniques. Business process modelling is a notation adapted from the Computer Science Corporation. The main idea used here is modelling business services and architecture. Logical data modelling employed here is used for wrapping relational (or other non-OO) databases to store data; obviously a key area in legacy database wrapping [2]. Obviously, the component model is largely geared to component development. However, most of the modelling techniques are applicable to both the development of specific solutions and to development of reusable components. The Figure 2-1 shows The Select Perspective approach used in modelling diagram.

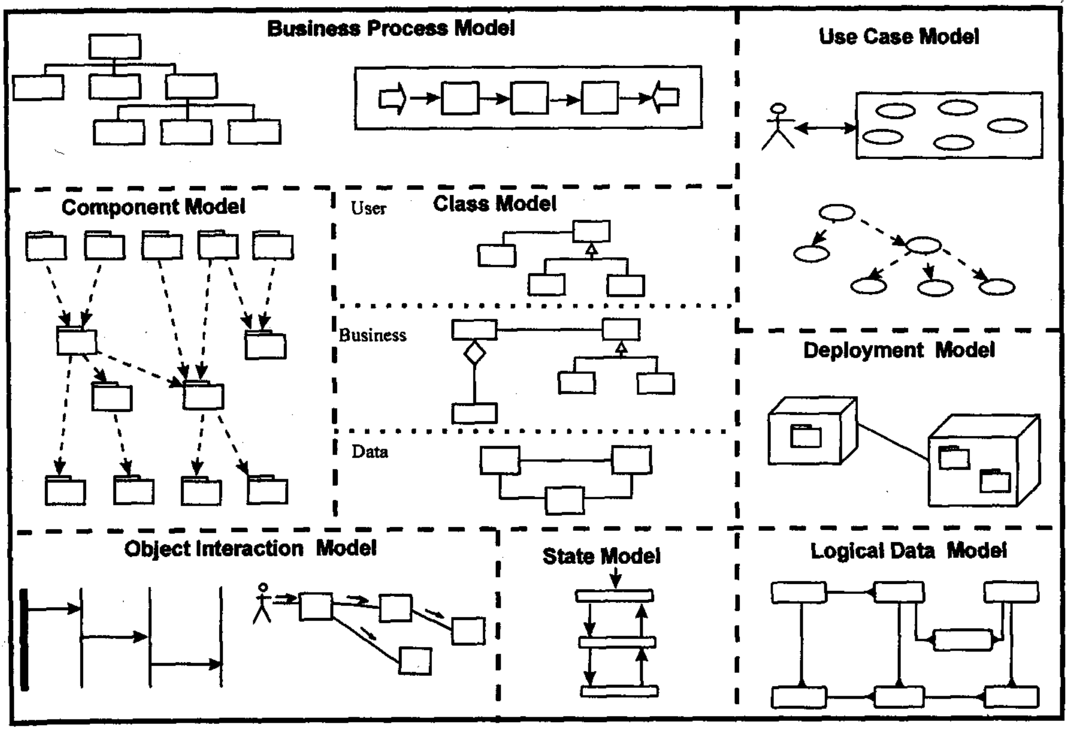


Figure 2-1 The Select Perspective models diagram [2]

As can be seen, The Select Perspective™ methodology mainly focuses on services or functional properties of a system. Its various modelling techniques are only capturing low-level systemic characteristics and services. Although, component modelling is based on system architecture and aimed at developing components that provide commonly used business and/or data services, it still just focuses on functional perspectives. Component users can't find higher-level systemic aspect information from component services. For example: systemic distribution, persistency, security or collaboration perspective services of components. This resulted in end-user comprehension difficulties, and hard to use or reconfigure components.

2.1.3 COMO Approach

COMO is a practical object-oriented component development methodology that can be used in developing software components. COMO component modelling technique only focuses on component development. It proposes modelling workflow between tasks and defines concrete modelling guidelines. COMO extended the current UML notations to model component through adding message flows and classes into component diagrams [13]. The message flows between components are mapped into interfaces of component diagram. Moreover, the COMO approach also considered reliability of component modelling. When identifying domain requirement set, the COMO technique only focuses on functional requirements from many application requirement specifications. When identifying components, COMO utilized two clustering techniques: use case clustering, and use case and class clustering technique [13].

The COMO methodology for component modelling only focuses on component lower level functional decompositions. There are no non-functional constraints addressed. As mentioned above, the COMO approach identifying components by using use case based clustering techniques, which is a vertical slice systems technology. System crosscutting issues are not identified at all such as distribution, transaction and persistency concerns. Although it classified component interfaces into provided interface and required interface for each component, it still focuses on component functional services, did not present any systemic high-level properties of components and non-functional issues. COMO like most component-based development technologies did not address non-functional constraints or crosscutting issues such as user interfaces, collaboration and security perspectives.

2.2 EJB Framework Architecture

Enterprise JavaBeans (EJB) is not a product. It is a specification for a Java server-side services framework [15]. It is part of a larger framework of the Java 2 Platform, Enterprise Edition (J2EE) [5]. This platform is architecture for developing, deploying and executing applications in a distributed environment. EJB server provides system-level services, such as transaction management, security, and database access (persistency) services. The benefit to application developers is that they can focus on writing the business logic necessary to support their application without having worry about implementing the surrounding framework.

The EJB server is the high-level process or application that manages EJB containers, along with providing access to system services. An EJB server is required to provide for availability of JNDI-accessible naming service and a transaction service [14]. The Figure 2-2 shows the EJB three tiers architecture.

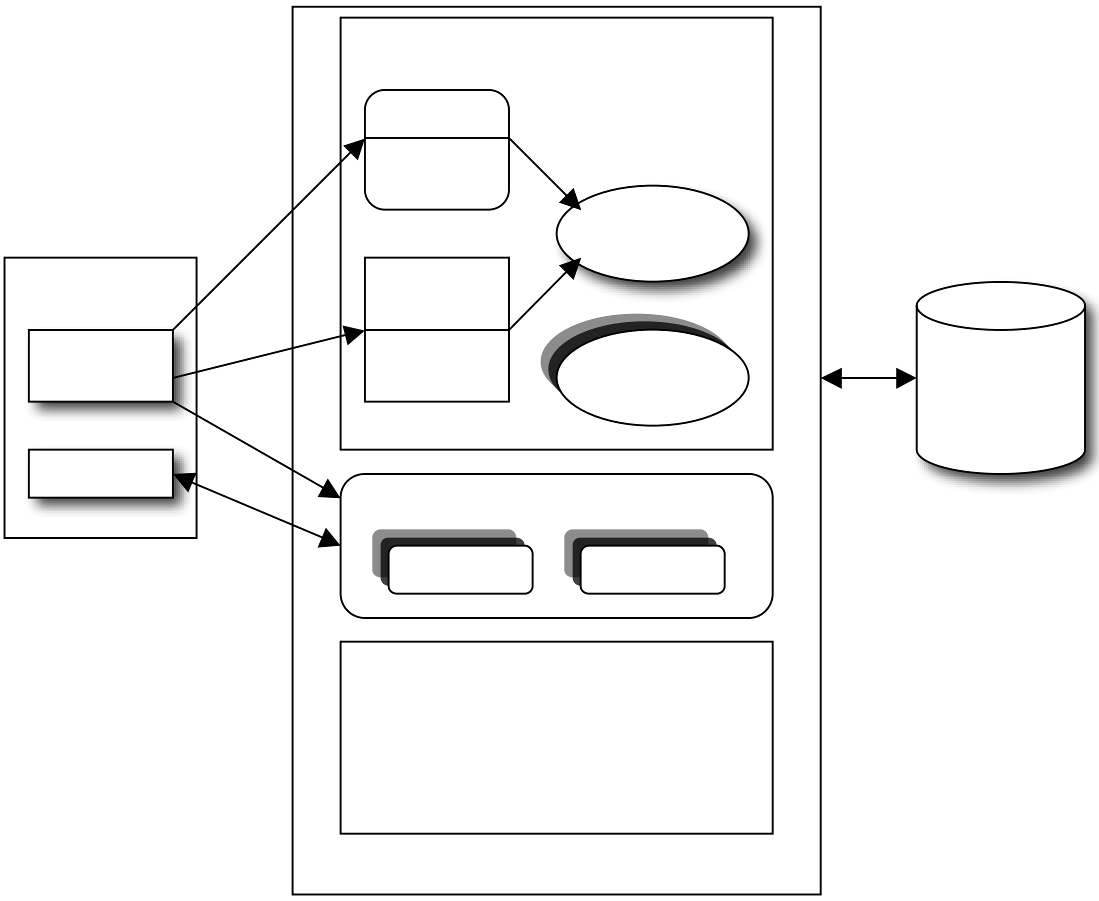


Figure 2-2 EJB three tier Architecture

The EJB container is an abstraction that manages one or more EJB classes and/or instances. The container is not visible to the client or to the contained bean. However, all method invocation made on the bean are intercepted by the container, allowing the container to provide various services to the bean transparently.

A web container manages the execution of all JSP pages and servlet components for EJB application. Web components and their container run on the EJB server.

Enterprise bean instances run within an EJB container. The container is runtime environment that controls the enterprise beans and provides them with important system-level services. The Container provides the following high-level services to enterprise beans: [5]

- Transaction management
- Security
- Remote Client Connectivity
- Life Cycle Management
- Database Connection Pooling (Persistence service)

Transaction Management

When a method of an enterprise bean has been invoked, the container intervenes in order to manage the transaction. Component developers do not have to code transaction boundaries in the enterprise bean, due to the container manages the transaction. The program code required to control distributed transaction can be very complicated. The component developers could just simply declare the enterprise bean's transactional properties in the deployment descriptor file, rather than coding and debugging complex code. The EJB container will load the file and handles the enterprise beans transactions.

Security

In the enterprise bean's deployment descriptor, developers could declare some specified roles and methods that may invoke by clients. Then, the container will permit only authorized clients who belong to a particular role to invoke an enterprise bean's method. That means each client belongs to a particular role, and each role is permitted to invoke certain methods. Therefore, developers do not necessary to code routines that enforce security, due to this declarative approach.

Database Connection Pooling (Persistence service)

Setting up a database connection is time-consuming and the number of connections may be limited. A database connection is a costly resource. To avoid these problems, the EJB container manages a pool of database connections. An enterprise beans can quickly obtain a connection from the pool. When the bean releases the connection, it may be reused by another bean. It increases the persistency services performance.

In general, EJB is a network middleware and application server that provides special functionalities such as transaction, persistence, location transparency, and security services [16]. These services can be considered as non-functional aspects and EJB can be regarded as an Aspect-Oriented Component Engineering (AOCE) ideal programming environment. The EJB technology successfully reduced domain specific component developer additional works for struggling on those non-functional aspects. Therefore, most of component aspects could be mapped to EJB provided aspect services. However, some of other aspects that EJB did not provided still need developers to implement in their code.

Both Sun's J2EE and Microsoft .NET framework include a complex middle tier infrastructure that supports resources sharing among clients. This infrastructure is important for supporting great numbers of clients, and thereby achieving high system performance. In J2EE, this middle tier infrastructure is called Enterprise JavaBeans (EJB) container. In the .NET framework, it is called COM+ [30]. However, the .Net/COM+ technology there is limitations on language and platforms. It can only be run on Microsoft Windows™ platform. And, the EJB server provided more system-level services such as transaction, persistency and security etc.

2.3 Summary

Component-based software development ideas were introduced into software engineering in recent several years. Because modern software systems have become larger and larger, software engineering field emerge crisis for developing large-scale software. The CBD technology increases software reusability, scalability and maintainability, and reduces the software developing cost and time.

Some component-based development techniques were proposed such as The Catalysis, The Select Perspective and COMO. However, most of these CBD approaches only focus on functional decomposition, they vertically slice system functionality to model and group components. Therefore the components developed by applying these methodologies are lack of non-functional properties and characteristics such as high-level systematic transaction, persistency and security aspects etc.

Enterprise JavaBeans is a popular component framework middleware and application server. The EJB technology can provide some system-level services, such as transaction management, security and persistency. These systemic services benefit component developers, that they can concentrate on business logic rather than system-level services. It reduces component development time and cost. Moreover, these services can be considered as aspects of services, and EJB can be regarded as an aspect-oriented component development environment.

Chapter 3 AOP and Aspect-Oriented Component Engineering

The Aspect concept was introduced into software developing fields to address the problems that not procedural and object-oriented programming technologies can sufficiently address. There are a number of techniques came up around aspect-oriented notation in software developing and programming. The most successful one is Aspect-Oriented Programming (AOP) [17] technique. The others are Adaptive Programming (AP), Programming with Aspectual Components [18] and our Aspect-Oriented Component Engineering (AOCE). This chapter will give a briefly introduction of these methodologies and mainly focus on our Aspect-Oriented Component Engineering methodology.

3.1 Aspect-Oriented programming

The definition of "aspect" has evolved over time. It is difficult to make terms like cross-cutting, tangling, intermingling, interleaving and cross-cutting precise. The current working definition is (May 99, Gregor Kiczales) [19]:

An aspect is a modular unit that cross-cuts the structure of other modular units.

Aspects exist in both design and implementation. A design aspect is a modular unit of the design that cross-cuts the structure of other parts of the design. A program or code aspect is a modular unit of the program that cross-cuts other modular units of the program.

In summary, an aspect is a unit that encapsulates state, behavior, and behavior enhancements in other units.

Aspect-Oriented Programming was named by Gregor Kiczales and his group [17]. The main purpose of AOP is to address some issues that neither procedural nor object-oriented programming techniques are sufficient to clearly capture in software design and implementation decisions. These issues usually crosscut a system's basic functionality and spread out in the implementation code. Moreover, during implementation they result tangling code that is very difficult to develop and maintain.

The Xerox PARC aspect-oriented programming definition distinguishes components and aspects. A component is considered a behavioral view. The Xerox PARC definition of AOP stresses the importance of "crosscutting". However, the Xerox PARC definition of AOP has evolved over time. For a while, AOP was meant to be only for systemic aspects like synchronization, distribution, failure handling, etc. Crosscutting collaboration issues were not considered as aspects for some time [20].

The aspect-oriented programming methodology initially is intended to clearly express and solve crosscutting aspect issues, including appropriate isolation, composition and reuse of the aspect code [17]. Code weaving is the main approach of AOP to solve these issues. Aspect weavers must be able to handle component and aspect languages, composing them properly to result the desired total system functionality [17]. The current aspect-oriented programming technology is mainly focus on controls tangling of concerns by isolating aspects that cross-cut each other into "building blocks" [18].

AOP clearly separated components and aspects from each other. Gregor Kiczales and his group defined component and aspect like following in [17]:

A component, if it can be cleanly encapsulated in a generalized procedure. Components tend to be units of the system's functional decomposition. For example, in an e-commerce system customer and shopping cart are functional decomposed components.

An aspect, if it can not be cleanly encapsulated in a generalized procedure. Aspects tend not to be units of the system's functional decomposition, but rather to be properties that affect the performance or semantics of the component in system ways. For example, object distribution, persistency and collaborations are system aspects.

AOP use these two terms to support the programmer separating components and aspects from each other clearly, by providing mechanisms that make it possible to abstract and compose them to produce the overall system [17]. For example, there are several aspects of concern for a digital library system – include communication, coordination and failure handling. AOP uses aspect language, component language and aspect weaver to solve these cross-cutting issues [17].

3.1 Adaptive Programming

Karl J. Lieberherr introduced adaptive Programming (AP) as a name around 1991. In adaptive programming, programs are decomposed into several crosscutting building blocks. Initially AP separated out object representation as a separate building block. Then it added structure-shy behavior and class structure as crosscutting building blocks [19]. Actually, the adaptive programming is an early instance of aspect-oriented programming.

AP book [21] stresses the importance of the coexistence of multiple organizations - views. Those views are composed by referring to each other, but the references may be to internal parts of other views leading to crosscutting. The referenced internal parts may be spread over the entire referenced view. Aspect-oriented programs consist of complementary, collaborating views, each one addressing a different concern of the application [21]. But *Karl* used a different terminology to describe AOP.

Adaptive programming methodology described components and aspects as views, not clearly separate each other. The code-weaving notation was used to describe enhancements of views. The goal is to separate views by minimizing dependencies between views so that a large class of modifications in one view has a minimum impact on the other views [21].

Recently, the Programming with Aspectual Components [18] technique was proposed by *Karl* and his group. They try to integrate AOP and component-based programming by introducing component construct for programming class collaborations, and call it aspectual component. The aspectual components extend adaptive plug-and-play

Iorizontal Slices" . Aspects, rspectives Process Stage Overall Software application i.e. objects, components Event History Process View related services Security related services

components (AP&P) with a modification interface that turns them into an effective tool for AOP [18]. A key point of aspectual components is that they are programmed in a generic data model, which is used to ensure the proper development of components. Other techniques use aspect notation is AspectJ [31] that is an aspect-oriented extension to the Java™ programming language.

3.2 Aspect-Oriented Component Engineering

Aspect-Oriented Component Engineering (AOCE) is not a programming technology, unlike Aspect-Oriented Programming and Programming with Aspectual Component. AOCE is a new software engineering methodology for developing more reusable, extensible and dynamically adaptable software components. The technique focuses on the whole software developing lifecycle, like requirement engineering, design, implementation, testing and deployment. We use the notation of "aspects" to horizontally slice overall system through vertically decomposed software components, to characterize crosscutting functional and non-functional properties of components. Figure 3-1 shows the general concepts of components vs. component aspects [22].

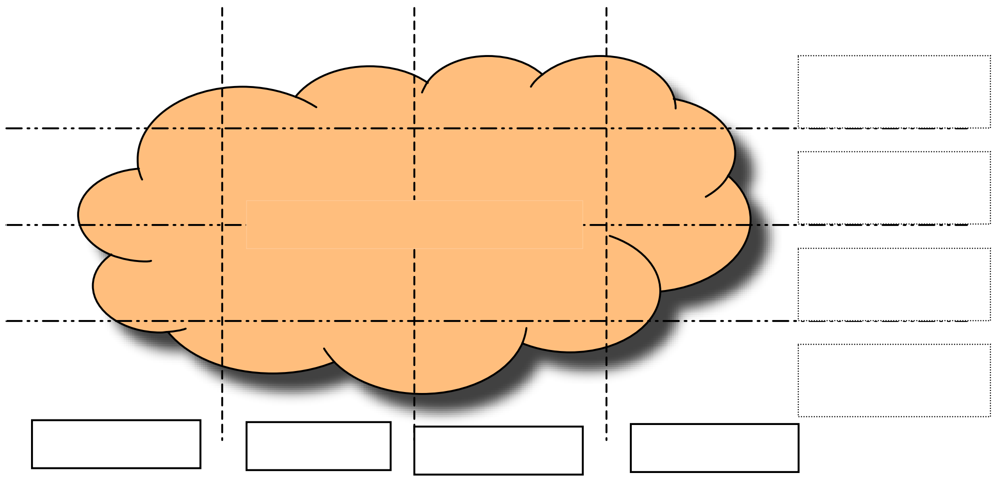


Figure 3-1 General concept of components vs. component aspects.

Common systemic aspects include user interfaces, transaction processing, persistency, security, distribution, memory management, collaboration and so on. These aspects may vary with different domain, for example, for real-time control system event response time (performance), memory management and concurrency aspects maybe important; security-critical systems have various additional security-related aspects; safety-critical systems have redundancy and high assurance aspects [22].

Normally, each component in a specified system will provide one or more aspect-oriented functional or non-functional services for other components/end user to use. Meanwhile, the component may require one or more aspect-oriented functional/non-functional services from other components in order to well functional working. When component developer identify aspect-related services, the AOCE methodology will allow user to reason about components interaction in various systemic aspects and crosscut the system vertically slicing into software components. Component developers have to address the important system level crosscutting concerns of their application components will have, identify proper “aspect details” and detailed “aspect properties” so that they can be used to exchange component higher-level information or component properties retrieval and introspection.

Each aspect of a component provided to or required from others has some “aspect details”, which is used to more precisely describe systemic properties of component related to the aspect. For example, one component may provide data retrieving and storage services of persistency aspect; one component may require data encoding or encrypting services of security aspect. These are aspect-details, which may also have one or more aspect detail properties that used to further characterize aspect information [23]. The aspect detail properties will relate to functional and/or non-functional characteristics of the aspect detail. Fore example, for security aspect we might be interested data encoding algorithms, encryption algorithm, encryption key length and key type etc detail properties.

The notion of an aspect in AOCE not only used to capture functional information of components but also used to capture non-functional constraints of components. For example, component developer may like to describe persistency functional aspect of retrieving and storage data, meanwhile, they also like to describe non-functional aspect of persistency such as performance of retrieving data or maximum data size can be retrieved and required LAN/WAN bandwidth for a good performance etc. The aspects of AOCE focus on enriching developers and end users knowledge about components by applying a more effective characterization and implementation mechanism for component crosscutting services.

The AOCE methodology has been applied at component and system requirement engineering, component design, implementation and run-time deployment levels [24, 25, 26]. The following section described how AOCE has been used in these software engineering stages.

3.2.1 Aspect-oriented Component Requirements Engineering

The Aspect-Oriented Component Requirements Engineering (AOCRE) is a subset of our aspect-oriented component engineering (AOCE). The AOCRE aims to identify and specify the functional and non-functional requirements, which related to important aspects of a system. These key aspects usually are the system components provided or required crosscutting systemic services. For example, a component or application developer may want to identify user interface, transaction, persistency and security related functional and non-functional aspects of a component, and may also want to document these aspect services of the component provided and required.

During component requirements specification, developers may find that some components have many aspects services and others may just few aspects. Moreover, some components may share aspect services, not only the provided aspects are important to characterize component, but also the required aspects. Sometimes more than one other component may provide or require a component's aspects that are overlapped aspects. These overlapped aspects are the reflection of a component high-level systemic characterization; they are also very helpful for requirements engineers to understand related components properties.

The AOCRE process started when general system application requirements or individual or groups of components requirements analyzed. The process scenario will refine the requirements iteratively top-down and bottom-up. The basic process flow shows in Figure 3-2 [24].

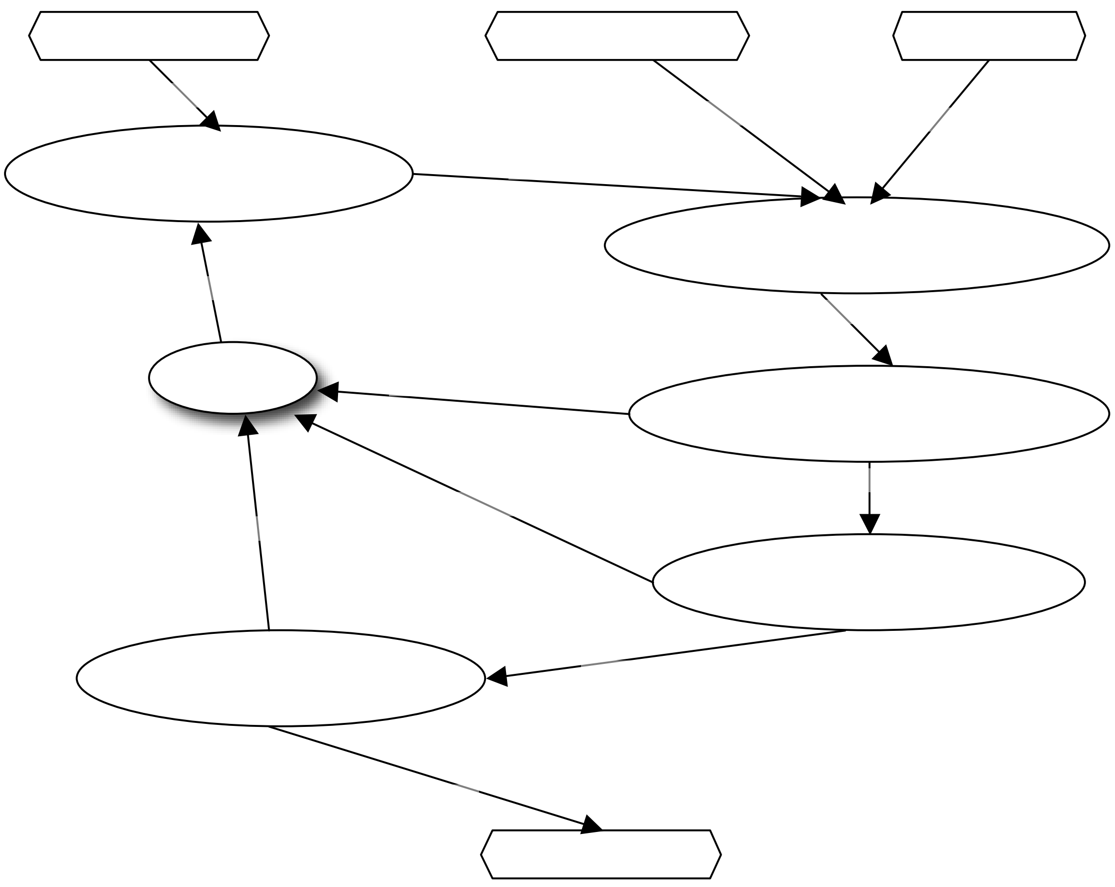


Figure 3-2 Basic AOCRE process

3.2.2 Software Component Design with Aspects

During requirement specification, developers have identified a set of functional and non-functional constraints from systemic aspects. These constraints/aspects can be used to refine into component design with aspects. Requirements-level aspect details can be refined into software detailed design-level component aspects, which characterize design-level component aspect services. For example, the detailed design-level aspect specifications are user interfaces, component persistency and distribution, security and transaction models and component configuration.

In the design stage, developers will refine design-level aspects from implementation-neutral requirements into aspect details and detailed aspect properties that specify services relevant to selected implementation strategies of software components.

Requirements-level aspects can be split, merged or refined to define aspect detail types and aspect detail properties more precisely, and associating these aspect details to component implementation design technology. At design level component services will be documented with related aspects, and may also documented aspects that overlapped with others in order to allow designer to track these perspective links [22]. Chapter 5 illustrated systemic aspect services refined into component design.

3.2.3 Component Implementation and Run-time with Aspects

Using AOCE designed components can be implemented by using any component-based implementation technology or framework, such as Enterprise JavaBeans™ and JViews [26] component-based framework. As discussed in chapter 2 of section 2.2, Enterprise JavaBeans™ provided system-level services that can map onto aspect characterizations reasonably well.

Component aspects can be implemented via interfaces, language reflection or design patterns. A component implemented with aspects can provide a mechanism for related components to access each other's functionality and to guide inter-component interface definition. Therefore, the component may invoke other components functionality indirectly via operation provided by aspect implementation or may invoke component function directly [24].

Aspect services implemented in a component can be used at run-time by other components or end-users. They can be used to query the component aspects it provided or required, meanwhile the component could also query other components aspect information to perform consistency checks for a configuration and validation of a component. They can also be used to introspect a component's capabilities at run-time.

3.2.4 Component Testing with Aspects

Aspect-oriented component engineering has been applied at component requirements and specification, design, implementation and run-time deployment levels. However, after components that implemented with aspect services deployed or plugged into other applications, we need to verify how well the component's aspects worked and how well they actually met the specification and design of components. Therefore we need to generate testing plans and oracles from aspects. The test plan we generated is that by defining component aspects descriptor and use our special testing agent to test component with aspect services. The detailed information of component aspects testing agent please refer to chapter 8.

3.3 Summary

Aspect-oriented programming and adaptive programming are very successful and popular approaches for handling cross-cutting issues of object-based systems. Programming with aspectual components focus on low-level generic data model. AOP use a technique of "code weaving" to solve systemic crosscutting aspects and avoid tangling code. However, our AOCE technology avoided the concept of "code weaving" and use of run-time reflection mechanism.

We use the concept of components providing services for one or more systemic aspects and requiring one or more aspect services from other components. The AOCE focus on factoring components' crosscutting systemic issues into component interfaces so that components can be run-time reconfigured, dynamically composed and reasoned about. The main advantages of AOCE are: it makes components provide richness of multiple perspectives (aspects), structures component requirements and design very well, and encourages implementing better dynamic configuration and decoupled components interaction [22].

AOCE allows developers document component more precisely and completely. Component with aspects gives developer and end user different view of the component capabilities. Aspect information of component can also aid developers better indexing and storage components, it also gives developer and end user higher-level systemic characteristics, which provide end-user better understanding for components.

Chapter 4 The E-Furniture System Component-Based Design

Components and Component-Based Development (CBD) methodologies are a very important force in the e-business revolution and large-scale software systems. They are the way to reach enterprise solutions in the Internet age. In recent reports by 2003 most new software intensive solutions will be constructed with "building blocks" such as pre-built components [1]. It is clear that over the next few years, enterprise-scale solutions in the Internet age will mainly focus on components and component-based approaches [1]. This chapter will briefly illustrate component-based system design by applying The Select Perspective™ approach [2] using an example system of furniture online selling and backend enterprise maintenance system. Of course it is impossible to illustrate the entire process, here we just illustrate main stages of the approach. These phases are: system requirements (business modelling), system analysis (use case, object interaction modelling) and system design (architecture modelling).

4.1 The System Requirements

Wanli Furniture Company is one of the biggest furniture manufacturers in China. They specialise in solid-wood furniture. Current trends of Internet Business are to use computer software systems to manage products inventory efficiently. Therefore, ***Wanli*** decide to research their products selling online and the management integrated software system with a view of more scalability, usability and flexibility. They also thinking about large scale integrated systems for later development, such as raw material supplier electronic data exchange system and retailer data exchanges via the Internet (the Virtual Private Network (VPN) system). In the first step, they would like to prototype an online shopping cart and back-end database maintaining system, which we call the E-Furniture system. The following sections illustrate the system's requirements and system modelling.

4.1.1 Functional Requirement Specifications:

Web User requirements:

1. Users via the WWW Internet anywhere in the world can access the system.
2. The system may use different languages to provide friendly user interfaces.
3. The system should allow users to view all the products that the company offers and their details.
4. The system should also allow users to search any products that the company offers.
5. The system should allow users to sign up as a member of the company. Then if the customer wants to place an order later, they can login first so they don't need to fill in the detailed information about themselves for each order.
6. The system should allow users to maintain their own information.
7. The system should allow users to pick up more than one item when shopping.
8. The system should also allow users to put back items they don't want to buy when they are shopping.
9. The system should allow users to place and cancel orders (within some period) and view their order list and order deliver information.
10. The system should allow users to submit messages using the company web site when they are looking for products or doing online shopping as to get feedback information in order to improve services.
11. The system should allow users to check out when they have finish shopping and make online payments using credit card.
12. The system must notify users via email (if the user has an email address) when their orders have been delivered or if they could not be delivered on time.

Maintain requirements:

13. The system should only allow specified staff or management, access to the system database.
14. The system should permit selected staff to view and add/delete/update products information.
15. The system should allow selected staff to view and maintain product stock information.
16. The system should permit some staff/management to view customer information.
17. The system should allow input of single sale details into database for when customers come to the company to buy products.
18. The system must allow selected staff to view/maintain order information that customers have placed.
19. The system must keep information of customer-returned products.
20. The system should also allow staff to view the messages that customers have submitted using the web site.
21. The system should also allow staff to maintain order delivery information.
22. The system should also allow managers to view/maintain staff information.

4.1.2 Non-functional requirements:

Security and Integrity Aspects:

1. Access to the system is via an authentication module. All packets involving authentication details need to be encrypted.

2. The web server should be secured very carefully covering any known and popular security holes and exploits.

3. The local unit running the web site is responsible for the security, and care should be taken to ensure that it doesn't become a potential point of entry into the entire system.

4. Only managers can access and change staff information details.
5. Passwords are needed to access all staff functions.

6. Customer requests (especially for payment information) should be encrypted.
7. Backups done every night.

Quality Aspects:

8. Reporting is not critical and can be off line for up to 24 hours.
9. Selling is critical – downtime must be less than 10 minutes.
10. On-line access is not very critical, but downtime should be limited to 1 hour.
11. All programs should generate error logs.
12. In case of delays or error, the system should indicate this by some sort of visual display.
13. Power failure needs to be handled by UPS for the main server machine and database.

Performance Aspects:

14. Response time should be less than 5 seconds under heavy loading (up to 100 simultaneous customer connections allowed), for retail search and on-line search (regardless WAN transmission delay)

These functional and non-functional requirements give developers a clear description of the system to build. The non-functional constraints are also important for the system; they describe the system from higher-level perspectives.

4.2 The System Object-Oriented Analysis - OOA

In the system analysis phase, Select Perspective approach will mainly focus on use case modelling, class modelling, and object interaction modelling. These models will simulate a system from different perspectives giving developers a very good understanding of a system's functional properties.

4.2.1 E-Furniture system use case modelling

Use case modelling has become a very popular object-oriented analysis technique. The “non-technical” nature of use case allows users to participate in a way that is seldom possible using the abstraction of object modelling alone. It also helps the analyst get to grips with specific user needs before analysing the internal mechanics of a system. They also provide a means of trace ability for functional requirements upstream in the process, and for constructing test plans downstream in the process.

Figure 4-1 is the system’s main use case view. It shows the web user (can be either individual or organizations) who can register as a member in order to easily shopping on the web site later. When a user is registered as a member of the furniture store, he/she can directly login to place orders or maintain their personal information. However, any user visiting the site can view the product’s catalogue to search products and view detailed information, whilst being able to purchase some products. Any user would also be able to submit a message to the furniture store using web site.

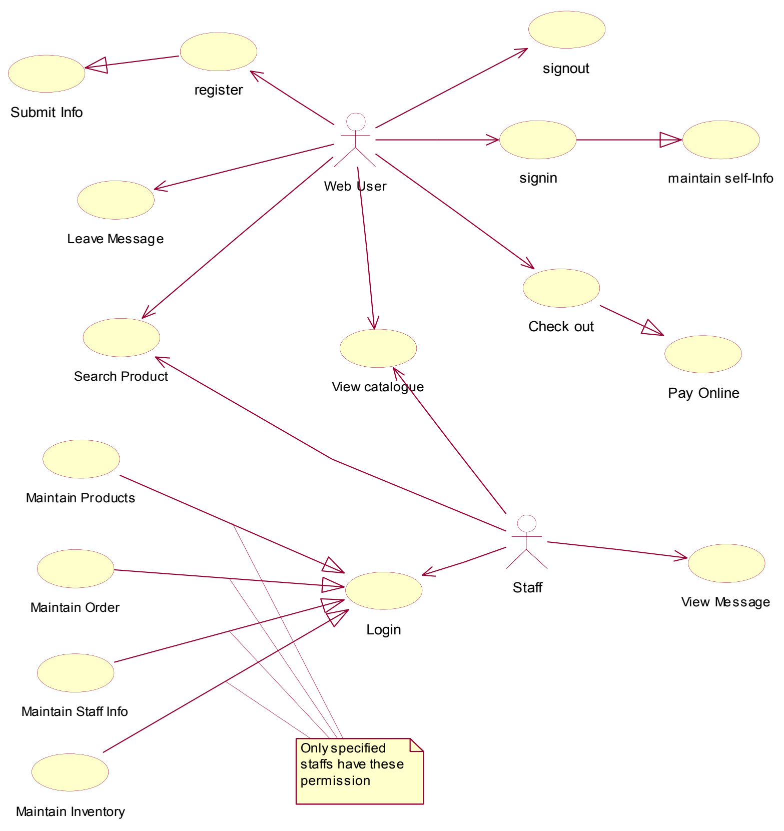


Figure 4-1 E-Furniture system main use case diagram

Staff who have relevant permissions could make sales, view messages from customer sent through the site, and, could also handle delivery and returned products. Some authorized staff could login to the maintenance system to maintain products, orders, and staff information.

Following are two detailed use case descriptions and their event flow. For more use case details descriptions please refer Appendix I.

Search Product	
Actor	Customer
Precondition	Customer views or searches the products or services on line
Postcondition	System shows the search result on the screen
Description	Customer searches the company information (products/services) or individual information
Basic course of action	Products/Services ID or relevant keywords are entered and searching results are displayed

<u>Use Case event flow for "Search Product":</u>

1. Used by customers via Web browser to query for products by product ID, product Name or show all.
2. Event flows:

   2.1 Repeat until customer leaves Web page:

   2.1.1 Customer types in Product ID, Product Name or select from Category list. Customer also can view whole range of product by category at once. Categories might include "Office furniture", "Home furniture", "Bedroom furniture", "Lounge furniture" etc. The system records the following information about furniture: a unique ID, name, price, description, size, and category.

   2.1.2 Customer clicks "Find" button and a list of matching furniture are returned showing furniture ID and Name. If no furniture found, go to step 2.2. If error from server, go to 2.3.

2.1.3 Customer clicks on furniture ID/Name. More detailed information about the furniture will display e.g. picture, description, size, category and prices.

2.2 No furniture found – error message displayed. Go to 2.1.1.

2.3 Server error – error message displayed. Go to 2.1.1.

3 Related Actors/Use cases: Used by Customer actor

4 Special conditions: Uses WWW Http JSP interface

Checkout	
Actor	Customer
Precondition	Customers have selected products in shopping cart
Postcondition	System records the detail of the orders, payment and customer information for further processing.
Description	Customers checkout and make payment online
Basic course of action	System assigns a new order ID and system responds whether the order is accepted or not. If the order is accepted then the order information will store into database for further processing, otherwise, report error back to the customer.
Alternative courses of action	If the customer is a new customer, then the system will request the customer registering first.

Use Case event flow for "Checkout":

1. Used by customer to buy furniture and make payment online.
2. Event flows:
   2.1. Repeat until exit:
      2.1.1. Customer login as member or register new customer. If login as exist member, go to 2.1.3.
      2.1.2. Register new customer, enter customer detail information and submit go to 2.1.4.
      2.1.3. Input customer ID and password, System looks up customer, if login fail, go to 2.3. If database or server error, go to 2.4.

2.1.4. Show shopping cart items summary, total cost and credit card fill in form. Enter credit card information and submit. If credit card is not valid, go to 21.4.

2.1.5. Customer enters billing information and submits or customer can terminate checkout by clicking sign out. If billing information not valid to 2.1.5.

2.1.6. Customer enters shipping information and submits or customer can terminate checkout by clicking sign out.

2.1.7. Show summary of detail order, billing information and shipping information. Customer could commit or cancel order. If commit go to 2.2.

2.1.8. Show main furniture shop, end checkout use case.

2.2. Insert detailed order information into database. If order create successful, use case end. If there is server error go to 2.4.

2.3. Login failed – error message shows login failed, go to 2.1.1.

2.4. Server error – error message indicating server error occurred make order failed go to 2.1.1.

3. Related Actors/Use cases: Customer
4. Special Conditions: Uses Http/JSP interface.

4.2.2 Class modelling

The quality of the building system in component-based development is essentially a reflection of the quality of the class model. This is because the class model sets the underlying foundation on which objects will be put to work. A quality class model should provide a flexible foundation on which systems can be assembled in component-like fashion.

Class modelling focuses on static system structure in terms of classes and association. A class is a set of objects that share a common structure and a common behavior. An object is something you can do things to and it has state, behavior and identity. The structure and behavior of similar objects are defined in their common class. Similarly and association between classes is an abstraction of its constituent links between objects

[2]. Based on Select Perspective class-modelling principles, E-Furniture system class modelling structure was built, which is shown in Figure 4-2. As can be seen, the class diagram figure out main business classes of the system, and also shows the relationships of these classes. That'll be very helpful for capturing and extracting components.

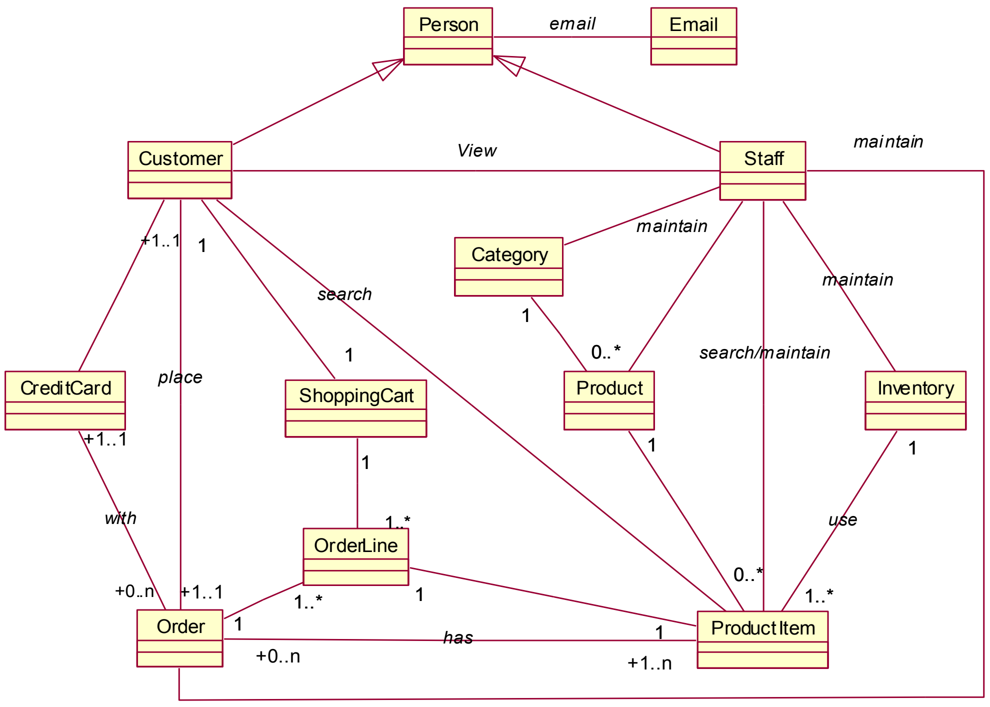


Figure 4-2 E-Furniture system classes diagram

The following tables show briefly descriptions of main classes in figure 4-2:

Classes	Descriptions
Customer	Encapsulate all the information of client include personal information and account detail. Customer shopping need to register an account. A set of business methods will define to access its information.
Order	Record customer ordered products information include billing information, shipping information and these information access methods.
Staff	Back-end maintaining person defined personal information and permission groups that specify its access scope. It also will specify a set of business method access its information.

Classes	Descriptions (continue)
Product	Encapsulate all the properties of products and a set of access method include searching functions etc
Inventory	Recording products stock information such as amount and minimum stock level etc. and its information access methods.
CeditCard	Customer order payment information that need very high secure levels.

4.2.3 Object interaction Sequence modelling

Sequence diagram is used to describe a use case or operation that is named according to use case name. The required objects declared across the top of the sequence diagram. This involves reasoning the class diagram in relation to the needs of the use case and looking for candidate objects [2]. These are illustrated in the following diagrams.

1. *Search products Sequence*

Figure 4-3 is the sequence diagram of search products, which can allow use to find some appropriate products according to the key words they want to find. The diagram shows the scenario how the relevant products been found. A web user involved with the sequence, and a controller to receive and dispatch web user's action. It's also will use Category and Product class to search for relevant information. However, the diagram just shows the lower-level functional event flow, no other information could be found, such as persistency and transaction etc.

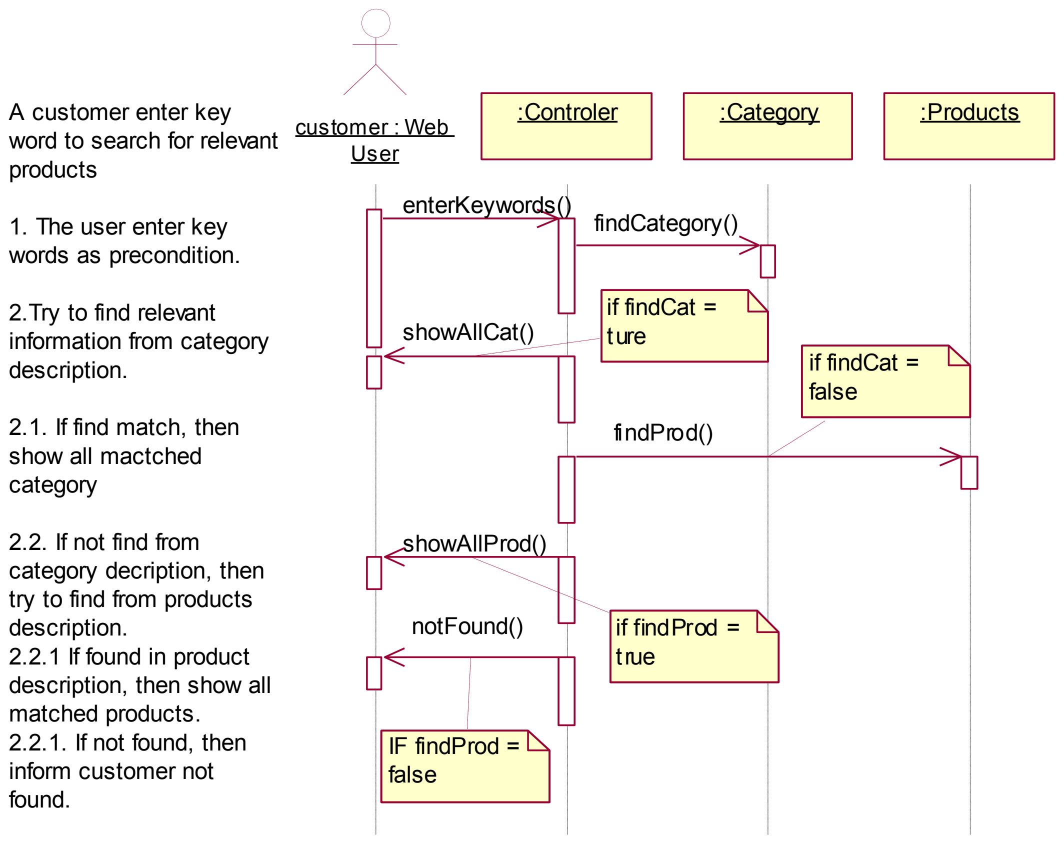
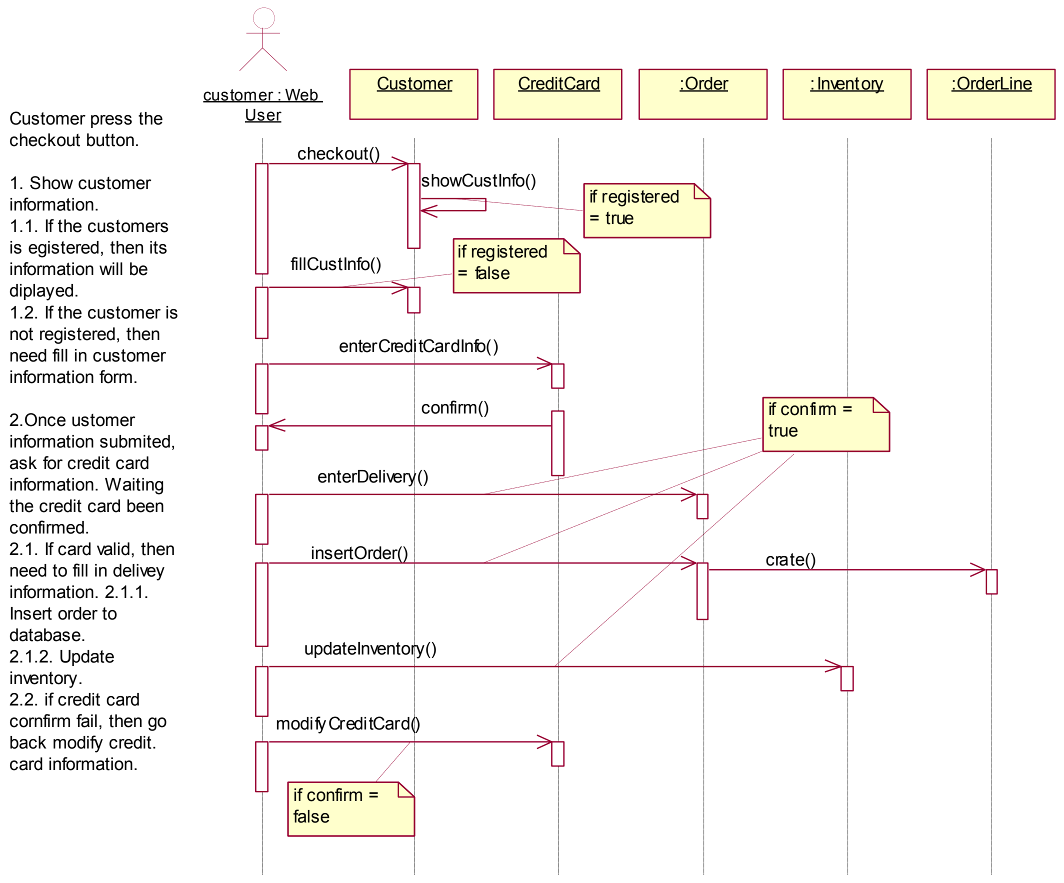

A customer enter key word to search for relevant products
1. The user enter key words as precondition.
2.Try to find relevant information from category description.
2.1. If find match, then show all mactched category
2.2. If not find from category decription, then try to find from products description.
2.2.1 If found in product description, then show all matched products.
2.2.1. If not found, then inform customer not found.
customer : Web User
:Controler
:Category
:Products
enterKeywords()
findCategory()
if findCat = ture
showAllCat()
if findCat = false
findProd()
showAllProd()
if findProd = true
notFound()
IF findProd = false

Figure 4-3 Search products sequence diagram

2. *Checkout Sequence diagram*

Figure 4-4 shows the scenario once the customer finish shopping invoked the checkout event. The system will go through the event flow to check credit card, insert relevant delivery information and order information into databases, and update inventory. Some of events occur have to satisfy some conditions, such as, if the customer is registered, and then show the customer information, else the system will require user register customer information. These constrained conditions clearly indicated how the whole event flow going. It also shows the objects involved with the event, may also need create new objects to make the event flow reasonable.



Customer press the checkout button.

1. Show customer information.
1.1. If the customers is egistered, then its information will be diplayed.
1.2. If the customer is not registered, then need fill in customer information form.

2.Once ustomer information submited, ask for credit card information. Waiting the credit card been confirmed.
2.1. If card valid, then need to fill in delivey information. 2.1.1. Insert order to database.
2.1.2. Update inventory.
2.2. if credit card cornfirm fail, then go back modify credit. card information.

Figure 4-4 Checkout sequence diagram

3. *Maintain products sequence*

The Maintain products sequence diagram Figure 4-5, shows how staff maintain products and category information event flows. The staff can add new categories and products, can also update and delete category and product items. The update and delete operation will need to find the item first, then do update or delete. The sequence diagram gives us a clear event flow for the maintain sequence and objects involved with the event. It'll help developer figure out more objects the event needs. Then developer could refine their class diagram from bottom up.

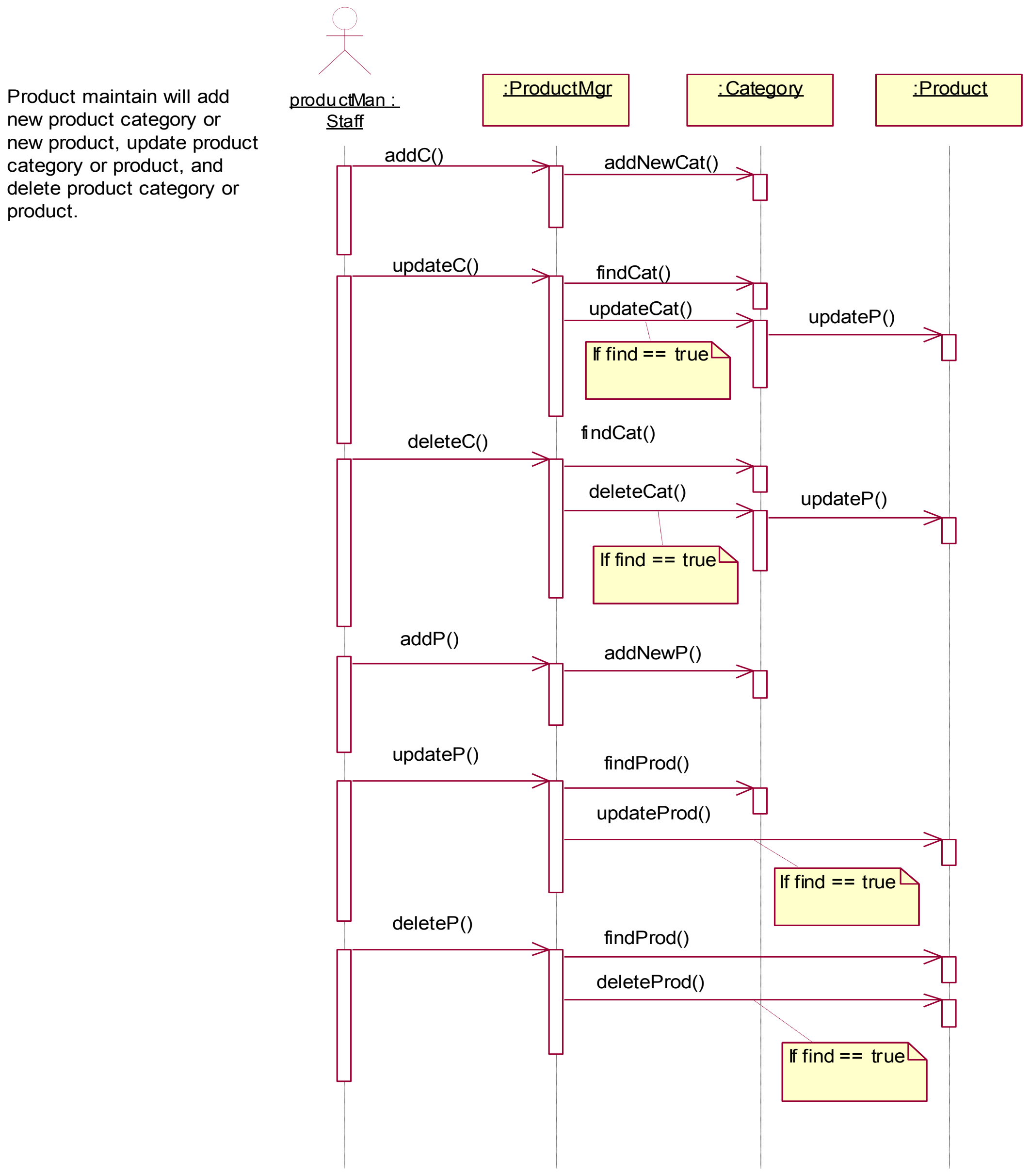


Figure 4-5 Maintain Products sequence diagram

4. *Maintain Inventory sequence*

The maintain inventory use cases used by staff to modify and maintain the inventory database. Figure 4-6 shows the scenario of add new inventory items, update and delete inventory items. The actor staff will invoke these operations via user interfaces. Then the function and method calls will go through relevant objects to fulfil the operation. It also defined the objects that involved with this event.

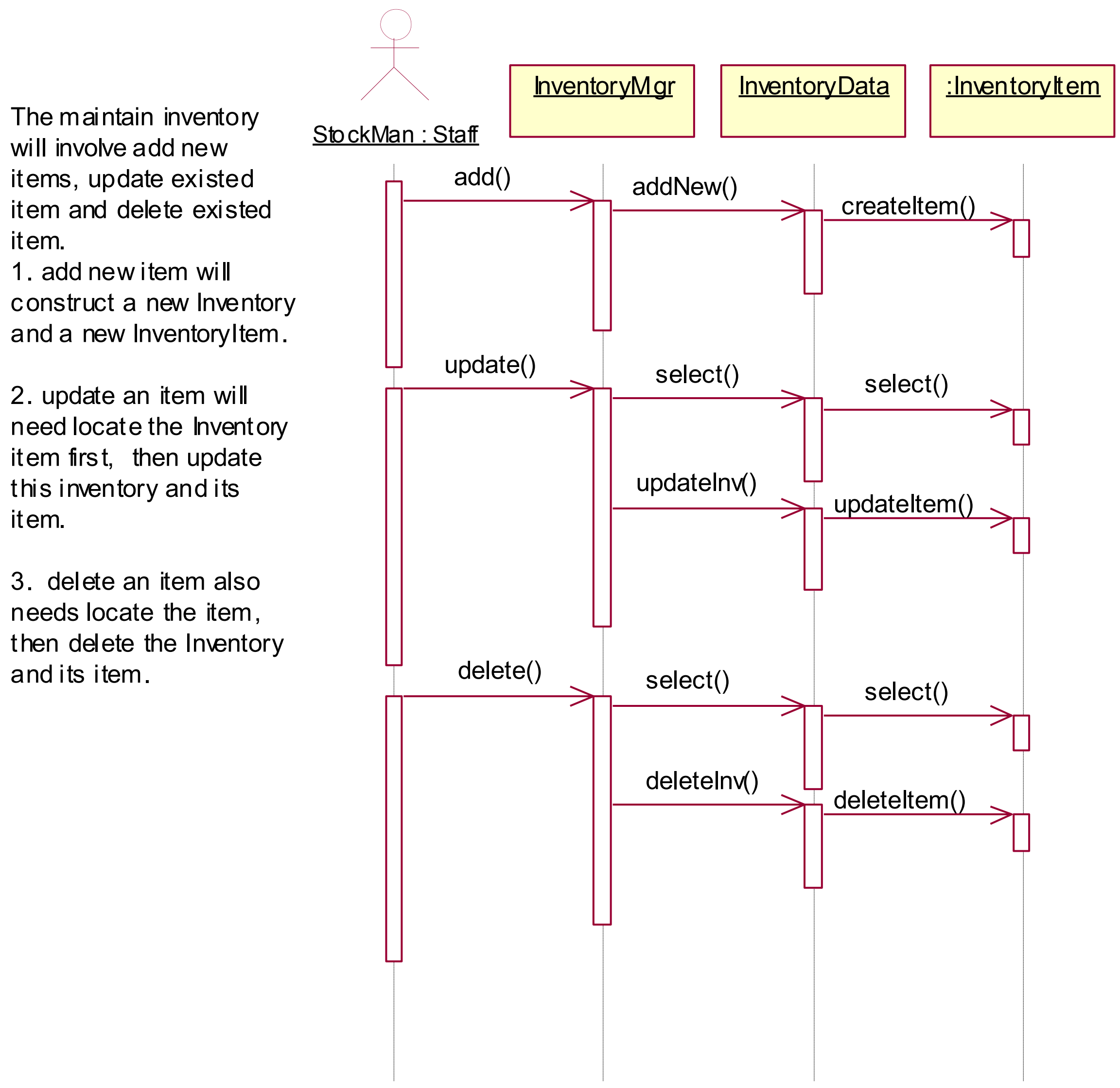

The maintain inventory will involve add new items, update existed item and delete existed item.
1. add new item will construct a new Inventory and a new InventoryItem.
2. update an item will need locate the Inventory item first, then update this inventory and its item.
3. delete an item also needs locate the item, then delete the Inventory and its item.
StockMan : Staff
InventoryMgr
InventoryData
:InventoryItem
add()
addNew()
createItem()
update()
select()
select()
updateInv()
updateItem()
delete()
select()
select()
deleteInv()
deleteItem()

Figure 4-6 Maintain inventory sequence diagram

4.3 The System Design – Component-based design

Object modelling has taken a firm position on the software industry transforming the way of software development. The unified modelling language has been widely used in software design and modelling. However, object-oriented modelling and design are not enough for some complex, scalable, reusable and integrate able focused systems. The Select Perspective component-based approach identifies and groups components from functional dependent perspectives rather than scratch from low-level objects. It vertically slices overall system.

The system we are trying to develop is focus on component-based system to increase the system's scalability and reusability etc. This section will give the system component modelling and detailed design.

4.3.1 Deployment Modelling

Deployment modelling explores and defines the configuration of run-time processing elements and the component packages. Deployment architecture will affect the use of the system in terms of such as basics cost, response time, convenience of access to the system, business efficiency, and usability of the system [2]. Figure 4 – 7 shows the e-furniture system deployment architecture. The application/web server located in the same centre node, and there is a mirror server to ensure the system highly availability. The database occupied a dependent node, which will be more easily to manage, backup data and securing. Enterprise back-end management applications will be located in the other nodes such as staff desktop or manager desktop. The protocol used for enterprise application will be CORBA/RMI. The web application user will use standard HTTP protocol via web browser in the Internet. The deployment diagram will give component developer a main architecture of a software system.

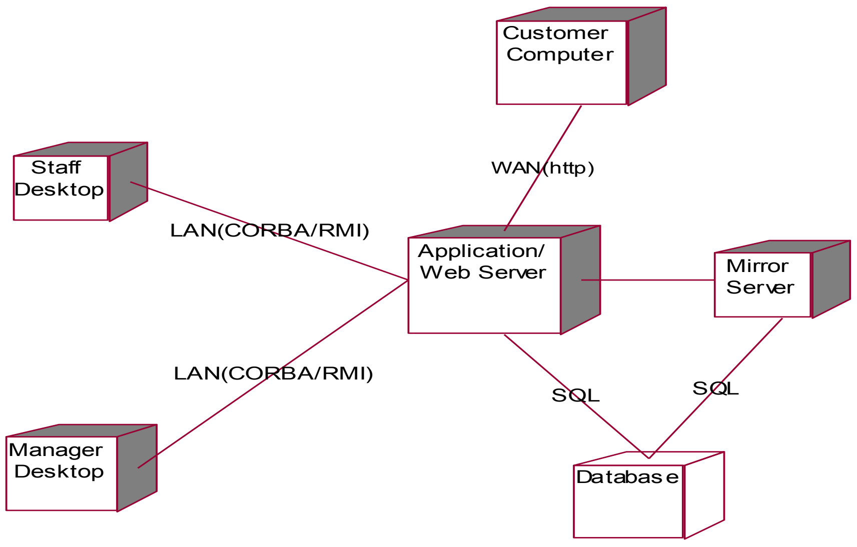


Figure 4-7 E-Furniture systems Deployment diagram

4.3.2 The System Architecture modelling

System architecture is the overall organization of domains into service packages [2]. Architecting service packages in the development process will help us to identify components and set-up the reusable infrastructure. It also allows later incremental design to focus on specific implementation detail without being overloaded with wider architectural concerns. The select perspective uses UML packages component modelling abstract software architecture and component. And component detail architecture uses UML class diagram like component modelling.

Figure 4-8 illustrates E-furniture system package architecture. User interface act as user services package, which is the top services level. The middle level are the business services packages, which are acting the main role utilize the system functional services. All data have been used data services packages to wrap onto relational databases.

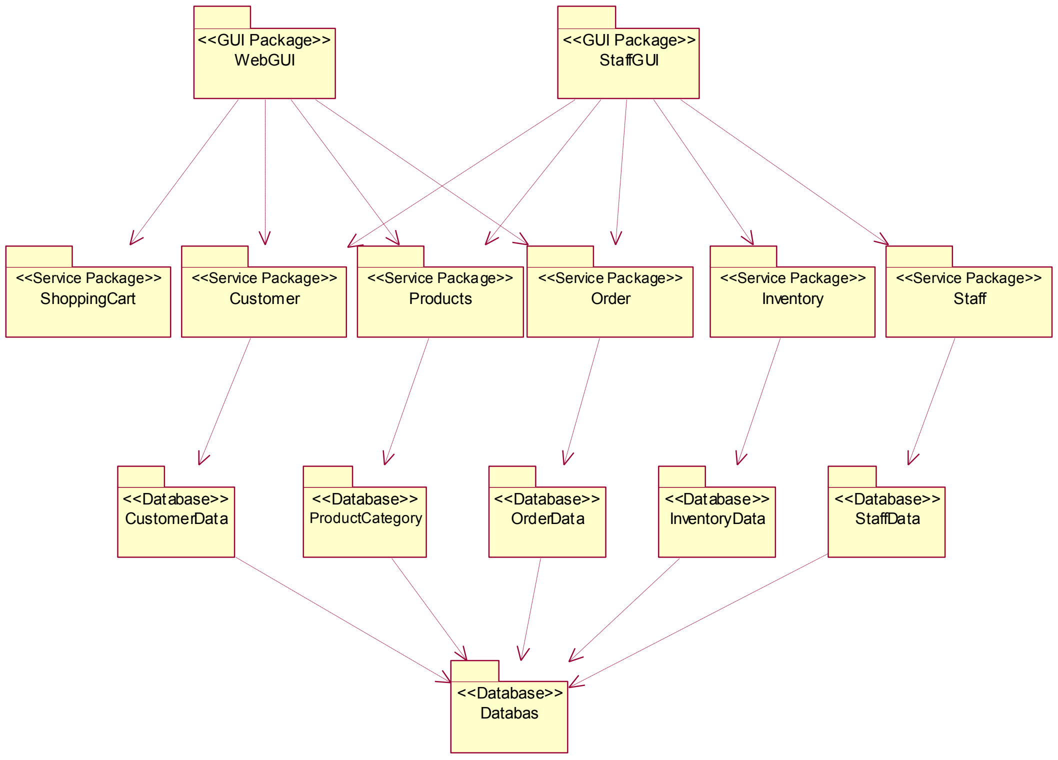


Figure 4-8 Business service package diagram

4.3.3 Component design.

Components are very flexible and powerful means of implementing reusable services through a consistent, published interface that includes an interaction standard. This section outlined components based on the services architecture and business-oriented modelling techniques. Detailed components were designed by using UML notations. Each component composed by interfaces, which published for others, some services objects and data objects that are used to wrap relational database in order to communicate with databases. The followings are the example system's components detailed design structure.

1. Customer component structure

   Customer component is most e-commerce domain-related reusable component, which published interfaces are identified from requirements of the system client actor needed and iterated from use case, dynamic and class model. The CustomerMgrImp (customer data manager object) is designed to manage all information of customer in the databases. And, the CustoerData object is designed to wrap rational database information, its attributes and access methods iteratively refined from requirements, use case and object dynamic interaction model. The component also designed data connection facility to connect with relational databases that customer information stored. The Figure 4 – 9 shows the customer component detailed architecture and design.

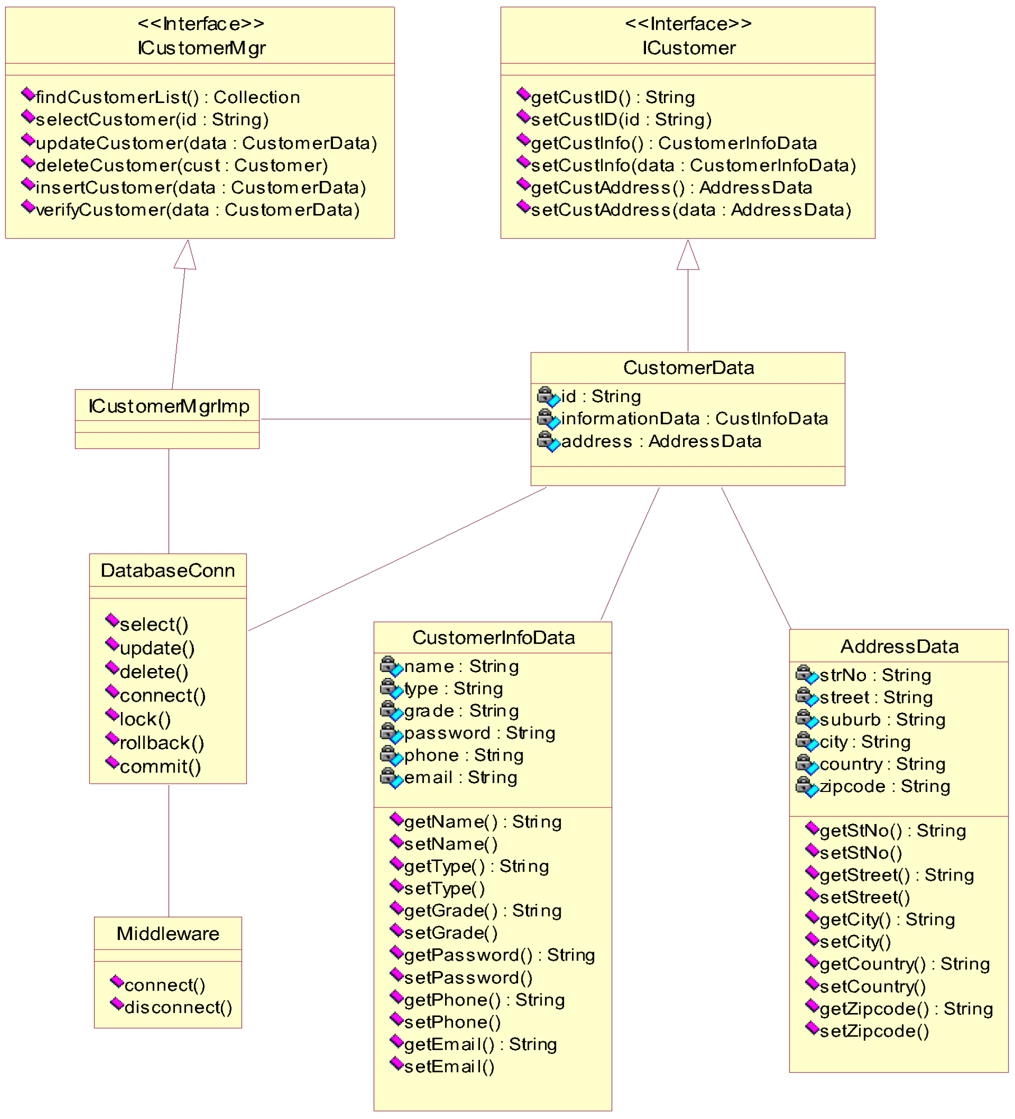


Figure 4-9 Customer component diagram

2. Order component

The component is used for handling all the information of associated with customer orders - include customer ID, delivery, payment information and order item etc. information. The information was wrapped by object OrderData and OrderLine that attributes and access methods were refined from requirements, use case and dynamic models. The component interfaces were also captured from system requirements and class model. Figure 4-10 shows the component detailed structure and its services can provide to others. It also shows the component need other component support to fully functional such as Product, Customer, and Inventory components. The data storage and retrieval are via middleware through SQL statement.

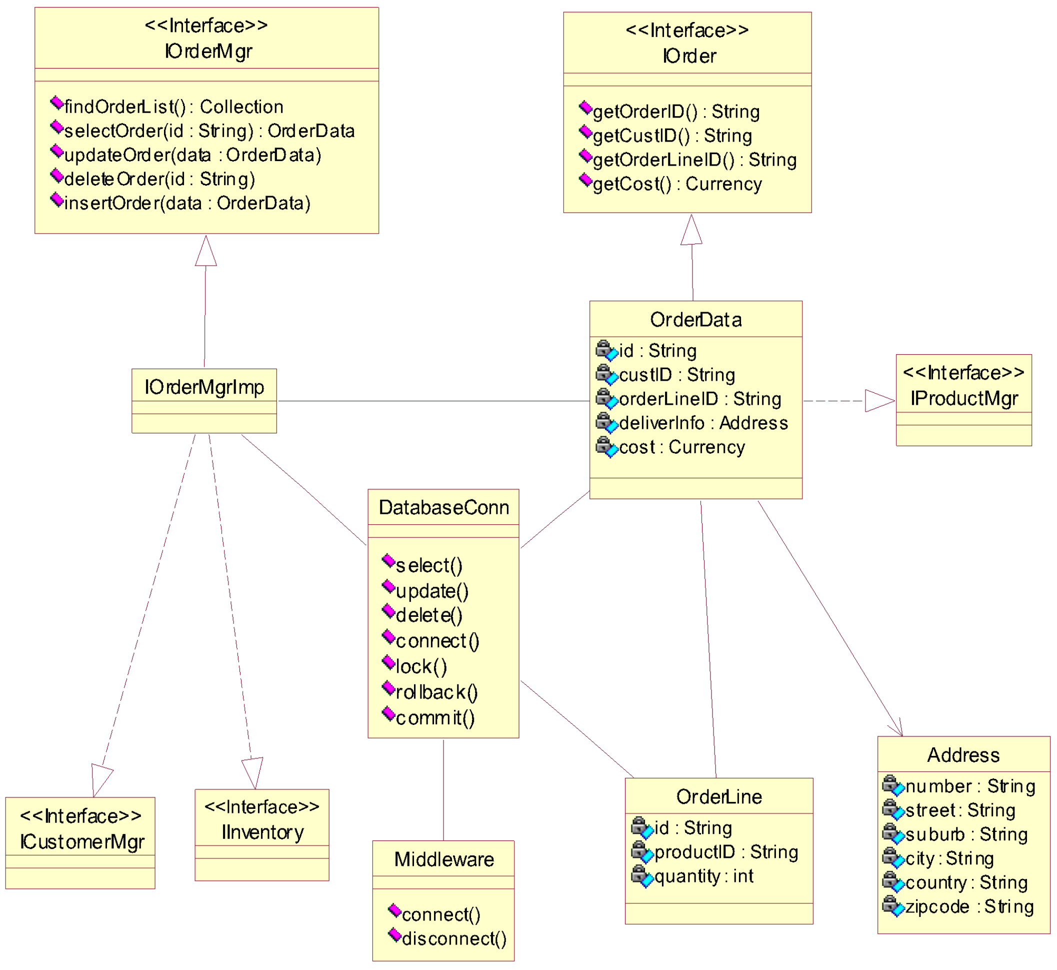


Figure 4-10 Order component diagram

3. ShoppingCart and Product component

When customer shopping in an e-store such as e-furniture store, most of services will be provided by a component that need to encapsulate business logics, may also provide information about products they are selling. Therefore, we designed the ShoppingCart and Product combined component that provide more interfaces for others to access. Because most services of the shopping cart depend on product services. This component combined shopping cart functionality and products catalog information handling functions together. However, the shopping cart services also need other two components to corporate, they are Order and Inventory components. Figure 4-11 shows the component structure and detailed services.

The component also designed for handling products catalogue information - include detailed products information and its category classification. It can provide product information maintain services via IProductsMgr interface. Users could search products by calling the component service; it could also used for maintaining products information that stored in databases.

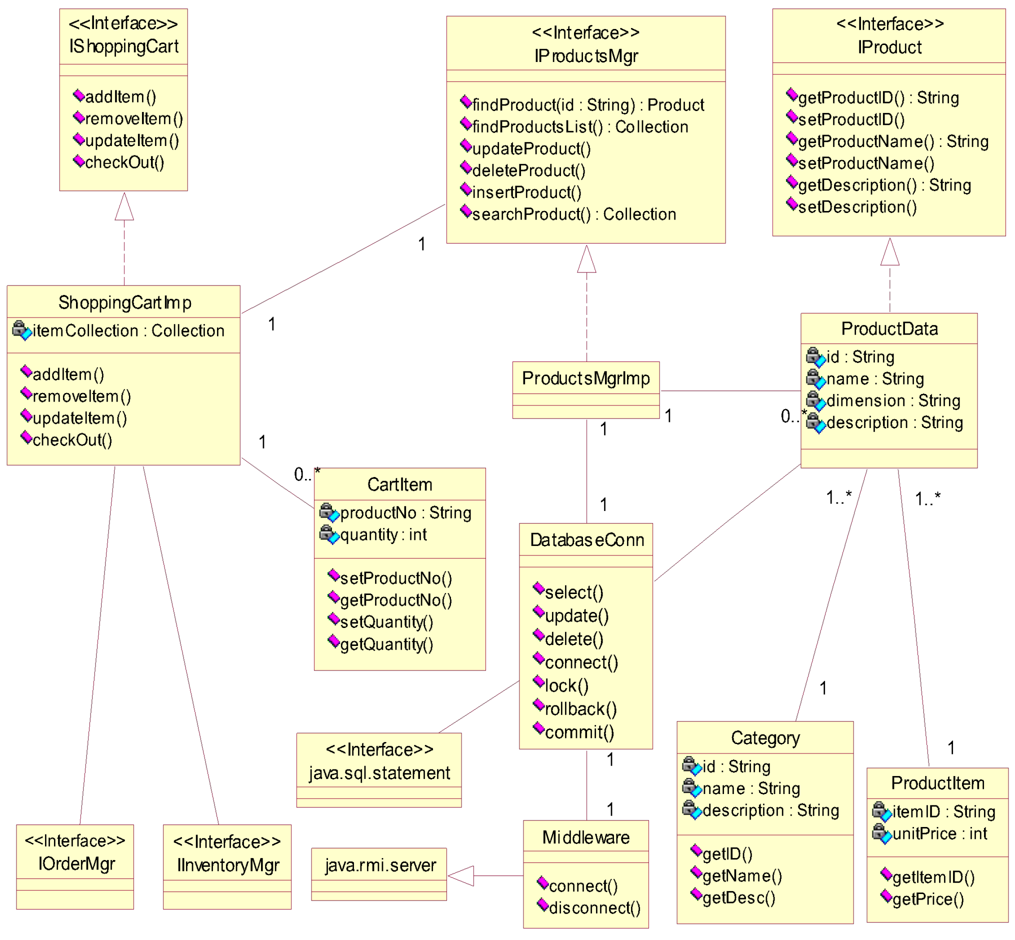


Figure 4-11 ShoppingCart and Product component diagram

4. Staff component

According to the system requirements, the back-end database information need to maintain by staffs. And staff personal information and the permission of access databases need to define as well. So, we designed the staff component to fulfill the functionalities and wrap the information of staff should have. Its interfaces are iteratively refined from requirements and previous models such as use case and objects interaction model.

The structure of the Staff component is shown in figure 4-12. This component provides all the functional services associated with staff information via its interfaces. The StaffData class extends the PersonData class, which abstract general person properties. These classes are the wrapper of relational databases. They communicate with database via middle ware by SQL statements. The component could provide staff information creation and maintaining services.

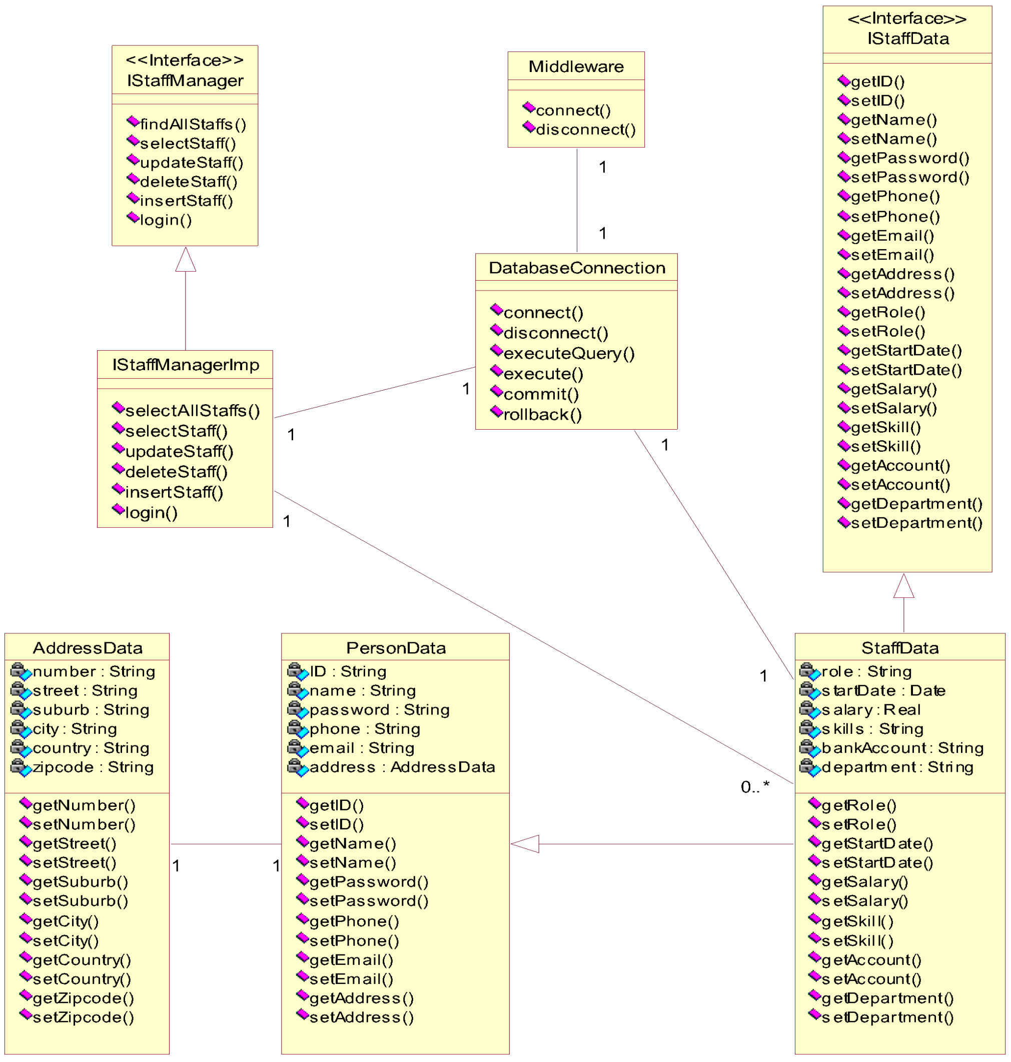


Figure 4-12 Staff component diagram

5. Inventory component

The system products stock information need to record and manipulate according to requirements. Hence, we designed the inventory component to dealing with stock information to insert new inventory item, to update inventory item and to delete inventory items etc. These functionalities and the inventory attributes are iteratively refined from requirements and other models i.e. use case and sequence diagrams. Figure 4-13 gives the component's detailed structure and its services provided via its interfaces. The component also has to be supported by ShoppingCart and Product component to fully functional well. The component will communicate with database as well to maintain the information stored in database.

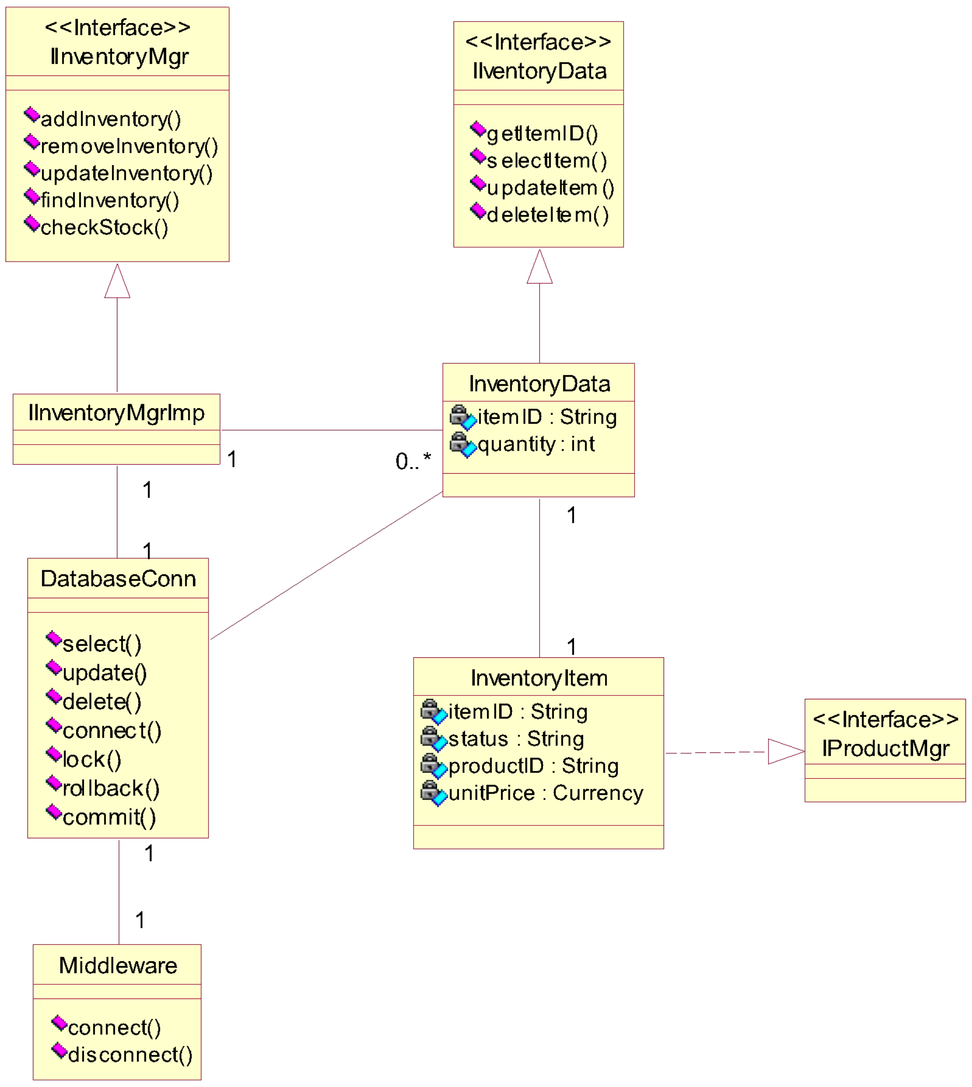


Figure 4-13 Inventory component diagram

6. Application UI component

To prototype a complete system and according to the system requirements, the e-furniture system needs an application to manage back-end database information. Therefore, we designed the component to provide user interface services and hook user access functions to the component. The functionalities mainly come from use case and requirements. They were carefully refined iteratively from use case sequence to component and back to use case again.

Figure 4-14 shows the application user interfaces structure and other components it will use. The main user interface functionality will different according to user's permission, when a user login to the system. Any communication with the server and database will go though the middleware class.

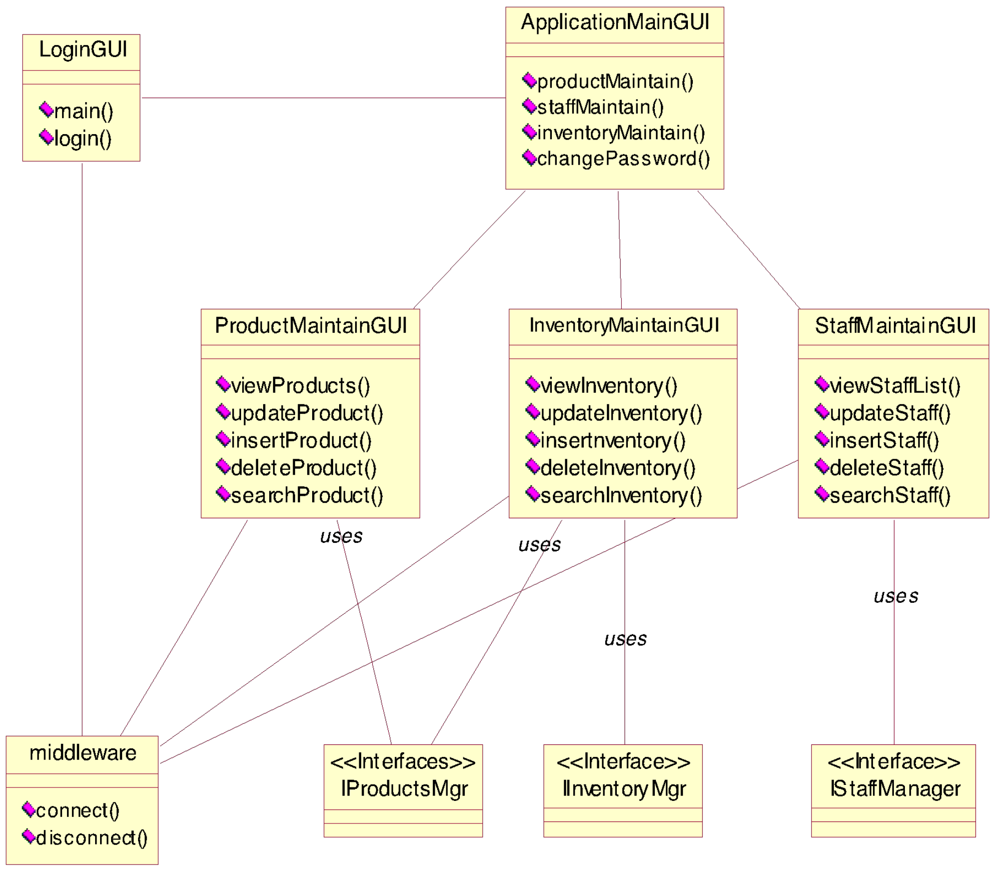


Figure 4-14 Application UI component diagram

7. Customer UI component

The e-furniture system main functionalities are that can provide online shopping environment for worldwide users. So the ideal way is design a web-based thin client that will provide any functions of online shopping and browsing products information for online customers. Therefore, based the system web user requirements and use cases, we designed the component. Its functionalities are iteratively refined from requirements, use case to the component and back to requirements and use cases.

Most of functionalities of customer can do are invoked through this component. The Figure 4-15 gives the component structure, its main and sub user interfaces, and detailed functionalities. It also shows the component will interact with other components, such as Customer, ShoppingCart, Order, and Inventory components. This component illustrated overall structure of online user interfaces and its functionality.

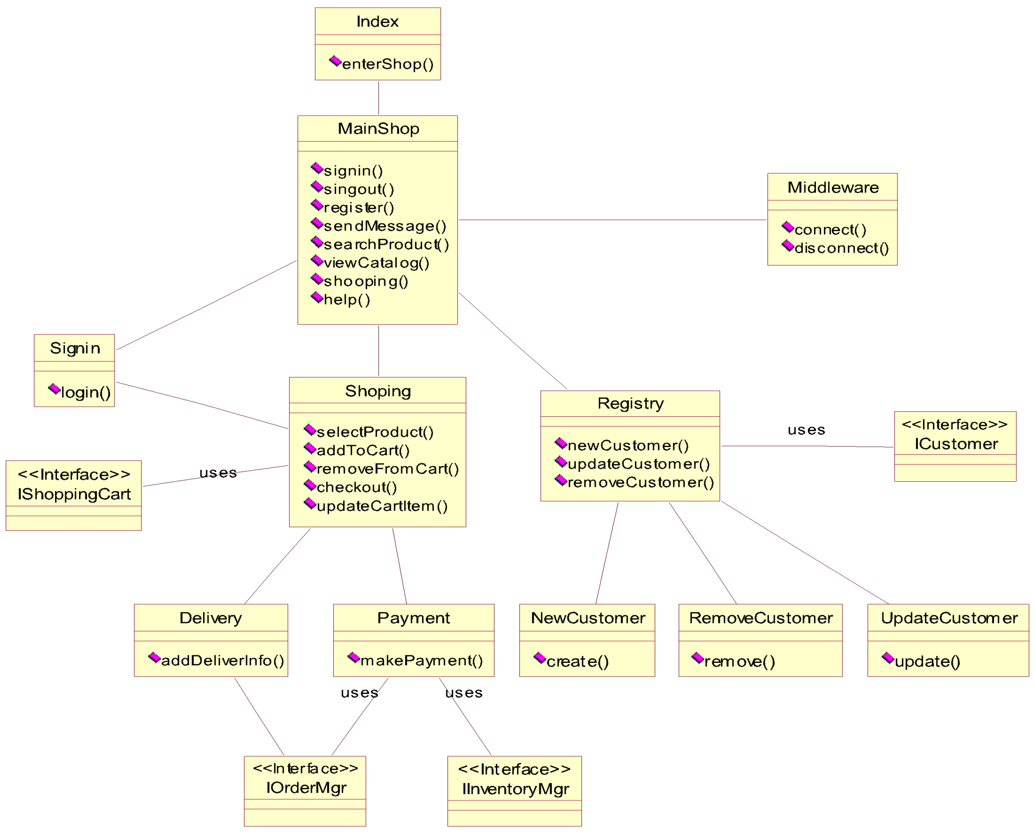


Figure 4-15 Customer UI component diagram

4.3.4 Summary

From the system requirements, use case diagram, class and dynamic objects interaction diagrams such as the sequence diagram, we have captured most of the system's service functionality. For example: user services, business services and data services. According to these services, (especially the business services), we designed the system service and deployment architectures. Furthermore, based on this analysis and architectural design, we decomposed the system into several components with functional services. As can be seen that all these components are mainly focused on functional perspectives by vertical slicing the system. These components provide lower-level functional information about the system and component itself. They are lacking of systemic higher-level characteristic properties such as transaction, persistency and security. We'll address these issues in the next chapter.

Chapter 5 Aspect-Oriented Analysis for E-Furniture System Design

From the previous chapter, we find the whole system analysis and design is mainly focused on the system's vertical functional slices used to address components and their associated data wrapping objects. Like most component-based techniques The Select Perspective™ approach also focuses on identifying component interfaces to support the vertical slice and functional decomposition. These approaches will lead components to encode low-level information about the component interfaces for run time use [22]. In this chapter, we'll try to analyze the component design by adding aspect information, most of which will come from horizontal slices of the overall system including functional and non-functional constraints. The methodology of Aspect-Oriented Component Engineering (AOCE) [22] will be applied to analyze the system design in order to address the issues of crosscutting component services. Identifying aspect-categorized services by using AOCE will allow us to reason about components interaction from various systemic aspects.

Aspects are horizontal slices through a system. They typically affect many components identified by functional decomposition of common system characteristics such as user interfaces, persistency, transaction, performance and security [22]. Component developers use aspects to describe different perspectives on systemic component capabilities during the software engineering lifecycle. The aspects might be very different in different domains, such as real-time response and memory management may useful real time control system. In the e-commerce system domain, those aspects were identified and found useful, especially for the e-furniture system. They are illustrated in figure 5-1. The following sections will illustrate system analysis and design with aspects. However, representing these aspects information in UML diagram is an issue. We'll discus that in the next chapter.

Aspect	Aspect Details	Description
User Interface	Views Frame Feedback Extensible parts	Components supporting or requiring user interfaces, including extensible and compos able interfaces for several components
Performance	Speed Robustness	Components supporting or requiring performance loading speed or stable ability
Transaction	Rollback Commit Locking	Components supporting or requiring transaction management capabilities
Persistency	Store/retrieve data Locate data Lock Data	Components supporting or requiring data persistency management facilities
Security	Encoding model Key distribution Authentication Access Control	Components supporting or requiring inter-component security models, data encoding and cryptography

Figure 5-1 Some of e-furniture system aspects and some aspect details

5.1 Class Modelling with Aspects Analysis

Class models only give developers a static system structure and associations of classes. However, the classes diagram lacks cross-cutting higher-level systemic information, such as transaction, persistency aspect information. Each class may provide/require various aspects that cross-cut multiple components, but they are not documented in class diagram and other UML diagrams. Moreover, these aspects information sometimes better characterize important system structure and associations. The following is some detailed aspect information of the e-furniture system class diagram potentially could be represented.

Aspects	Aspect details, detail properties and brief reasoning
<<Security>>	<<– required>> access authentication, detail properties can be password authentication mask. To vie customer information need staff login first.
	<<– required>> encode/decode data, can be Base 64 encode/decode or using some cryptography algorithms. The services can be provided by Encoder/Decoder component or cryptography package provider, such as, Bouncy Castle [32] JCE provider package.
<<Transaction>>	<<+ provided>> send/receive data, the transaction can use WAN/LAN connections, CORBA/RMI/COM protocol.
	<<+ provided>>lock/rollback data, some class can provide the service to users when access some information or error occur. Meanwhile, the class may require database provide same services.
<<Persistency>>	<< + provided >> store/retrieve data, detail properties can be insert, update, select and delete data. The services will be required by application or web users.
	<< – required>> store/retrieve data, some classes and associations may require other class or database provide the services as well.
	<< – required >> storage media, the detail property could be file or database.

Figure 5-2 shows some classes with aspects information and associations with aspects information annotated. From the annotated class diagram, the developers can much more clearly see that the customer class will provide and require persistency aspects (it saves data, but needs a component to provide file or database access to do this) to the system as a whole. It also will provide transaction and security aspects. The association of staff to view customer information will require security, transaction and persistency aspects (security as seen that data is exchanged; transaction as result data update; and persistency to save data). Therefore, developer could reason about the association to see whether the associated class could provide those aspects.

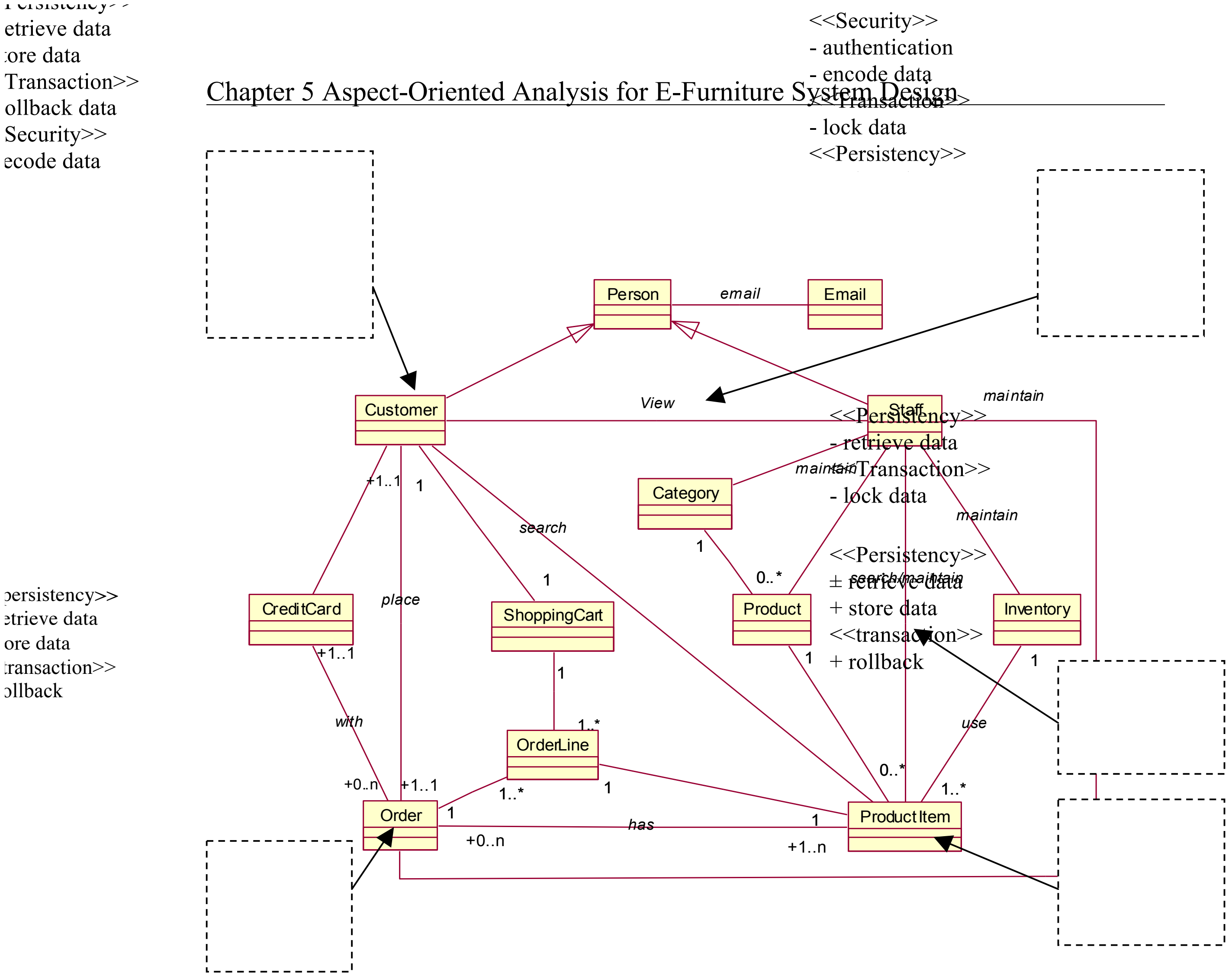


Figure 5-2 Class diagram with aspect analysis

By applying AOCE to analysis the class diagram can be found that most of classes provided persistency and transaction aspect service except ShoppingCart and Email classes. These two classes provide distribution aspect service and require transaction and persistency aspect services that we can address from system requirements and cross-cutting points of view.

5.2 Dynamic Object interaction modelling with Aspects

Objects interaction modelling such as sequence diagrams and collaboration diagrams can give developers information about dynamic class operations. Although object interaction modelling are very helpful to find classes, operation, services and group components, they also lack information about systemic non-functional aspects, for example, performance, persistency, security, transaction and distribution aspect information. The following section illustrates object interaction diagrams extension with aspect information.

5.2.1 Search products sequence diagrams with aspects analysis

The search products sequence diagram illustrates some function calls and involved classes. However, it lacks systemic information, for example, user interfaces, performance and distribution aspect information. Searching performance and search results user view aspects are very important information for a searching sequence. When analysis the sequence diagram from systemic perspectives, we found that the following detailed aspects are very useful to annotate the diagram in order to present more information. Figure 5-3 shows those aspect information related to function call or classes.

Aspects	Aspect details, detail properties and brief reasoning
<<User Interface>>	<<+ provided>> results views, present search result information to application or web users. The services required by web users or inventory maintaining staff, such as online shopping customer.
	<<– required>> frame/form, detailed properties will be GUI component or HTML tags. That can be provided by HTML and Swing components.
	<<+ provided>> response time, less than 5 seconds. Required by web users.
<<Distribution>>	<<+ provided>> locate/identify data. The service required by customers. Some class will also require others to provide same services, such as database connection JDBC.
	<<– required>> send/receive data, need transport services to transfer data. It will be provided by WAN/LAN connection through middleware facilities like CORBA/RMI. The transfer speed maybe limited.
<<Persistency>>	<<– required>> data retrieve, the sequence will required to retrieve information from other class or database such as products class. The data size may limit up to 10 MB.

+locate object ± retrieve data
- send/receive data

A customer enter key word to search for relevant products

1. The user enter key words as precondition.

2.Try to find relevant information from category description.

2.1. If find match, then show all mactched category

2.2. If not find from category decription, then try to find from products description.
2.2.1 If found in product description, then show all matched products.
2.2.1. If not found, then inform customer not found.

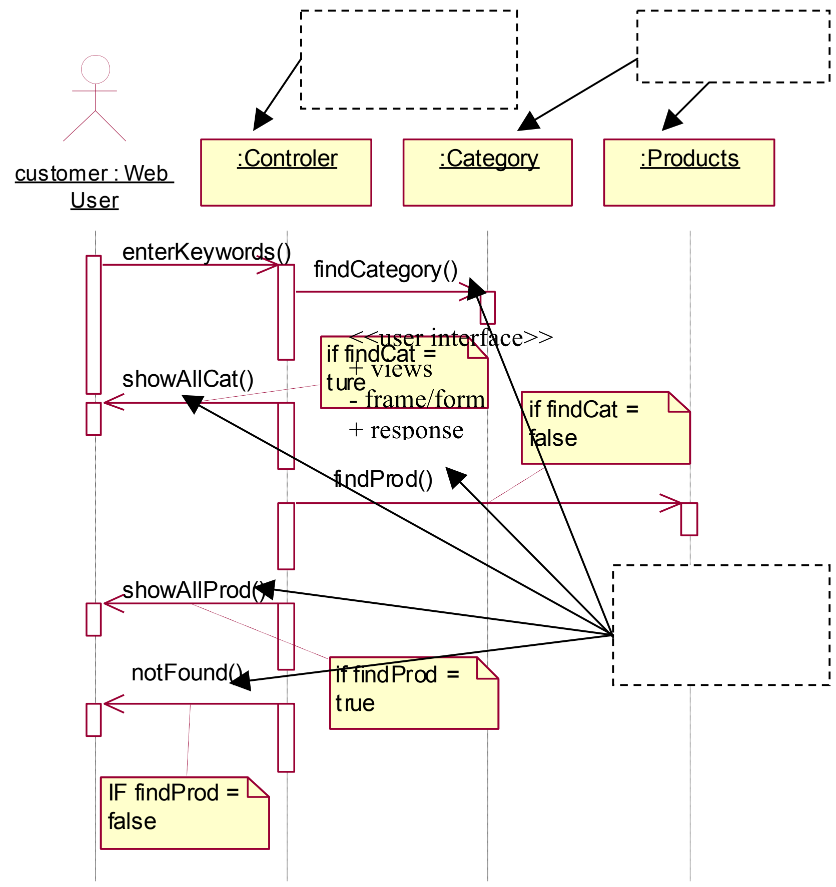
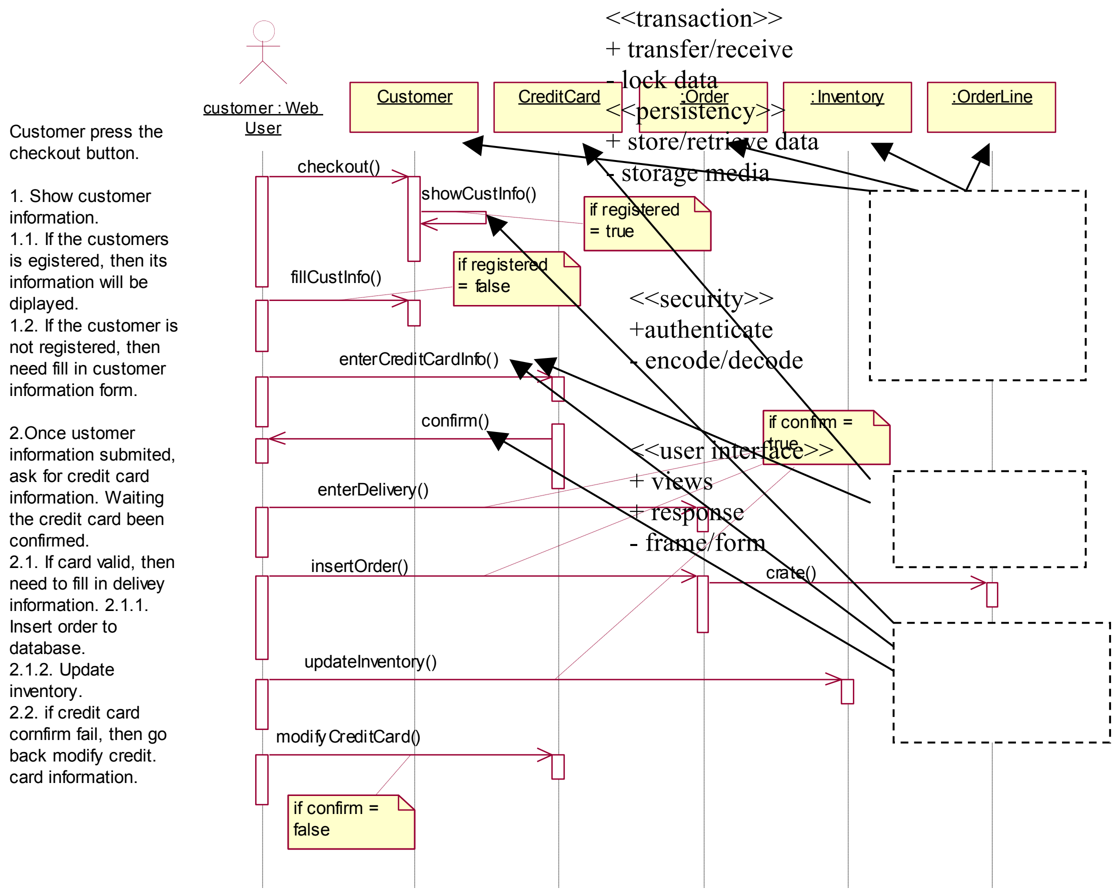


Figure 5-3 Search products sequence diagram with aspect analysis

5.2.2 Checkout sequence diagram with aspects analysis

Checkout sequence diagram shows to developer lower-level functional event flow and objects interactions. However, some systemic higher-level information is not clearly represented, such as user interfaces, transaction, and performance aspect information. The following aspects details illustrate the systemic aspect information. Figure 5-4 shows these aspects information associated classes and functions.

Aspects	Aspect details, detail properties and brief reasoning
<<User Interface>>	<<+ provided>> customer info/credit card info views, present process information and feedback information to web users. The services required by web users, such as online shopping customer.
	<<– required>> frame/form, detailed properties will be HTML tags or GUI component. That can be provided by HTML, JSP and swing components.
	<<+ provided>> response time, less than 5 seconds. Required by web users.
<<Transaction>>	<<+ provided>> data transferring/receiving, need transport services to transfer data. It will be provided by WAN/LAN connection through middleware facilities like CORBA/RMI.
	<<–required>> lock/rollback data, used to avoid concurrent access and error handling.
<<Persistency>>	<<+ provided>> store/retrieve data, some objects information need to be persistent and can be retrieve. The data size store or retrieve once may limit up to 10 MB.
	<<– required>> storage media, such as file or database.
<<Security>>	<<+provided>> access authentication, when user checkout need to login or register to gain access right. The detail properties can be password masks.
	<<– required>> encode/decode data, such as base 64 encoding scheme or cryptography algorithms, which will be provided by cryptography packages like JCE Provider.



<<transaction>>
+ transfer/receive
- lock data
<<persistency>>
+ store/retrieve data
- storage media
customer : Web User
Customer
CreditCard
:Order
:Inventory
:OrderLine
Customer press the checkout button.
1. Show customer information.
1.1. If the customers is egistered, then its information will be diplayed.
1.2. If the customer is not registered, then need fill in customer information form.
2.Once ustomer information submited, ask for credit card information. Waiting the credit card been confirmed.
2.1. If card valid, then need to fill in delivey information. 2.1.1. Insert order to database.
2.1.2. Update inventory.
2.2. if credit card cornfirm fail, then go back modify credit. card information.
checkout()
showCustInfo()
if registered = true
if registered = false
fillCustInfo()
<<security>>
+authenticate
- encode/decode
enterCreditCardInfo()
confirm()
if confirm = true
<<user interface>>
+ views
+ response
- frame/form
enterDelivery()
insertOrder()
crate()
updateInventory()
modifyCreditCard()
if confirm = false

Figure 5-4 Checkout sequence with aspect information analysis

5.2.3 Maintain products sequence diagram with aspects analysis

In chapter figure 4-5 shows the maintain products use case functional sequence method calls and objects involved with, but some systemic perspective information are not represented in the diagram, for example, persistency aspect, distribution aspect and performance aspect. The following aspects are detailed aspect information the diagram potentially should be represented. We did not annotate the following aspects in its diagram, because they are quite similar with previous sequence diagrams.

Aspects	Aspect details, detail properties and brief reasoning
<<Distribution>>	<<+ provided>> locate object, detailed properties can be locating products object. Application GUI component required the services.
	<<– required>> send/receive data, may use LAN connection as transport. The service provided by middleware component for example CORBA/RMI.
<<Persistency>>	<<+ provided>> data store/retrieve, like select, update, insert or delete product item. The services required by application GUI component to maintain product information. Meanwhile, the services will require other classes or database to provide same services as well.
	<<– required>> reliability of storage media, the products information need available at any time to avoid customer couldn't find information of the products i.e. need database available in 24hours 7days a week.

The persistency aspect usually provided by data manager like classes, which communicate with database, so it also required persistency aspect services from database connection. The distribution aspect provided by middleware, the aspect involved with some function calls. So developers can specify database and middleware capabilities from above aspects.

5.2.4 Maintain inventory sequence with aspects analysis

Figure 4-6 of chapter 4 illustrated the information of maintain inventory functional sequence methods invocation and associated objects. However, some systemic aspects information is not indicated in the diagram, such as, distribution aspect, persistency aspect information. The aspect information can be potentially illustrated in the sequence diagram to annotate the diagram that presents more information for developers and users. The followings are detailed aspects information that can be represented in the sequence diagram. We did not annotate the following aspects in the sequence diagram here, because the idea is same with previous diagram indication.

Aspects	Aspect details, detail properties and brief reasoning
<<Distribution>>	<<+ provided>> locate object, detailed properties can be locating products object. Application GUI component required the services.
	<<– required>> send/receive data, may use LAN connection as transport, transfer speed may higher. The service provided by middleware component for example CORBA/RMI.
<<Persistency>>	<<+ provided>> data store/retrieve, like select, update, insert or delete product item. The services required by application GUI component to maintain inventory information. Meanwhile, the services will require other classes or database to provide same services as well. Data size may limit up to 10MB.
	<<– required>> reliability of storage media, the inventory information need available at any time to avoid customer couldn't shopping sometimes i.e. need database available in 24hours 7days a week.

5.3 Component Design Specification with Aspects

Component specification and design present component's lower-level vertical decomposed functional properties. From last chapter component design, we can find that component published interfaces only give us functional perspective information and services. Component's systemic and non-functional aspect information didn't identify at all. However, the systemic and non-functional aspect information sometimes is very useful to better characterizing and indexing components. The aspect information will slice components horizontally and present higher-level component properties.

5.3.1 Deployment modelling with aspects analysis

System deployment architecture diagram illustrated system runtime processing elements, component static residential state and protocols a system used. On the other hand, there isn't any systemic aspect information captured in the diagram, such as, security aspect and persistency aspect information. The following shows the detailed aspect information could be potentially represented by deployment diagram.

Aspects	Aspect details, detail properties and brief reasoning
<<Security>>	<<+provided>> access authentication, detail properties can be password authentication mask. Web GUI component or application component required the service to verify customer or staff login.
	<<– required>> encode/decode data, can be Base 64 encode/decode scheme or using some cryptography algorithms. The services can be provided by Encoder/Decoder component or cryptography package, such as, JCE provider components.
<<User Interface>>	<<+ provided>> process/feedback views, user information presentation. Required by application GUI and web customer GUI components.
	<<+ provided>> system response, it could be detailed properties of user views, it can shared by multiple aspects. Constrained response time less than 5 seconds.
	<<– required >> form/frame, supplied by Html tag or GUI components such as swing components.
<<Distribution>>	<<+ provided>> object transfer, the distribution can use WAN/LAN connections, CORBA/RMI/SQL protocol. The service will be provided by middleware facility.
	<<– required>> send/receive data. The data transaction capabilities can be provided by database connection, such as JDBC by using SQL statement.
<<Persistency>>	<< + provided >> store/retrieve data. The services maybe used by GUI component to present or update information. Meanwhile, the server components will require database provide same services.
	<< – required >> storage media, the detail property could be file or database. May require minimum spaces great than 1 GB.

The figure 5-5 shows deployment diagram with annotated aspects detail analysis associated with each node.

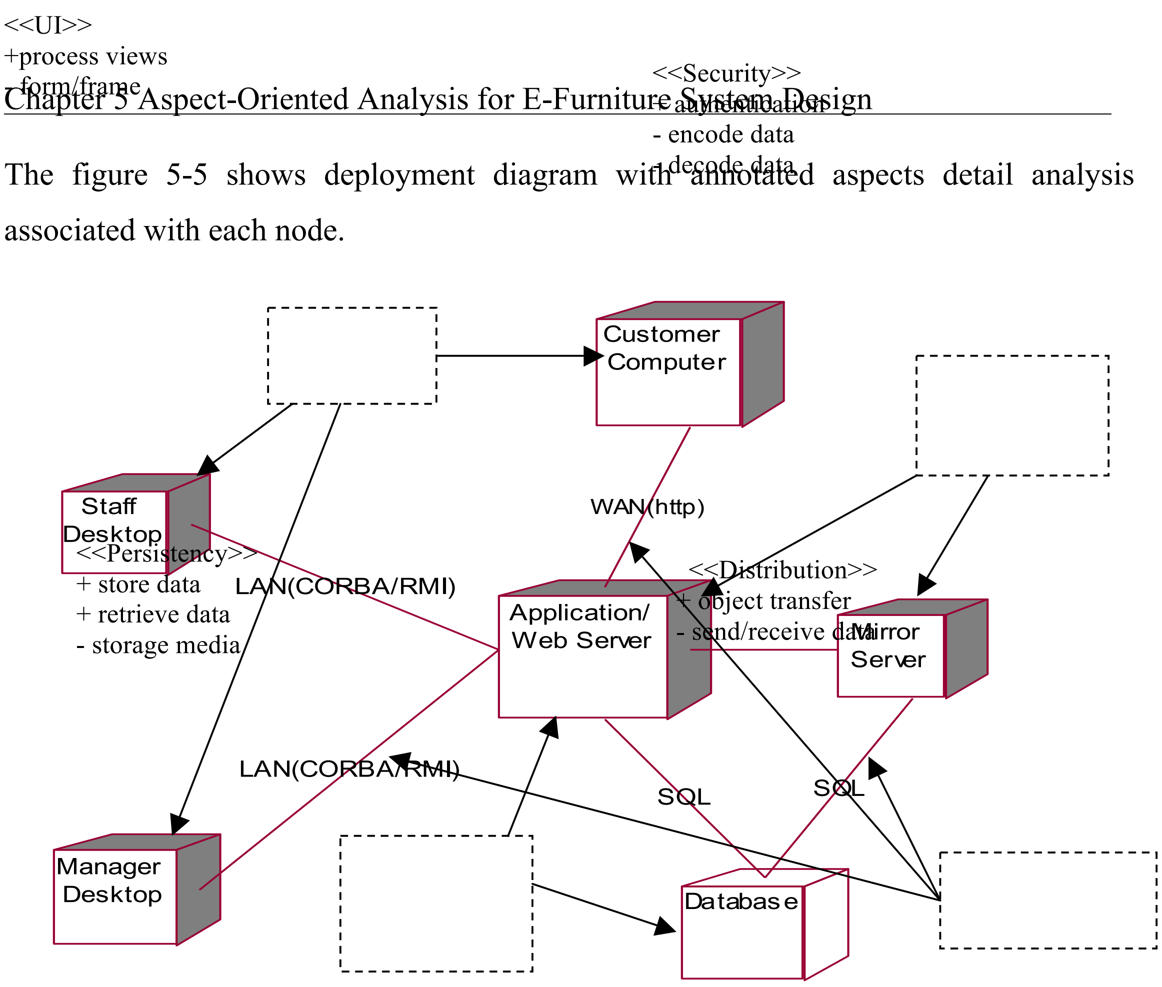


Figure 5-5 Deployment diagram with aspect information

5.3.2 Component design with aspects

Most of component design methodologies focus on lower-level functional decomposition, the Select Perspective™ approach as well. These approach designed components lack of systemic-level aspect information. This section we focus on the systemic aspect information to analyse component design and specification.

1. Customer component

As can be seen from last chapter the component design, the component's functional services are decomposed very well. It well specified those functional services and properties in details. However, systemic aspect information is not indicated, for example, security aspect, persistency aspect etc.

The following detailed aspects are potentially can be indicated clearly for the customer component to enrich the component characteristics. Figure 5-6 shows these aspect information annotated in customer component design diagram.

Aspects	Aspect details, detail properties and brief reasoning
<<Security>>	<<+provided>> access authentication, detail properties can be password authentication mask. Web GUI component may need the service to verify customer login.
	<<– required>> encode/decode data, can be Base 64 encode/decode or using some cryptography algorithms. The services can be provided by Encoder/Decoder component or cryptography package provider, such as, Bouncy Castle [32] JCE provider package.
<<Distribution>>	<<+ provided>> object transfer, the distribution can use WAN/LAN connections, CORBA/RMI/COM protocol. The service will be provided by middleware facility.
	<<– required>> send/receive data. The data transaction capabilities can be provided by database connection, such as JDBC.
	<< – required>> transaction for update customer info
<<Persistency>>	<< + provided >> store/retrieve data, detail properties can be insert, update, select and delete data. The services maybe used by web GUI component to present or update information. Meanwhile, the customer component will require database provide same services. The service can also limited data size up to 10MB.
	<< – required >> storage media, the detail property could be file or database. May require minimum spaces great than 1 GB.

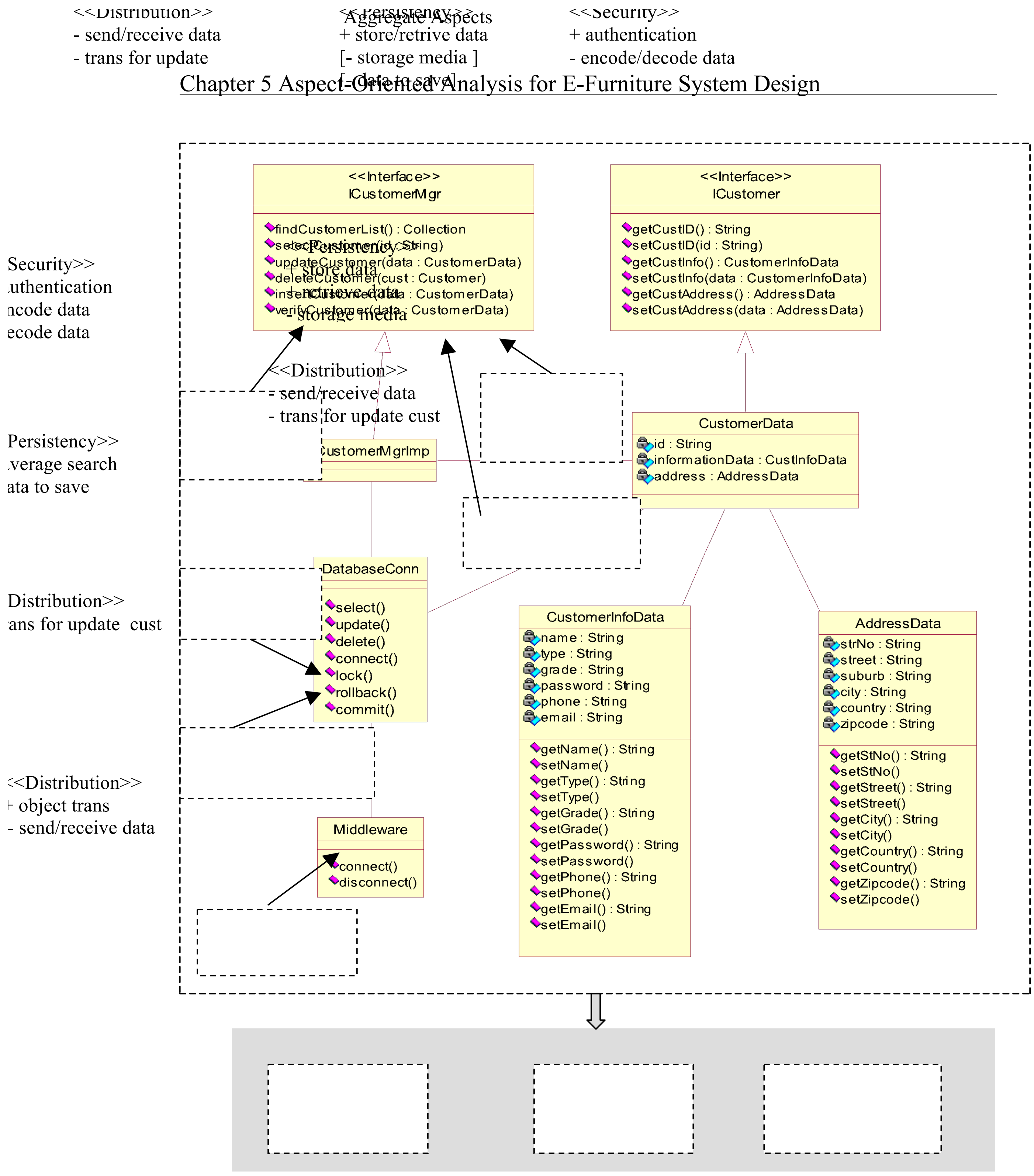


Figure 5-6 Customer component with aspect information

2. Order Component

Although the Order component (figure 4-10 of chapter 4) has been outlined functional services very well, the systemic cross-cutting issues and non-functional constraints are not indicated at all. In some point of view, the aspect information can give developers or end user a good understanding of the component, because they are characterized the component higher-level properties. The followings are the order component detailed aspect information that could be presented in its design diagram. We are not showing them in its diagram, but the idea is same with Customer component aspect annotation.

Aspects	Aspect details, detail properties and brief reasoning
<<Security>>	<<+provided>> encode/decode data, the service may use Base 64 encoding scheme or cryptographic algorithm. Web user GUI component may require the service when create orders.
	<<– required>> encode/decode data. Meanwhile, the component may require encoding scheme services as well, they are supported by cryptography components such as JCE provider components.
<<Distribution>>	<<+ provided>> object transaction, may use WAN/LAN connection as transport. The service supported by middleware facility such as CORBA/RMI/COM.
	<<– required>> send/receive data, transfer data between storage, may provided by database connection component such as JDBC.
<<Persistency>>	<<+ provided>> data store/retrieve, detailed functions will be insert, select, update and delete data. The services required by web user GUI component and staff component to create order or modify order information.
	<<– required>> data store/retrieve, the component will required the services as well to find information in storage media such as database. The services will be supported by database through SQL protocol.

3. ShoppingCart and Product component

Figure 4-11 of chapter 4 shows the component functional services that can provide to other components or users. It also shows the component required lower-level functional services. All functional services have been captured by its published interfaces. However, the component higher-level systemic aspect information is not presented yet.

The following detailed aspects were captured for the component by using our AOCE. These aspects will give developer/user a higher-level various view of the component.

Aspects	Aspect details, detail properties and brief reasoning
<<Distribution>>	<<+ provided>> locate object. Detail properties can be search products or catalog information. Web user GUI component required the services, which are supported by middleware facility such as CORBA/RMI/COM.
	<<– required>> object transaction, may use WAN/LAN connection as transport. The service provided by middleware component for example CORBA/RMI.
<<Persistency>>	<<+ provided>> data store/retrieve, detailed functions will be insert, select, update and delete data. The services required by web user GUI component and staff component to create products or maintain product information.
	<<– required>> data store/retrieve, the component will required the services as well to retrieve information in storage media such as database. The services will be supported by database and its connection component through SQL protocol.

4. Staff Component

All functional business services have been well captured for the staff component in Figure 4-12 of chapter 4. The component's properties also have been identified completely and specified clearly. However, its systemic and non-functional aspect information is not shown in the component design diagram at all. The following aspects may better characterize staff component higher-level properties.

Aspects	Aspect details, detail properties and brief reasoning
<<Security>>	<<+ provided>> access authentication, details can be password and permission group authentication. Application GUI component required the service to verify user and configure application functionalities according to permission group.
	<<– required>> encode/decode data, may use Base 64 encode scheme and cryptography algorithms. The service can be provided by cryptography components such as JCE provider components.
<<Distribution>>	<<+ provided>> locate object. Detail properties can be locate and identify staff object. Application GUI component required the services.
	<<– required>> object transaction, may use WAN/LAN connection as transport. The service provided by middleware component for example CORBA/RMI.
<<Persistency>>	<<+ provided>> data store/retrieve, detailed functions will be insert, select, update and delete data. The services required by application GUI component to create new staff or maintain staff personal information.
	<<– required>> data store/retrieve, the component will required the services as well to retrieve information in storage media such as database. The services will be supported by database and its connection component through SQL protocol.

5. Inventory Component

In chapter 4 the figure 4-13 has been clearly specified inventory component business services, which are functional decomposed behaviors. It also has shown the components encapsulated properties. However, the component system-level cross-cutting properties are not presented in its design diagram, such as distribution aspect and persistency aspect etc. The following aspects will be potentially better characterize the component systemic properties.

Aspects	Aspect details, detail properties and brief reasoning
<<Transaction>>	<<+ provided>> lock data, locking a specified record to avoid concurrent access. The service required by shopping cart component when checkout update inventory. The inventory component will require database or server component provide locking server.
	<<– required>> rollback/commit data, when there are some error occur will need rollback services or confirm service. The services will be provided by database or server components.
<<Distribution>>	<<+ provided>> locate object. Detail properties can be search stock information. Application GUI component required the services when staff maintains stock records. The services are supported by middleware facility such as CORBA/RMI/COM.
	<<– required>> object transaction, may use WAN/LAN connection as transport. The service provided by middleware component for example CORBA/RMI.
<<Persistency>>	<<+ provided>> data store/retrieve, detailed functions will be insert, select, update and delete data. Data size may limit up to 10MB. The services required by application GUI component when staff creates inventory items or maintain inventory data.
	<<– required>> data store/retrieve, the component will required the services as well to retrieve information in storage media such as database. The services will be supported by database and its connection component through SQL protocol.

5. ApplicationGUI component

The component's business functional services have been well captured in Figure 4-14 of chapter 4. The figure also shows the component state properties. However, it didn't show any information of systemic non-functional properties such as user interface aspect information. The following aspect information may illustrate ApplicationGUI component more clearly from high-level point of views.

Aspects	Aspect details, detail properties and brief reasoning
<<User Interface>>	<<+ provided>> process views, present process information and feedback information to users. The services required by application user, such as staff or manager.
	<<– required>> frame/form/panel, detailed properties can be GUI component such as JFrame and other swing components or HTML tag if use web-based interface.
<<Persistency>>	<<– required>> data store/retrieve, the component will required to retrieve information other components encapsulated such as inventory component.

6. CustomerGUI Component

Although the CustomerGUI component (figure 4-15 of chapter 4) specified lower-level business functional services in details, the systemic cross-cutting behaviors and non-functional constraints are not explicitly indicated at all. There isn't non-functional requirements been specified in the diagram such as user interface aspect. The following detailed aspect information described the component higher-level systemic characteristics. And, they potentially could be shown in the component design diagram to document the component higher-level properties.

Aspects	Aspect details, detail properties and brief reasoning
<<User Interface>>	<<+ provided>> process views, present process information and feedback information to web users. The services required by web users, such as online shopping customer.
	<<– required>> frame/form, detailed properties will be HTML tags. That can be provided by HTML and JSP components.
	<<+ provided>> response time, less than 5 seconds. Required by web users.

<<Distribution>>	<<+ provided>> locate data, detail can be search products or locate user information. The service required by online user. The component will also require other components same services, such as shopping cart and customer components.
	<<– required>> data transferring, need transport services to transfer data. It will be provided by WAN/LAN connection through middleware facilities like CORBA/RMI.
<<Persistency>>	<<– required>> data store/retrieve, the component will required to retrieve information other components encapsulated such as customer component and shopping cart component.

5.4 Summary

From component analysis and design using aspect information, we can find that each component has some aspects that may vary from different domains, and each aspect has several aspect details that are used to more precisely characterise the component associated with the aspect. Each aspect detail has some aspect “detail properties”. These particular aspects mainly focus on the systemic, crosscutting issues and non-functional constraints. These aspect details are very helpful in reasoning components by looking at its provided and required aspect details. Therefore, after analysing aspect information for object interaction modelling, deployment modelling and component specification, we can find that the system’s lower-level functional services, higher-level systemic information, and non-functional constraints are captured very well.

Moreover, the high-level aspect information will impact on component implementation, testing and deployment. The component aspects will also impact the end-user reconfiguration and indexing component repository. Developers and end-users will greatly benefit from these aspects information in understanding those components that were documented with aspect information. However, there is a trade-off for this extra work that has been done to analysis the aspect information. These issues will be discussed in the next chapter.

Chapter 6 Component Aspects Representation

The Rational Rose™ CASE tool is considered as one of the standard tools used for modelling software during analysis and design, such as use case diagrams, sequence diagrams, collaboration diagrams, state transition diagrams, class diagrams, deployment diagrams, and component diagrams etc. These diagrams will give the user a very good understanding for modelling a software system. Moreover, these diagrams can capture a software system's functionality and business logic very well. However, these tools lack elements used to indicate aspect information with the system's non-functional requirements. In this chapter, we'll illustrate how we can express aspect information in Rational Rose for sequence diagrams, collaboration diagrams, class diagrams, and deployment diagrams. These notes will give component designer a good understanding of designing and documenting components.

6.1 Sequence diagrams with aspects in Rational Rose

Sequence diagrams represent dynamic system behaviors. Most of functional invocations flow and objects involved with will be captured. Hence, we are not only can stick aspect information on objects, but also we can show dynamic aspects involved with function call.

There are two ways to indicate aspect information for objects in sequence diagrams. One is shown in Figure 6-1 that we use notes to indicate aspects for each object involved. Every object aspect information details can be found in the notes at the bottom of the object in the sequence diagrams. Then we could reason about these objects relationship by analyzing their provided and required aspects.

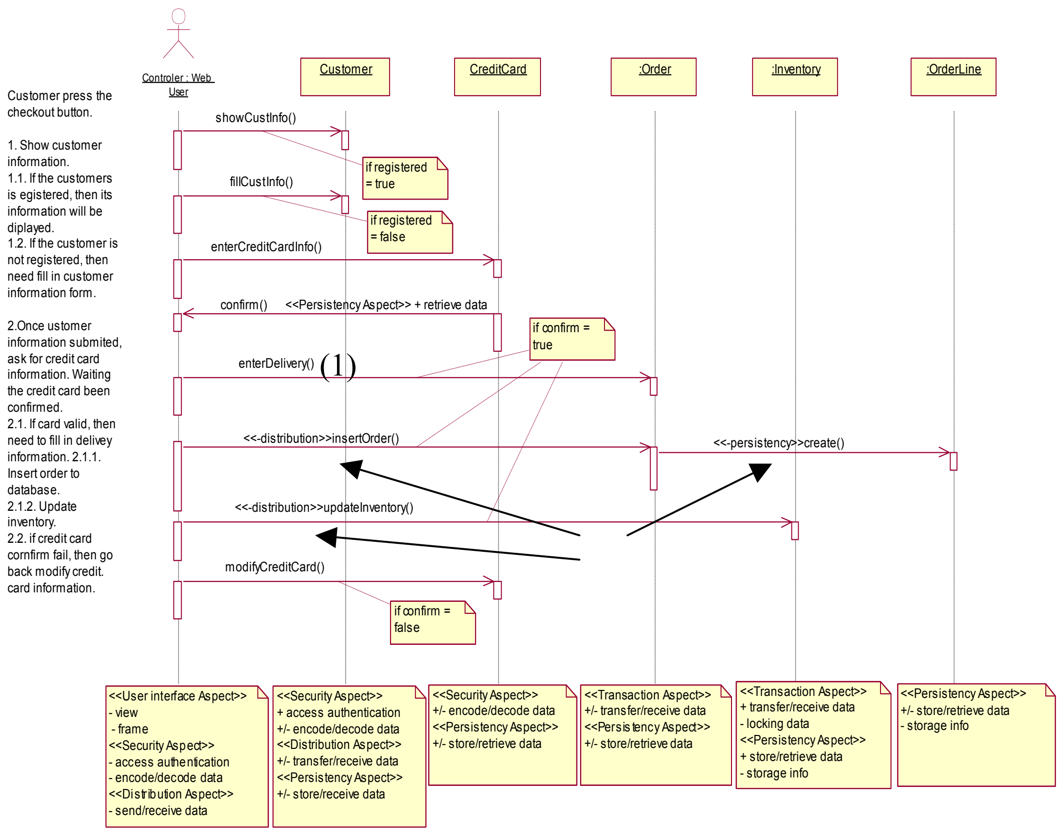
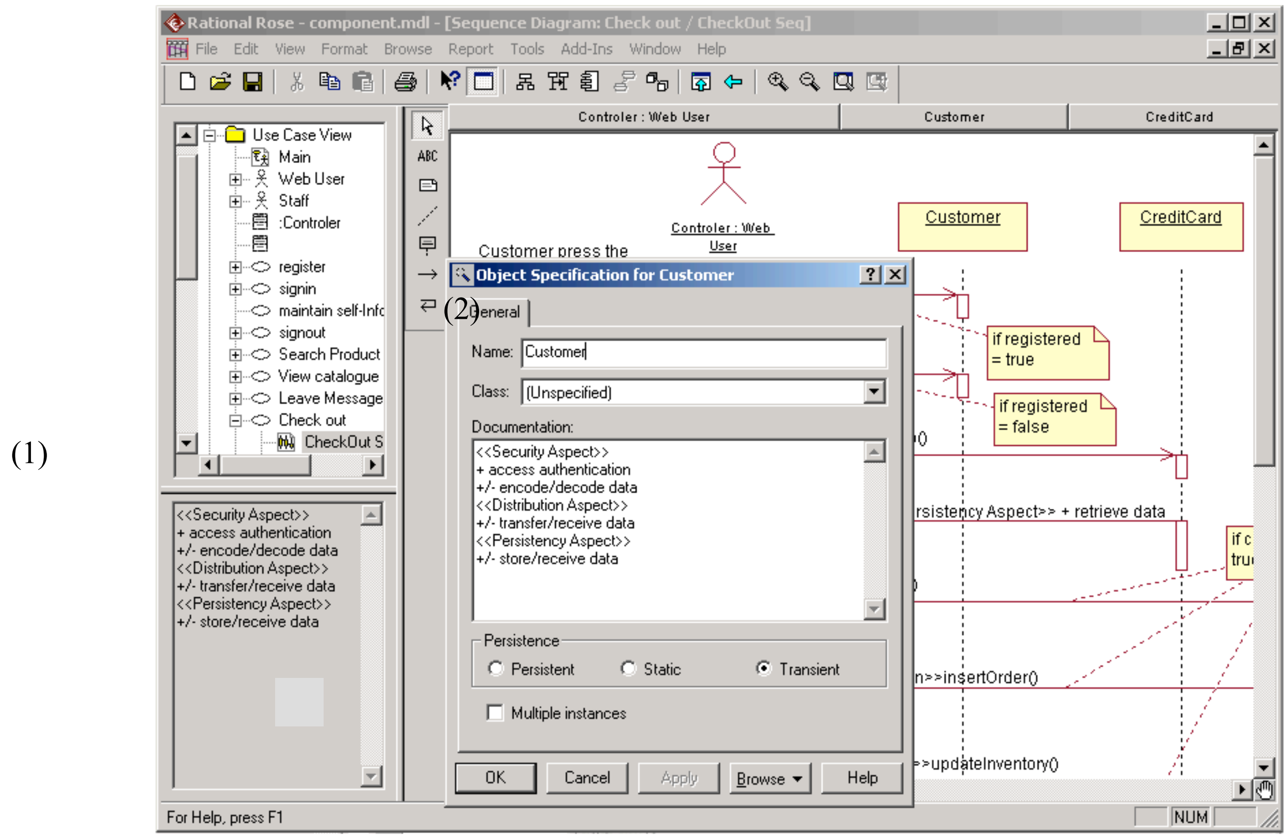

Controler : Web User
Customer
CreditCard
:Order
:Inventory
:OrderLine
Customer press the checkout button.
1. Show customer information.
1.1. If the customers is egistered, then its information will be diplayed.
1.2. If the customer is not registered, then need fill in customer information form.
2.Once ustomer information submited, ask for credit card information. Waiting the credit card been confirmed.
2.1. If card valid, then need to fill in delivey information. 2.1.1. Insert order to database.
2.1.2. Update inventory.
2.2. if credit card cornfirm fail, then go back modify credit. card information.
showCustInfo()
if registered = true
fillCustInfo()
if registered = false
enterCreditCardInfo()
confirm() <<Persistency Aspect>> + retrieve data
if confirm = true
enterDelivery()
(1)
<<-distribution>>insertOrder()
<<-persistency>>create()
<<-distribution>>updateInventory()
modifyCreditCard()
if confirm = false
<<User interface Aspect>> - view - frame <<Security Aspect>> - access authentication - encode/decode data <<Distribution Aspect>> - send/receive data
<<Security Aspect>> + access authentication +/- encode/decode data <<Distribution Aspect>> +/- transfer/receive data <<Persistency Aspect>> +/- store/receive data
<<Security Aspect>> +/- encode/decode data <<Persistency Aspect>> +/- store/retrieve data
<<Transaction Aspect>> +/- transfer/receive data <<Persistency Aspect>> +/- store/retrieve data
<<Transaction Aspect>> + transfer/receive data - locking data <<Persistency Aspect>> + store/retrieve data - storage info
<<Persistency Aspect>> +/- store/retrieve data - storage info

Figure 6-1 Checkout sequence diagram with aspect information

Another way to express the aspect information on objects is use object documentation dialog of the Rational Rose to indicate aspect information of objects. Figure 6-2 illustrated this notation. Developers can select a class and write its aspect information in the documentation window (1) of figure 6-2. We can also view and write the aspect information via the class specification dialog by right click the class then select *open specification,* the specification dialog will popup (2) of figure 6-2.

The dynamic function call involved aspects usually are transaction, distribution and persistency aspects, such as checkout sequence function call *insertOrder()* and *updateInventory()*, in which distribution aspect as involved. The *create()* function call is required persistency aspect. We can illustrate the information by using stereotype, which is shown in sequence diagram (1) of Figure 6-1. So, we can use these notations to well capture system dynamic behaviour aspects information. And we can gain more high-level understanding of a system.



<<Security Aspect>>
+ access authentication
+/- encode/decode data
<<Distribution Aspect>>
+/- transfer/receive data
<<Persistency Aspect>>
+/- store/receive data

Object Specification for Customer

General

Name: Customer

Class: (Unspecified)

Documentation:
<<Security Aspect>>
+ access authentication
+/- encode/decode data
<<Distribution Aspect>>
+/- transfer/receive data
<<Persistency Aspect>>
+/- store/receive data

Persistence: ( ) Persistent ( ) Static [✓ Transient]

[ ] Multiple instances

Figure 6-2 Sequence diagram with aspects in documentation dialog

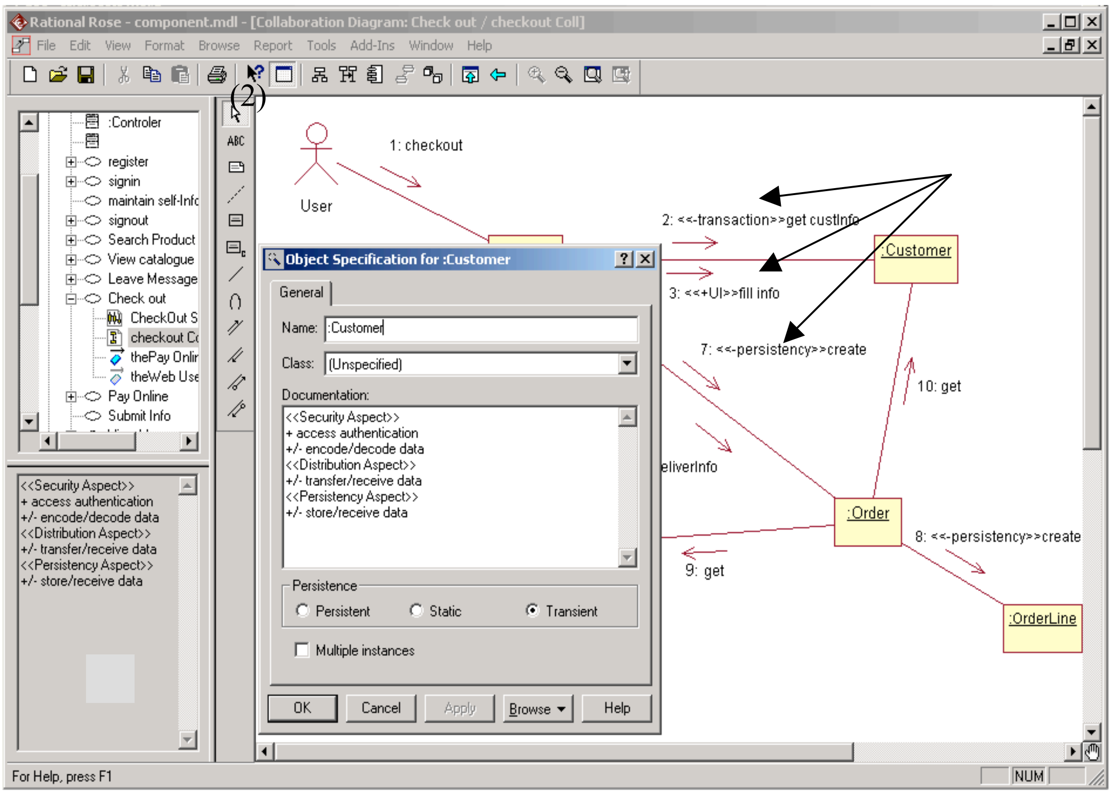
6.2 Collaboration diagrams with aspects in Rational Rose

The collaboration diagrams are different from sequence diagrams that layout is more restricted and tidy. In the collaboration diagrams, the objects layout may variety especially when a collaboration diagram involves many classes. If we use attach notes to each class, the diagram will look quite messy and hard to understand. Therefore, we adopted class documentation dialog to indicate the class aspect information and we can also use stereotypes stick to function calls. For example, when we select customer object, its detailed aspect information can be written and shown in the documentation window (1) in Figure 6-3. The user also can right click object and select open specification to edit or view the detailed aspect information like (2) in figure 6-3. To indicate dynamic function call required or provided aspects, we can use stereotypes of function call, such as *getCustInfo* method call required transaction aspect and *create*

(3)

order function required persistency aspect that is shown in (3) of Figure 6-3. So, when aspect information was shown in collaboration diagram, we can find that it will give user more systemic cross-cutting and non-functional information.

(1)



<<Security Aspect>>
+ access authentication
+/- encode/decode data
<<Distribution Aspect>>
+/- transfer/receive data
<<Persistency Aspect>>
+/- store/receive data

Name: :Customer

Class: (Unspecified)

Documentation:

<<Security Aspect>>
+ access authentication
+/- encode/decode data
<<Distribution Aspect>>
+/- transfer/receive data
<<Persistency Aspect>>
+/- store/receive data

Persistence: Persistent [✓ Transient] Static

Multiple instances ☐

Figure 6-3 Checkout Collaboration diagram with aspect information

6.3 Deployment diagrams with aspects in Rational Rose

The deployment diagrams aspect information is quite important, because deployment diagram will present system deployment architecture and system connection etc. properties. So, when aspect information was shown in the diagram, the system's properties and non-functional requirements will be quite clear. It'll be very helpful for designers to think about system architecture design by considering various constraints and cross-cutting issues.

There are three ways to illustrate aspect information for deployment diagrams in Rational Rose. The first approach to express aspect information is by attaching aspect detail notes to each deployment node and attach aspects note on connections as well. For example, *Database* will have security aspect and persistency aspect information stick on it. Moreover, the LAN or WAN connections between deployment nodes are required security aspects, such as SSL connection tunnel. This notation has been illustrated in Figure 6-4.

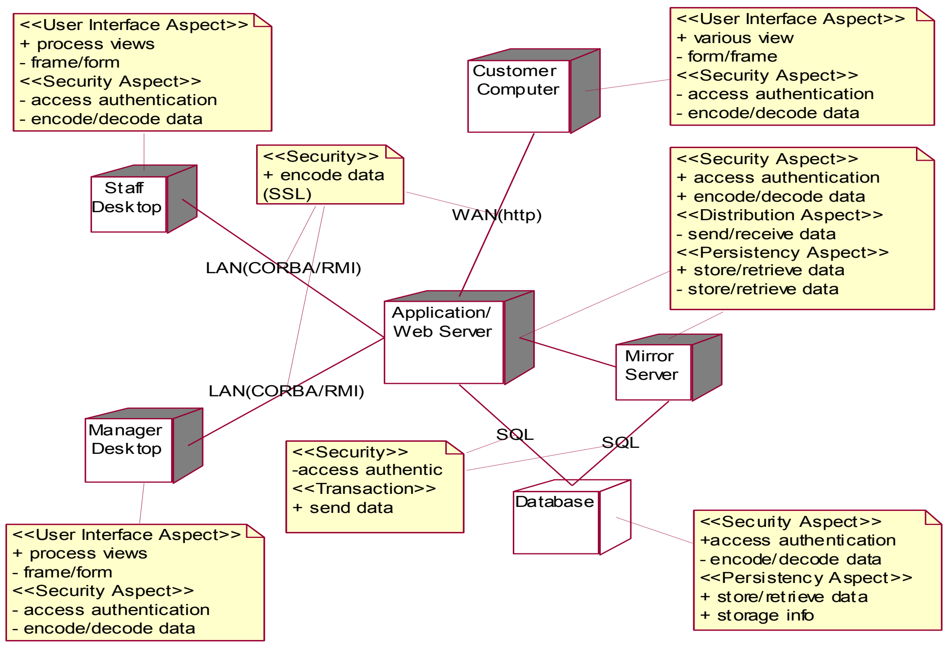
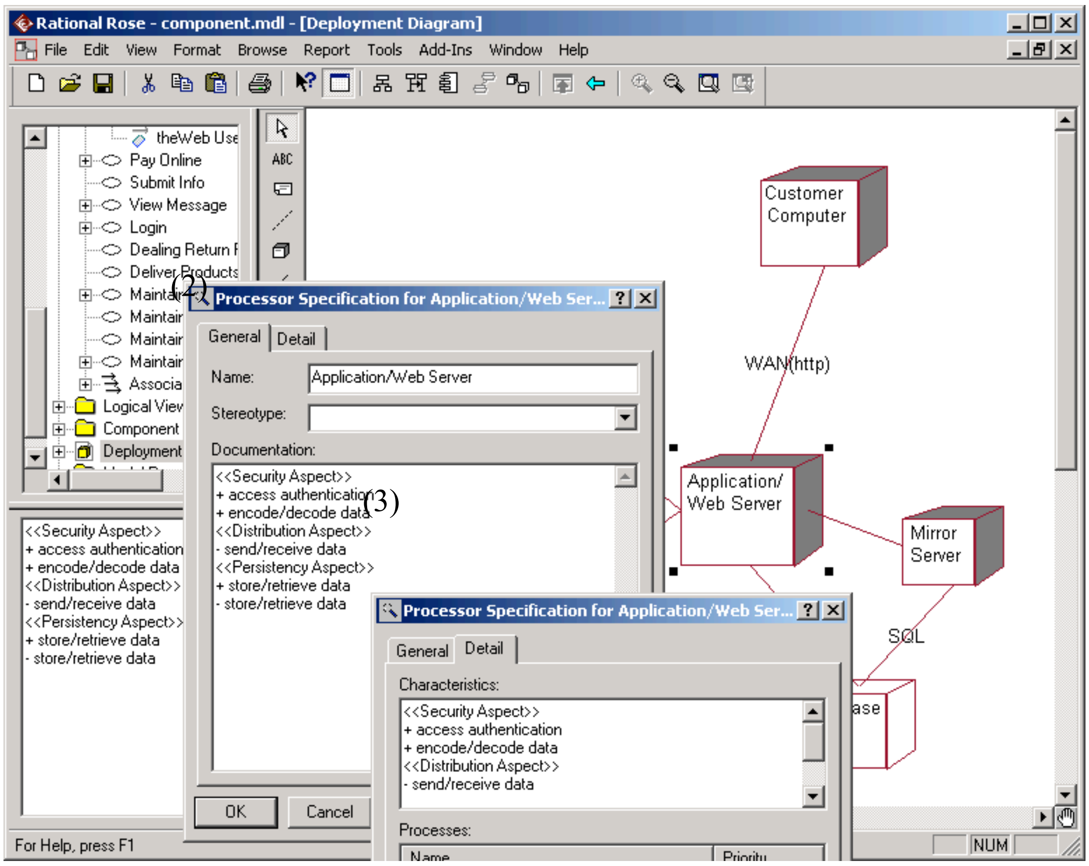


Figure 6-4 Deployment diagram with aspect information

The other approach is use the documentation window of Rational Rose provided for each deployment node. When a deployment node is selected, the aspects information can be written in processor or device documentation widow (1) in Figure 6-5. Designer could also edit or view aspect information either by selecting the deployment node to view aspect detail or right click the node selecting *open specification* use the *"General"* tab pane documentation window, which is showed in (2) Figure 6-5.

The last approach to express the aspect information is using specification "detail Characteristics" windows to show the aspect information, which is illustrated in (3) figure 6-5. The dialog window invoked by right click a processor or device and select *open specification*, then select *"Detail"* tab-pane.

(1)



Name: Application/Web Server

Stereotype:

Documentation:

<<Security Aspect>>
+ access authentication
+ encode/decode data
<<Distribution Aspect>>
- send/receive data
<<Persistency Aspect>>
+ store/retrieve data
- store/retrieve data

Characteristics:

<<Security Aspect>>
+ access authentication
+ encode/decode data
<<Distribution Aspect>>
- send/receive data

Processes:

Name	Priority

Figure 6-5 Deployment diagrams with aspects in documentation dialog

6.4 Component class diagrams with aspects in Rational Rose

Aspect information is used to capture the component high-level functional and non-functional information. So the aspect information should be found in component published interfaces and its implementation. To indicate aspect information for components class diagrams in Rational Rose tool, there are two approaches will be illustrated in the paper.

The first approach is using class diagram’s operation stereotype to indicate aspect information of the component provided or required aspects, and put these stereotyped aspect information in the component interface. For example, Customer component can provide persistency aspects and security aspects services, it also require transaction aspects, distribution aspects services. These aspects have been shown by stereotyped methods, such as *<<+Persistency>>insertCustomer()*, *<<-Security>> verifyCustomer()*, which is shown in Figure 6-6. From the customer component aspects information diagram, we can find that its non-functional aspects also indicated in its published interface, for example, required distribution aspect and encode data of security aspects. In figure 6-6, there are some explanation notes for those aspects in the component interface at the bottom. This approach is more explicit and easily to reason about the component relationships and gives component developer a high-level understanding of these components.

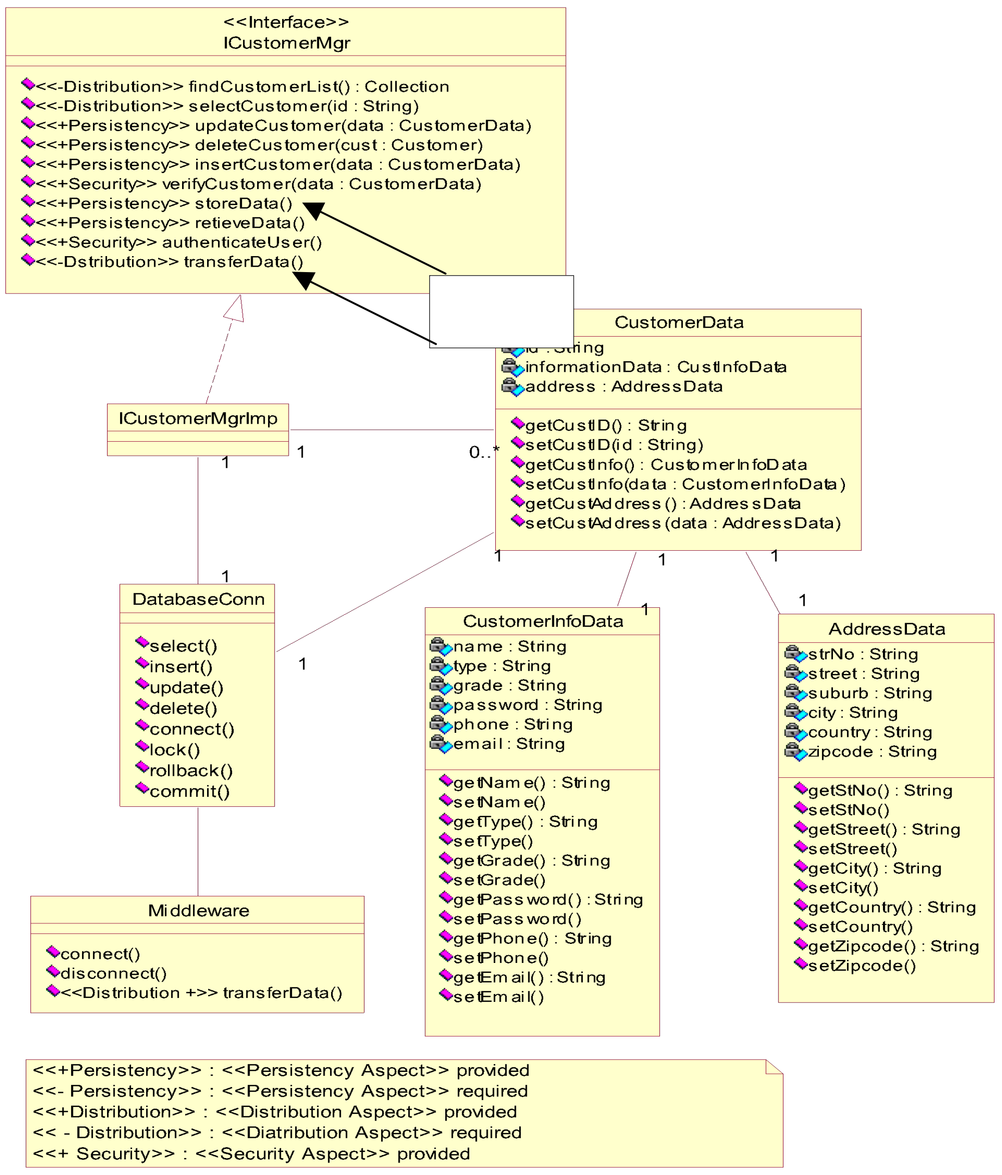
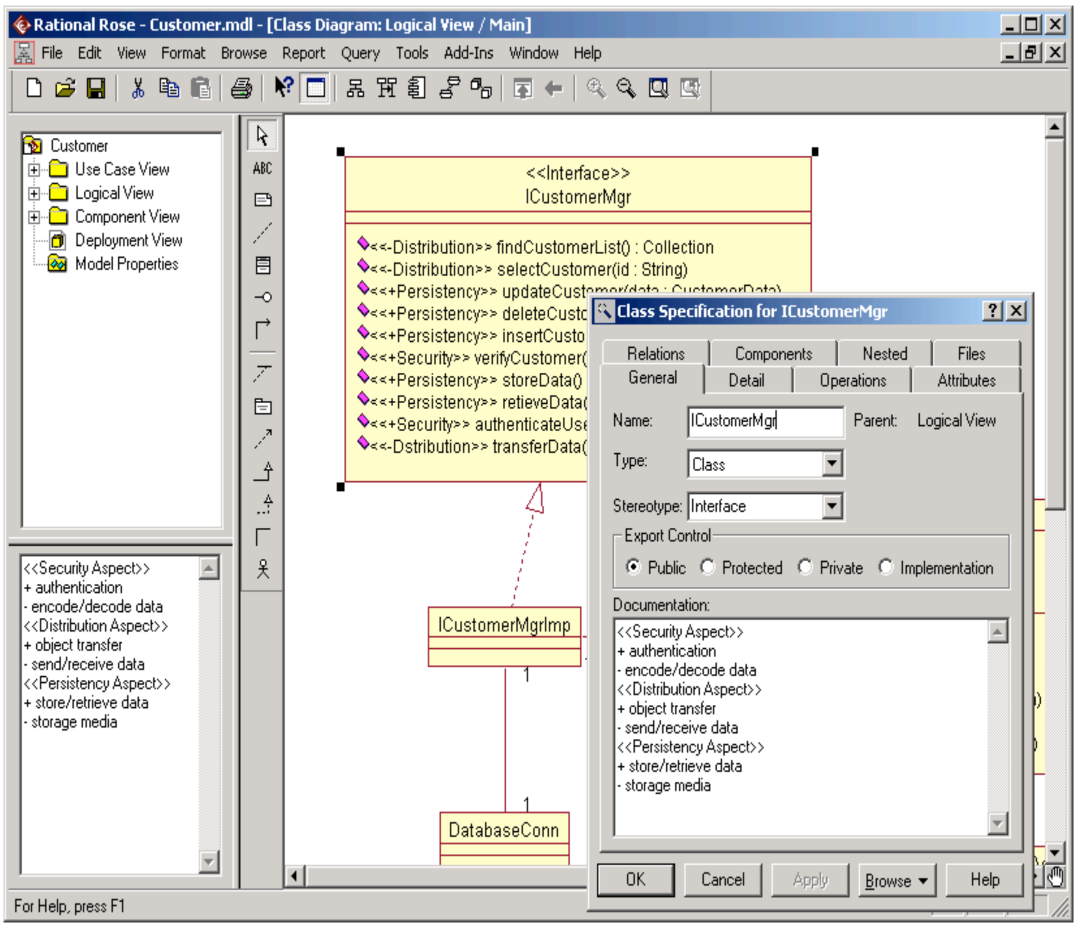

Aspect Information
<<Interface>>
ICustomerMgr
<<-Distribution>> findCustomerList() : Collection
<<-Distribution>> selectCustomer(id : String)
<<+Persistency>> updateCustomer(data : CustomerData)
<<+Persistency>> deleteCustomer(cust : Customer)
<<+Persistency>> insertCustomer(data : CustomerData)
<<+Security>> verifyCustomer(data : CustomerData)
<<+Persistency>> storeData()
<<+Persistency>> retieveData()
<<+Security>> authenticateUser()
<<-Dstribution>> transferData()
CustomerData
id : String
informationData : CustInfoData
address : AddressData
getCustID() : String
setCustID(id : String)
getCustInfo() : CustomerInfoData
setCustInfo(data : CustomerInfoData)
getCustAddress() : AddressData
setCustAddress(data : AddressData)
ICustomerMgrImp
1
1
0..*
1
1
1
1
1
1
1
1
DatabaseConn
select()
insert()
update()
delete()
connect()
lock()
rollback()
commit()
CustomerInfoData
name : String
type : String
grade : String
password : String
phone : String
email : String
getName() : String
setName()
getType() : String
setType()
getGrade() : String
setGrade()
getPassword() : String
setPassword()
getPhone() : String
setPhone()
getEmail() : String
setEmail()
AddressData
strNo : String
street : String
suburb : String
city : String
country : String
zipcode : String
getStNo() : String
setStNo()
getStreet() : String
setStreet()
getCity() : String
setCity()
getCountry() : String
setCountry()
getZipcode() : String
setZipcode()
Middleware
connect()
disconnect()
<<Distribution +>> transferData()
<<+Persistency>> : <<Persistency Aspect>> provided
<<- Persistency>> : <<Persistency Aspect>> required
<<+Distribution>> : <<Distribution Aspect>> provided
<< - Distribution>> : <<Diatribution Aspect>> required
<<+ Security>> : <<Security Aspect>> provided

Figure 6-6 Customer Component with aspect information

Another approach is using the Rational Rose documentation window of classes and interfaces in the logical class diagram. When an interface of the component is selected, its aspects information can be written or edited in its documentation window, which is shown in (1) of Figure 6-7. And user can also right click the component interface select *open specification* to show or modify its aspect information that is illustrated in (2) of Figure 6-7. These two approaches documented component systemic functional and non-functional properties more explicit and complete. The component design diagram with aspects information will be very helpful for component implementation, testing, component deployment and end user understanding. (2)

(1)



<<-Distribution>> findCustomerList() : Collection
<<-Distribution>> selectCustomer(id : String)
<<+Persistency>> updateCustomer(data : CustomerData)
<<+Persistency>> deleteCustc
<<+Persistency>> insertCusto
<<+Security>> verifyCustomer(
<<+Persistency>> storeData()
<<+Persistency>> retieveData(
<<+Security>> authenticateUse
<<-Dstribution>> transferData(

Name: ICustomerMgr Parent: Logical View
Type: Class
Stereotype: Interface
Export Control: [✓ Public] Protected Private Implementation

Documentation:

<<Security Aspect>>
+ authentication
- encode/decode data
<<Distribution Aspect>>
+ object transfer
- send/receive data
<<Persistency Aspect>>
+ store/retrieve data
- storage media

Figure 6-7 Customer component with aspects in documentation window

6.5 Component architecture with aspects in Rational Rose

In Rational Rose, there are not many elements that can be use to annotate components. The description notes will be the best way to indicate component aspects in component architecture diagram. Therefore, the first approach we used here to illustrate the aspect information in the component architecture diagram is using notes to document aspects information. This approach explicitly illustrated the aspect information provided and required and also can be very easily reason about components relationships. For example, the *ApplicationGUI* component required user interface aspects such as frame etc. the *javax.swing* components will provide these aspects for it. Moreover, the component also required distribution aspects, and middleware component will provide distribution aspect services. Therefore these components relationship have been reasoned about from these aspects. The e-furniture system components architecture detailed aspects information has been shown in Figure 6-7.

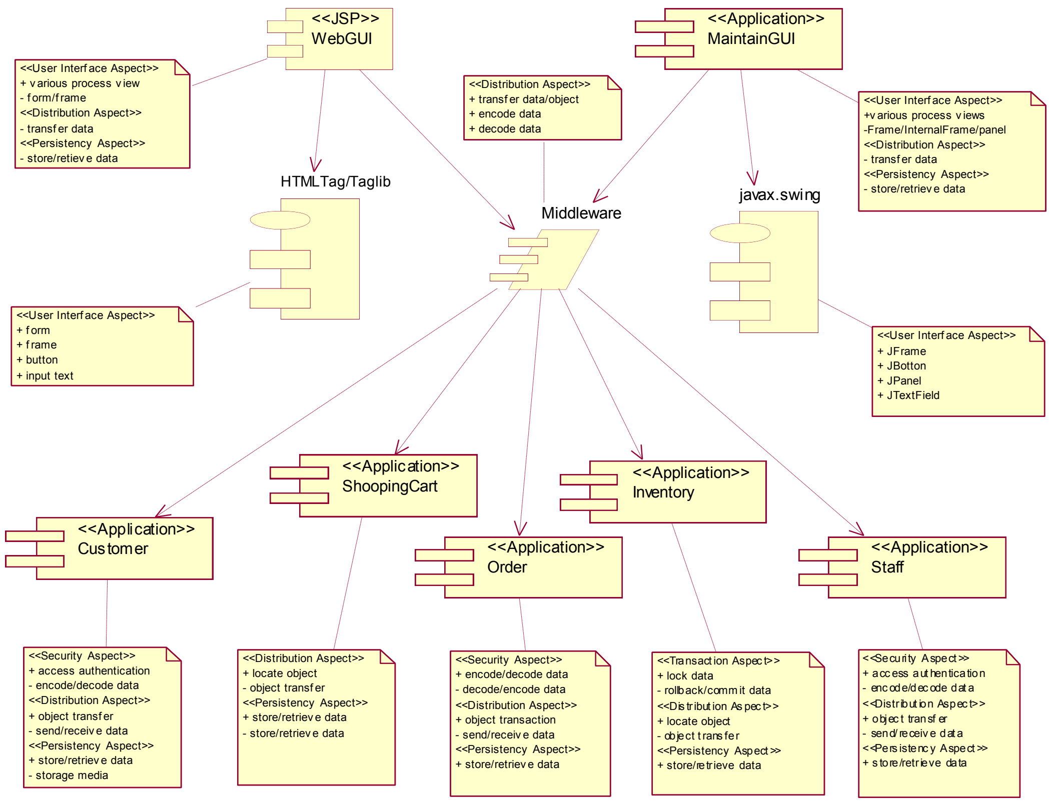
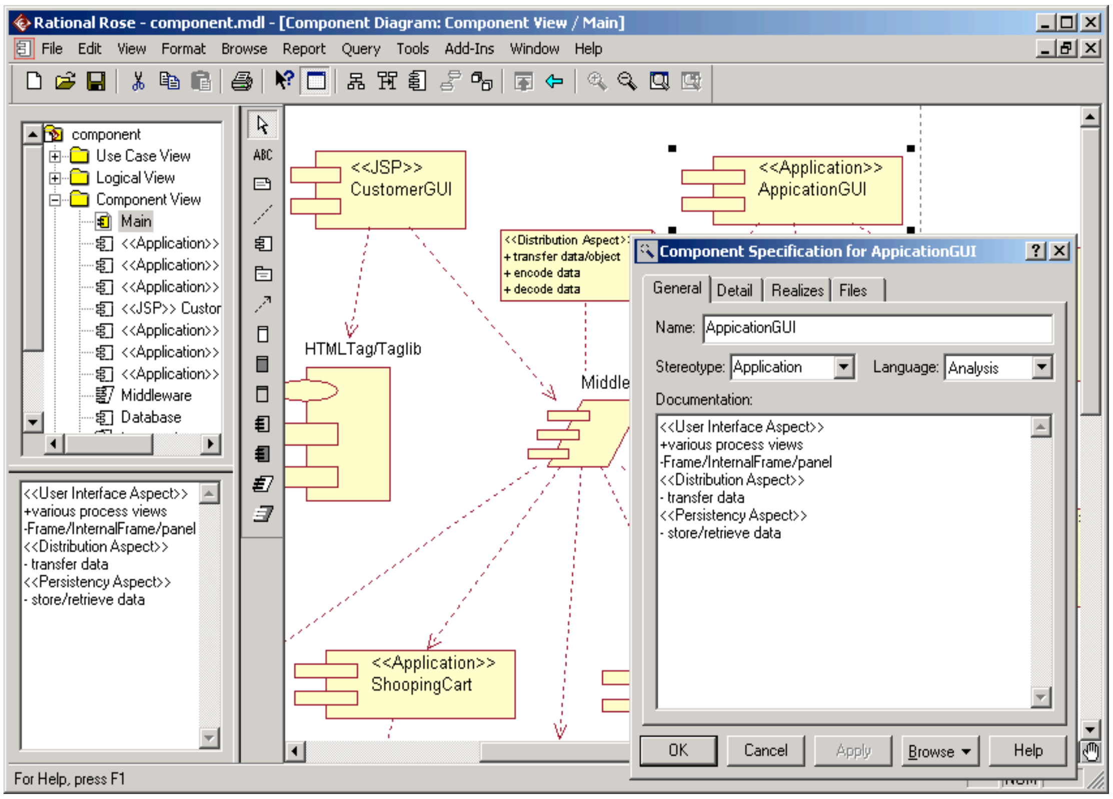

<<JSP>> WebGUI
<<Application>> MaintainGUI
<<User Interface Aspect>> + various process view - form/frame <<Distribution Aspect>> - transfer data <<Persistency Aspect>> - store/retieve data
<<Distribution Aspect>> + transfer data/object + encode data + decode data
<<User Interface Aspect>> +various process views -Frame/InternalFrame/panel <<Distribution Aspect>> - transfer data <<Persistency Aspect>> - store/retrieve data
HTMLTag/Taglib
Middleware
javax.swing
<<User Interface Aspect>> + form + frame + button + input text
<<User Interface Aspect>> + JFrame + JBotton + JPanel + JTextField
<<Application>> ShoopingCart
<<Application>> Inventory
<<Application>> Customer
<<Application>> Order
<<Application>> Staff
<<Security Aspect>> + access authentication - encode/decode data <<Distribution Aspect>> + object transfer - send/receive data <<Persistency Aspect>> + store/retrieve data - storage media
<<Distribution Aspect>> + locate object - object transfer <<Persistency Aspect>> + store/retrieve data - store/retrieve data
<<Security Aspect>> + encode/decode data - decode/encode data <<Distribution Aspect>> + object transaction - send/receive data <<Persistency Aspect>> + store/retrieve data
<<Transaction Aspect>> + lock data - rollback/commit data <<Distribution Aspect>> + locate object - object transfer <<Persistency Aspect>> + store/retrieve data
<<Security Aspect>> + access authentication - encode/decode data <<Distribution Aspect>> + object transfer - send/receive data <<Persistency Aspect>> + store/retrieve data

Figure 6-8 Component architecture diagram with aspects in notes

However, when there are too many components/packages in a system, the diagram will look not very clear if aspects information attached with each component. In this circumstance, component designers can also use the other approach, which is documenting aspect information in the component documentation window (1) in Figure 6-9 illustrated. These aspects information can also be viewed or modified in component specification window by right click the component and select *open specification* in the (2) *general* tabbed pane, which is shown in (2) of Figure 6-9.

(1) These two approaches illustrated component aspects information in system components architecture diagram explicitly. The former one will give designer or implement coder a direct and clearly information of component high-level systemic requirements. Although the later approach hides the aspects information in documentation window, designer could also grasp these components high-level information from it.



<<User Interface Aspect>>
+various process views
-Frame/InternalFrame/panel
<<Distribution Aspect>>
- transfer data
<<Persistency Aspect>>
- store/retrieve data

Name: ApicationGUI
Stereotype: Application
Language: Analysis
Documentation:
<<User Interface Aspect>>
+various process views
-Frame/InternalFrame/panel
<<Distribution Aspect>>
- transfer data
<<Persistency Aspect>>
- store/retrieve data

Figure 6-9 Component architecture with aspects in documentation window

6.6 Comparison of Component with and without Aspects

Like the paper mentioned in chapter 2, there are many technologies that have illustrated component based development methods. They mainly focus on functional decomposition of requirements into objects and components. At the component design stage, these technologies focus on detailed component interface design and service implementation. As a result, these components are characterized as low-level component interfaces and properties. They lack systemic high-level component properties and non-functional cross-cutting information. This creates difficulties in understanding components for both components developers and end users. In some systems, end users themselves may need to extend the system environment. Those difficulties in understanding and capturing components made the work even hard, and even more difficult to interact with third-party components when the third-party components are also lacking high-level characteristics.

By using AOCE technology, the analyzed, designed, and implemented components will have more systemic high-level properties, and even systemic non-functional constraints are found in component properties and interfaces. AOCE technology can also present component characteristics aspect information at different analysis and design level. More importantly, the aspect information can be documented in different UML diagrams by using stereotypes and annotations. These notations provide developers high-level abstractions of components characteristics in documentations, which will document components more completely and richly. For example, this chapter has illustrated aspect information with a deployment diagram, which gives user systemic higher-level aspect information. Whereas, sequential and collaboration diagrams give developers low-level systemic and crosscutting aspect information. Component class diagrams can give developers both low-level and high-level component characteristics whilst presenting non-functional properties. Aspect information can be used to reason about components inner relationships, and in grouping and indexing components. These characteristics increase components reusability and reconfiguration.

The main advantages aspect-oriented of component engineering are better characterization of component requirements and design, providing extra richness of multiple perspectives into components, encouraging implementation of better run-time configuration and de-coupled components communication, and dynamic access to detailed component properties. Developers can obtain different viewpoints on component capabilities by using AOCE. Aspect-based perspectives encourage more flexible coupling, runtime configuration and dynamic deployment strategies in the design and implementation stages.

AOCE methodology suggests added components complexity. This requires developers to design components using various aspects, specifying provided and required aspect details, and reasoning about component interaction with each aspect. Therefore, there is a trade-off between this extra work and AOCE benefits. Some complex systems require enhanced reusable, re-configurable, and understandable components, AOCE is worth these extra specification and reasoning efforts. In some systems, the technology may be less effective due to problems identifying suitable aspects and the lack of tools support [22].

6.7 Summary

Rational Rose™ has become one of the standard CASE tools used for modelling software system analysis and design. Those modelling diagrams act as critical roles in software development life cycles. They are also important documentation of software system developing. Hence, in this chapter we illustrated how aspect information has been shown in Rational Rose tools. Those notations and approaches are very helpful for applying AOCE methodology in component-based development (CBD) for software systems by using standard tools rather than to design special tools support. Although we could design specific CASE tools to support AOCE, these tools will be quite limited in other usages and will be hard to use by designers/developers. However, most of software designers/developers are more familiar with the Rational Rose tool, so they can use these approaches illustrated in this chapter to capture systemic aspects of components, and grasp high-level characteristics of components in order to put them in Rational Rose diagrams. These diagrams documented with aspect information will be of benefit to component designers and implementation coders. This trade-off of extra effort for the benefit of using AOCE needs to be considered by component developers.

Chapter 7 EJB and JSP Components Design and Implementation

The Enterprise JavaBeans™ (EJB) technology mapped AOCE reasonably well. Therefore, we choose the technology to implement our example the e-furniture system with aspect services. This chapter will briefly introduce the EJB technology and map the e-furniture system components to EJB beans or JSP components. The chapter will also briefly discuss the system database design, and the whole system's implementation and show some screen shots.

7.1 Enterprise Beans – the Server Components Design

Enterprise Java Beans are server components, which are implemented in the Java programming language. These enterprise beans contain the business logic of the application. There are three types of enterprise beans in EJB 2.0 specifications [33], which include session beans, entity beans and message-driven beans. However, in the E-Furniture system, we just used two types of enterprise beans - Session Beans and Entity Beans, which are built on Java 2 platform, Enterprise Edition (J2EE 1.2.1). The tools used for deploy server components are Sun Microsystems J2EE Deploytool comes with J2EE 1.2.1. The systems implementation, coding and compiling is used Borland JBuilder 4.0 as the IDE tools. This made the project prototype more quickly and efficiently.

7.1.1 Session Beans Component

A session bean is a representation of a client in the J2EE server. A client communicates with J2EE server by invoking the methods that belong to an enterprise session bean. A session bean communicates with client and can be thought of as an extension of the client. Each session bean can have only one client. When the client terminates, its corresponding session bean also terminates. Therefore, a session bean is transient, or non-persistent [5]. Session beans can also provide transaction aspects services for other enterprise beans, and can require transaction aspects from other enterprise beans to collaborate. For example, in the e-furniture system EJB design *ShoppingCart* component will provide transaction aspects for *Category* component, and it also requires *Category* and *Inventory* beans transaction services as well to finish some functions such as checkout. For the EJB implementation of the system, EJB container will manage session bean's transaction aspects. So, session bean component implementation is not necessary to add extra code to manage transaction aspects.

By reviewing E-Furniture system component design, we can find that there is several components can be mapped to session beans, such as, shopping cart, category and email components. These components can provide transaction aspects for other components, and also will require transaction aspects from others for some services. Most of client requests for online shopping will invoke these component business methods to communicate with other enterprise beans to utilize transaction aspects of the system.

And furthermore, the customer component can also separate its functionality by extract a customer session bean and a customer account entity bean to map the component in EJB framework. The customer session bean will mainly provide transaction and distribution services for account component and order component. The Figure 7-1 is the example of shopping cart and category (mapped from shopping cart and products component design of chapter 4) session bean design diagrams in EJB architecture, other session bean components mapped to EJB design frameworks can be found in Appendix II.

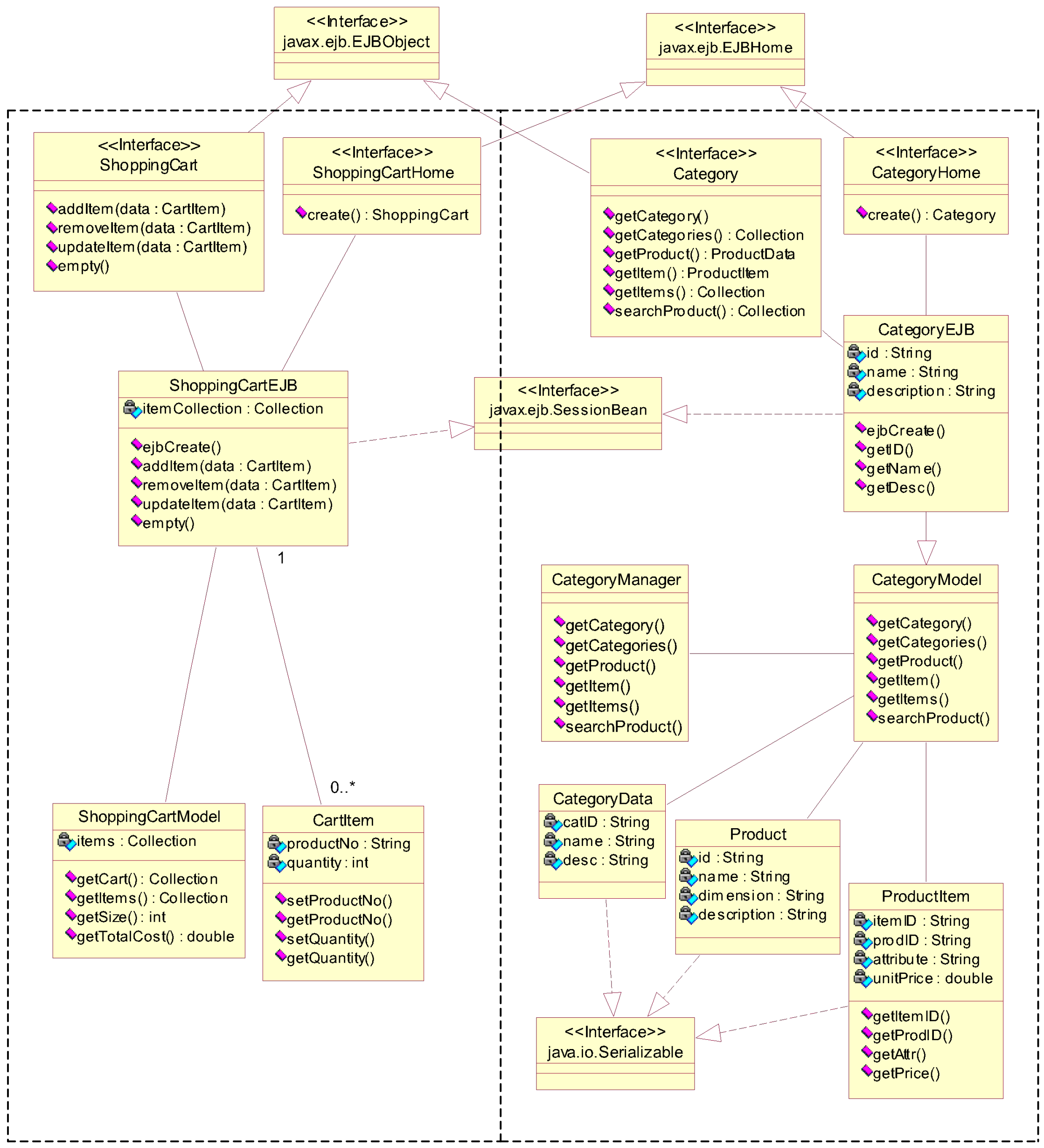


Figure 7-1 ShoppingCart and Category component EJB design diagram

7.1.2 Entity Beans Component

An entity bean is a representation of a business object in a persistent storage mechanism such as a database. An entity bean's information can be stored as a raw in a relational database table, but it does not have to store in a relational database. It could be also stored in an object-oriented database, a legacy system, a file or some other storage mechanism [5].

Component persistency aspects are well mapped to entity beans implementation. The entity bean components will require data storage/retrieving services from storage media and can also provide storage/retrieving of persistency aspects services for other components. The persistency aspects of an entity bean can be managed by either the entity bean itself that provide persistency management, or by the EJB container, then the bean will require persistency managing. If use bean-managed persistence, developers have to write data access code in the entity bean to implement persistency aspects in code, such as use SQL command to access a relational database via JDBC. However, container-managed persistence is that the EJB container handles data persistency aspects automatically. This is the EJB container provided aspect services.

In the E-Furniture system, we used *Cloudscape* relational database as the storage mechanism for the entity beans data, and we adopted bean-managed persistence, which means that the system's entity beans have been designed and implemented using SQL and JDBC connection to deal with persistency aspects of entity bean components.

From the system component design we can find that most of the e-furniture system components will be mapped as entity beans. These components are customer account, order, inventory and staff components. The Figure 7-2 shows an example of staff component EJB entity bean structure; other components EJB entity beans design diagrams can be found in Appendix II.

As can be seen in figure 7-2, the staff component defined two interfaces, *Staff* remote interface and *StaffHome* interface, which extends *EJBObject* and *EJBHome* interfaces of EJB library interfaces respectively. The *EJBObject* and *EJBHome* will give distribution aspects service support. *StaffEJB* class implements EJB *EntityBean* interface, and data persistency aspect services were handled by data access object *StaffDAO* class to communicate with database via Java SQL command and JDBC connection. Other helper classes are used for encapsulate relevant staff information in the database. The component will provide services via *Staff* remote and *StaffHome* interfaces include aspect services.

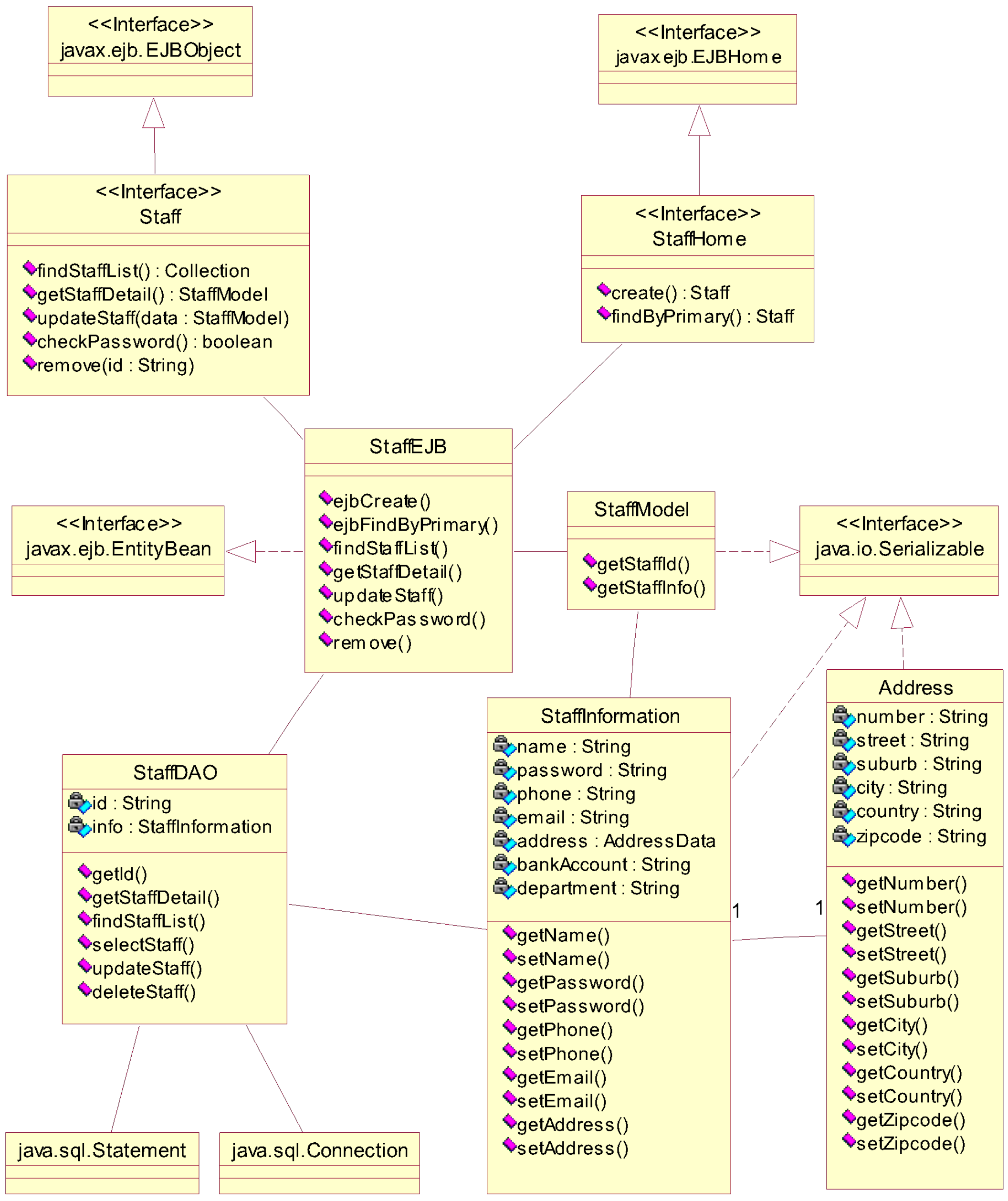


Figure 7-2 Staff component EJB diagram

7.1.3 Enterprise Bean Implementations

From the enterprise beans design and services architecture we can find that enterprise beans are using framework design patterns. Each of bean components has to either implement EJB *SessionBean* or *EntityBean* interfaces, and also have to define all the business methods in remote interface that client may invoke and bean creation methods in home interface that a client may invoke as well. The two interfaces will be published to provide services to other component and client. Meanwhile, the two interfaces will encapsulate some aspects services as well, such as transaction aspect services and persistency services.

A remote interface extends the *EJBObject* interface and will provide transaction aspects services for the AOCE design. When the remote interface mapped into EJB bean class, the component transaction aspects will be implemented in the bean class and managed by EJB container. The method defined in remote interface must to follow some rules [5]:

- Each method in the remote interface must match a method implement in the enterprise bean class. For example, method *addItem(CartItem)* in *ShoppingCart* remote interface, there is a exactly matched method *addItem(CartItem)* in *ShoppingCartEJB* class.
- The signature of the methods in the remote interface must be identical to the signature of the corresponding methods in the enterprise bean class.

Whereas, home interfaces can provide persistency aspect services in some situation, such as entity bean home interface creation method. EJB bean home interface has to extend *EJBHome* interface of the EJB library, in which defined methods also have to follow some rules:

- Every methods in the home interface must have a corresponding *ejbXXX()* method in enterprise bean class, for example, method *findByPrimary()* in *StaffHome* interface there is a corresponding method *ejbFindByPrimary()* in *StaffEJB* class.
- The number and type of arguments in a *create()* method must match those of its corresponding *ejbCreate()* method.
- A *create()* method return type of the remote interface of the enterprise bean, but an *ejbCreate()* method return void.

The EJB container manages above two interfaces type objects, and it can provide distribution and security aspect services.

By applying design framework and these rules, we designed and implemented the system EJB components and successfully deployed these components onto J2EE server by using Sun's J2EE server and deploytool. In next chapter we'll discuss how these components functionality and some of aspects information have been tested.

7.2 Database Design

In the e-furniture system, we designed a relational database for the systems persistence information storage, which are some of the system components persistency aspects required. Although we used MS Access database tool designed the database Entity-Relationship-Diagram (ERD) that is shown in Figure 7-3, when we implemented the system we used cloudscape database, which is more compatible with J2EE. From the ERD diagram we can find that the database table are not formalized, because some of derived fields are still in the database table, such as, total price of order table. This design is the consideration of some components performance aspect required. This can be increasing some entity beans performance. The Customer and Staff tables are linked to a password table respectively. This is because the two components security aspects required. Then the two password tables can be stored in different more secure host or the realm of system.

The following are explanation of the database table two main relationships:

1. Category – Product – Item – Inventory

   Category table links to *Product* table by provide a foreign key constraints *cat_id* in the *Product* table. That means all products belong to certain category. *Product* table links to *Item* table is by put a foreign key constraints *product_id* in *Item* table. This means every item belongs to certain product. So, we can use the item table to classify same product and different attributes, such as round table with different color. *Inventory* table used to record products quantity and stock information. Its just keep an *item_id* foreign key constraint link to item table, then we could find any information about the item and its details.

2. Customer – LineItem – Order – Item

Customer and *Order* table are many to many relationships, so we create an association entity *LineItem* table, which keeps *customer_id* and *order_id* as foreign key constraints to link these two tables. In the *LineItem* table has an item_id foreign key constraint as well to like to item table. So, we could find any information of a customer ordered products.

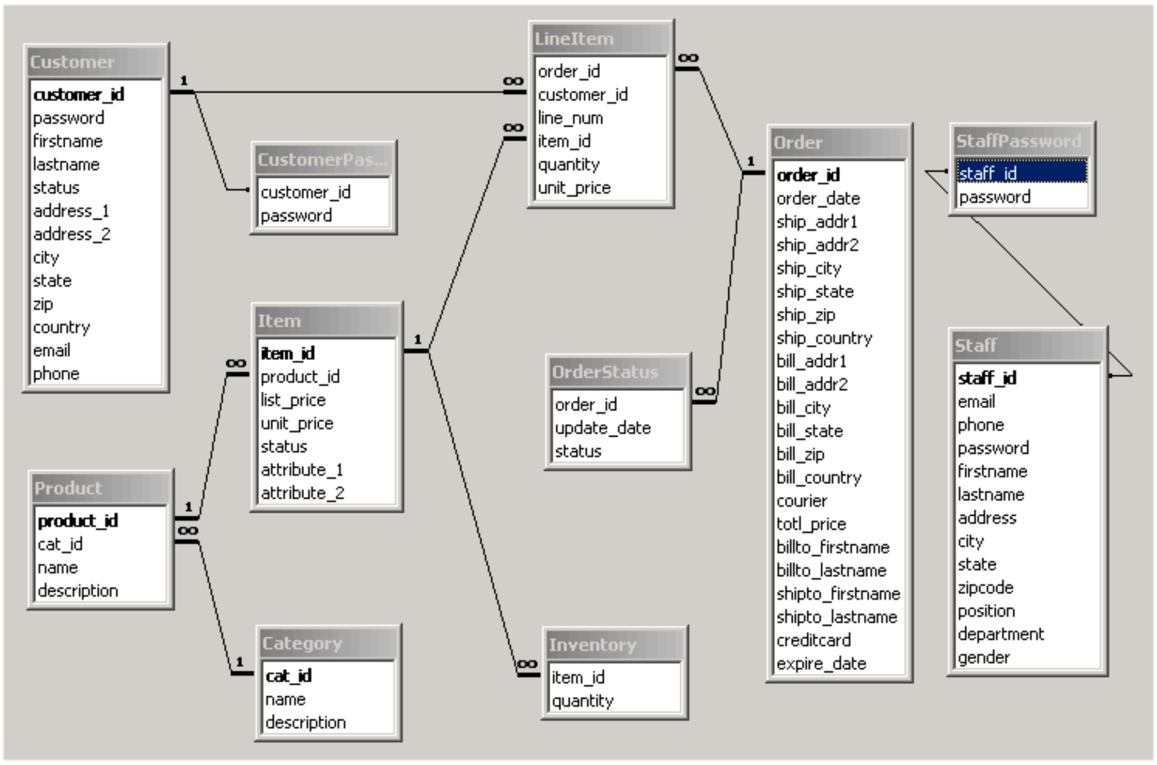


Figure 7-3 E-Furniture System database ERD diagram

7.3 Data Maintenance Application Implementation

The E-Furniture systems back-end maintaining sub-system was designed to manage inventory, order and personal information of staff for the enterprise. The sub-system designed for administrators/managers of the enterprise using. So, we implemented the maintaining system as a Java Swing components-based application system, which is comprised by login component, inventory maintain component, order maintain component and staff maintain component. In implementation stage, we also realized that internationalization (user interface aspects required) is possible and important for the system. Therefore, we implemented the system as multiple language supported by utilizing Java resource bundle components. When the system need to be deployed to non-English users, we can just translate resources bundle properties file to the users language, there isn't any works have to be done for the source code. This provided the application some commercial features and easily to deployed as an international application. When we analysis these components user interface aspects, we can find that these components required multi-language supported user interface aspects. And furthermore, when we use resources bundle properties, these component can provide different views of user interface aspects.

These data maintenance components all required GUI components to provide user interface aspects for clients. Therefore, we implement the main application component by using a *JFrame* to provide main views and sub-maintenance components by using *JInternalFrame* to provide other processing views. And we plugged in those sub-components into the main application frame. These sub-maintenance components user interface aspect services can be invoked inside the main frame and displayed as an internal frame application that is very convenient to use. The Figure 7-4 shows the application main user interface.

All the functionalities of the application were placed on a menu bar. The tool bar provided quick access of each application and helps. The menu bar and tool bar provide user interface aspects that the component required. When users first time launch the application, the splash window and login dialog will appear in the center of screen, which are shows in Figure 7-5. The login dialog required authentication of security aspect services from server component to authenticate the user. If the user provided correct user name and password, then the main application frame will appear, and the user could access some of the system functionalities according to its permissions and some of the functions may be disabled due to the user's limited permission. If login failed, the user couldn't launch main application and has to try login again. If a user logged into the system later logged out, the system all functionalities are disabled except login and help. So other users couldn't do anything if the application not logged in. This is security aspect of the main application component provided. Therefore, the data maintenance application not only required and provided user interface aspects information, but also required and provided security aspects information.

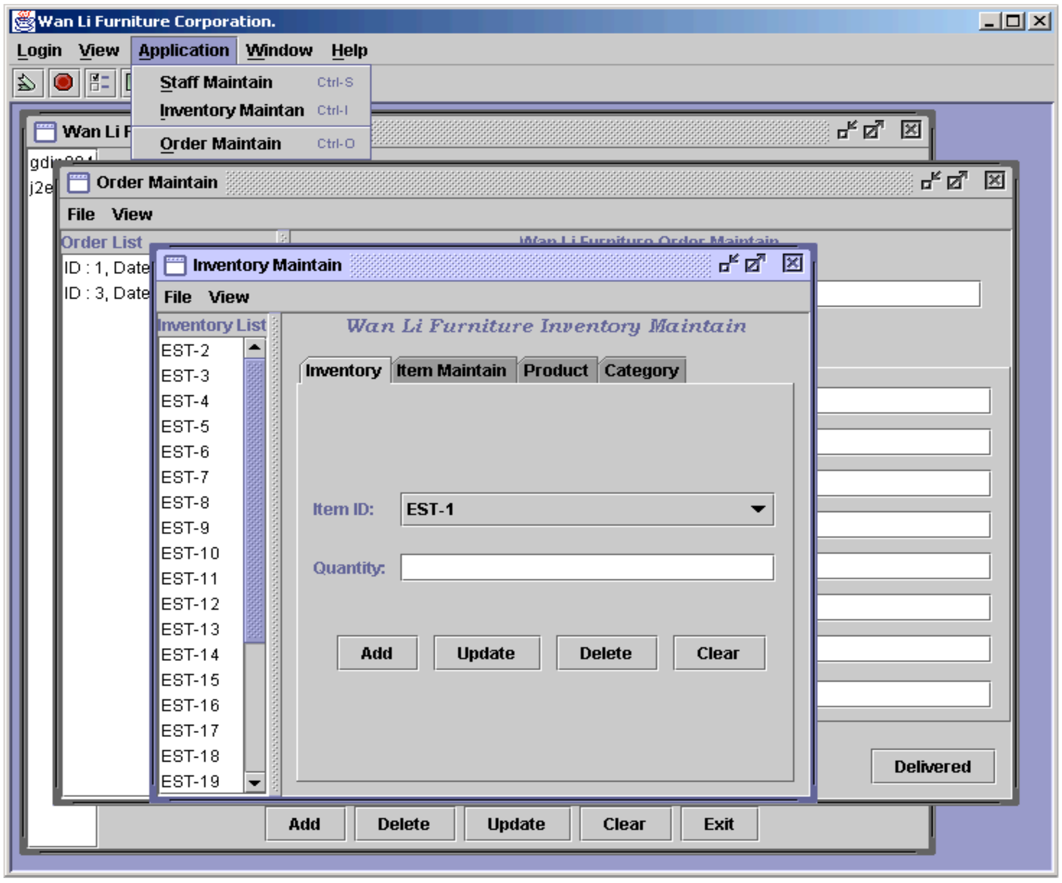


Figure 7-4 Main internal application user interface

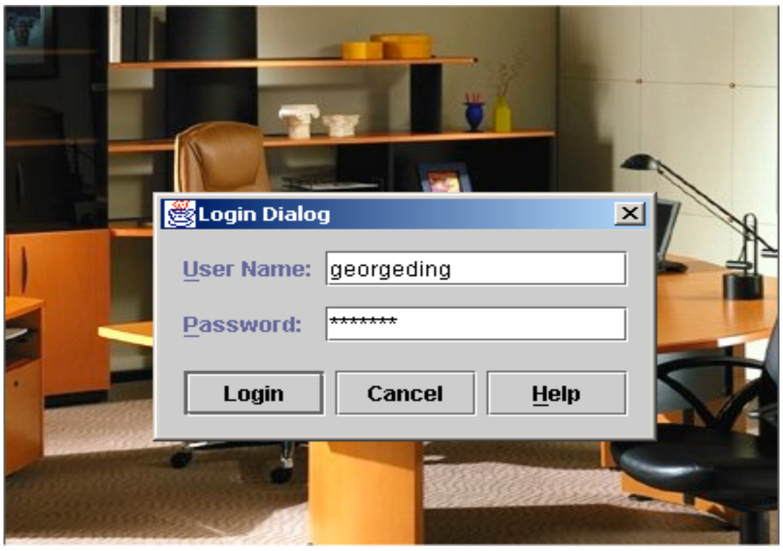
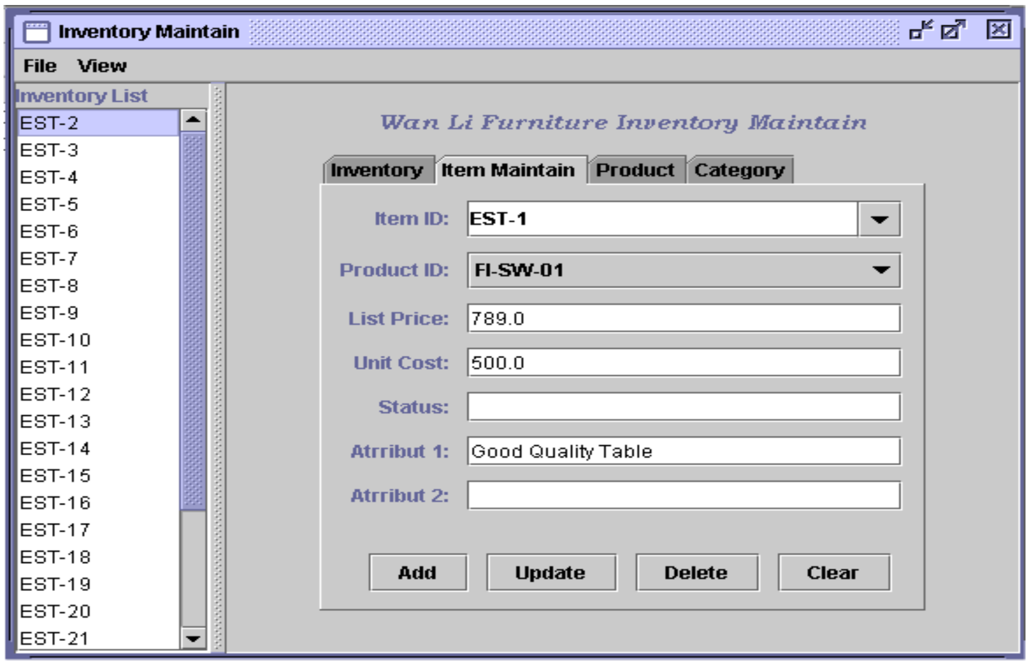


Figure 7-5 Application splash window and login dialog

In the e-furniture data maintain system include three sub-maintenance components, staff information maintenance, inventory data maintenance and order information maintenance. Here we just illustrate inventory data and staff information maintenance applications.

7.3.1 Inventory maintaining sub-component

The inventory-maintaining component is designed to maintain inventory stock data, product item, products and product category information. The component required user interface aspect of frame to present information and also provided UI aspects to main application component. Figure 7-6 shows the inventory maintenance user interface screen shot. The left panel lists all the inventory items currently available in databases. When user selects one item in the left panel, the item detailed information will be displayed in right side tabbed panes. The user then could manipulate the whole inventory database information by its functions provide by each tabbed pane, such as add new item, update item attributes and delete item etc. and could also add, update, delete products and categories information.

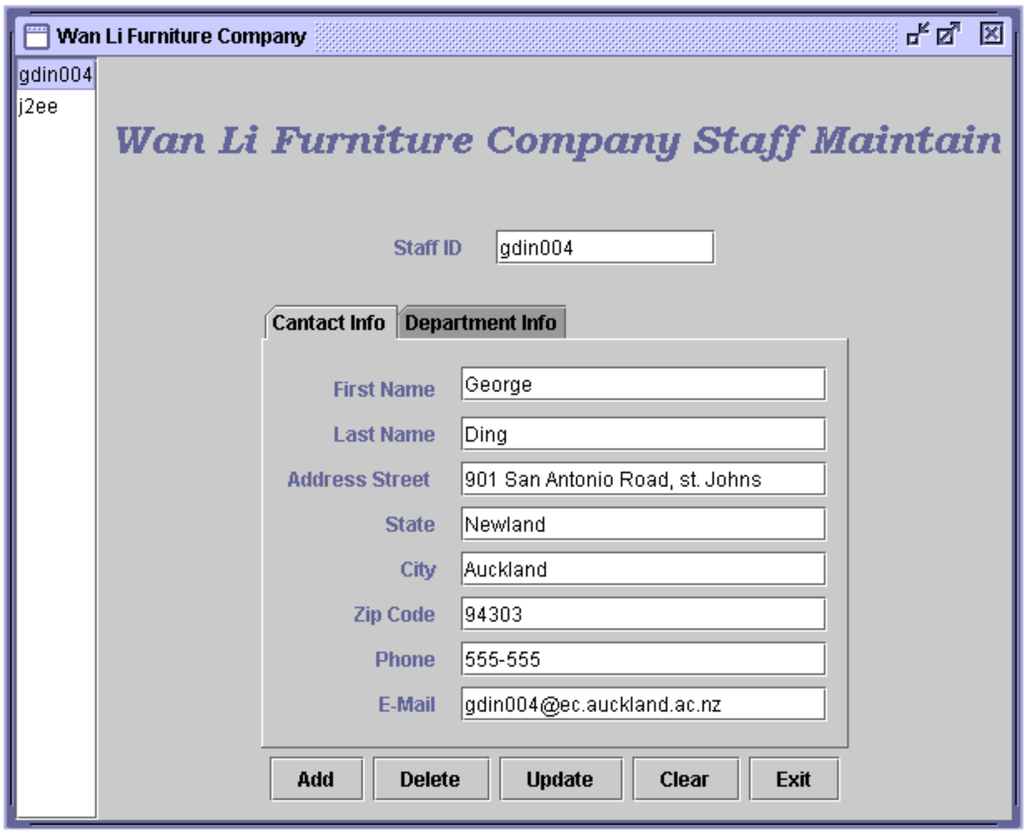


Item ID:	EST-1
Product ID:	FI-SW-01
List Price:	789.0
Unit Cost:	500.0
Status:	
Atrribut 1:	Good Quality Table
Atrribut 2:	

Figure 7-6 Inventory Maintain user interface

7.3.2 Staff maintaining sub-component

Staff maintenance component required frame UI aspects to present staff information and meanwhile provided UI aspects to main application component. The Figure 7-7 shows Staff maintenance user interface screen dump. We design and implement these maintenance user interfaces are in similar layout to keep the application consistency and convenient to use. Users could add new staff into database, and also could update, delete staff information via the maintenance tool.


Wan Li Furniture Company

gdin004
j2ee

Wan Li Furniture Company Staff Maintain

Staff ID: gdin004

Cantact Info | Department Info

First Name	George
Last Name	Ding
Address Street	901 San Antonio Road, st. Johns
State	Newland
City	Auckland
Zip Code	94303
Phone	555-555
E-Mail	gdin004@ec.auckland.ac.nz

Figure 7-7 Staff Maintain user interface

These maintenance components underlying mechanism is implemented by looking up the system relevant EJB server components, and then invokes their remote interfaces or home interfaces methods to communicate with database and do transactions. So these components are also required transaction aspects, which provided by server components through middleware. For example, inventory maintenance is looked up inventory home server component to create or locate remote inventory objects. Via the server components transaction aspect service, these maintenance components could complete its creation, updating and deleting functionalities. So, when user log into the system, the application will locate all the server components used by the application, and keep them in memory that will improve the applications performance aspects. If the application couldn't find any server components, it will inform user couldn't launch the application, server maybe down etc. The user has to try again later. And, some data loaded from database also have been cached to improve the applications performance. These are the consideration of components performance aspects required.

From component design and implementation, we can find that some components required security aspect and some components can also provided security aspects, for example, customer and staff component can provide security aspects for other components that try to communicate with them, and also required security aspects for its information storage such as password storage. So these components have been implemented by providing password authentication for users. Moreover, these components confidential information has been implemented by using cryptography algorithm encrypted then stored in database such as credit card information. All these are components security aspects consideration.

7.4 JSP – Web Components Design and Implementation

The one main part of E-Furniture system is the business to customer (B2C) web components, which were designed and implemented using Java Server Page (JSP) technology, that is very compatible with Enterprise Java Beans. JSP technology provides an easy way to develop servlet-based dynamic content, with the additional benefit of separating content and display logic. JSP technology is scalable and easy to write and maintain. The technology can provide dynamic user interface aspects in a platform-independent way. In the J2EE programming model web components can serve tow roles: as presentation components and as front components. JSP page acting as presentation component and front components used for managing other components and handle HTTP requests. Generally there are two types of architecture for design and implement web components – Model-View-Controller (MVC) architecture and just Java Beans based architecture.

The MVC architecture provides more flexibility, scalability and manageability. It is easy to customize in order to apply for other types of domain, but its design and implementation are very complex and very hard to debug. Figure 7-8 shows the architecture of using JSP page, Enterprise Beans and controller components.

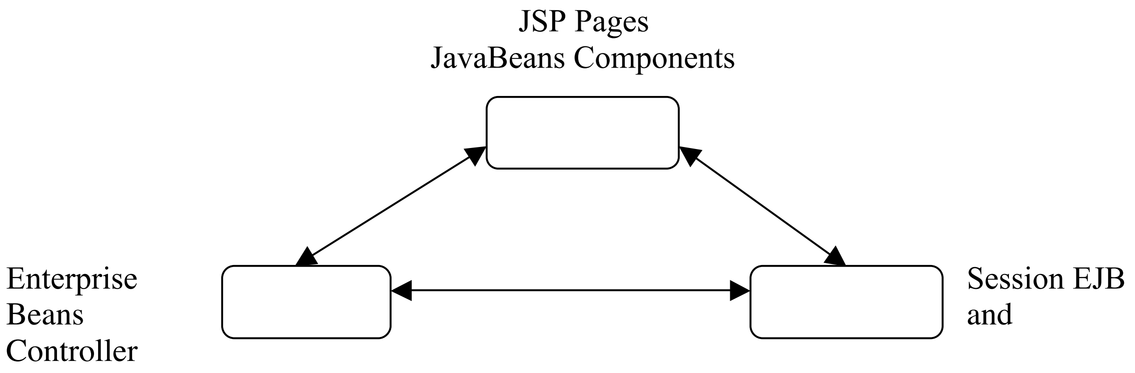


Figure 7-8 Model-View-Controller Architecture

Whereas, just Java Beans based JSP architecture are easier implementing, debugging and quickly achieving some specified goals. However, the architecture is not very flexible, scalable and customisable. Due to time constraints of the research, we adopt the second architecture to quickly achieve prototype of the system. Figure 7-9 illustrated aspect information with the architecture.

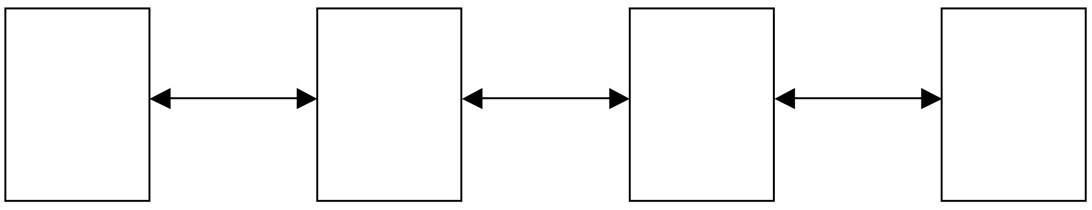


Figure 7-9 Web components architecture and aspects information

From the above architecture we can find that the web browser required user interface, transaction and security aspects, JSP will provide user interface, transaction and security aspects services. The security aspects we implemented using SSL secure tunnel via browser and JSP. Java beans required transaction, persistence and security aspects, and EJB beans will provide these aspects services. The Java beans components can also provide transaction and distribution aspects for JSP components. Furthermore, the EJB beans can also require persistency aspects, which are provided by databases. So the aspects information of a component is relative to the component its serviced for and required for.

In the e-furniture system, we designed a centralized Java Bean as the front component to manage and process all the requests and responses to JSP page. The centralized JavaBean communicates with EJB beans retrieving and updating data, and keeping the processing status for JSP pages within a session. The central JavaBean provided transaction aspect to JSP and required transaction aspect from EJB beans. Its like a bridge between JSP and EJB beans to service both sides.

The Main JSP page is designed to fulfill the whole functionalities of the system, users can go anywhere from the main page and invoke any functions in any orders. It required user interface aspects that provided by frame, and provided UI aspects to web browser. The main page screen dump is shown in Figure 7-10. It shows the left frame are all functionalities that invoking results will be shown in right frame. Left frame are the user navigation menus:

1. Users can choose the two main categories of Office Furniture and Home Furniture to browse any products of the system provided.
2. Users can also use the search engine to find any products by using relevant keywords, such as “table”.
3. User can sign in to the system at any time by clicking sign in menu and sign out the system as well by clicking sign out menu if already signed in.
4. The shopping cart content can also be viewed at the any time, however, if customer haven’t put anything in, it will give you an empty cart message.
5. Use can also maintain their personal information and view orders they placed in the store by clicking Maintain Information menu.

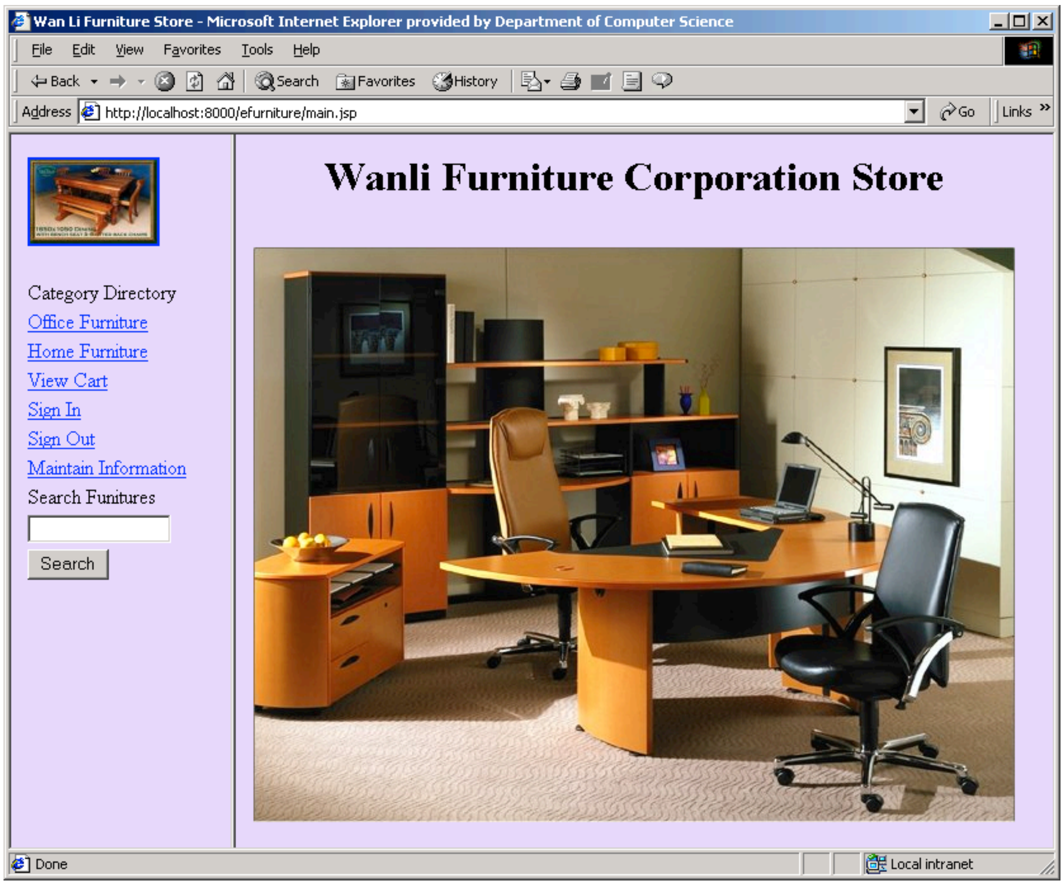
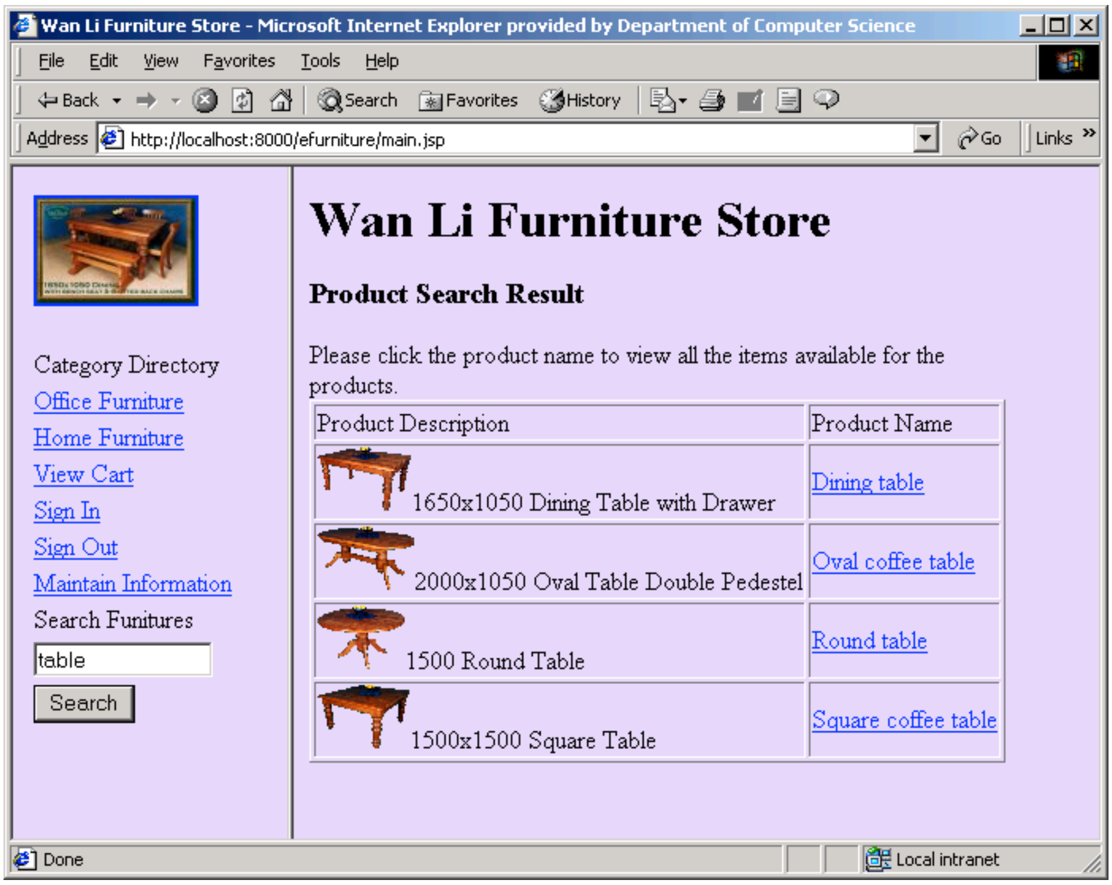

Wan Li Furniture Store - Microsoft Internet Explorer provided by Department of Computer Science
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8000/efurniture/main.jsp
Go Links
Wanli Furniture Corporation Store
Category Directory
Office Furniture
Home Furniture
View Cart
Sign In
Sign Out
Maintain Information
Search Funitures
Search
Done
Local intranet

Figure 7-10 Furniture store main page

The system web search engine JSP component is implemented not only can provide transaction aspect services, but also can provide user interface aspects for the search results. For example, if customer typed in a *"table"* as keywords in the search field, then click search button. Any item in the databases related with *"table"* category or products will be displayed in the right frame, which screen shot shows in Figure 7-11. The user can click products name to view its detailed information, which is include item code, item attributes and item unit prices attribute of the product. Further operations on product item are select an item added it into shopping cart and view the item more detailed description etc. Finally, when the user got all needed product in shopping cart, he/she can checkout to finish the shopping place an order in the store.


Wan Li Furniture Store - Microsoft Internet Explorer provided by Department of Computer Science
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8000/efurniture/main.jsp
Go Links
Done
Local intranet

Wan Li Furniture Store

Product Search Result

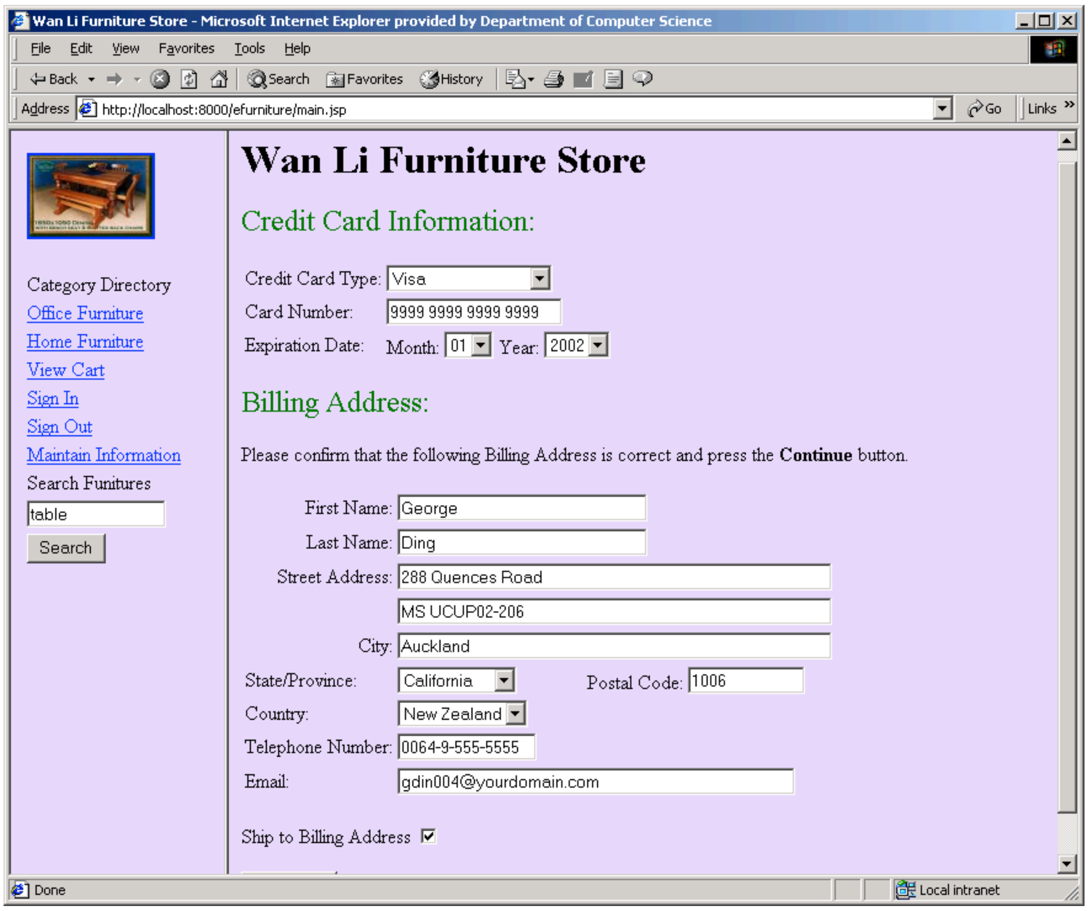
Please click the product name to view all the items available for the products.

Product Description	Product Name
1650x1050 Dining Table with Drawer	Dining table
2000x1050 Oval Table Double Pedestel	Oval coffee table
1500 Round Table	Round table
1500x1500 Square Table	Square coffee table

Category Directory
Office Furniture
Home Furniture
View Cart
Sign In
Sign Out
Maintain Information
Search Funitures
table
Search

Figure 7-11 Product Search result page

Like most of web components, the checkout JSP component is also can provide user interface aspect, and required transaction and distribution aspects from Java bean component to do the order creation etc operation. If a user has already added some product items in the shopping cart, the cart can be viewed at any time and go to checkout to make the order. If the user not logged in, user sign in page will prompt to let the user sign in first or create new user profile. However, if user signed in or new user created, then credit card fill in form and shipping information form will appear to ask user fill in these information. The checkout screen shot shows in Figure 7-12.


Wan Li Furniture Store - Microsoft Internet Explorer provided by Department of Computer Science
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8000/efurniture/main.jsp
Go Links
Category Directory
Office Furniture
Home Furniture
View Cart
Sign In
Sign Out
Maintain Information
Search Funitures
table
Search
Done
Local intranet

Wan Li Furniture Store

Credit Card Information:

Credit Card Type:	Visa
Card Number:	9999 9999 9999 9999
Expiration Date:	Month: 01 Year: 2002

Billing Address:

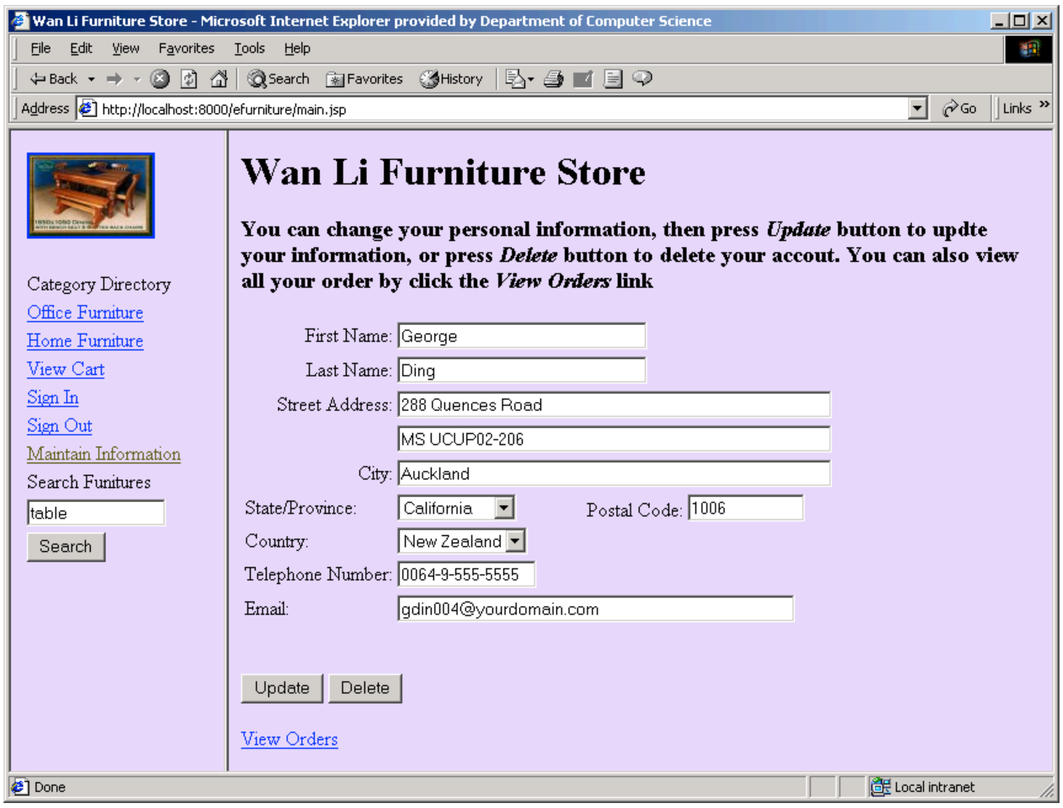
Please confirm that the following Billing Address is correct and press the **Continue** button.

First Name:	George
Last Name:	Ding
Street Address:	288 Quences Road
	MS UCUP02-206
City:	Auckland
State/Province:	California Postal Code: 1006
Country:	New Zealand
Telephone Number:	0064-9-555-5555
Email:	gdin004@yourdomain.com

Ship to Billing Address ☑

Figure 7-12 Checkout order information form page

Another main functional web component is customer information maintenance component. The component can provide user interface aspect and require transaction, distribution and persistency aspects. These required aspects are provided by Java bean component and EJB beans components. The component working scenario is like the following: If a customer has ever been registered in the store, then it's information and orders ever placed will be stored in the system databases. Customers could update itself information or delete them, and also could view its order lists and processing status when user has signed in. The customer information maintaining web page interface shows in Figure 7-13.


Wan Li Furniture Store - Microsoft Internet Explorer provided by Department of Computer Science
File Edit View Favorites Tools Help
Back Search Favorites History
Address http://localhost:8000/efurniture/main.jsp
Go Links
Category Directory
Office Furniture
Home Furniture
View Cart
Sign In
Sign Out
Maintain Information
Search Funitures
Search
Update
Delete
View Orders
Done
Local intranet

Wan Li Furniture Store

You can change your personal information, then press *Update* button to updte your information, or press *Delete* button to delete your accout. You can also view all your order by click the *View Orders* link

First Name:	George
Last Name:	Ding
Street Address:	288 Quences Road
	MS UCUP02-206
City:	Auckland
State/Province:	California
Postal Code:	1006
Country:	New Zealand
Telephone Number:	0064-9-555-5555
Email:	gdin004@yourdomain.com

Search Funitures: table

Figure 7-13 Customer information maintain page

7.5 Summary

EJB design and implementation framework provides much more flexible, scaleable, manageable and aspect-oriented services, which offer more opportunities for developers to concentrate on business logic, rather then systemic aspects services. These aspects services provided by EJB container has been illustrated in chapter 2, from which and the implementation of these components, we can find that the Enterprise JavaBeans technology is well fitted to our AOCE methodology. Therefore, the system EJB design and implementation didn't concern too much for aspect services. For example, entity beans or EJB container have managed persistency aspect, security aspects has been managed by EJB container etc. However, web component security aspect just used SSL, for some domains only SSL is not strong enough, some cryptography algorithm should be applied on it. Those domain web components will require more strong security aspects, which can be provided by java security packages and JCE provider packages.

The whole system components have been well designed, implemented and successfully deployed to J2EE server. From the e-furniture system components design and implementation for some specified aspects, we can find that these components aspect information are actually matched after we deployed the EJB beans and JSP pages to J2EE server. Aspect-oriented component analysis, design, implementation and deployment have made us gained high-level understanding for these components.

Chapter 8 Component Aspects Testing Agent

When AOCE technology and aspect notation set in UML have been applied to components specification and design, components with provided and required aspects information can be mapped to and implemented by EJB technology. However, after these EJB components deployed to EJB server, there is an issue comes up, how developers could verify that the EJB implemented components actually meet the component aspects they designed with aspects specification. To address this, we need some automatic testing agents to test these deployed components in EJB server by using their enabled aspects from these components aspects design. The chapter will illustrate the testing tools design and testing results in details.

8.1 Component Aspects Testing Tools Architecture Design

There are two types of components – EJB beans and JSP web components have been implemented and deployed to EJB server. Therefore, two testing tools have been designed to test the components respectively. Figure 8-1 shows the component tester tools design architecture.

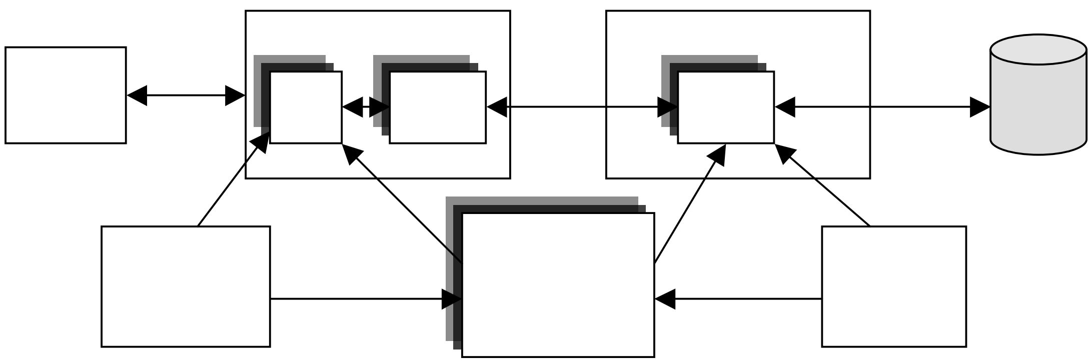


Figure 8-1 Component testers design architecture

The web components tester used for testing any web components include JSPs, whereas, EJB beans tester designed only for testing EJB components. Component aspects information – either web or EJB components will be described by component aspects descriptor XML documents, which can be loaded by tester tools to test components and present visual results for developers.

eXtensive Markup Language (XML) is a formalization of rules for marking up documents with meta-data to convey extra information (the data's purpose) to the user. XML has been widely used for data exchanging and storage, the first successful use is in business-to-business commerce and eventually creating new concepts, opportunities and extrapolated technologies [6]. So, XML has become the de facto standard for storing corporate data that can be easily manipulated stored and transformed by any computer languages. The XML's structure and easy to retrieve features are well fitted to describe components aspects for the AOCE technology.

Therefore, XML has been adopted in our testing tools to describe component aspects that are illustrated in component analysis and design phrase in chapter 4. According to component aspects analysis, each component can be described in variety aspects, which were defined in aspect descriptor XML Document Type Definition (DTD). Because, we are focus on component aspects testing using the descriptor, we are not only describing component aspects, but also aspects testing methods and parameters required by the method. Once we described component aspects in very specified detail, then we can load the component descriptor parsing it and retrieve its aspects and methods to test the component aspects information. The following two parts illustrated aspects descriptor DTD and an example of Staff component aspects descriptor and testing results in details.

8.1.1 Aspects Descriptor DTD

In order to marking up component aspects, we can specify the rules that govern the component aspect descriptor. These rules will determine whether a component aspects descriptor is valid. And, these rules can be specified in Document Type Definition (DTD) document. Therefore, a generic component aspects testing description DTD document has been designed to validate well-formed XML documents of component aspects. Developers who will apply AOCE to components can formalize component aspects descriptor instance according to the component aspects DTD. By declaring document type in component aspects descriptor instance, standard XML parsers then could validate the well-formed descriptor when parsing it. If there are some elements or attributes not consistent with DTD in the descriptor, the XML parser will throw exceptions to inform user what's the error in it. If no errors parsing successful, the parsed descriptor can be used to retrieve relevant information and perform the actual testing. The Figure 8-2 shows the component aspects descriptor DTD.

```
<?xml version="1.0" encoding="UTF-8"?>

<!ELEMENT Component (MappingName, URL, Aspects)>
<!ATTLIST Component Name CDATA #REQUIRED>

<!ELEMENT MappingName (#PCDATA)>
<!ELEMENT URL (#PCDATA)>
<!ELEMENT Aspects (Aspect)>
<!ELEMENT  Aspect  (Performance  |  Persistence  |
Transaction | Security)+>
<!ATTLIST Aspect name CDATA #IMPLIED>

<!ELEMENT Performance (Speed|Robustness)+>
<!ELEMENT Speed (Method)+>
<!ATTLIST Speed TestNumber CDATA "1">
<!ELEMENT Robustness (Method)+>

<!ELEMENT Persistence (Retrieve|Store)+>
<!ELEMENT Retrieve (Method)+>
<!ELEMENT Store (Method)+>
<!ATTLIST Persistence
          User_Num CDATA "1"
          Data_Size CDATA "0">

<!ELEMENT Transaction (Commit|Rollback|Speed)+>
<!ELEMENT Commit (Method)+>
<!ELEMENT Rollback (Method)+>

<!ELEMENT Security (Encryption|Authentication)+>
<!ELEMENT Encryption (Method)+>
<!ELEMENT Authentication (Method)+>

<!ELEMENT Method (Parameter)*>
<!ATTLIST Method name CDATA #REQUIRED>
<!ELEMENT Parameter (#PCDATA)>
```

Figure 8-2 Component aspects descriptor DTD

From above DTD we can find that the high-level element *Component* will encapsulate whole information of component and its aspects. The *Component* element's children nodes can be *MappingName*, *URL* and *Aspects*, but it only have one attribute *Name* that used indicate which component described, and it is required attribute. The *MappingName* element is a convenient tag used to specify some technology's component binding name, such as CORBA server component binding name or EJB enterprise beans JNDI name. It is an optional tag. If a component will be tested don't need some binding names, the *MappingName* element is not necessary. The *URL* element used to specify the component universal resource location, mainly designed for testing web components or components not running in the same testing host. And, the *Aspects* element will encapsulate any single specified *Aspect* information for the component we would like to describe. The every single *Aspect* element can have one of the aspects *Performance*, *Persistence*, *Transaction*, or *Security* etc. The DTD can also be extended to include other aspects when some components needed, such as to include user interface aspect. The following table shows each aspect element brief description.

Table of component aspect DTD element/attribute description:

Element/Attribute	Description
Performance	Aspect element performance
Speed	Specified performance aspect to test component perform speed
TestNumber	Attribute of Speed element to define the number of concurrent testing for the component, default value 1
Robustness	Specified performance aspect to test component robust ability
Persistence	Aspect element of persistency
User_Num	Attribute of persistence element to simulate the number of concurrent users to call the function, default value 1
Data_Size	Attribute of persistence to specify data size for persistency aspect function call, default 0 means can be any size

Retrieve	Specified element of persistence aspect to test data retrieve ability of the component
Store	Specified element of persistence aspect to test data storage ability of the component
Transaction	Aspect element of transaction
Commit	Specified element of transaction aspect to test transaction successful or not
Rollback	Specified element of transaction aspect to test rollback ability of transaction when transaction failed
Speed	Specified element of transaction aspect to describe transaction performance
Security	Aspect element of security
Encryption	Specified element of security aspect to test secret data encrypted or not
Authentication	Specified element of security aspect to test access right of the component
Method	Aspect testing call function element
Name	Attribute of method element to specify aspect testing function name
Parameter	Nested element of method to define the parameters used by the tested function

8.1.2 Staff component Aspects XML Descriptor Example

To test e-furniture system's staff component aspect information, we figured out the following XML descriptor in Figure 8-3. In the XML descriptor, we defined document type using component aspects DTD in the second line of the document. The component name of "*Staff*" has been specified in top-level element *Component* and also defined the component JNDI mapping name of *"java:comp/env/ejb/staff"* in the J2EE platform. The JNDI mapping name will be used for locating staff component in EJB container when testing, then operate aspects testing methods on the component. All the testing aspects have been nested in *Aspects* element, for example performance aspect. We just illustrated performance aspect here, for the Staff component full aspects testing descriptor please refer to Appendix III.

The Staff component performance will be tested in two aspects, *Speed* and *Robustness*, in which used testing function is *"StaffHome.findByPrimaryKey()"* and passed in parameter for the function is a staff user ID *"J2ee"* and *"J2ees"* respectively. The expected result is that for performance speed aspect, we'll successfully find the staff information of *"J2ee"* and the method consumed time, for performance robustness aspect, we couldn't find the staff information of *"J2ees"*. The reason for this is that there is staff information for *"J2ee"* in our database, but no information for *"J2ees"*. That is we try to pass in a wrong argument to test the component robustness aspect. Moreover, for the *Speed* performance aspect, the descriptor specified *TestNumber="3"* which means the speed testing will concurrently test three times and the *AcceptableSpeed="5"* means the minimum acceptable speed for the performance. The detailed testing results analysis will be discussed in following section of the chapter.

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE Component SYSTEM "componentaspects.dtd">
<Component Name="Staff">
  <MappingName>java:comp/env/ejb/staff</MappingName>
  <Aspects>
    <Aspect name="Performance">
      <Performance>
          <Speed TestNumber="3" AcceptableSpeed="5">
              <Method name="StaffHome.findByPrimaryKey()">
                <Parameter>J2ee</Parameter>
             </Method>
         </Speed>
          <Robustness>
            < Method name="StaffHome.findByPrimaryKey()">
               <Parameter>J2ees</Parameter>
            </Method>
          </Robustness>
      </Performance>
    </Aspect>
  </Aspects>
</Component>
```

Figure 8-3 Staff component aspect descriptor

8.2 Server Component Aspects Testing Agent

Once server components design and implementation have been finished, we can define some components aspect descriptor to test these components aspects that we are interested. In the example e-furniture system, when EJB server components have been deployed to J2EE server, we can invoke its remote and home methods to test their functionalities. However, we don't know anything about these components aspect information, and the aspects information is not easy to test. Therefore, we designed component aspects testing tool, which can load component aspects descriptor and test those aspects specified in the descriptor, and also can show the testing results in java swing GUI components supplied user interface. Based on component aspect descriptor and the e-furniture system server component implementation technology, we designed an EJB dependent component aspects testing tool. However, the idea can be applied to any technologies.

The testing tools we designed composed by two components, one is the tools user interface GUI components, and the other is an EJB session bean component, which will do the actual testing task in the J2EE server. Figure 8-4 shows the tester tool EJB component of session bean design architecture. As we can see, the component remote interface *AspectTester* has defined all the business methods corresponded to the aspects that specified in component aspect DTD. So, developers could get any tested aspect results via the remote interface. The returned results will be a collection of serialized *AspectData* object, which encapsulated an aspect testing results.

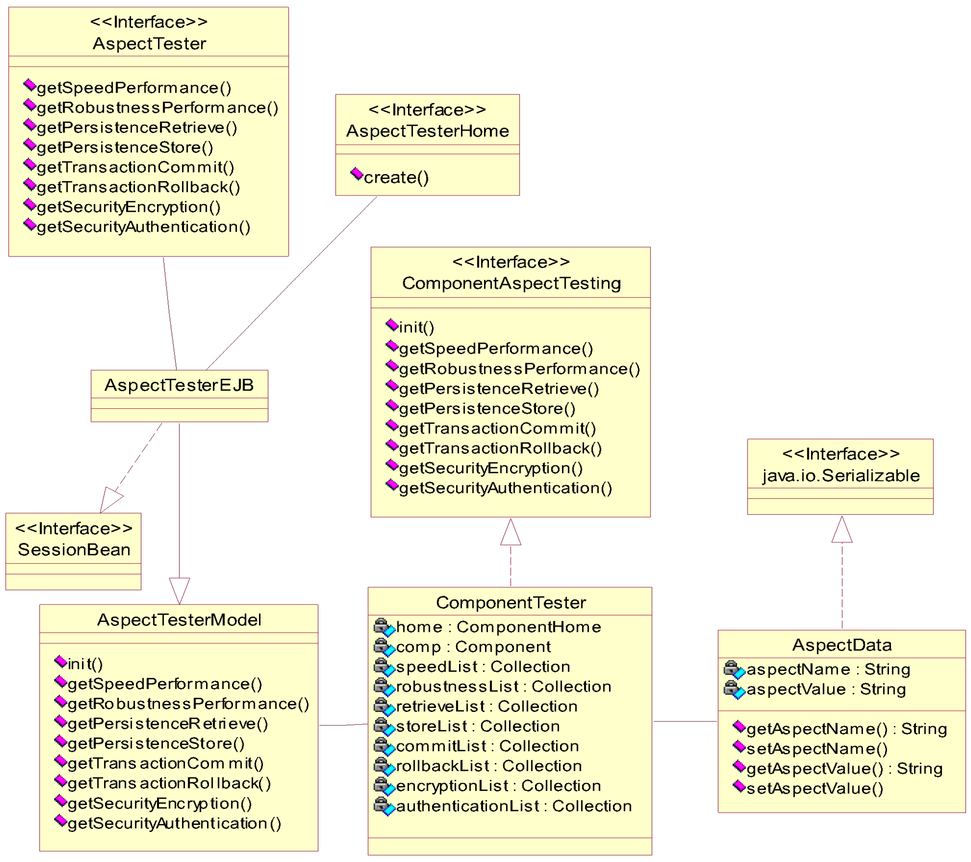
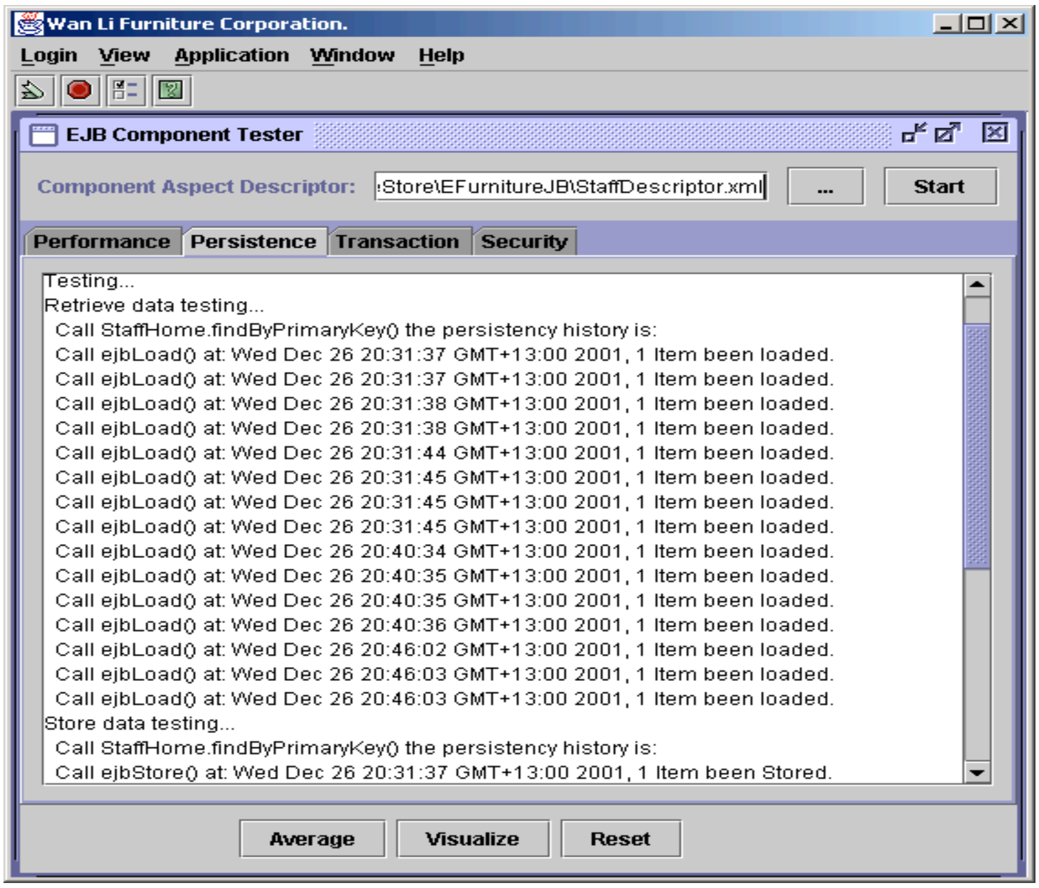


Figure 8-4 Aspect tester session bean

In the implementation of the testing tool, component aspects descriptor will be passed to the aspects tester session bean to do the test in J2EE server side. In the tool's client side, by retrieving the returned collection of those aspect data, the tested results will be shown in the testing tool GUI component. Figure 8-5 shows the testing tool's user interface that implemented by using java swing components. From the user interface, we can find that user can select component aspects descriptor by clicking the button *"…"* to invoke a file chooser. When descriptor has been selected, user can press the *"Start"* button to start the testing. Once testing finished, the relevant results will be shown in its aspect tabbed-pane respectively. For the performance aspect, user can press the *"Average"* button to calculate the average performance of the component. The tool also provide a reset function used to reset the tool to initial state and to clear the test results. To invoke the function just press the button of *"Reset"*. Although there is a *"Visualize"* button at the bottom of the tool for showing graph of results the visualization function not been implemented yet.



Component Aspect Descriptor: :Store\EFurnitureJB\StaffDescriptor.xml

```
Testing...
Retrieve data testing...
 Call StaffHome.findByPrimaryKey() the persistency history is:
 Call ejbLoad() at: Wed Dec 26 20:31:37 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:37 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:38 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:38 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:44 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:45 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:45 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:31:45 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:40:34 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:40:35 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:40:35 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:40:36 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:46:02 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:46:03 GMT+13:00 2001, 1 Item been loaded.
 Call ejbLoad() at: Wed Dec 26 20:46:03 GMT+13:00 2001, 1 Item been loaded.
Store data testing...
 Call StaffHome.findByPrimaryKey() the persistency history is:
 Call ejbStore() at: Wed Dec 26 20:31:37 GMT+13:00 2001, 1 Item been Stored.
```

Figure 8-5 Component tester tool

The underlying aspects testing scenario is shown in Figure 8-6 aspect testing sequence diagram.

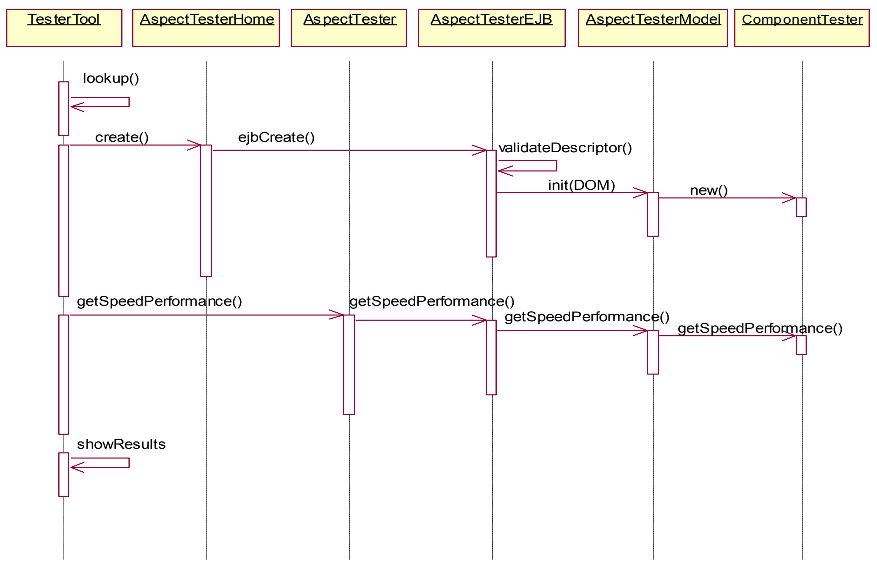


Figure 8-6 Aspect Testing sequence diagram

The Figure 8-7 is the tested results of performance aspect for Staff component. The testing machine used for the testing results is Intel 500MHz processor, 256MB memory; the platform used is Windows 2000, J2EE server, cloudscape database and all these are running in the same machine.

```
Compnent Name: Staff

Testing...
Speed testing...
  Call StaffHome.findByPrimaryKey() once Time used: 40 MilliSeconds
  Call Staff.getDetails() once Time used: 170 MilliSeconds
  Call Staff.changeStaffInfo() once Time used: 170 MilliSeconds
Robustness testing...
  Call StaffHome.create() once :  Ok
Another test:
Speed testing...
  Call StaffHome.findByPrimaryKey() once Time used: 40 MilliSeconds
  Call Staff.getDetails() once Time used: 140 MilliSeconds
  Call Staff.changeStaffInfo() once Time used: 110 MilliSeconds
Robustness testing...
  Call StaffHome.create once: Ok
Another test:
Speed testing...
  Call StaffHome.findByPrimaryKey() once Time used: 60 MilliSeconds
  Call Staff.getDetails() once Time used: 130 MilliSeconds
  Call Staff.changeStaffInfo() once Time used: 130 MilliSeconds
Robustness testing...
  Call StaffHome.create() once: Ok

Average speed performance:
  Call StaffHome.findByPrimaryKey() 3 times Average time used: 50 MilliSeconds
  Call Staff.changeStaffInfo() 3 times Average time used: 135 MilliSeconds
  Call Staff.getDetails() 3 times Average time used: 142 MilliSeconds
```

Figure 8-7 Staff component aspects information testing results

As can be found from the results, the staff component's main functionality performance are reasonable well, although when first time loading the component and call its methods are little bit slower. The average performances we tested are met the requirements of 5 seconds. If the performances fail to meet the requirements constraints, the system components designed and implemented are not successful. Robustness of the component is quite stable as well. Other aspects of the component are also been tested, its results are well reflected the component aspects information, which gives user a good understanding of the component higher-level systemic and crosscutting issues.

In general developers can use the testing tool and component aspects descriptor DTD to outline any components aspect information and test them. Although the tools now just support for Enterprise Java Bean technology implemented components, its idea and design frameworks can be applied to any technologies.

8.3 Web Component Aspects Testing Agent

Web components mainly provide user interfaces aspect services dynamically for users. There are two ways that can be used to test web component aspects information, one is grab web components as plain HTML file and parse the HTML file to find relevant information for the web component loading results. Another is load web components and displays the loading results in a browser like viewer to give user visual results for the component aspect testing. We adopted later technology and use the same component aspects descriptor DTD with server components. The Figure 8-8 shows the web component tester design class diagram. The *WebCompTesterIFrame* class shows the testing tools functionalities and *WebCompTestThread* class will do the actual testing work, callback to return results or loading web component and display in *SimpleBrowser*.

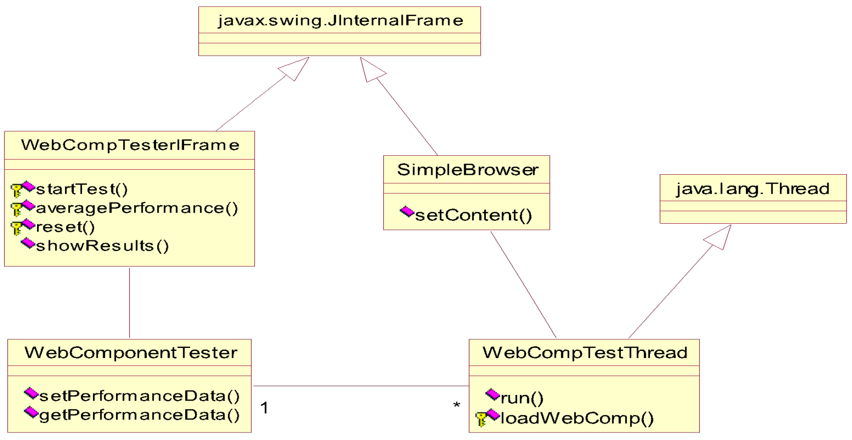


Figure 8-8 Web component tester class diagram

The Web components testing scenario is shown in Figure 8-9 Web component aspects testing tool sequence diagram.

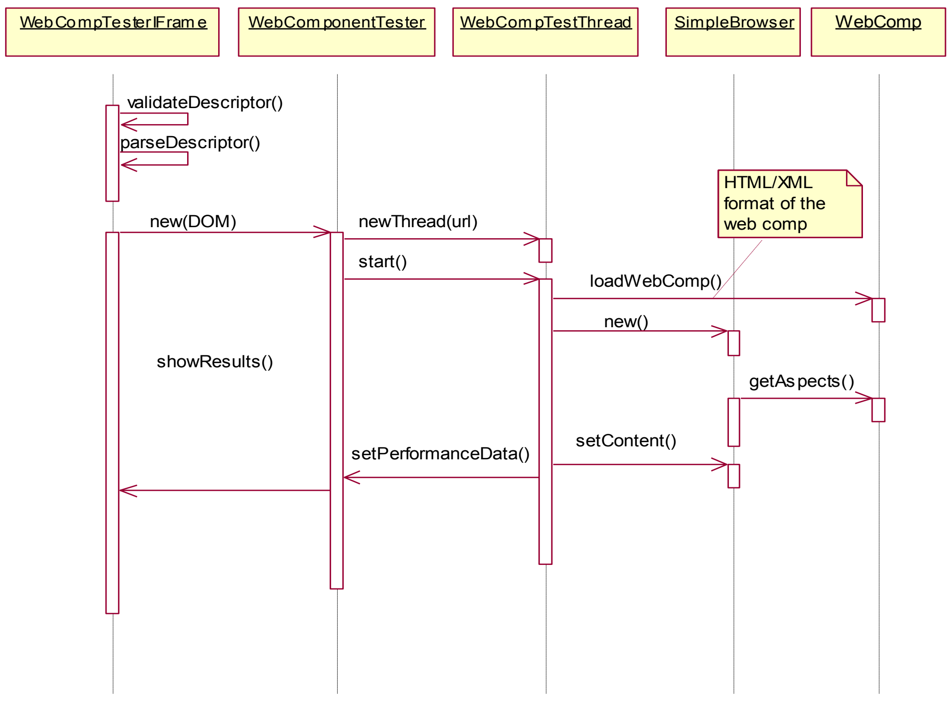


Figure 8-9 Web component aspects testing tool sequence diagram

The Figure 8-10 is an example of a partial of web components descriptor XML file, in which specified some web components performance aspects to be tested. The web component aspects testing tool parse the component descriptor as different way. For example, the *Method* element specifies one of the web pages to be loaded and tested. And the *URL* element becomes more important for web components testing. The *URL* element defines universal location of these web pages to be loaded and tested.

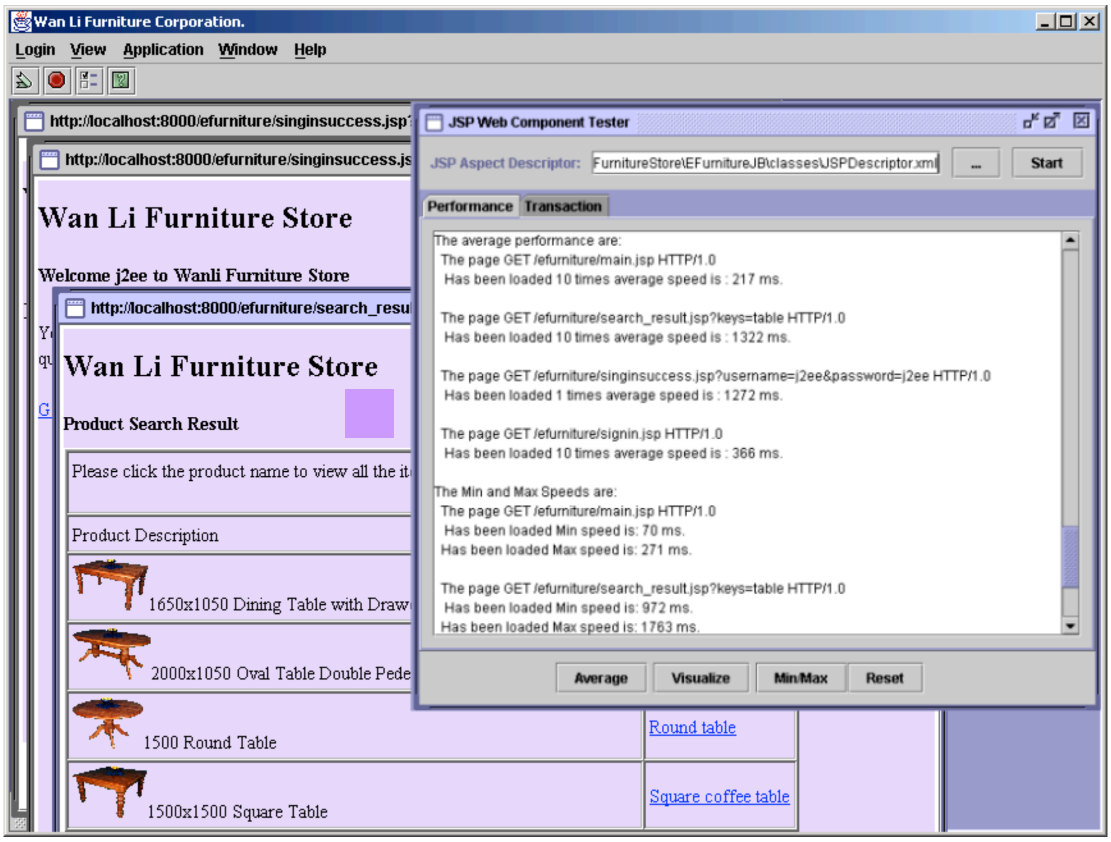
From figure 8-10 (the part of aspect descriptor), can be seen that there is three web pages have been specified in *Speed* element with attribute *TestNumber="10"*, which means that each of the three pages will be concurrently loaded ten times by starting a thread for each test. If the *TestNumber* attribute not specified means that the *TestNumber* is 1 by default, just test once. For example, the *singinsuccess.jsp* just tested once by passing in a username and its password to test whether the user can sing in to the furniture store. Some aspects of a web component have to examine their loading results by passing some specified parameters, such as robustness aspect, which testing is by passing correct and incorrect parameters to see its results.

```
<?xml version="1.0" encoding=”UTF-8”?>

<!DOCTYPE Component SYSTEM "componentaspects.dtd">
<Component Name="JSP">
  <URL>http://localhost:8000</URL>
  <Aspects>
    <Aspect name="Performance">
      <Performance>
          <Speed TestNumber="10">
            <Method name="GET /efurniture/main.jsp HTTP/1.0" />
            <Method name="GET /efurniture/signin.jsp HTTP/1.0" />
            <Method name="GET
/efurniture/search_result.jsp?action=Search+keys=table HTTP/1.0" />
         </Speed>
         <Speed>
            <Method name=
"GET /efurniture/singinsuccess.jsp?username=j2ee+password=j2ee HTTP/1.0" />
         </Speed>
      </Performance>
     </Aspect>
  </Aspects>
</Component>
```

Figure 8-10 Part of web components descriptor

There is an aspects descriptor of web components in Appendix IV, which tested-results have been shown in Figure 8-11. As can be seen that the testing tool’s user interface is almost same with EJB components’ testing tool, the one more function is the tool can find some web pages loading minimum and maximum loading speed by pressing button *“Min/Max”*. The number (1) performance tabbed pane shows performance aspects testing results of four web components and each of them average performance. In the main desktop pane, there are several web pages browsers have been created to show some web components tested results. The simple web page browser has been designed for some web components aspect testing results display, such as for robustness aspect testing. (2) The number (2) internal frame simple browser shows the web search engine component transaction aspect tested results. We invoke (1) the search engine web component by passing in a key word of *"table"*, and then the component retrieves databases find relevant results and display them in the internal frame browser. From the displayed results and databases actual data, we know the component correctly retrieved data from databases and present right results for us, and furthermore the transaction's consumed time can be found in the testing tool's transaction tabbed pane.


Wan Li Furniture Corporation.
Login View Application Window Help
http://localhost:8000/efurniture/singinsuccess.jsp
http://localhost:8000/efurniture/singinsuccess.js
Wan Li Furniture Store
Welcome j2ee to Wanli Furniture Store
http://localhost:8000/efurniture/search_resul
Wan Li Furniture Store
Product Search Result
Please click the product name to view all the it
Product Description
1650x1050 Dining Table with Draw
2000x1050 Oval Table Double Pede
1500 Round Table
Round table
1500x1500 Square Table
Square coffee table
JSP Web Component Tester
Performance
Transaction
Average
Visualize
Min/Max
Reset
Start

JSP Aspect Descriptor: FurnitureStore\EFurnitureJB\classes\JSPDescriptor.xml

```
The average performance are:
 The page GET /efurniture/main.jsp HTTP/1.0
  Has been loaded 10 times average speed is : 217 ms.

 The page GET /efurniture/search_result.jsp?keys=table HTTP/1.0
  Has been loaded 10 times average speed is : 1322 ms.

 The page GET /efurniture/singinsuccess.jsp?username=j2ee&password=j2ee HTTP/1.0
  Has been loaded 1 times average speed is : 1272 ms.

 The page GET /efurniture/signin.jsp HTTP/1.0
  Has been loaded 10 times average speed is : 366 ms.

The Min and Max Speeds are:
 The page GET /efurniture/main.jsp HTTP/1.0
  Has been loaded Min speed is: 70 ms.
 Has been loaded Max speed is: 271 ms.

 The page GET /efurniture/search_result.jsp?keys=table HTTP/1.0
  Has been loaded Min speed is: 972 ms.
 Has been loaded Max speed is: 1763 ms.
```

Figure 8-11 Web component testing tool and tested results

More tests have been made for web components. The component descriptor *TestNumber* attribute value changed to 20 and 100, to simulate concurrent users of these numbers. That means the testing tool will simulate 20 and 100 concurrent users to load several web pages concurrently. Although the tested results are much slower, it didn’t crash machine and J2EE server. Due to the testing tools and J2EE server database running in the same host, the tested results much slower are reasonable.

8.4 Summary

Component aspect information can give developers and component users a higher-level systemic aspect understanding. However, component aspect information is very hard to test and simulate, although aspects have been designed and implemented with components. In the chapter, component aspect descriptor and testing tools have been designed and implemented, and successfully tested server component – EJB beans and web component – JSP aspect information. Hence component developers could use the tools to verify AOCE designed components to see whether the designed aspects actually met the aspects when components deployed to server.

For application server components, the testing tools just implemented testing EJB server components, but it can be used for testing any web components and its notation also can be applied to any technologies implemented components. Component aspect descriptor could also be used for any technology designed and implemented components with aspects. The component aspects testing results will give component developers a good understanding of components systemic crosscutting aspects information, from which developers will benefit to improve components service qualities. However, components user interface aspects haven’t been tested in the testing tool, maybe this can add into the tools in the future works.

Chapter 9 Conclusion

This chapter is a summary of the thesis. It presents what we have done, the main contributions, and points out some possible future works we think maybe worth extending and researching.

9.1 Contributions of the Thesis

This thesis illustrates the use of our new methodology of aspect-oriented component engineering in the whole software engineering lifecycle. This includes engineering requirements, designs, implementations and testing. The main contributions made are:

- Showing how to develop components with provided and required aspect services by using AOCE.
- Suggesting a set of new notation to express the concept of component aspects in standard UML-based software development tools such as Rational Rose™.
- Proposing a notation of component testing framework – XML based component aspects descriptor to outline component aspect services.
- Designing and implementing component aspect testing agents, focusing on Enterprise JavaBeans technology implemented components with aspect information. This testing framework can be applied to any technologies.
- Prototyping a fully functional e-commerce system by applying our AOCE approach, and implementing the system using EJB technology.

There are number of component-based development methodologies such as The Select Perspective™, The Catalysis™ and COMO. Most of these approaches focus on low-level functionalities vertically slicing overall system and group components. These resulting components lack systemic crosscutting concerns and non-functional constraint information. Whereas, our AOCE approach successfully solves these issues by identifying and specifying provided and required aspect services in component interfaces. The aspect services could be used for reasoning about components to provide good quality.

The concept of component aspects can be indicated in UML-based standard software development tools. The aspect services of components can be documented in component requirements and design by using our proposed notation sets. They can be codified in component implementations to provide high-level multiple perspective views and run-time dynamic reconfiguration capabilities of components.

To verify component with aspect services actually met the requirements and designs, our testing framework and agents can give component developers better testing plans and tested results analysis for components they developed.

We implement the example system very well by using Enterprise JavaBeans. These are mapped aspect-oriented services can be considered as aspect-oriented component development environments.

9.2 Future Research Works

AOCE is quite a new methodology for developing more reusable components. There are a number of different fields, which need to do further research. In this thesis, we only give some feasible fields and some natural extensions of the thesis.

- Extending the component aspect testing framework and agents to cover the user interface aspect testing. The user interface aspects are not very easy to test as it's usually involved with GUI components plug-in. Therefore, the GUI component's detailed aspect description and introspection will be more important for user interface aspect testing.

- Extending the aspect testing agents to visualize testing results that will give developers a better results-testing analysis.

- Designing and implementing a component aspect descriptor generation agent for combining with the testing agents. Developers could then just specify component aspects and the agent introspect component aspect services would generate a component aspect descriptor.

- For tool support of the AOCE methodology, it maybe worth future researchers integrating AOCE with a standard CASE tool such as Rational Rose™.

- The UML diagrams with aspects in this thesis can be further augmented, maybe using Object Constraints Language (OCL) in UML to express aspect-based constraints.

- Furthering dynamic configuration support for AOCE, specifically a relatively complete aspects repository and details for different domains. These are the possible researching fields of AOCE.

- Further use of AOCE integration methodology with various other component-based development technologies is also a possible investigation area.

Bibliography

[1] *Alan W. Brown*, Large-Scale, Component-based Development, 2000.

[2] *Paul Allen, Stuart Frost*, Component-Based Development for Enterprise Systems Applying The Select Perspective™, 1998.

[3] *Grundy, J.C., Mugridge, W.B. and Hosking, J.G.,* Constructing Component-based Software Engineering Environments: Issues and Experiences, *Journal of Information and Software Technology*, Vol.42, No. 2, January 200, pp.117-128.

[4] *John Grundy*, Storage and Retrieval of Software Components using Aspects

[5] The Java 2 Enterprise Edition Developer's Guide, May 2000, Sun Microsystems

[6] *Michael C. Daconta and AI Saganich*, XML Development with Java 2, Oct. 2000

[7] *G. Pour,* Software Component Technologies: JavaBeans and ActiveX, Proceedings of Technology of Object-Oriented Language and system, 1999, pp.398-398.

[8] *Xia Cai, Michael R. Lyu, Kam-Fai Wong, Roy Ko*, Component-baed software Engineering: Technologies, Development Frameworks, and Quality Assurance Schemes, IEEE 2000.

[9] *G. Pour*, Component-based Software Development Approach: New Opportunities and Challenges, Proceedings Technology of Object-Oriented Laguage, 1998, Tools 26.

[10] *C. R. Guareis De Farias, L. Ferreira Pires, M. van Sinderen, D. Quartel*, A combined Component-Based Approach for the Design of Distributed Software

Systems, Proceedings of the Eighth IEEE Workshop on Future Trends of Distributed Computing System, 2001

[11] *A. W. Brown, K.C. Wallnau*, The Current State of CBSE, IEEE Software, Volume 155, Sept.-Oct. 1998.

[12] *Desmond Francis D'Souza, Alan Cameron Wills,* Objects, Components, and Frameworks with UML, The Catalysis™ Approach, 1999, Addison Wesley Longman, Inc.

[13] *San Duck Lee, Young Jong Yang, Eun Sook Cho, Soo Dong Kim, Sung Yul Rhew*, COMO: A UML-Based Component Development Methodology, Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific, 1999 Page(s): 54 –61

[14] *Chris Crenshaw*, The Developer's Guide to Understanding Enterprise JavaBeans Applications, NOVA Laboratories, www.nova-labs.com

[15] *Sun Mirosystems, Inc.*, Enterprise JavaBeans (EJB) specification 1.1, http://www.javasoft.com/products/ejb

[16] *Jung Pil Choi*, Aspect-Oriented Programming with Enterprise JavaBeans, Enterprise Distributed Object Computing Conference, 2000. EDOC 2000. Proceedings. Fourth International , 2000.

[17] *Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, John Irwin.* Aspect-Oriented Programming, In *proceedings of the European Conference on Object-Oriented Programming (ECOOP),* Finland. Springer-Verlag LNCS 1241. June 1997.

[18] *Karl Lieberherr, David Lorenz, Mira Mezini*, Programming with Aspectual Components, 1999.

[19] *Karl Lieberherr*, Connections between Demeter/Adaptive Programming and Aspect-Oriented Programming (AOP), http://www.ccs.neu.edu/home/lieber/connection-to-aop.html 1999

[20] *Karl J. Lieberherr*, Early Definition of Aspect-Oriented Programming, http://www.ccs.neu.edu/research/demeter/AOP/early-def/AP-AOP.html

[21] *Karl Lieberherr* , Adaptive Object-Oriented Software - The Demeter Method http://www.ccs.neu.edu/research/demeter/biblio/dem-book.html

[22] *John Grundy*, Multi-perspective Specification, Design and Implementation of Software Components Using Aspects, *Journal of Software Engineering and Knowledge Engineering*, Vol. 20, No. 6, December 2000, World Scientific

[23] *John Grundy, Rakesh Patel*, Developing Software Components with UML, Enterprise Java Beans and Aspects, In Proceedings of the 2001 Australian Software Engineering Conference, Canberra, Australia, 26-28 August 2001, IEEE CS Press.

[24] *John Grundy*, Aspect-oriented Requirements Engineering for Component-based Software System, In *Proceedings of the 1999 IEEE Symposium on Requirements Engineering, Limmerick*, Ireland, 7-11 June, 1999, IEEE CS Press, pp. 84-91.

[25] *Grundy, J.C.* Supporting aspect-oriented component-based systems engineering, In *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany, 16-19 June 1999, KSI Press, pp. 388-395.

[26] *Grundy, J.C.*, An implementation architecture for aspect-oriented component engineering, In *Proceedings of the 5th International Conference on Parallel and Distributed Processing Techniques and Applications: Special Session on Aspect-oriented Programming*, Las Vagas, June 26-29 2000, CSREA Press.

[27] *Monson-Haefel, R.,* Enterprise JavaBeas, Oreilly, 1999.

[28] *Sessions, R.,* COM and DCOM: Microsoft's vision for distributed objects, Wiley, 1998.

[29] *Ho, W.M., Pennaneach, F., Jezequel,* J.M. and Plouzeau, N., Aspect-Oriented Design with the UML, *InProceedings of the ICSE2000 Workshop on Multi-Dimessional Separation of Concerns in Software Engineering, Limerick, Ireland, Jun 6 2000.*

[30] *Roger Sessions*, Java 2 Enterprise Edition (J2EE) versus The .NET Platform Two Visions for eBusiness, ObjectWatch Inc.March 28, 2001, http://www.objectwatch.com/_Toc511347196.

[31] *AspectJ*, http://aspectj.org/servlets/AJSite

[32] *Legion of the Bouncy Castle*, http://www.bouncycastle.org

[33] *Sun Microsystems, Inc.*, Enterprise JavaBeans™ Specification,Version 2.0, http://java.sun.com/Download5

Appendix I: E-Furniture system use case description

➢ Maintain Order

Maintain order	
Actor	Customer
Precondition	Customers want to update or cancel orders.
Postcondition	Customer's orders have been updated, cancelled.
Description	Customer updates or cancels orders through online system.
Basic course of action	System shows current orders for Customer. Customer enters order ID and the screen shows the information of the order. Customer updates or cancels the order.
Alternative courses of action	If the customer maintains order via fax, phone or face to face, staff should record the messages; staff enters the information and maintains order.

➢ Leave Message

Leave message	
Actor	Customer
Precondition	Customer wants to inform company some information
Postcondition	Customers email company, make phone call or face to face to inform company about products/services, complaint and other messages.
Description	Customers message company through email, phone or face to face and the messages are recorded for further handling.
Basic action	Messages is accepted and recorded in the system.

Alternative courses of action	If the customer is a new customer, then create new customer information and a customer ID.

➢ Register

Register	
Actor	Customer
Precondition	New customer come and requires a service.
Postcondition	Name, address, and phone numbers are recorded into system and a customer ID is created
Description	New customer supplies their detail information.
Basic action	Customer information is stored into the system

➢ Online Payment

Online Payment	
Actor	Customer
Precondition	Customers want to pay online.
Postcondition	Payment is accepted or rejected, and information is send to the customer.
Description	Customer use credit card to pay and system response the customer whether the payment is accepted or not.
Basic course of action	Customer credit card number is accepted and transferred to Banking System. Banking System checks the card validation and response the system whether the payment is accepted or not. The results send to the customer.

➢ Maintain Product

Maintain Product	
Actor	Staff
Precondition	Staff maintains products' information by adding, deleting, updating.
Postcondition	Product information is updated.
Description	Staff maintains products information.
Basic action	Searching product and update products' information.
Alternative courses of action	If the product is new then add product information to the system.

➢ Maintain Staff

Maintain Staff	
Actor	Manager
Precondition	Manager maintains staff information.
Postcondition	Staff information maintained or updated.
Description	Manager wants to add new staff information, update exiting information or delete it from the system. The information will be staff ID, name, address, position, salary etc.
Basic action	Manager search exiting staff and maintain the information.
Alternative courses of action	If staff is new then add new staff information to the system.

Appendix II: EJB Enterprise Beans design diagrams

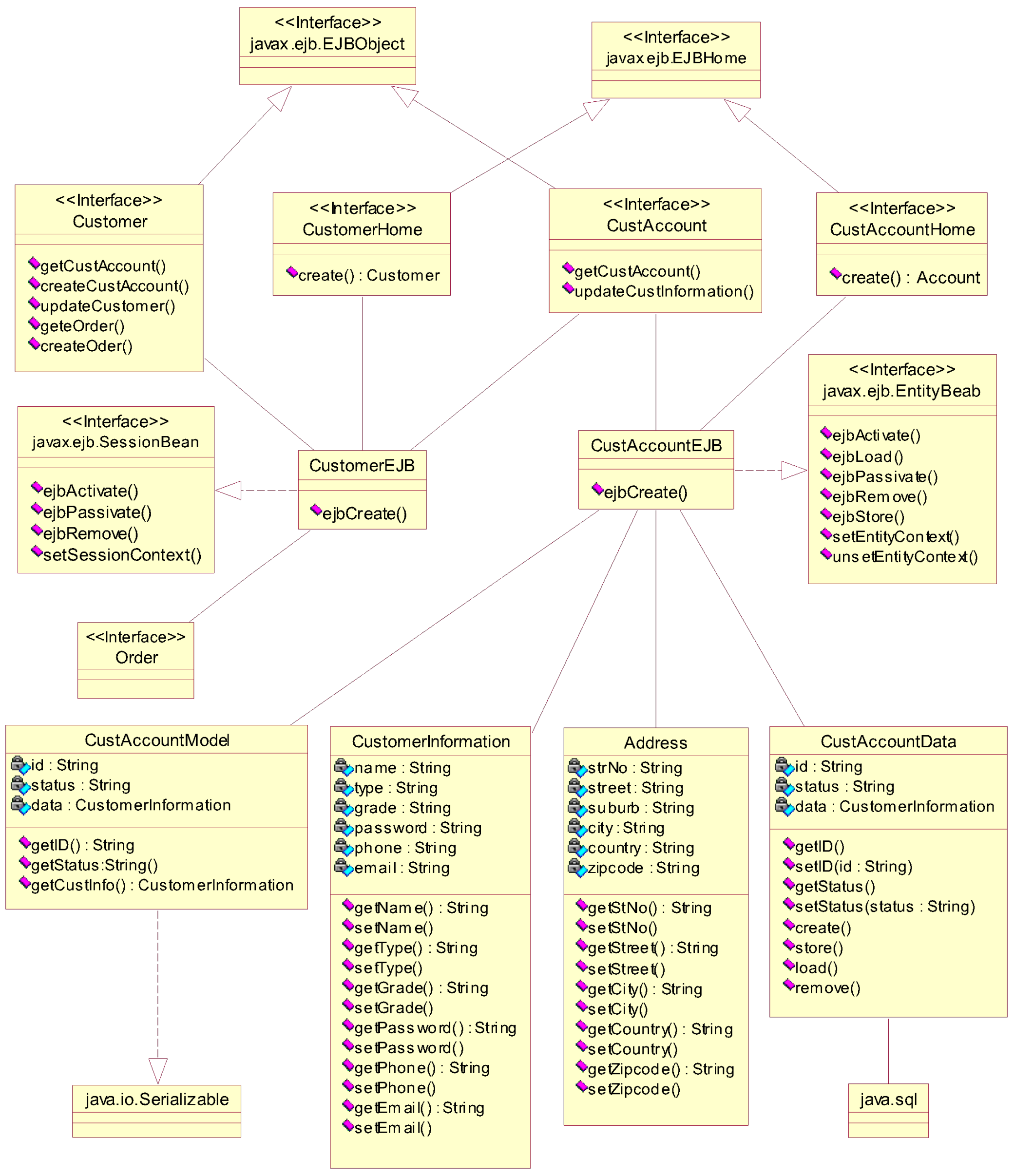


Figure Appendix II – 5 Customer session bean and customer account entity bean

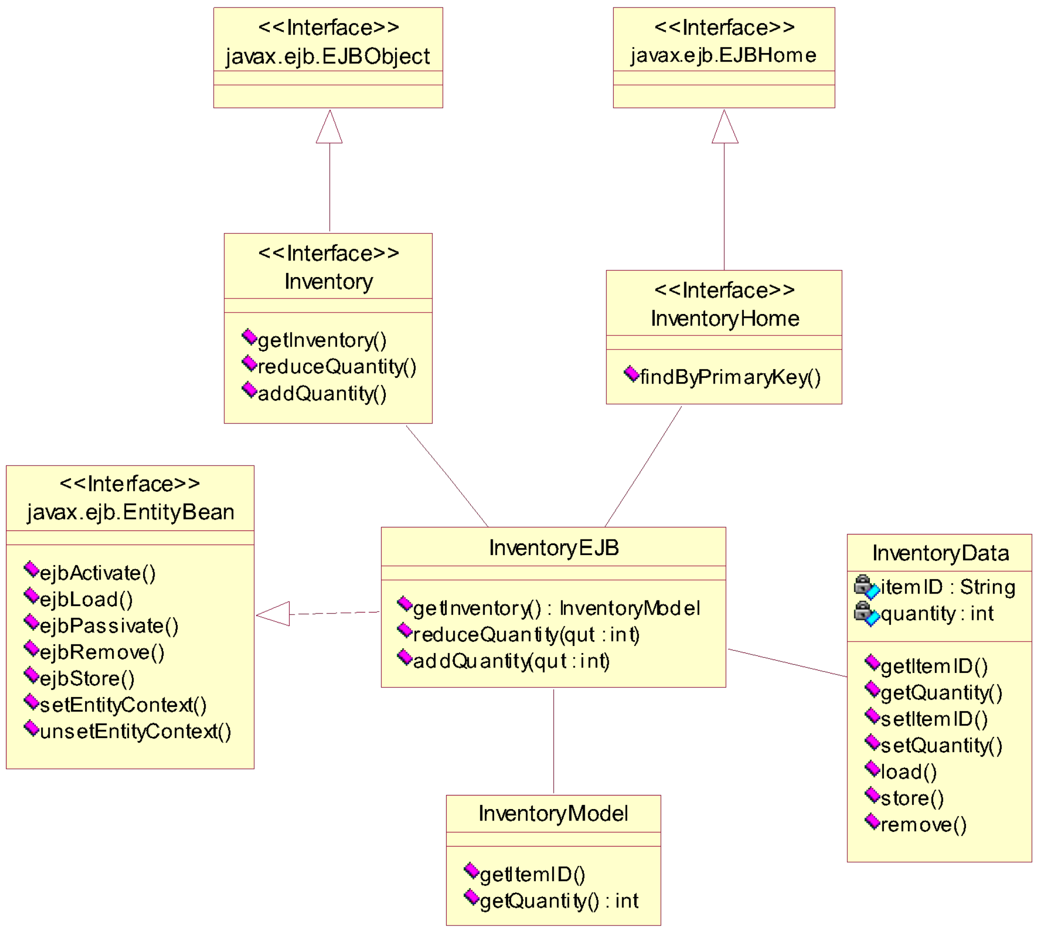


Figure Appendix II – 6 Inventory Entity Bean

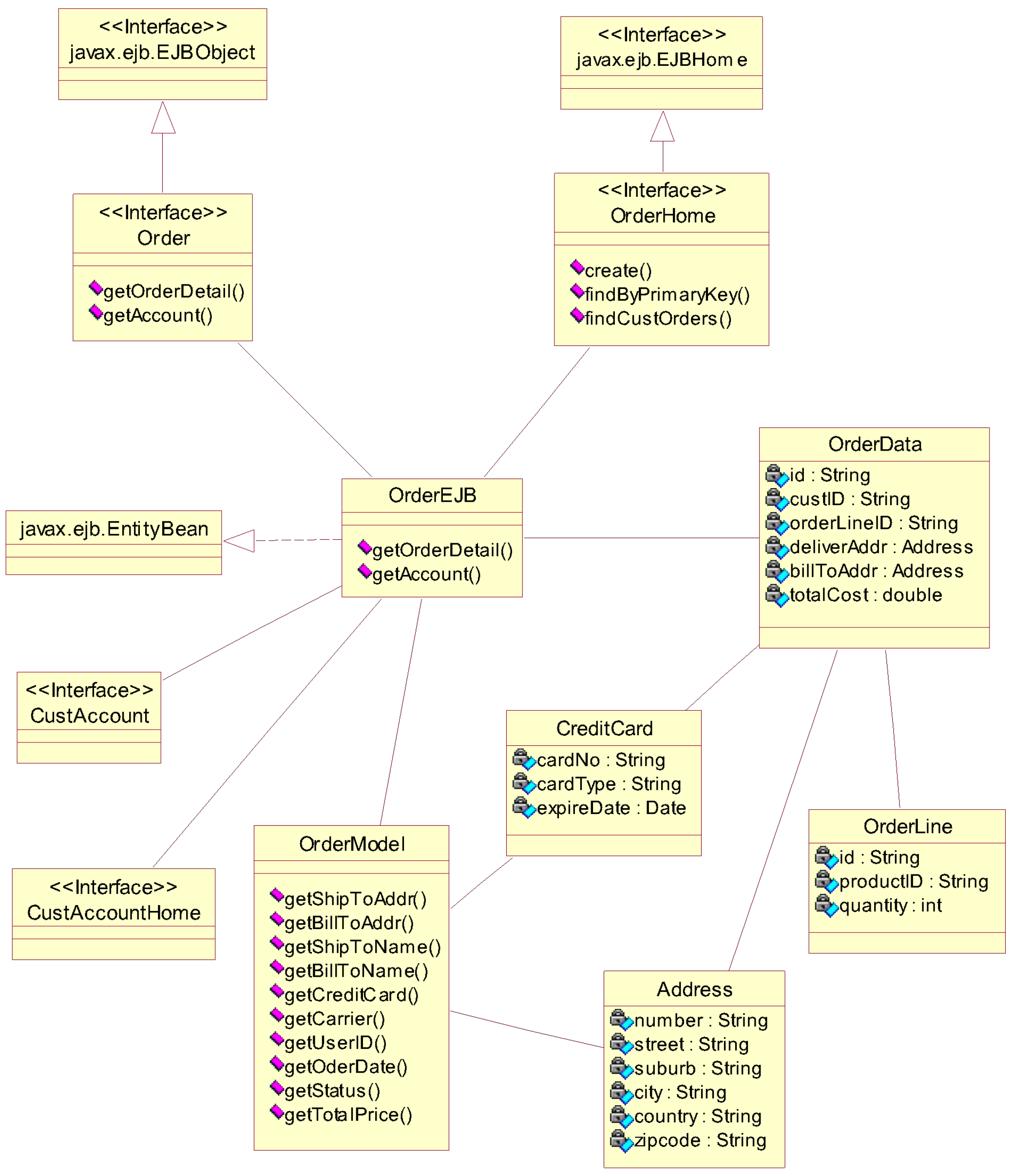


Figure Appendix II – 7 Order Entity Bean

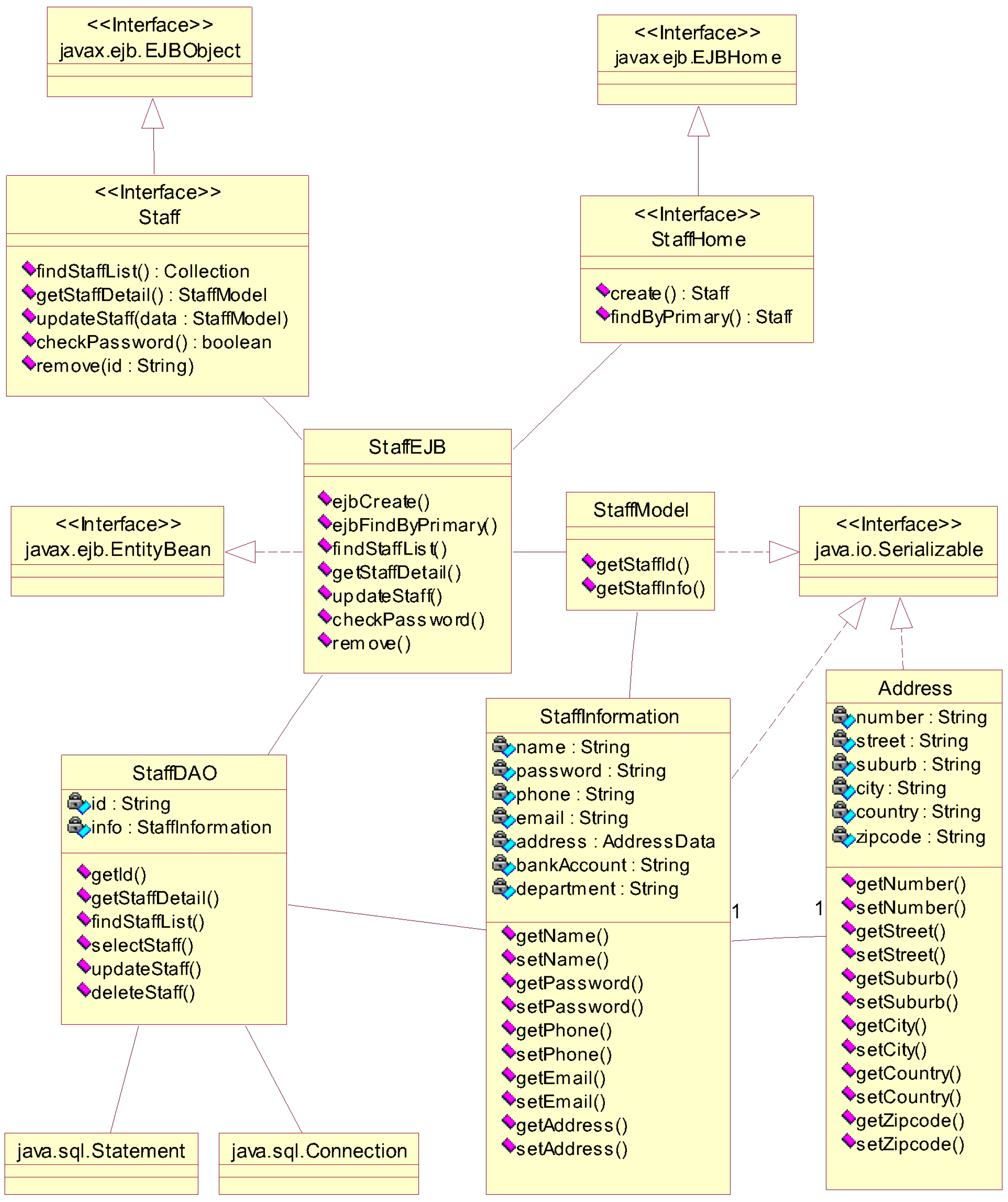


Figure Appendix II – 8 Staff Entity Bean

Appendix III: Component Aspect descriptor

Example of Staff Component Aspect Descriptor

```
<?xml version="1.0"?>

<!DOCTYPE Component SYSTEM "componentaspects.dtd">

<Component Name="Staff">
 <JNDIName>java:comp/env/ejb/staff</JNDIName>
  <Aspects>
   <Aspect name="Performance">
      <Performance>
         <Speed TestNumber=”3” AcceptableSpeed=”5”>
            <Method name="StaffHome.findByPrimaryKey()">
               <Parameter>J2ee</Parameter>
            </Method>
         </Speed>
         <Robustness>
            < Method name="StaffHome.findByPrimaryKey()">
               <Parameter>J2ees</Parameter>
            </Method>
         </Robustness>
      </Performance>
      <Performance>
         <Robustness>
            <Method name="StaffHome.create()">
               <Parameter>J2eeSun</Parameter>
            </Method>
            <Method name="StaffHome.findByPrimaryKey()">
```

```
                    <Parameter>J2ee</Parameter>
            </Robustness>
        </Performance>
    </Aspect>

    <Aspect name="Persistence">
        <Persistence>
            <Retrieve>
                <Method name="StaffHome.create()">
                    <Parameter>J2eeSun</Parameter>
                </Method>
                <Method name="StaffHome.findByPrimaryKey()">
                    <Parameter>J2ee</Parameter>
                </Method>
            </Retrieve>
        </Persistence>
        <Persistence>
            <Store>
                <Method name="StaffHome.create()">
                    <Parameter>J2eeSun</Parameter>
                </Method>
                <Method name="StaffHome.findByPrimaryKey()">
                    <Parameter>J2ee</Parameter>
                </Method>
            </Store>
        </Persistence>
    </Aspect>

    <Aspect name="Transaction">
        <Transaction>
            <Commit>
                <Method name="Staff.changeStaffInfo()">
                    <Parameter>J2ee</Parameter>
                </Method>
                <Method name="Staff.remove()">
                        <Parameter>J2eeSun</Parameter>
                </Method>
        </Commit>
        <Rollback>
                <Method name="Staff.changeStaffInfo()">
                        <Parameter>J2ee</Parameter>
                </Method>
                <Method name="Staff.remove()">
                        <Parameter>J2eeSun</Parameter>
                </Method>
        </Rollback>
    </Transaction>
</Aspect>

<Aspect name="Security">
    <Security>
        <Encryption>
                <Method name="Staff.checkPassword()">
                        <Parameter>********</Parameter>
                </Method>
                <Method name="Staff.changePassword()">
                        <Parameter>********</Parameter>
                </Method>
        </Encryption>
    </Security>
    <Security>
        <Authentication>
                <Method name="Staff.checkPassword()">
                        <Parameter>********</Parameter>
                </Method>
                <Method="Staff.changePassword()">
                        <Parameter>********</Parameter>
                </Method>
                </Authentication>
        </Security>
    </Aspect>
  </Aspects>
</Component>
```