DEVELOPMENT OF BUSINESS MANAGEMENT SYSTEM IN MODEL DRIVEN APPROACH AND BUILDING OF MODEL-DRIVEN-ENABLED DEVELOPMENT ENVIRONMENT

By Ming-Jen Huang

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Takuya Katayama

September, 2004

DEVELOPMENT OF BUSINESS MANAGEMENT SYSTEM IN MODEL DRIVEN APPROACH AND BUILDING OF MODEL-DRIVEN-ENABLED DEVELOPMENT ENVIRONMENT

By Ming-Jen Huang (210205)

A thesis submitted to
School of Information Science,
Japan Advanced Institute of Science and Technology,
in partial fulfillment of the requirements
for the degree of
Master of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Takuya Katayama

and approved by
Professor Takuya Katayama
Professor Koichiro Ochimizu
Professor Atsushi Ohori

August, 2004 (Submitted)

To Dad: your life, your courage, your mercy. I miss you

Abstract

Model-driven approach is a software developing method that proposes programming from higher abstraction level and the computerized model transformation. Current software developing frameworks applies languages that can only be understood by developers to develop software. Users can not understand them, thus the developing materials can not be validated by users. This hampered smooth communication between the users and the developers. Although OMG proposed MDA and defined a set of specifications to develop software from higher abstraction level, their works do not solve this issue. There is a gap that a model-driven software developing framework for enterprise computing that use a language that can be understood by the users and use as the communicating tool between the users and the developers is needed.

This thesis describes the details of ECSDF, a model-driven software developing framework for enterprise computing. ECSDF contains four architectural parts, the Business Model, the object model generator, the source code generator, and the domain model virtual machine. The Business Model allows users and developers to use the same language to communicate and to define the details of software projects. The creation of the Business Model is inspired by the characteristics of enterprise computing: document-centric business activities, request-processing-response model of business activities, and business rules that confine business activities and business entities. Object model and source code generating mechanism are based on the ideas of roles, responsibilities, and collaborations of software objects. The object model generator and the source code generator use the Business Model to generate object model and source code respectively. The mechanism can be simplified to “each system responsibility is taken by a set of collaborative software objects and, in turn, each software object takes smaller responsibilities and collaborates with its neighboring software objects.” There are three types of rules, interaction rules, responsibility rules, and architectural rules to generate object model from the Business Model. They provide a clean and trustworthy way to realize model-driven approach. The implementation for ECSDF, which is called ECSDF-DE, is developed as Eclipse plug-in. The purpose of ECSDF-DE is for practical rapid development and for academic expanded researches and experiments. An evaluation of the effectiveness of ECSDF is performed. The evaluation shows that the generated software system achieves two quality attributes, flexibility and reliability. And it also shows productivity of using ECSDF.

ECSDF fills the gap of the previous works. For the future work, an

autonomous virtual machine that is both executable of the Business Model and adaptable to desired quality attributes will be studied.

Table of Contents

List of Figures

List of Tables

0. INTRODUCTION

This chapter describes background and previous work of my research and what gap my research would like to bridge. It states the overall purpose and the tasks have to perform. Finally it gives the following contents of the thesis.

0.1. Background

0.1.1. Overview of Model Driven Approach

Software developers design software system with software objects[1]. They design those objects with various roles and assign various responsibilities to them. They fill objects with instructions to instruct software objects how to behave in accordance with their roles. Software objects then form a software system to take bigger responsibilities to accomplish desired works. A software system is complex machinery constructed from software objects which affects each other [1].

In 2001, Object Management Group (OMG) proposed a new software development approach, Model Driven Architecture (MDA) [2]. MDA is a development approach and a set of standards that raises the abstraction level of programming to modeling. The general idea of MDA is to develop software system by modeling rather than by creating source code, such as Java or C#. It defines the notion of Platform Independent Model (PIM) and Platform Specific Model (PSM) to present technology-independent concepts and technology-dependent concepts respectively. Then the computerized model transformation transforms PIMs to PSMs to source code. Currently MDA uses Unified Modeling Language (UML) as its standard modeling language. In short, the core concepts of MDA is raising programming abstraction level and utilizing the computerized model transformation.

0.1.2. Benefits of Model-Driven Approach

Why raising programming abstraction level and utilizing the computerized model transformation are necessary? How the software developers can be benefit from the model-driven approach? As mentioned

---

[1] This thesis only discusses object-oriented software system.

before, a software system is complex machinery. It is an artifact that designers of the software system try to comprehend real world facts and to reconstruct these facts as conceptual software objects. The transformation from real world facts to software objects is not trivial at all. The mapped software system has also to modify accordance with the ever-changing world. Doing these kinds of transformation manually is not easy. Any small change in the real world may have larger effects on the software system which can result in further effects on other parts of the system. With the help of the model-driven approach, these issues can be addressed. By model-driven approach, the developers only consider how to perceive real world facts. The complex transformation to software objects are handled by machines. It raises the productivity of software development. The resulted software system can be both flexible and reliable.

0.1.3. Enterprise computing and Model-Driven Approach

Model-driven approach can bring large benefits specially to enterprise computing. Although there are many researches on or commercial implementation of model-driven approach, the researches or the implementations for enterprise computing are rare [3]. Within this ever-changing business environment, software system that is developed for enterprise suffers pressure of matching the speed of changes and frequency of changes. Enterprises change their business behavior and business rules to meet their operating environments. In the same time, the software systems that are designed for them are also modified to meet these changes. Software system is complex and it should also satisfy multiple goals, such as reliability, flexibility, or security etc. By implementing model-driven approach concepts, software system can match the desired features easily and rapidly, and also be modified without corrupting the whole system.

0.2. Previous work

Wirfs-Brock and Wiklerson proposed the responsibility-driven approach for software design [4]. They focused on the very essential of software objects responsibility, actions an object was responsible for and information an object shared. Beck and Cunningham proposed using CRC Card to record classes, responsibilities, and collaborations for object-oriented software design [5]. These three aspects provided a clean and easy-to-comprehend framework for object-oriented design. Responsibility-driven approach was leveraged by characterizing software objects into different stereotypes by Wirfs-Brock Rebecca [6]. These stereotypes were characterized by their object behavior. The characterization provided a clean way to define objects.

Model Driven Architecture (MDA®) was proposed by OMG [2]. OMG submitted a set of specifications trying to cover every aspect of software development. One of the most important specifications was Unified Modeling Language (UML) [7] that was the de facto standard modeling language for MDA. The other was Meta-Object Facility [8]. It was designed as a common meta-model for all other specifications. It could be easily overwhelmed by quantity and thickness of the specifications. All in all, the idea was higher-abstraction-level software developing materials (which were coined as Platform Independent Platform) should be specified by a set of specifications for various purposes that were based on the same common meta-model, and the specifications also defined how to transform the developing materials into lower-abstraction-level design materials (which were coined as Platform Specific Platform). Then these lower-abstraction-level design materials were further transformed into implementation materials, such as source code or database schema. It sounded like if I gave you a dictionary, and then by looking it up, you could speak foreign language well. Without mentioned it could be unsuccessful in marketing [9], it failed in two points. The first was the idea overlooked complex and changeability of underlying technologies. The second was it overlooked the complex and diversity of software modeling to real world facts.

Regarding the first point, existing framework or programming environments were too complex to map. Java 2 Platform, Standard Edition, Version 1.4.2 [10] had totally 2723 classes and interfaces. Java 2 Platform, Enterprise Edition, Version 1.3 [11], had 432 classes and interfaces. One of the most popular web frameworks, Struts [12], had 284 classes and interfaces. Nuance existed across them and fluidity among them made it very hard to have any direct mapping from analysis artifacts to design artifacts to implementation artifacts [9].

Regarding the second point, the most important thing of software development was the intention of the problem domain. If UML should be the next generation programming (modeling) language, then if should be productivity and expressiveness enough to replace current text-based programming languages. To examine this point, we should consider who would use it, i.e. who would be the customer of UML in MDA? If software developers would be the customers, UML did not provide productivity and expressiveness to replace current text-based programming languages [13][14]. If end-users would be the customers, the raising of programming abstraction level was needed. If it rose to end-user level, then UML should also provide productivity and expressiveness to the end-users. But different domain had different needs. UML might be useful for some domains, but couldn’t incorporate enough semantics to be a universal modeling language [9][14].

0.3. Gap in the Research

There is a gap that a framework for enterprise computing based on the model-driven approach which provides programming from higher abstraction level and the computerized model transformation is needed.

Programming from higher abstraction level is model-centric. The models shall be reviewable by non-technical people. And the transformed software system shall be reliable and flexible.

0.4. Purpose and Tasks of the Current Research

The purpose of the research is to build a software developing framework for enterprise computing that is based on the model-driven approach and to build real software system for enterprise computing to verify the framework. For the purpose, there are four tasks:

1. To build a business management system (by hand)
2. To conceive a mechanism for mechanical object model and source code generation
3. To construct a modeling language for enterprise computing
4. To evaluate the software developing framework

Before exploring into the model-driven jungle, a real and workable demo application is needed. Firstly, the business management system (BMS, for short) is developed by hand-coded. It shall provide features for basic business operations, such as sales, procurement, and inventory management etc. It shall follow currently known “good design principles” of enterprise application. It shall meet two architectural characteristics, maintainability and flexibility. By developing this demo application, the resulting specifications can be used to build another system by the model-driven framework. The hand-coded BMS can be evaluated against the machine-generated BMS.

Secondly, the mechanism for the object models and the source code generation shall be conceived. The mechanism is based on the idea of roles, responsibilities, and collaborations of software objects. Responsibilities are the things the system has to do and the information it shall provide. Roles are abstraction of object characteristics [1]. By distilling into these higher abstraction concepts, mechanical transformation can be achieved.

Thirdly, business models used to describe daily business activities of enterprises shall be constructed. For the users and the developers to describe the business world facts that could be handle by machines, a domain specific

language for enterprise computing is needed. The resulting business models are expressible enough to capture the essential business world facts and also rigorous and unambiguous to be processed by machines to produce the object models and the source code.

And finally, the evaluation of the framework will be performed. The evaluation is performed against the hand-coded BMS. To this end, a development environment bases on the framework shall be built. It will be used to build another business management system but has the identical specifications as the hand-coded BMS. The specifications will be used to develop the business models. The business models are then used to generate the object model. The generated BMS is built with goals of reliability and flexibility. The main focus of the evaluation is if the generated object model can reliably accomplish responsibilities define in the business models and can flexibly match any changes in its specifications without affecting other parts not concerning with the changes.

0.5. Organization of the Thesis

The following chapters are organized into three major parts.

The first part, chapters 1 - 4, introduces the proposed model-driven framework for enterprise computing, Enterprise Computing Software Developing Framework (ECSDF for short). Chapter 1, Overview of ECSDF, gives a short introduction to ECSDF. Chapter 2, Role Stereotypes of Objects, explains the fundamental ideas of the object model generator in ECSDF. Chapter 3, Modeling Language for Enterprise Computing, states a higher-abstraction-level domain specific language, the Business Model, which is used to describe the business activities and the business entities of enterprise computing. Chapter 4, Business Models and Model Transformations, gives the details of mechanical transformations from the business model to the software object models to source code.

The second part, chapters 5 – 6, presents a development environment implemented for ECSDF and its applications. Chapter 5, Development Environment for ECSDF, describes the Eclipse-based developing environment (ECSDF-DE) developed for ECSDF. Its features, architecture, and implementation are given. Chapter 6, Applying and Evaluating of ECSDF, discusses the details of applying ECSDF and using ECSDF-DE to develop a business management system and of evaluating the resulted system.

And the final part, Chapter 7, Conclusion, is the conclusions of the

thesis.

Appendix A, Specification of BMS, gives a general specification of the demo software system (BMS for short).

Appendix B, Design of BMS, shows the design materials of the hand-coded BMS and the ECSDF-generated BMS.

1. OVERVIEW OF ECSDF

This chapter describes the conceived model-driven software developing framework for enterprise computing. It is called enterprise computing software developing framework (ECSDF for short). ECSDF is based on the concepts of programming from higher abstraction level and the computerized model transformations. ECSDF mainly contains four architectural parts, the Business Model, the object model generator, the source code generator, and the virtual machine. ECSDF uses only one language for both users and developers to describe business activities and business entities of enterprise computing. ECSDF bases on the ideas of MDA proposed by OMG but don't comply with any OMG's MDA specifications.

1.1. What is ECSDF?

ECSDF is a software developing framework that helps both users and developers to build software by specifying developing materials that are readable by non-technical people and are also computable by machines. It can bring time to market and customer satisfaction to the users and the software developers. The generated software systems shall be reliable and flexible.

A software system is reliable when it is both "doing the right things" and "doing the things right". To produce a reliable software system, specifications shall be readable by non-technical people and also be computational by machines. ECSDF solved these two requirements simultaneously.

1. Readable specification for customers[1]: Specification shall be readable by customers. Without reviewing specifications by the customers, the first part of software reliability, "doing the right things", can not be achieved. The customers usually do not understand jargon of software or obscure mathematical symbols. A model that is used as specifications to describe behavioral part and structural part of real world shall state in a language that can be understood and verified by the customers. The model shall also be expressible to represent problem domain. The only way to have specifications be readable, verifiable, and expressible is to adopt a language that is used by customers in their daily activities. Within ECSDF, four models, which

[1] Customers are the end-users of the developed application. Customers, users, and end-users are used interchanged.

are called the Business Model collectively, are included. They capture business entities and business tasks that customers performed in their daily business activities.

2. Computational specifications: Specifications shall be processed and transformed to source code by machines to achieve the second parts of software reliability, “doing the things right”. Specifications are statements to human’s perceived world. To have specifications computable by machines, a rigorous and unambiguous mapping between specifications and software objects is needed. ESCDF abstracts system behavior and object characteristics to provide a rigorous and unambiguous model transformation that generates structural part and behavioral part of software objects from the four business models. And it can even further generate source code.

A software system is flexible when it meets the changes of real world facts it maps. One of the problems of current software development is the ripple effect of changes. ECSDF abstracts the software object characteristics to help to transform the business models. If there is any change in real world facts, the object model generator and the source code generator transform the business models rapidly, and also preserve reliability and flexible.

1.2. An Architecture Overview

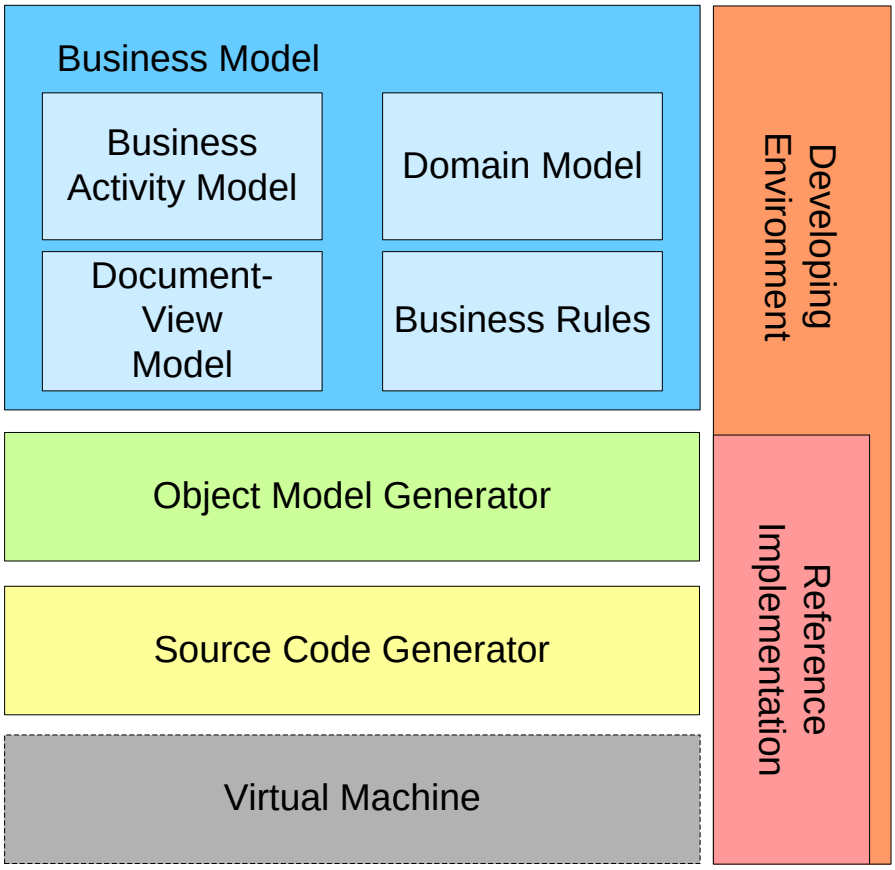


Figure 1-1 Architecture of ECSDF

Figure 1-1 shows the architecture of ECSDF. ECSDF provides the ability to develop software system for enterprise computing from higher abstraction level. The customers and the developers work together to describe requirements and specifications of an application in a language that customers understand. The requirements and the specifications are described in the ***Business Model***, which consists of business activity model, document-view model, domain model, and business rules.

The object model generator is used to generate object models in accordance with the Business Model. The generation is based on the concept of role stereotypes of software objects, responsibilities of the software systems and the software objects, and collaborations of the software objects. Business entities and business tasks described in the Business Model are considered as software system responsibilities. These software system responsibilities are

behavior that software systems must perform and information they must provide. These larger responsibilities are decomposed into smaller responsibilities and are assigned to the role stereotypes. Role stereotypes are abstractions of software characteristics. Each of role stereotypes carries out these smaller responsibilities. The source code generator is used to generate implementation code specific to a programming language. The generating mechanism of the object model generator and the source code generator are accomplished by rule-based engines. Pre-defined rules describe how to assign responsibilities in the Business Model to objects and source code.

The last piece of ECSDF is the virtual machine that runs a software system by only specifying the four business models. It is not considered yet in the thesis, and it will be the studying subject of my dissertation.

1.3. Motivation of ECSDF

Human form various communities who have the same interests. Software objects in an application also form various object communities. Their interests are to conduct a set of software system responsibilities. Software Objects play various roles and interact with each other to form collaborations in the community to take all the designated software system responsibilities. Objects in the community are smart. They have different and clearly defined roles and take one or more object responsibilities. Their responsibilities are decomposed from software system responsibilities. They know what they have to do. Some of them may provide information to others. Some of them may provide services to others. Some of them may have to interact with other communities. Some of them may work as a commander to direct the interactions within an object community. ECSDF is based on these concepts of roles, responsibilities, and collaborations.

Before I go further discussion, clearly definition of terms appear above shall be given. The following definitions are quoted from Rebecca Wirsf-Brock [1]:

- An ***application***[1]: a set of interacting objects
- An ***object***: an implementation of one or more roles
- A ***role***: a set of related responsibilities
- A ***responsibility***: an obligation to perform a task or know information
- A ***collaboration***: an interaction of objects or roles (or both)

---

[1] Application and software system are used interchanged.

Generally development of software system starts from gathering requirements from customers. When gathering requirements, technical people try to understand what customers (who pay money to build the application) want. Gathering requirements is no easier than coding. If one just asks customers *"What do you want the system to do?"* directly, customers always answer *"Well, I am not sure what I want…"* Gathering requirements becomes a serious discipline which is called ***Requirement Engineering*** [15]. One of the most obstacles of gathering requirements lies on customers who provide information and technical people who gather information speak different languages [16]. Customers have their own language to describe their business activities. Technical people have their own language to describe their knowledge to build an application. This kind of language barrier hampers customers and technical people to communicate with each other smoothly.

After some requirements are gathered, technical people make decisions about how to describe the requirements in their language. This stage is generally called ***analysis***. Diagrammatic representation (such as UML) may be used. For example, UML class diagrams are used to define the structural relationship of conceptual elements of problem domain. UML interaction diagrams are used to depict interactions of those conceptual elements. These diagrams help technical people to understand knowledge related to problem domain. They are also used to communicate with customers and developers. When to transform from requirements to analysis materials, the intention of customers' requirements may be misunderstood and misinterpreted by technical people. As mentioned before, this is due to the language barrier.

After technical people learn important facts in the analysis stage, it is time to do design in software terms. The design works contain software objects that do different works to accomplish desired features. There is a misunderstanding that software objects are designed to represent real world facts directly. When doing design, we can reinvent the real world in the domain of software even the design materials may largely deviate from the world the software intent to manage [1]. Developers may misunderstand and misinterpret the analysis materials again in this stage.

After some design materials are based, developers implement application with source code. Developers fill software objects with the application logic or domain logic base on the design materials and his knowledge to the problem domain and the software domain. Again, the transformation from the design materials to source code might be misunderstood and misinterpreted.

Since customers who use the application and technical people who

develop the application speak different languages and the transformations of materials in various stages of software development can lead to misunderstanding and misinterpretation, I came up with an idea that if it is possible to raise the abstraction level of programming to have customers and technical people both speak the same language to describe the specifications of software system and then applies mechanical transformations to transform the specifications to software system? ECSDF includes the Business Model to help customers and developers using the same language to describe the specifications. It also includes mechanical model transformations mechanism to transform the specifications to the object model and source code without distorting semantics in the specifications.

1.4. Why ECSDF is NOT MDA®?

ECSDF is NOT an MDA-compliant framework because it does not comply with any specifications that are specified by OMG. ECSDF even not incorporates the now prevailing phases, platform independent model (PIM) and platform specific model (PSM) within it although I formerly used them in my thesis proposal. I have mentioned why OMG's MDA will not be a future of software development from generally in Section 0.2. The following discussion describes specific reasons that indicate why I do not try to be MDA-compliant.

Most of the end-users do not say any word of UML

The first reason is very simple and clear. Using UML as the language to communicate with customers is not possible. First and the foremost point to be successful in a software project is the communication between the customers and the developers shall be smoothly. Using a language that the customers can understand is necessary. Specifications shall be read and validated by the customers. UML is a unified modeling language for software development. It is not a universal modeling language for every domain in the world. Technology is not almighty. Saying UML will be a general-purpose language for all problem domains is only a wishful thinking. Every domain has its own needs and requirements. We shall apply languages that customers use in their daily activities to describe specifications of software projects.

UML works for specific domains but not all domains

UML is good at giving visual representation of source code. It is especially useful to present structural aspect of source code. But it is limited to present behavioral aspect of source code [17]. State charts work for specific domains but not for some domains as complex as enterprise computing. Writing algorithms in text-based programming languages or textual pseudo-code is much productivity and easily comprehensible than diagrammatic representation,

such as UML.

There is no model that can be “platform-independent”

Every model is platform-dependent. Even defining PIM in OMG’s specifications is platform-dependent. Regarding this point, it needs more explanations. In the following, I will use a very simple example, a classical Hello, World, to explain this reason.

Following is Java source code for printing “Hello, World!” to a standard output device.

```
public class MyApp {
	public static void main(String[] args) {
		MyApp app = new MyApp();
		app.printHelloWorld();
	}

	public void printHelloWorld() {
		System.out.println("Hello, World!");
	}
}
```

Figure 1-2 “Hello, World” Java Source Code

This Java program contains a `MyApp` class which has a static entry point `main` method and an instance `printHelloWorld` method which prints “Hello, World!” to the output device. It is surely a PSM according to OMG’s definition. It is dependent on Java platform.

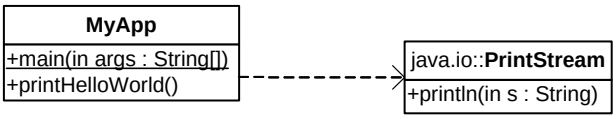


Figure 1-3 “Hello, World” UML Class Diagram Dependent on Java

Figure 1-3 is a UML class diagram that gives diagrammatic representation of the Java source code in Figure 1-2. It is still a PSM because it is still dependent on Java platform. UML class diagram only provides concise visual representation.

If we need to represent the Hello World application as a PIM that independent from any technology platform, such as Java, the UML class diagram may look like Figure 1-4. Can we now tell what the class intents to do? We can not. There is nothing in this diagram tells us what the meaning of the operation `main` and `printHelloWorld`. From the naming of the methods, we may guess it is a class that print string “hello world”. We can tell what the class

does because if its good naming. But we still do not know where the string “hello world” may go to.

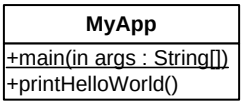


Figure 1-4 “Hello, World” UML Class Diagram Independent from Any Programming Language

My argument is how a model can be possible to be independent? A model always bases on something to describe other things. As the example shows in Figure 1-4, we can guess what the class `MyApp` does from linguistic form. But if we want to describe rigorous and unambiguous specifications of the operation `printHelloWorld`, we need libraries. There is no such I/O library defines in OMG’s standards. It is also in contradiction to the definition of PIM which states it shall be independent from any technology platform. It is in a dilemma. Fowler described this point in his web site [18].

Frankel argued [3] that a PIM shall specify what the PIM is independent from. It is also strange because a model can possibly be independent from many things. It may be independent from an operation system or from a hardware platform. Saying a model is independent from a thing is meaningless.

OMG’s specifications are as much as platform

Can it possible to provide another universal model to be an abstraction level above any technology platform? The answer is surely not. OMG provides many specifications such as EDOC, CWM etc which are indented to be the universal models. If changeability is the cause that OMG to create these specifications, then the problem may also happen to them. In fact, all these specifications are as much as platform as Java [18]. Fowler gives PIM a witty name, Platform Independent Malapropism [18].

Trying to completely dispel technology considerations from initial stage of software development is another wishful thinking, and not very practical. Without considering technology aspect from initial stage will only drag down software projects in later stages. Current enterprise application development is hard and complex. Implementation details may be deferred until later developing stages. But considering executing environments and technology choices are necessary for developing software projects successfully. All these factors affect developing ideas and decisions. Even when applying a model-driven approach, such as ECSDF, we still have to put technology aspect up front, or we will not be possible to find complete system responsibilities to be carried out by software

system.

ECSDF is not MDA® but MDA

From statements above, it may have an impression that I against model-driven approach. It is not the case. All in all, I will not use:

- UML,
- terms of PIM and PSM, and
- any specifications from OMG

to develop ECSDF. They may be helpful for specific domains but not for other domains, at least not for enterprise computing. Saying these things is a future of software development is too early. The ideas within MDA are beneficial. But we need more thorough studies and considerations about raising programming abstraction level and computational model transformation, rather only try to match the steps of OMG.

2. ROLE STEREOTYPES OF OBJECTS

This chapter describes the fundamental concepts used in the object model generator. A simple example is represented as the illustration of the concepts. With regard to object generation, the abstraction of object characteristics is needed. Software objects play various roles in a software system. Roles can be categorized into six role stereotypes. They follow a set of rules to interact with. The set of rules are called collaboration patterns. The object model generator of ECSDF applies collaboration patterns to generate a set of collaborative objects. The set of collaborative objects are called collaboration groups.

2.1. Hello User Example, Episode I

One day, the only architect, system analyst, and developer of ACME software Inc., Michael, wins his first contract to create an application. The requirement is to print a string “Hello” and a user name which is typed by a user to a terminal. The specifications are described as the following:

- A user of the application types a name on screen.
- The application then has to show “Hello” and the name on the screen. For example, if a user types a name, Jenson, then it prints “Hello, Jenson”.
- The screen is a terminal type input/output device.

To make a change tracking, Michael calls it “HU 1.0”. After a deep thought, Michael decides to create an object to store the string “Hello” and to print out to the terminal. The class diagram is shown in Figure 2-1 in UML.

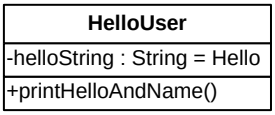


Figure 2-1 HU 1.0 Class Diagram

The application is finished with careful implementation and thorough test. Before he hands the application to the customer, the customer (who pays money to Michael) tell him the requirements are changed. It is fine, anyway, a customer, no matter what blood type he[1] has, always changes his mind. The new

---

[1] It does not imply the customer is male or female; “he” or “his” is used hereafter for convenience sake.

requirement is to print a “Hello” which is concatenates by the user name which is further concatenated by a string “what a beautiful day”. The specifications are described as the following:

- A user of the application types in his name on the screen.
- The application then has to show “Hello”, follows by the typed name, and follows by “what a beautiful day” on the screen. For example, if a user types a name, Jenson, then it prints “Hello, Jenson, what a beautiful day”.
- The constant strings “hello” and “what a beautiful day” shall be stored in a text file.
- The screen may be a terminal type input/output device or a web page browser.
- More screen types may be added in the future.
- Storage of the constant strings may be changed in the future such as using a database instead.

Again, Michael gives it a tracking name, “HU 1.1”. The specifications add more constrains and need much thorough consideration. The main changes of the new requirements are listed as the following:

- The constant strings shall be stored in a text file
- Different I/O device is added
- The persistent storage and I/O device may be changed in the future

To cope with the new changes, Michael first comes up an execution model in a request-processing-response fashion. There are two outside actors to the system, one is the customers and the other is the string storage. The actors do not belong to the application but the application has to interact with them. Users issue request, type their name on the I/O device. The application processes operations according to users’ request, which is to retrieve the strings from the text file. The application made a response to the users, which is to concatenate the constant strings and the typed name and to display the assembled strings to the users.

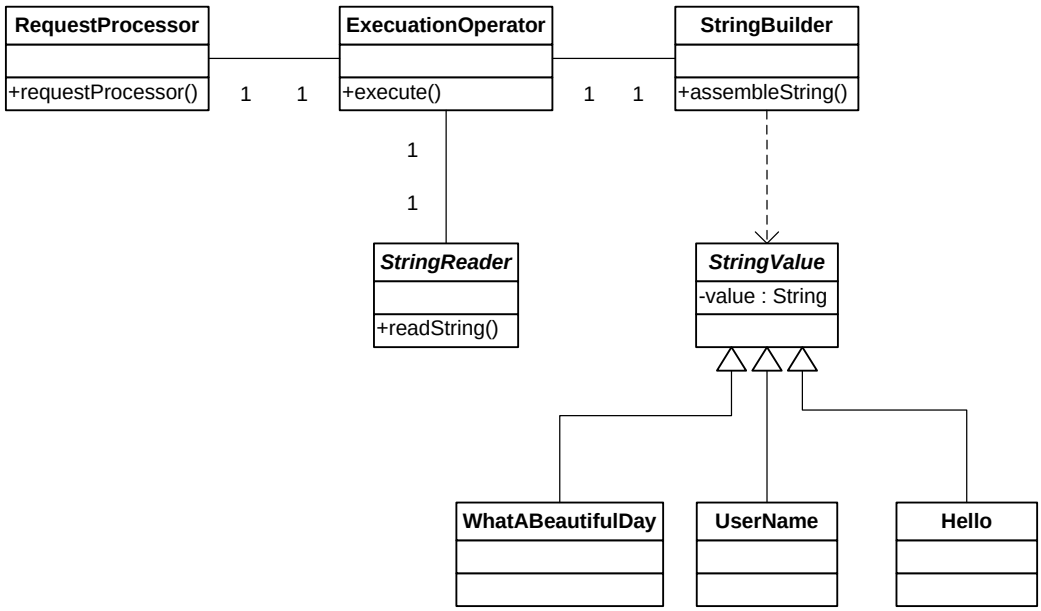


Figure 2-2 HU 1.1 Class Diagram

According to the execution model, Michael makes a change to the object model. The new class diagram is shown in Figure 2-2. An abstract `RequestProcessor` class is responsible for communicating with the users. It extracts the user name from incoming request message. It then hands the extracted user name to an `ExecuationOperator` class. The `ExecuationOperator` class has the responsibility to work as a director which mange the whole operation of the application. An abstract `StringReader` class is responsible for reading strings from storages. The implementation specific to a type of storage media is leave to its subclasses. A `StringBuidler` class has a responsibility to concatenate constant strings and the user name to a complete string to return to the I/O device. The `ExecuationOperator` class directs subclasses of the `StringReader` class to read stored strings and then asks the `StringBuilder` class to assemble the whole string. It then returns the assembled string to the `RequestProcessor` class. The `RequestProcessor` is also responsible for displaying the execution results to the I/O device.

Besides the classes mentioned above, Michael also defines a set of domain-specific classes to present the concepts of problem domain. An abstract `StringValue` class represents those various constant strings. Three inherited classes, `WhatABeautifulDay` class represents “what a beautiful day”, `UserName` class represents user typed name, and `Hello` class represents “hello”. The `StringBuilder` class operates on the domain objects and sends

the response to the `ExecuationOperator` class.

2.1. Role Stereotypes

As we can see from the example, objects in an application take responsibilities to achieve larger responsibilities. Each object plays one or more clearly defined roles. The roles are categorized as a number of role stereotypes. Role stereotype characterizes roles and responsibilities an object takes. The example is simple, but it demonstrates a set of stereotypes in a common software system.

The `RequestProcessor` class plays a role as an ***interfacer*** to interact with the users. It accepts request from the users and responds results to the users.

The `ExecuationOperator` class plays a role as a ***controller*** to direct the operations of the application. It takes the responsibility to ask other classes to read strings from persistent storage, to build strings from domain-specific objects, and to display the resultant string to an I/O device.

The `StringReader` class and `StringBuilder` class plays a role as ***service provider***; it provides persistent service and string operation service to the controller class.

The domain-specific object hierarchy, the abstract `StringValue` class and its inherited subclasses play a role as ***information provider***s; they store strings. Other classes that need the values ask them to provide information.

By using the role stereotypes, we can simplify objects design according to the roles an object plays. And according to their interaction relationships, the collaboration of a set of object can be decided.

In *Object Design: Roles, Responsibilities, and Collaborations*, Wirfs-Brock defines six stereotypes and their responsibilities as the following [1]:

- Information holder - knows and provides information
- Structurer - maintains relationship between objects and information about these relationships
- Service provider - performs works and, in general, offers computing services
- Coordinator - reacts to events by delegating tasks to thers
- Controller - makes decisions and closely direct other `s actions
- Interfacer - transforms information and requests between distinct parts of our system

In the following section, the collaboration patterns, what role stereotypes are permissible to interact with what other role stereotypes, are described

2.2. Collaboration Patterns of Role Stereotypes

A software object plays one or more clearly define roles in an application. Roles can be categorized as role stereotypes that follow a set of rules to constrain themselves that other role stereotypes they interact with. I call the set of rules as ***collaboration pattern***s. Each collaboration pattern defines two role stereotypes to interact with. The collaboration patterns are arbitrary defined based on the domain under consideration. In the thesis, a set of patterns for enterprise computing is defined. A role stereotype actively interacts with the other role stereotype. The role stereotype which is called may replay with response. For example, if a collaboration pattern defines interfacers interact with service providers, it implies interfacers may ask service providers to do some works but not vice versa.

For each role stereotype, a table (see Table 2-1) is used to show if a role stereotype is permissible to interact with other role stereotypes. First column presents the role stereotypes. Second column presents if the role stereotype under discussion is permissible to interact with the role stereotype listed left. A role stereotype not only interacts with other role stereotypes, it may also interact with its own role stereotypes. Symbol ✓ indicates the role stereotype under discussion is permissible to interact with the stereotype listed left. Otherwise, it is marked with symbol ✗. In the following sections, a set of collaboration patterns for enterprise computing is described.

Table 2-1 Collaboration Patterns Table

Role Stereotype	Interact With
Information holder	✓
Structurer	✗
Service provider	✗
Coordinator	✓
Controller	✓
Interfacer	✗

2.2.1. Information Holders

The collaboration pattern of the role stereotype information holder is

shown in Table 2-2.

Table 2-2 Information Holder Collaboration Pattern

Role Stereotype	Interact With
Information holder	✓
Structurer	✓
Service provider	✗
Coordinator	✗
Controller	✗
Interfacer	✗

Information holder rarely interacts with other role stereotypes. After all, as its name implies, its responsibility is hold and provide information. A best way to manage information holders is to provide a factory object or an aggregate object as a structurer. A structurer works as a gateway to all information holders it knows. Sometimes an information holder needs services from other service providers, such as, a logging service provider.

2.2.2. Structurers

The collaboration pattern of role stereotype structurer is shown in Table 2-3.

Table 2-3 Structurer Collaboration Pattern

Role Stereotype	Interact With
Information holder	✓
Structurer	✓
Service provider	✓
Coordinator	✗
Controller	✗
Interfacer	✗

Structurer is used to organize and maintain a bunch of related objects especially information providers and service providers. Any other role stereotype needs to access them shall ask structurer first.

2.2.3. Service Providers

The collaboration pattern of role stereotype service provider is shown in Table 2-4.

Table 2-4 Service Provider Collaboration Pattern

Role Stereotype	Interact With
Information holder	✗
Structurer	✓
Service provider	✓
Coordinator	✗
Controller	✗
Interfacer	✗

Service provider always does some things. It is the workhorse of an application. It takes responsibility as heavy as to execute a series of business activities, such as creating salary information for the whole company; to responsibility as easy as logging what happens inside a software object.

2.2.4. Coordinators

The collaboration pattern of role stereotype coordinator is shown in Table 2-5.

Table 2-5 Coordinator Collaboration Pattern

Role Stereotype	Interact With
Information holder	✗
Structurer	✓
Service provider	✗
Coordinator	✗
Controller	✗
Interfacer	✗

A coordinator passes information to other role stereotypes. It receives events and asks other role stereotypes to handle events. It is not a smart object. It only connects between software objects. It can be viewed as a downgraded controller. It usually receives requests from an interfacer and then asks a

structurer to provide information from information holders or to provide services from service providers.

2.2.5. Controllers

The collaboration pattern of role stereotype controller is shown in Table 2-6.

Table 2-6 Controller Collaboration Pattern

Role Stereotype	Interact With
Information holder	✗
Structurer	✓
Service provider	✗
Coordinator	✗
Controller	✗
Interfacer	✗

Controller is the upgrade version of coordinator. A controller makes decisions on what next action to take bases on a certain situation. It may hold information internally or may ask a structurer to provide information in order to make decisions. It may also interact with a structurer to provide services when it knows what action to take next.

2.2.6. Interfacers

The collaboration pattern of role stereotype interfacer is shown in Table 2-7.

Table 2-7 Interfacer Collaboration Pattern

Role Stereotype	Interact With
Information holder	✖
Structurer	✓
Service provider	✖
Coordinator	✖
Controller	✓
Interfacer	✖

Interfacer interacts with actors outside of an application or other collaboration groups (collaboration group is discussed in Section 2.3). It is a gateway of a collaboration group. Its responsibility is to interact with outsiders that are not belonging to the collaboration group the interfacer sits in. An interfacer may have to process request messages from outside actors by itself, or it may ask a structurer to provide processing services. After the request messages are handled, it then passes information to a controller or a coordinator in order to decide what action to take next.

2.3. Collaboration Groups

Collaboration patterns provide rules for a machine to generate objects. The object model generator of ESCDF follows the collaboration patterns described above to decide how to form a group of objects. In ESCDF, a group of object following the collaboration patterns is called ***collaboration group***. An application is formed from many collaboration groups. Each collaboration group takes larger responsibilities of software system.

Collaboration groups work as logical grouping units or physical deployment units. It can be physically deployed to different processes or machines (see Figure 2-3). The details of the object model generator in ESCDF are discussed in Section 4.3.

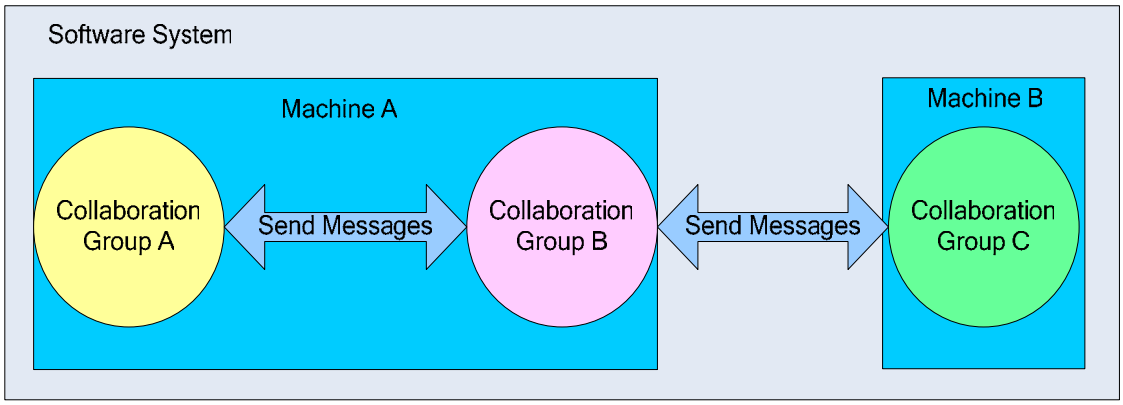


Figure 2-3 Collaboration Groups within Software

Collaboration group A and Collaboration group B sat in a same machine. Collaboration group C sat in another machine. Collaboration group B and collaboration group C communicate through networks.

3. MODELING LANGUAGE FOR ENTERPRISE COMPUTING

This chapter describes the personality of enterprise computing. By observing the concrete and abstract things and behavior within a company, we can abstract them to help us to build computational models. Within ECSDF, there are four business models to describe business world. They are document/view model, business activity model, business rules, and domain model. They capture the characteristics of enterprise computing: document-centric, business-rules-rich, and request-processing-response-pattern.

3.1. Enterprise Personality

Before delve into further discussion, clearly definition of terms are given:

- **Business operation**: represents an action that is performed on one or more business entities.
- **Business activity**: represents a sequence of business operations that brings business benefit to a company
- **Business entity**: represents a concrete or abstract element in a company

These terms are used within two worlds. One was the real world we are living in (real world facts). The other was the perceived world (software specifications).

Starting a business is not hard, although earning a lot of money may be hard. To develop software system for a company to manage its business information and to help people in the company to perform business activities will be harder. It is because there are no hard and fast rules to do business. Every domain has its own business environments and its own business conventions. Even companies in the same domain do business in their own way. Difficulties are added due to the fast changes of business environment.

In the thesis, I propose a model-driven framework for enterprise computing which is based on the ideas of raising the programming abstraction level and using computational model transformation. For raising the programming abstraction level, we have to know how to abstract business world facts. To this end, we shall first consider the personality of enterprise computing.

3.1.1. Business Activities

A company makes its living by selling goods or providing services to customers. At first glance, goods and services represent very different concepts. Customers who buy goods can place them in a place and the goods occupy a space but it is impossible to place services in a place. They have one thing in common; customers shall pay money or barter for goods and services. A company buys goods or services from other companies and sells goods or assembled goods (or divided goods) to other companies or individuals. From above discussion, people who operate the company, goods or services a company sells and provides, and money a company holds largely constituted business entities in enterprise computing. Business activities operate on the business entities to bring benefits to the company. A list (it is not exhausted) of major activities that concerns the entities (people, goods, and money) is shown bellowed:

- Buy goods or services from other companies
- Sell goods to other companies and individuals
- Provide service to other companies and individuals
- Manage goods stocking.
- Manage money flow
- Manage customers data
- Manage employee data

3.1.2. Recording Business Activities

A company not only performs the activities list above. It shall also record information of the activities in documents. The recorded information is placed in formatted documents for business or legal purposes. The recorded information that contained the activities shall be viewed by various roles of people in the company. For example, the owner of the company needs to know total amount of money it earns last month. The company creates and preserves tons of documents with regard to its business activities.

3.1.3. Business Constrains

The company also defines rules about how/when/who to conduct its business. Rules are constraint. They constrain entities and behaviors [19]. For example, to manage the customer’s data, a unique identifier may assign to each customer. The unique identifiers are not arbitrary defined but usually follow a predefined rule. Such as the length of identifier are 15 characters long and can contain only numbers and alphabets. Another example is before a company buys a product from other companies, it shall be approved by the owner of the company. These constraints about how/when/who to conduct business activities

and business entities are called business rules. Business rules are fluid. They change frequently, if not every day. They are modified to cope with the ever-changing environment the company sit in and the changes within itself.

3.2. Modeling Personality of a Company

Personality of enterprise computing, which is discussed in Section 3.1, is summarized as the following list:

- Documents record business activities
- Business activities operate on business entities
- Business rules constrain on business activities and business entities

In ECSDF, the personality of enterprise computing is described in four business models, which are collectively called the ***Business Model***, to cover different aspects of the business. The business models are abstraction of business world facts. They describe business entities and business activities.

3.2.1. Recording business activities in documents

As mentioned in Section 3.1.2, there are tons of documents a company creates and preserves. The primary work of people in the company is to handle the documents. People in the company then judge performance of the business activities and made business decisions by gathering information from these documents. Thus, the primary mission of enterprise computing application is to manage the documents and information gathering from different documents. In order to describe the documents and the gathered information, I propose the ***document/view model*** (d/v model for short).

Documents in the d/v model matches to those formal documents generated everyday by people in the company. View provides a window that shows different aspect of those documents. The views and the documents are a one-to-many relationship. A view represents information collected from the documents or parts of the documents. The d/v model is computable. It provides entities semantics of the human perceived world.

The Documents and the views in the d/v model are not “the documents” of real business world. Since they have to be processed by machines, the definitions of a document or a view shall be machine-readable. For example, in business world, a purchase order contains supplier information, such as its name and address, and purchasing items information. For a computable purchase order, another computable document called supplier address, which contains supplier information, shall be defined in addition to the purchase order. The computable purpose order only contains purchasing item information. It is

mainly for database processing. The relationship of the purchase order of business world and the computable documents in the d/v model is shown in Figure 3-1. Surely the computable supplier address can be combined into the computable purchaser order, but it is inefficient for computation. It is one of the limitations of the d/v model. I would like to solve this mismatch of business world documents and computable documents from the d/v model in my dissertation.

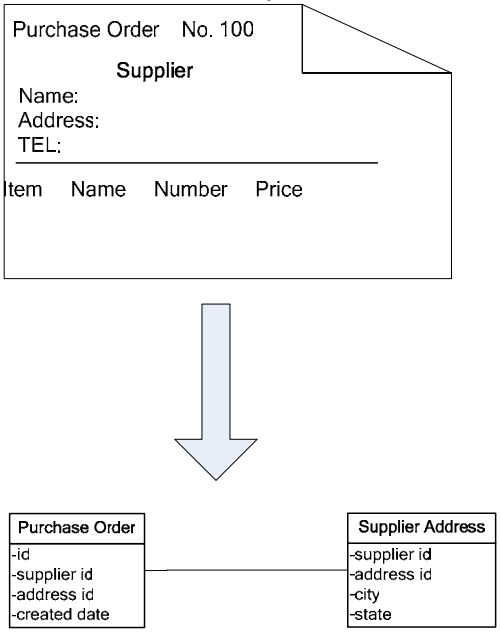


Figure 3-1 Creation of Computable Purchase Order and Supplier Address from Business World Purchase Order

3.2.2. Operating business activities on business entities

Exception d/v model, we need another model to describe how to process the management of the documents and the views.

When people in a company have to create some kinds of documents, they usually start from preparing data that are related to the documents from different sources. For example, if one would like to create a purchase order, he may have to know supplier information, such as supplier's ID, supplier's address etc. He may also have to know the details of purchasing items, such as items' ID, items' prices etc. With the information of the supplier and purchasing items on hand, he can create the purchase order. And then pass it to the supplier and waiting the supplier to acknowledge the purchase with acknowledged messages. Usually they are also in a form of documents. In order to describe this process, I

propose a request-processing-response-pattern model, which is called the ***business activity model***.

The request represents actions of people who collect information from various sources. The processing represents operations conduct on the documents or the views. With regard to enterprise computing, the operation types are generally limited to CRUD (creation, retrieval, update, and deletion). The response represents the results of the processing. It may be a newly created document, modification of an existing document, or an acknowledged message about if the process is success or failure.

3.2.3. Constraining business rules on business activities and business entities

Business process group defines that “a business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business” [20]. Business rules define how/when/who to conduct business activities and constraints on business entities.

4. BUSINESS MODELS AND MODEL TRANSFORMATIONS

This chapter describes the details of the object model generator and the source code generator. ECSDF abstracts the business into higher-abstraction-level model, which is called the Business Model. The Business Model describes how a business executes its business activities on business entities to gain profits. The simple example used in Chapter 2 is expanded to represent the ideas of the generating mechanism. The object model and source code generating mechanism base on the ideas of roles, responsibilities, and collaborations. The responsibilities of software system are assigned to role stereotypes. And then role stereotypes are transformed into software objects. Inheritance and composition of software objects are also based on the same ideas.

4.1. Business Model

The Business Model describes business activities, business operations, business entities, and constraints on them. The business activity model defines business activities and business operations. The d/v model defines business entities. The business rules define constraints on the business activities, the business operations, and the business entities.

4.1.1. Business Activity Model

I propose using business activity model to describe request-processing-response pattern of business activities. As I mentioned in Section 3.1.2, documents play an important role in a company. And as Section **Error! Reference source not found.** described, the request-processing-response pattern is used to describe how people in the company execute business activities to process the documents and the views. The range of a business activity is very broad. It can be from updating customer name in a customer record to the generation of a balance sheet. The customers define what and how they expect the application to perform in business activities. Each business activity represents a system responsibility of the generated application shall take. The meta-model of the business activity model is shown in Figure 4-1.

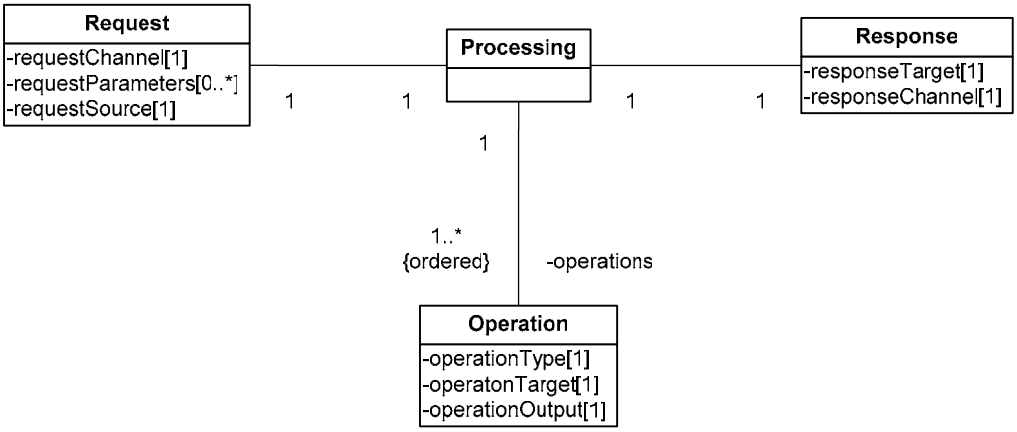


Figure 4-1 Meta-model of the Business Activity Model

A business activity definition is composed from three parts, ***request***, ***processing***, and ***response***.

The request represents a request message which is sent from a user to the application to perform a business activity. The request describes request channel and request parameters. It also contains information of the user. The request channel represents what protocol is used to transmit the request message. The request parameters represent necessary information provide by the user to the application to perform the business activity. For example, a request specifies a request channel of "HTTP" protocol and one request parameter of "purchasing order ID" to search a purchasing order with the designed identification number. Thus the request is "using HTTP to send a request message which contains purchasing order ID".

The processing represents business operations the application has to execute. The processing contains a sequence of operations which represents the decomposed business activity. An operation describes operation type, operation target, and operation output. The operation type represents what type of the operation it is. The operation type can be one of the basic CRUD operations or any extended domain-specific operation. When it is defined as the domain-specific operation, it must be further mapped to one of the CRUD operations. The operation target represents on which the operation to perform. It is always specific to a document or a view. The operation output represents the output of the results of the operation. There are two types of operation output. The first is the operation target itself. The other is if the results of the operation are success or not. For example, a processing contains one operation, and the operation specifies that the operation type of "place" and the operation target of "purchase order". The operation type "place" is further mapped to

"create" operation. Thus the processing is to execute a "Place purchase order" operation. The operations of the processing are executed in sequence, never overlapping.

The response represents a reaction comes from the application in replay to the request. The response describes response channel and response target. The response channel describes what protocol is used to transmit the response message. The response target describes what to transmit to. The response target is assigned one of the operation outputs. For example, a response contains the response channel of "SMTP" protocol and the response target is the operation output of first operation which is "an email with acknowledgement of a purchase order". Thus the response is "using SMTP to replay an email with acknowledgement of a purchase order".

4.1.2. Document/View Model

I propose using the document/view model (d/v model for short) to describe operation targets of the business activities. The documents and the views in the d/v model represent the business documents and the business information gathered from different business documents in the business world. A document records the results of business activities, such as purchase order for order processing, shipping invoice for shipment of goods. Business operations in the business activities only operate on the documents and the views, never on the domain objects. Domain objects are explained in the next section. The documents are defined from domain objects. The views are also used to record business activities. They are used in a way as a window on many documents, such as the monthly sales of product records. The line between the documents and the views are blurred. The documents are usually used to record the daily business activities. The views are usually used to show business performance in order to make any business decisions. The meta-model of the document/view model is shown in Figure 4-2.

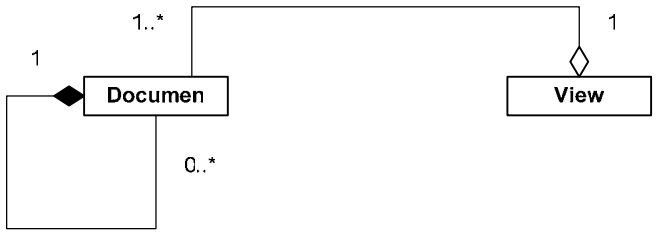


Figure 4-2 Meta-model of Document/View Model

Generally, the documents shall persist in storage media. The storage

media may be any kind of media that provides persistent service, such as database management system or file system. Each document or each view represents a system responsibility of persistent service the generated application shall take.

4.1.3. Domain Model

Domain model is not a new concept at all. It describes conceptual objects, which are called domain objects, an application concerns [1][16][21]. It characterizes an application from others. Different application for different customers has different domain objects with definitions of attributes and relationships among them. With regard to enterprise computing, it contains conceptual objects of people, goods/services, money, and many others.

Domain model is also use as a communicating tool to help the customers and the developers to speak the same language. It describes what the customers use in their daily activities. The developers try to explore it in order to build desired features in an application. Within ECSDF, it is not computable. But it plays an extremely important role in ESCDF. The object model generator can generate an interfacer or a controller role objects, but it can never be possible to “guess” a domain model for enterprise computing. In ECSDF, It is used as the communicating tool to help the customers and the developers to build other three business models.

4.1.4. Business Rules Definition

Business rules of ECSDF constrain how/when/what of business operations and business entities, the documents and the views. For example, a business rule defines in the business activity “Add purchasing item to purchase order” in its second operation “Add a purchasing item to a purchase order” of the processing states that “the total price of a purchase order cannot excess ¥10000”. This example shows the constraints on the business activity. Business rules definition also constrains on the value of the attributes of the business entities. For example, a business rules constrains that the length of a customer’s name cannot excess 30 alphabetic characters. Each business rule represents a system responsibility that the generated application shall take to prevent the constraint be violated.

The business rules are like IF-THEN statements. The IF part of the statement represents the ***constraints***, the THEN part of the statement represents ***violation reactions*** indicate what actions to take if the constraints are violated. For example, there is a business activity that a web customer adds an item to a shopping cart. The IF part defines a constraint that states a

shopping cart can only accommodate up to ten shopping items. The THEN part defines a violation reaction that states if the constraint is violated then shows a warning message to the web customer.

```
`     IF
            counts of shopping items of a shopping cart >= 10
      THEN
            operationOutput displays “The maximum counts of a
            shopping item is 10”
```

By defining constraints on the business activities and the business entities, the generating mechanism in ESCDF generates business validation and violation reactions service provider objects and source code.

The source code generator will generate source code to ensure the constraints will never be violated and to display a default warning message states that the rules are violated. The users and the developers can override the default violation reactions.

4.2. Hello User Example, Episode II

After an incredible success of the delivery of HW 1.1, customers decide to expand the Hello User application to be more capable. The new requirements are to print a document contained “Hello” followed by a user name and then followed by “you have accessed the application N times”. N is a count to show how many times a user has access the application. The specifications are shown as the following:

- A user of the application types a name on screen.
- The application then has to display a document contains “Hello”, follows by the name, follows by “you have access “, follows by the count of how many times the user has accessed the application, and follows by “times”. For example, if a user types a name, Jenson, and it is the second time he accesses the application, then it prints “Hello, Jenson, you have accessed the application 2 times”.
- The constant “hello”, “you have accessed the application”, “times”, and the count a user access the application shall be stored in a database.
- The screen shall be a terminal type input/output device or a web browser.
- Other screen types may be added in the future.
- Storage of the constant strings and the count may be changed in the future.

The new requirements are called HU 2.0. This time, Michael decides to use ECSDF to evaluate its promises.

Michael first defines the domain model. It is a good idea to start from finding domain objects. He discusses the domain model with the customer. After he gains confidence that he does really understand the requirements, he then creates the d/v model by referring the domain objects. The domain model and the d/v model for HU 2.0 are shown in Figure 4-3 and Figure 4-4 respectively.

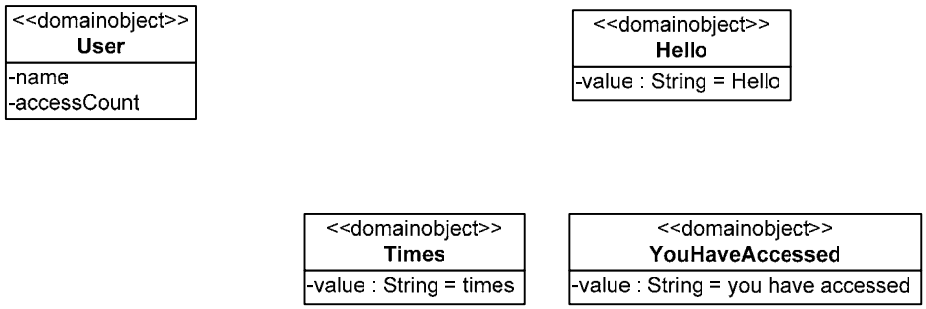


Figure 4-3 Domain Model of HU 2.0 in UML Class Diagram

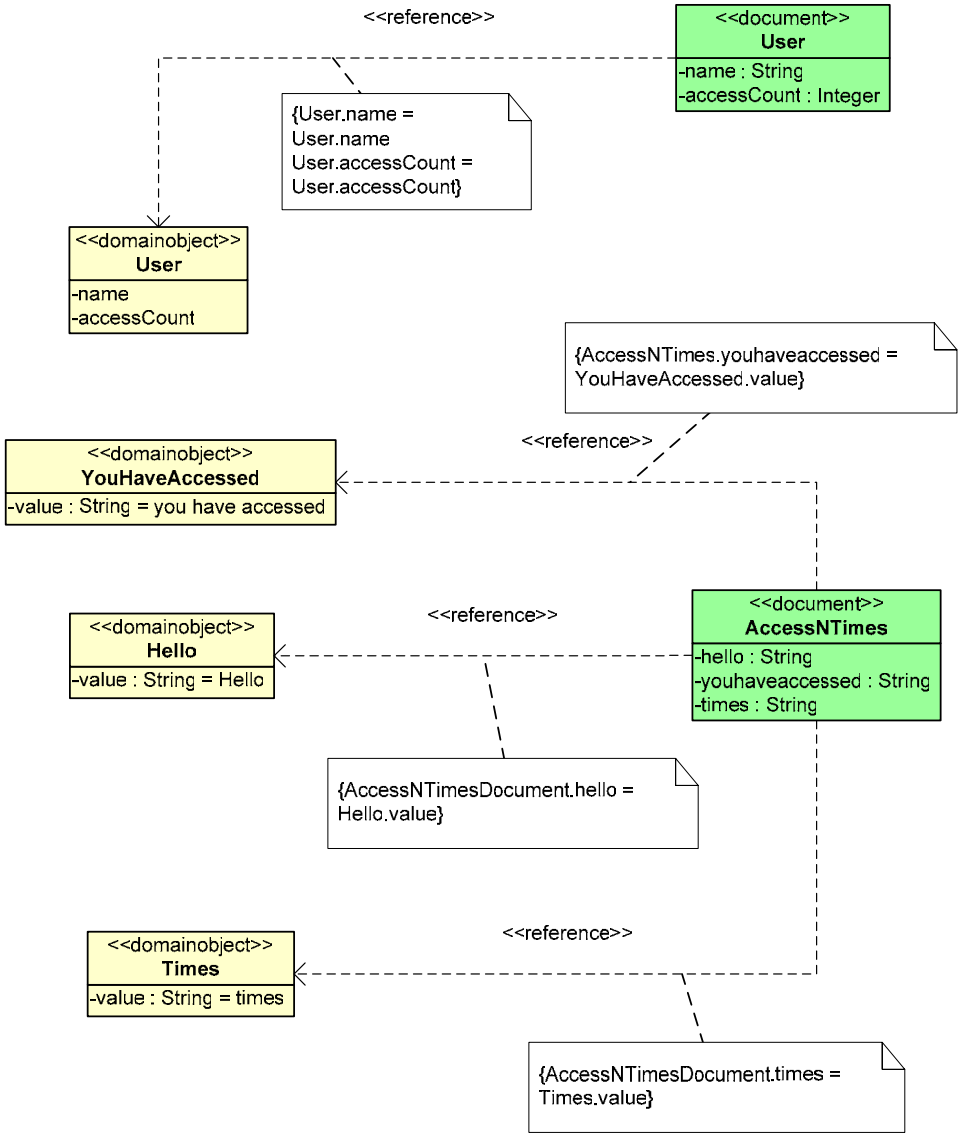


Figure 4-4 Document/View Model of HU 2.0 in UML Class Diagram

The domain model contains four domain objects. The `User` domain object represents the user who uses the application. Other three domain objects, `YouHaveAccessed`, `Times`, and `Hello`, represent the constant strings the application uses. The `User` domain object has two attributes which represent information about the user of the application. One is the `name` attribute, which represents the name the user types and the other is the `count` attribute, which represents the count the user accesses the application. All three constant string domain objects have the same attributes, `value`, to represent the string the domain object represent. The domain objects are labeled with UML-style

stereotype <<domainobject>>.

The d/v model contains two documents, the AccessNTimes document and the User document. These two documents are created by referring to the domain objects. The dashed line with arrowed head that is labeled with UML styled stereotype <<reference>> represents a document referring to the definition of the domain objects (see Figure 4-4). The User document refers to the User domain object and the AccessNTimes document refers to the other three domain objects, YouHaveAccessed, Times, and Hello. The documents are labeled with UML-style stereotype <<document>>.

The detailed reference specifications are shown in UML-styled constraints. The dot operator represents referring to the attributes of the documents or the attributes of the domain objects. Equal sign = represents definition of a document (left hand side) refers to the definition of a domain object (right hand side). For example, the User document refers to the User domain object which is represents with a dashed line with arrowed head that is labeled with UML styled stereotype <<reference>>. The constraints of the dashed line have defined that

User.name = User.name,

which means the definition of the name attribute of the User document (left hand side) refers to the definition of the name attribute of the User domain object (right hand side).

Michael then defines the business activities of the application. The business activities model is showed in Table 4-1.

Table 4-1 Business Activity Model of HU 2.0

Model Element	Value
Name: Get AccessTime and increase count by one	
Request	
requestChannel	Stream
requestParameters	User.name
requestSource	Terminal user
Processing.Operations	
operation:1	
operationType	Retrieval
operationTarget	AccessTimeDocument
operationOutput	operationTarget
operation:2	
operationType	Update
operationTarget	User.accessCount
operationOutput	operationResult
Response	
responseTarget	operation:1.operationOutput
responseChannel	Stream

The business activity model contains one business activity. The business activity is given a name “Get AccessTimeDocument and increment count by one”. The name plays two roles in ECSDF. One is used to identify the business activity and to convey the purpose of the business activity. The other is used to be the name of business service providers when the object model generator generates software class definition. The processing of the business activity has two operations, one is to retrieve the `AccessTimeDocument` document, and the other is to add 1 to the `accessCount` attribute of the `User` document. Although the details of update operation are not shown in Table 4-1, it shall always be defined for update operation.

Finally, Michael defines the business rules definition. There is only one rule:

- User.accessCount must not be smaller then 0.

4.3. Basic Generation Mechanism

After Michael finishes the definition of the business model, it is about time to have ECSDF to show its promises. In this section, I would like to use the example described above to present how ECSDF is possible to generate the object model and the implementation code.

Before I explain the “how” of the generation, I would to like discuss what the prerequisite necessities for ECSDF to generate are. The following is a list of high-level structural and behavioral elements that may exist in object model and source code.

- Entities
- Entities relationships
- Entities constraints
- Entities interactions
- Application logic and domain logic

The entities are the software objects in the object model. Each software object has attributes. The entities relationships represent the relationship among the software objects. The entities constraints limit the values of object attributes. The entities interactions represent a software object provides information or services to other software objects. Application logic and domain logic are source code that concretely state how things are done and how information are hold.

The four business models provide information for the object model generator and the source code generator to produce these elements. In short, the mechanism can be described as “each system responsibility is taken by a set of collaborative software objects and, in turn, each software object takes smaller responsibilities and collaborates with its neighboring software objects.” Following sections, Section 4.3.1 and Section 4.3.2, describe the details of the generation mechanism.

4.3.1. Object Model Generation

There are three types of rules, interaction rules, responsibility rules, and architectural rules, to generate object model. These rules are described as the following.

Interaction rules

The object model generator holds a ***collaboration template*** of the role stereotypes. Collaboration template defines a set of collaboration patterns for a specific domain (collaboration patterns are described in Section 2.2). The collaboration template for enterprise computing is shown in Figure 4-5. A role

stereotype in the collaboration template can be viewed as a placeholder. The placeholders contain two parts, structural part and behavioral part. Structural part is equaled to property and operation definitions of a class. Behavioral part is equaled to the implementation of a class. The object model generator fills the placeholders with classes. How the generator knows what classes to fill? The information comes from the Business Model.

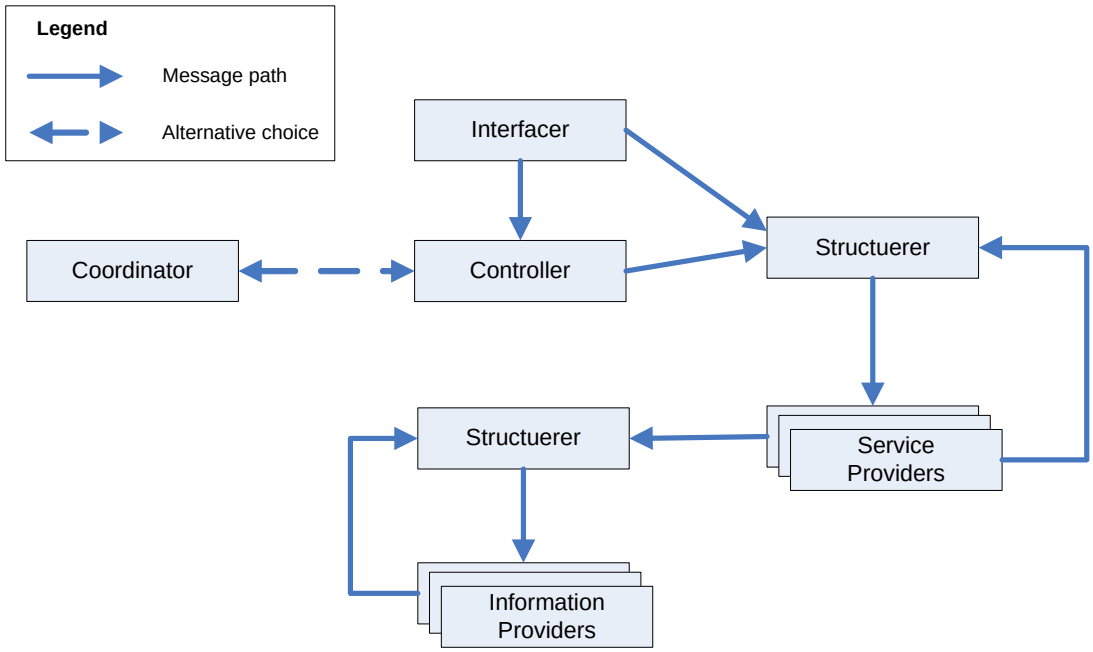


Figure 4-5 Collaboration Template of Application for Enterprise Computing

Responsibility rules

The Business Model contains all system responsibilities the application must carry out. If the application carries out all these system responsibilities, we can say the application is reliable because the customers can dependent on the application to help them doing business. The object model generator generates the object model that carries out all these system responsibilities. The business activities play an important role in generating classes to fill in the placeholders. A business activity defines a larger (system) responsibility an application has to realize. It has three parts, the request, the processing, and the response. Each part has smaller responsibilities. These smaller responsibilities are listed in Table 4-2.

Table 4-2 Responsibilities in Each Part of the Business Activity

Part	No.	Responsibility
Request	R-1	Process request message send by a specific channel defines in requestChannel.
	R-2	Extract request parameters from request message.
Processing	R-3	Decide what classes shall process the operations.
	R-4	Maintain classes that process operations
	R-5	Execute each operation in turn.
	R-6	Create document defines in operationTarget by values defined in requestParameters.
	R-7	Retrieve document defines in operationTarget by criteria defined in requestParameters.
	R-8	Update document defined in operationTarget by values defined in requestParameters
	R-9	Delete document defined in operationTarget by criteria defined in requestParameters.
	R-10	Return different types the results of the operation defines in operationOutput
	R-11	Maintain classes that provide information of the documents
	R-12	Hold information of a document
Response	R-13	Return one of the operationOutput defines in Processing.operations
	R-14	Return the operaitonOutput via a specific channel defines in responseChannel

Table 4-3 Role Stereotypes and Their Responsibilities

Role Stereotype	Responsibility
Interfacer	R-1, R-2, R-14
Controller	R-3, R-13
Service provider	R-5, R-6, R-7, R-8, R-9, R-10
Information holder	R-12
Structurer	R-4, R-11

These smaller responsibilities in Table 4-2 shall be taken by one or more role stereotypes. In Table 4-2, each responsibility is given an ID for convenient discussion sake. ECSDF predefines what role stereotypes shall take the responsibilities. Table 4-3 shows these predefined relationships. The relationships of the roles stereotypes and the responsibilities are discussed below.

Interfacer processes the request and the response of the business activity. It takes three responsibilities of the request and the response, R-1, R-2, and R-14. The generated interfacer role stereotype class for HU 2.0 is shown in Figure 4-6. The generated class has three operations, `processRequestStream`, `processRequest`, and `processResponseStream`. Operations `processRequestStream` and `processResponseStream` are used to take the responsibilities R-1 and R-14 respectively. Operation `processRequest` takes the responsibility R-2.

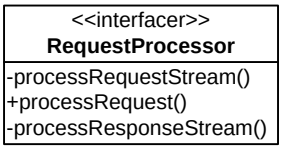


Figure 4-6 Generated Class of Interfacer Role Stereotype

Controller decides what to do next. It takes the responsibilities R-3 and R-13 to execute the operations of the processing in sequence. The generated controller role stereotype class for HU 2.0 is shown in Figure 4-7. The generated class has two operations, `getGetAccessNTimesService` with a parameter and `processOperations`. These two operations take the responsibilities R-3 and R-13. Operation `getGetAccessNTimesService` is a private operation which is used by operation `processOperations` to get the service providers which provide processing execution service.

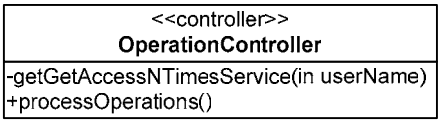


Figure 4-7 Generated Class of Controller Role Stereotype

Service providers execute the operations. Service providers take the responsibilities R-5, R-6, R-7, R-8, R-9, and R-10. The responsibilities R-6, R-7, R-8, and R-9 may not always be taken. Their responsibilities are taken by the generated application only when their operation types are

defined in the business activity. The operation type is always one of CRUD (creation, retrieval, update, and deletion) operations. Creation, retrieval, and deletion are always easier. ECSDF can infer implementations for all operation types except update. Update contains computation. The computation shall be defined by the customers or the developers. It is not possible be inferred by ECSDF. The source code generator uses the request parameters of the request and the operation target of the processing to infer source code for creation, retrieval, and deletion.

The generated controller role stereotype class for HU 2.0 is shown in Figure 4-8. The generated class has four operations, `GetAccessNTimesService` with a parameter, `execute`, `retrieveAccessNTimes`, and `updateAccessNTimes`. `GetAccessNTimesService` is the constructor for the class. It has a parameter which is defined in requestParameters in the request of the business activity. Operations `retrieveAccessNTimes` and `updateAccessNTimes` are both private and are used to perform the operations defined in processing part of the business activity. In HU 2.0, operation `retrieveAccessNTimes` takes the responsibility R-7 and operation `updateAccessNTimes` takes the responsibility R-8. Operation `execute` calls these two operations in sequence. It corresponds to the responsibilities R-5 and R-10.

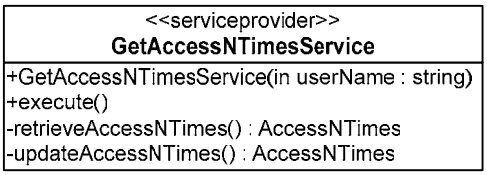


Figure 4-8 Generated Class of Service Provider Role Stereotype

Information holders hold information of the documents or the views. Information holders take the responsibility R-12 to provide information about the documents to the service providers. What information of the documents to provide is defined in operationTarget of the processing part in the business activity. The generated controller role stereotype classes for HU 2.0 are shown in Figure 4-9. The generated classes have no operation but only attributes to hold information of the documents.

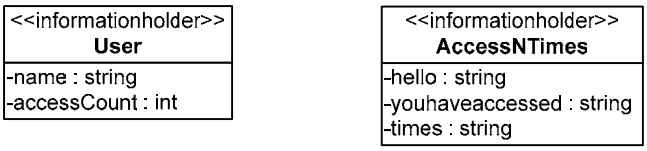


Figure 4-9 Generated Class of Information Role Stereotype

Structurers maintain relationships of information holders and service providers. When a service provider needs information from the information holders, it does not get it directly from the information holders. Instead, it asks a structurer to reach the necessary information holders. When a controller needs service providers, it also has to ask a structurer to reach the necessary service providers. The structurers take the responsibilities R-4 and R-11. The generated controller role stereotype classes for HU 2.0 are shown in Figure 4-10. Each class always has a static operation `getInstance` that is used to get the only permissible instance of the class. It is an implementation of Singleton pattern [22]. Each of them also has one or more operations to get classes it maintains. In HU 2.0, class `DocumentFactory` has two operations `getAccessNTimes` and `getUser`. Both operations take the responsibility R-11. Class `BusinessServiceFactory` has one operation `getGetAccessNTimesService` which takes the responsibility R-4.

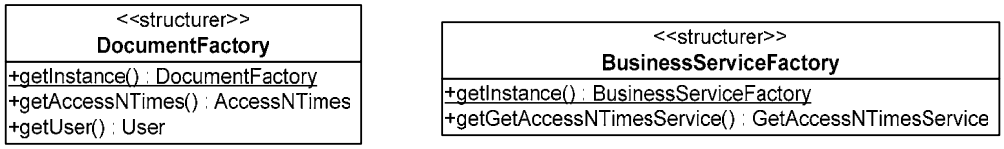


Figure 4-10 Generated Class of Structurer Role Stereotype

Coordinators may be used to take the responsibilities of controllers. But it is always a good idea to use the controllers instead of the coordinators because the controllers are much smarter than the coordinators. The controllers take more responsibilities, thus the burden of the controllers' neighboring objects is alleviated and those neighboring objects can focus on their own responsibilities.

Besides these responsibilities in the business activity model, the d/v model also contains responsibility. As mentioned in 4.1.2, each document represents a system responsibility of persistent service to the document that the generated application shall provide. Decomposed smaller responsibilities of the persistent service are listed in Table 4-4.

Table 4-4 Responsibilities in Each Document

Part	No.	Responsibility
Document	R-15	Create new data of document in persistent storage
	R-16	Find data of document in persistent storage
	R-17	Update data of document in persistent storage
	R-18	Delete data of document in persistent storage
	R-19	Maintain classes that provide these persistent service

There are four responsibilities, R-15, R-16, R-17, and R-18 in each document. Responsibility R-15, R-16, R-17, and R-18 may not always be taken. These responsibilities match to the operation types, CRUD, in the business activity model. Their responsibilities are taken by the generated application only when the operation types are defined in the business activity.

The generated service providers role stereotype classes for HU 2.0 are shown in Figure 4-11. There is one class of structurer role stereotype, `DAOFactory`, and two classes, `UserDAO` and `AccessNTimesDAO`, of service provider role stereotypes. These two service providers are used to manipulate data in database. Operations `insert`, `find`, and `save` in the class `UserDAO` correspond to the responsibilities R-15, R-16, and R-17. Operations `insert` and `find` in the class `AccessNTimesDAO` correspond to the responsibilities R-15 and R1-6. As usual, there is always a class that used to maintain the service providers. Class `DAOFactory` takes the responsibility R-19. It is an implementation of FactoryMethod pattern [22].

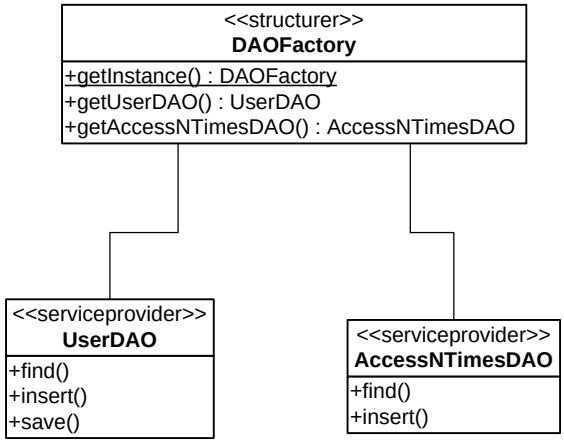


Figure 4-11 Generated Classes for persistent service

Business rules definition also represents responsibilities. Business rules apply to the documents and the views in the d/v model and operations in the processing of the business activity. Business rules shall be checked against any point where applied documents are operated on or applied operations are executed. Deciding when to check business rules is a trade-off between performance and reliability. If checking too often, the performance can be dragged down. Conversely, if checking too less, wrong data or malicious data may be slipped through. The following list is the possible points to check business rules:

1. In classes of interfacer role stereotype
2. In classes of controller role stereotype
3. In classes of structurer role stereotype which manage information holders and service providers
4. In classes of information holder role stereotype
5. In classes of service providers role stereotype which provide data persistent service

Deciding where to put the checking logic of the business rules are another trade-off between performance and maintainability. The checking logic can be spread in every place showed in the aforementioned list without calling any other classes for help. It may have some performance gain. But if there is any business rule change, it is very hard to hunt down every place where modification is necessary. Or it can be gathered up in only a few classes and be called by other classes that require the checking logic. It is best for maintainability.

In ECSDF, the generated application shall be flexible and reliable. And these two characteristics are guidelines for deciding when to check and where to place the checking logic. With regard to when to check, all aforementioned places shall be checked because it promises reliability. With regard to where to place the checking logic, the business rules are centralized in a few classes because it promises flexibility. More explanations of the checking logic creation are given in the next section.

Architectural Rules

Architectural rule is very simple. It defines that there shall be one collaboration group for a business activity. A business activity contributes a certain set of role stereotypes to a collaboration group. The d/v model and the business rules also contribute a certain set of role stereotypes to a collaboration group. All the collaboration groups forms the object model. For example, if there are two business activities, one is creating purchase order, and the other is updating customer record. Then there shall be two collaboration groups for these two business activities. And these two collaboration groups forms the generated

object model (see Figure 4-12).

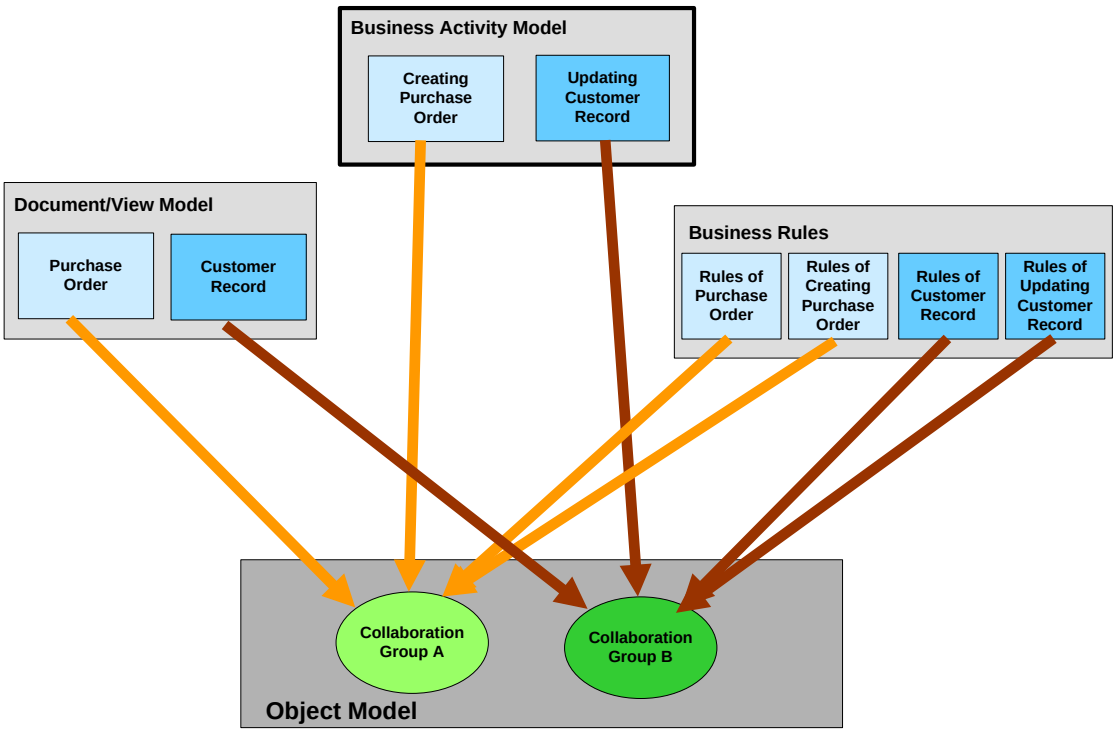


Figure 4-12 Example of Architectural Rules

4.3.2. Source Code Generator

The details of source code generation are not discussed in my thesis. It is the future work for my dissertation. Following discussions only provide ideas concerning this topic.

We can use recently emerged code generation engine to produce implementing logic. From the aspect of raising the abstraction level of programming, the declarative style of programming improves productivity. One of the most well-known declarative styles is JavaServer Page (JSP) [23]. JSP uses tags to provide the necessary information to the code generator to produce Java Servlet [24]. Generally, code generators hold code templates and the developers provide extra information as the tags in a declarative way. The idea of the source code generator in ECSDF is to use the Business Model as the tags to provide information to generate application logic and domain logic. The d/v model is used to create private fields and getter and setter methods. Business

rules are centralized in a few classes and Dependency Injection pattern [25] is used to have every necessary place to be checked.

4.4. Advanced Generation Mechanism

One of the hardest problems for the model-driven approach is how to have machines to know when and where to create relationships. There are two kinds of relationships in an object model, inheritance and composition. Before discussing of how to create these two relationships by machines, the purpose of these two relationships shall be considered first. Without knowing why these relationships are necessary, it is not possible to design the mechanical creation mechanism.

Both relationships are used to extend an object's responsibilities. But composition is dynamic, whereas inheritance is static [1]. Within a collaboration group, composition exists among collaborative objects to ask others to provide service or to provide information. There may be a situation that there are too many business activities and the object model just becomes bloated. It is possible to optimize the object model to be more compact. Composition and inheritance is helpful in the situation. For example, the interfacers of all collaboration groups can be merged into a hierarchy of interfacers to provide request message handling functionality. If multiple request channels exist, the common responsibilities of the request message handling can be placed in a superclass and specific responsibilities to each request channel are scattered over subclasses. In short, the responsibilities shall be divided into a hierarchy structure. Higher-level responsibilities shall be taken by superclasses and lower-level responsibilities shall be taken by subclasses. Table 4-5 shows the possible hierarchical responsibilities for R-1. Figure 4-13 shows the possible generated interfacer classes for the hierarchical structure of R-1.

Table 4-5 Responsibilities for Hierarchy Structure for R-1

No.	Responsibilities
R-1-1	Extract message head
R-1-2	Extract message body
R-1-1-1	Handle message head of request channel HTTP
R-1-1-2	Handle message head of request channel STREAM
R-1-2-1	Handle message body of request channel HTTP
R-1-2-2	Handle message body of request channel STREAM

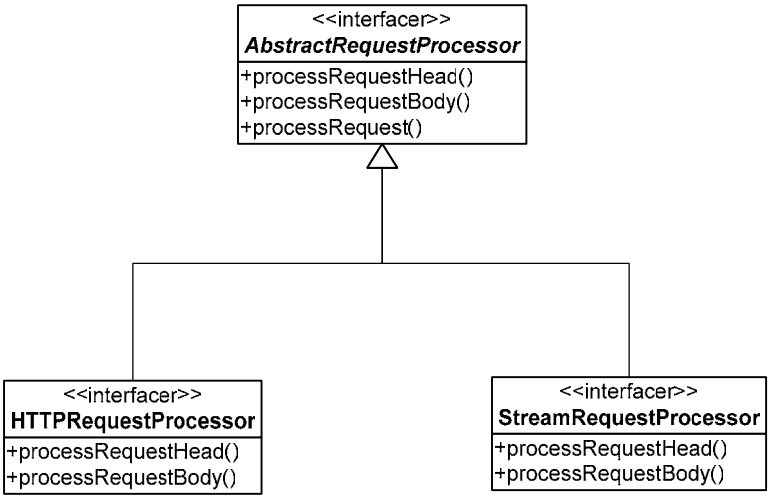


Figure 4-13 Class Diagram for R-1

There is no significant benefit of optimizing an object model with these two relationships for reuse. For human to handle complexity, this kind of optimization is helpful. But for machines it is skeptical. But the conclusion may be too early. I would like to conduct more studies on this topic in the future.

5. DEVELOPING ENVIRONMENT FOR ECSDF

This chapter describes the developing environment developed for ECSDF. It is developed as Eclipse plug-in. It is not a traditional integrated developing environment for text-based programming language. It provides functionality to programming from higher abstraction level. Current implementation provides project explorer, document/view model editor, business activity model editor, and rule-based object model generator. Others functionality expects to be implemented in my dissertation.

5.1. Overview

In Chapter **Error! Reference source not found.**, the details of the generation mechanism of ECSDF are discussed. The next work is building a developing environment based on the mechanism. Building a developing environment is not an easy job. A contemporary developing environment usually provides as an integrated, all-in-one solution application. Following is a list of common functionality.

- Elegant source code editor
- Source code visualization, such as UML diagrams supports
- Outliner

Some others provide more advanced functionality.

R-15. Source code rafactoring

R-16. Code skeleton generation from UML diagrams

All these common and advanced functionality are central to source code. For ECSDF, the focus is shifted from source code to modeling. There are two choices for the work.

- Implementing from scratch
- Using existing framework and expanding it

Surely our choice is the second one. A developing environment bases on ECSDF is developed as a plug-in of Eclipse platform [26], which is called ECSDF-DE (ECSDF developing environment). One of the most significant benefits of Eclipse is extensibility. Eclipse is not only a framework to provide functionality of GUI and resource management, it is effectively a platform to be extended into any creative works. ECSDF-DE is provided as an all-in-one solution for modeling.

5.2. Functional Requirements

ECSDF-DE has two goals:

- Rapid development for developers
- Expandable for researchers

Its functional requirements are listed in Table 5-1.

Table 5-1 Functional Requirements of ECSDF-DE

Function	Implemented	Future Work
Project explorer	✓	
Document/view model editor	✓	
Business activity model editor	✓	
Business rule editor		✓
Object model generator	✓	
Source code generator		✓
Verification and validation of the Business Model		✓
Virtual machine		✓

Not all of the requirements are implemented in this thesis. The project explorer, the document/view model editor, the business activity model editor, and the object model generator are implemented first. Others are leaved out and shall be implemented in the dissertation.

The project explorer is a tree-styled viewer for project contents. An ECSDF project contains the Business Model, the generated object model, and the generated source code. The document/view model editor, the business activity model editor, and the business rule editor are used to edit the d/v model, the business activity model, and the business rules of ECSDF respectively. The object model generator and the source code generator of ECSDF-DE are the implementation of generation mechanism described in Section 4.3. Verification and validation of the Business Model is used to verify and validate the correctness of the business model. Virtual machine provides a real-time running environment which directly executes software system from the Business Model. Following section described the implemented functional requirements.

5.3. Implementation

Figure 5-1 shows the main elements of ECSDF-DE. ECSDF-DE is comprised from three elements, Eclipse plug-in, rule-based engine, and database. The plug-in provides the Business Model graphical editing functionality. The rule-based engine generates the object model. The database stores the Business Model data. Lines with arrow head indicate the flow of data. The data of the Business Model are provided to the rule-based engine to generate the object model. The generated object model goes to the plug-in to provide visual representation. The data of the Business Model are stored in and retrieved from the database.

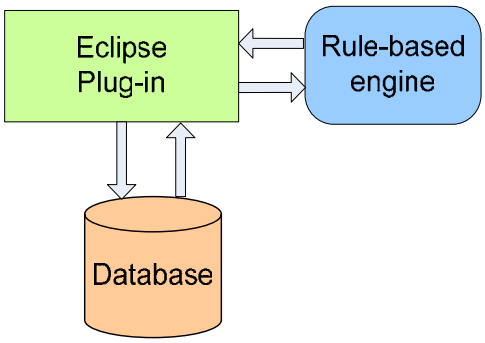


Figure 5-1 Main Elements of ECSDF-DE

5.3.1. Eclipse Plug-in

Figure 5-2 shows the screenshot of the ECSDF-DE. The basic user interface of Eclipse is a multi-paned window which could be customized for different purposes [26]. In ECSDF-DE, the windows are customized with the following items.

1. Project explorer is a tree-styled view which shows the contents of the Business Model and the generated object model.
2. Editing area is stacked with windows which has tabs for navigation. The document/view model editor and the business activity model editor are hosted in the editing area.
3. Console output is used to show the informative messages, such as the execution of the object model generator.

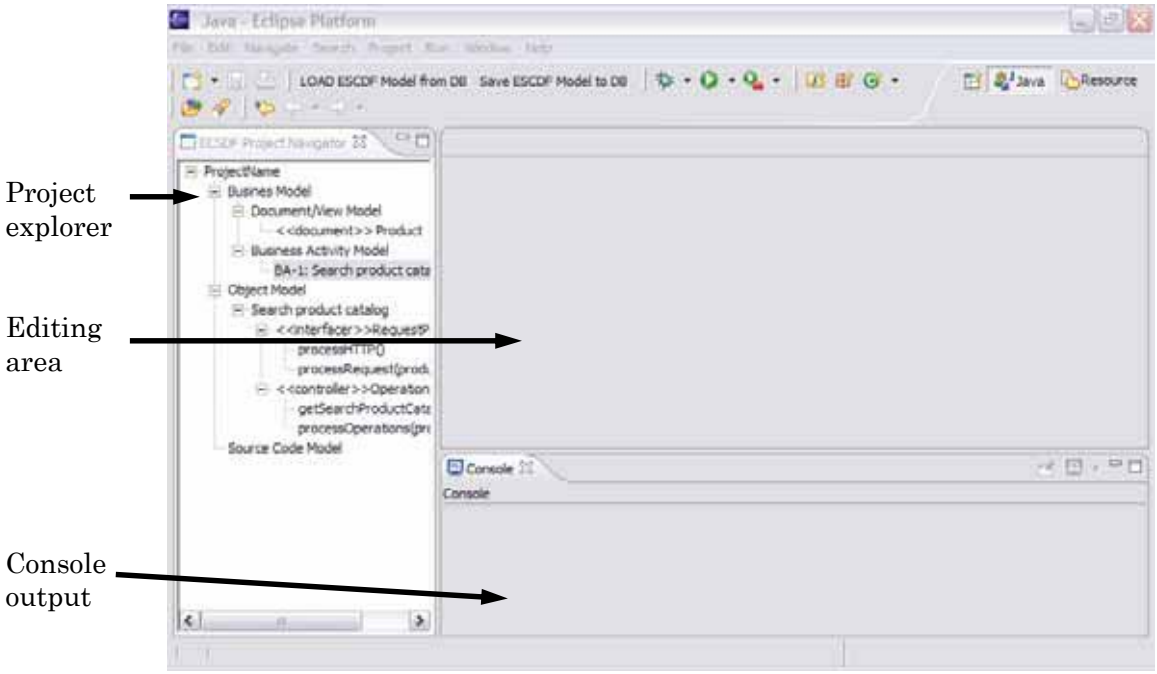


Figure 5-2 ECSDF Developing Environment

Figure 5-3 and Figure 5-4 show the document creation window and the business activity creation window respectively. These windows provide the creation functionality of the documents and the business activities.

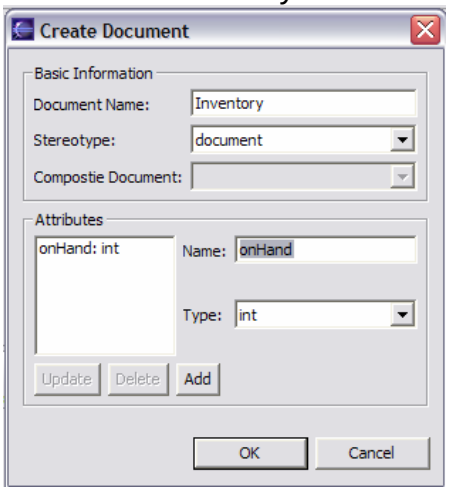


Figure 5-3 Document Creation Window

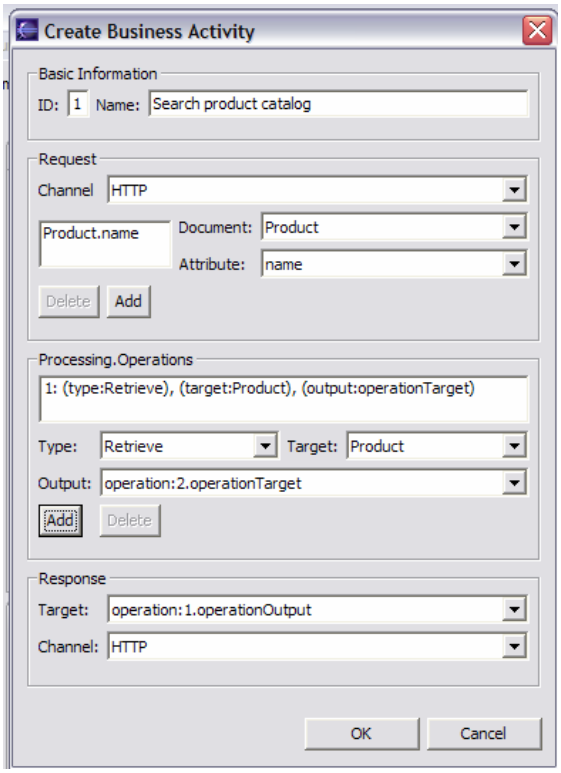


Figure 5-4 Business Activity Creation Window

Figure 5-5 and Figure 5-6 show the document editor and the business activity editor respectively. These editors provide the editing functionality of the documents and the business activities that already exist in the Business Model. The view editor is not implemented in the thesis.

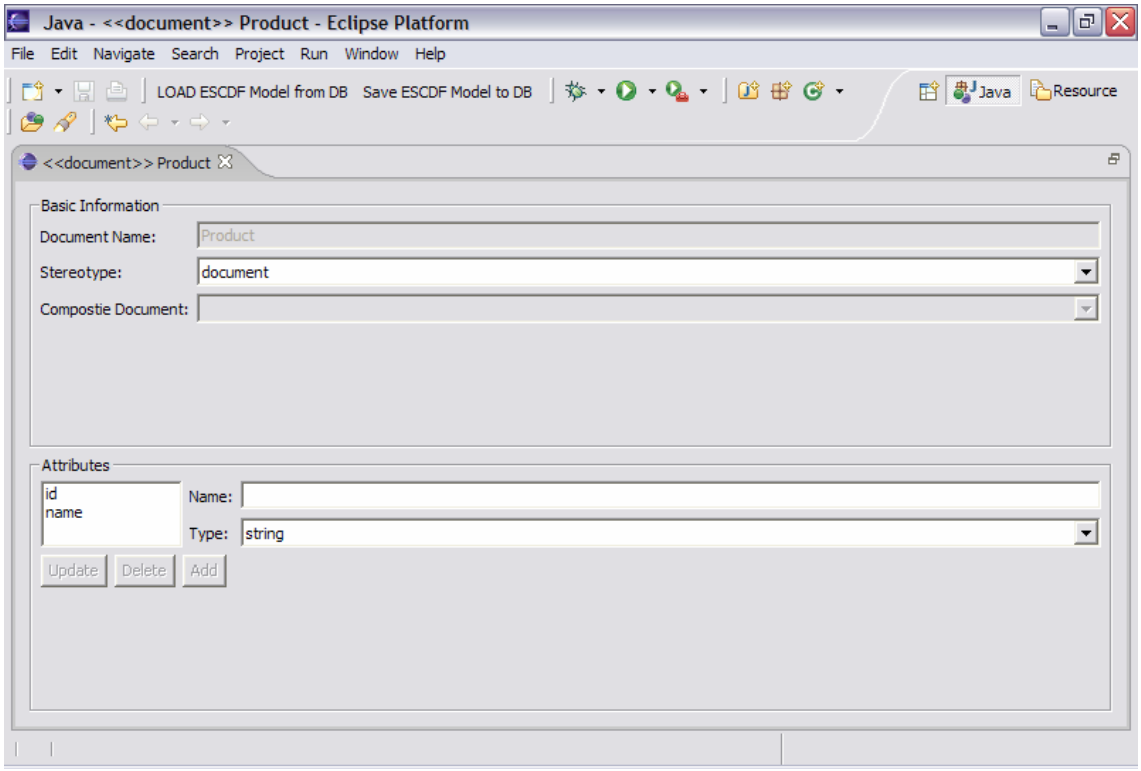


Figure 5-5 Document Editor

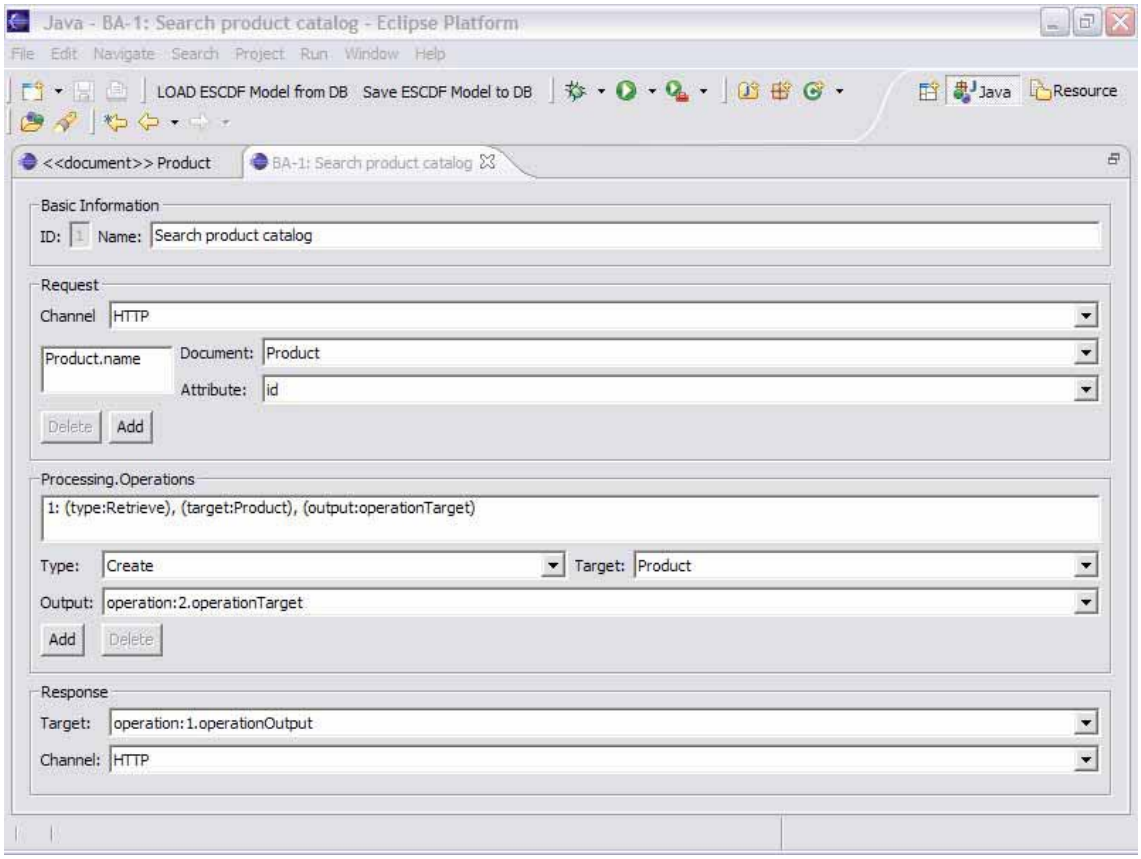


Figure 5-6 Business Activity Editor

5.3.2. Rule-Based Engine

The object model generator uses a rule-based engine to produce the object model structure. The rules are defined based on the mechanism described in Section 4.3. Currently the rules are hard-coded into a text file. In the future, a more flexible rules definition editor will be considered. In the current implementation, a collaboration group is created for each business activity. Figure 5-7 shows the generated object model for the business activity “BA-1: Search product catalog” of the demo application. If more than one collaboration group exists, they can be optimized into a merged collaboration group. The optimizing implementation is based on the mechanism described in Section 4.4.

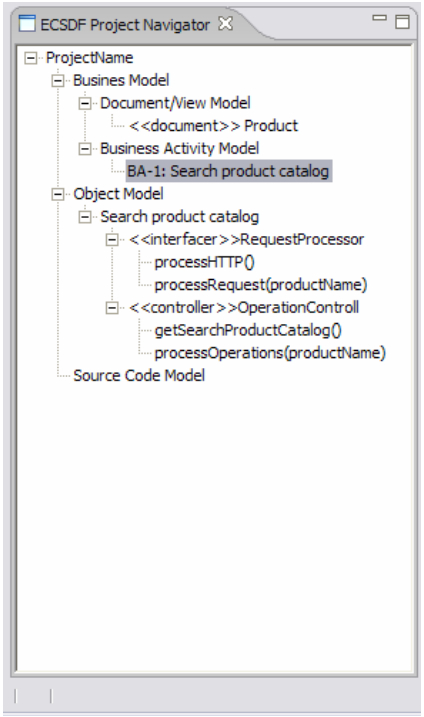


Figure 5-7 Generated Object Model

Currently, Jess is used as the engine [27]. It does not imply that ECSDF is designed specific to Jess. The mechanism of ECSDF can be implemented by any rule-based engine or even without a rule-based engine but code from scratch. Jess is comprised from two main elements, facts and rules. The facts represent the things about the real-world environment. The rules represent what to do if a fact occurred. Facts trigger rules, rules assert more facts. With regard to ECSDF-DE, the data of the Business Model are inputted as the facts. The generating mechanisms are solidified to the rules. The Business Model triggers the object generating rules, and the object generating rules assert the object model structure. The work is depicted in Figure 5-8.

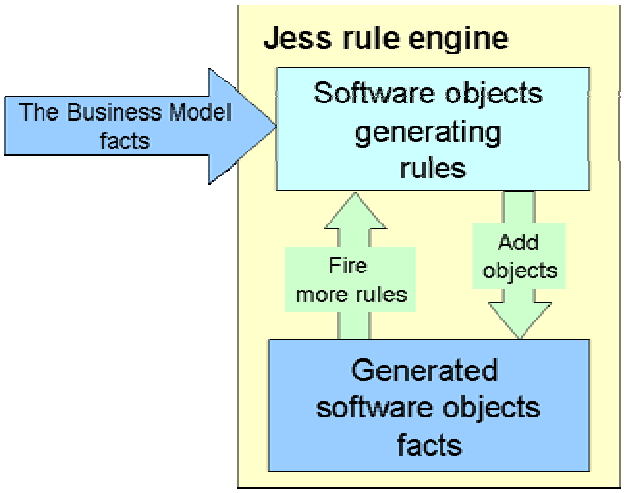


Figure 5-8 Works of Jess

6. APPLYING AND EVALUATING OF ECSDF

This chapter describes the application and the evaluation of ESCDF. To verify the effectiveness of ECSDF, ECSDF-DE is used to develop a software system for enterprise computing. Another three-layered software system is developed by hand-coded as a comparison. The details of the software specification are described in Appendix-A. The design materials of both software systems are illustrated in Appendix-B. The evaluation is carried out against software architecture and object models.

6.1. Overview

In Chapter **Error! Reference source not found.**, the mechanism of ECSDF is described. In Chapter 5, the implementation based on the mechanism is introduced. Then, I would like to evaluate the effeteness of ECSDF. The specifications of software system for enterprise computing (BMS for short) described in Appendix-A is developed into two software systems. One is done by hand and the other is done by ECSDF-DE. The context diagram of BMS is shown in Figure 6-1. BMS provides web store, basic information, procurement, selling, and inventory functionality. Customers order products from the web store via the Internet. Basic information functionality provides management of data of customers, suppliers, and products. Procurement functionality provides management of purchasing documents. Selling functionality provides management of sales documents. Inventory functionality provides management of product storage.

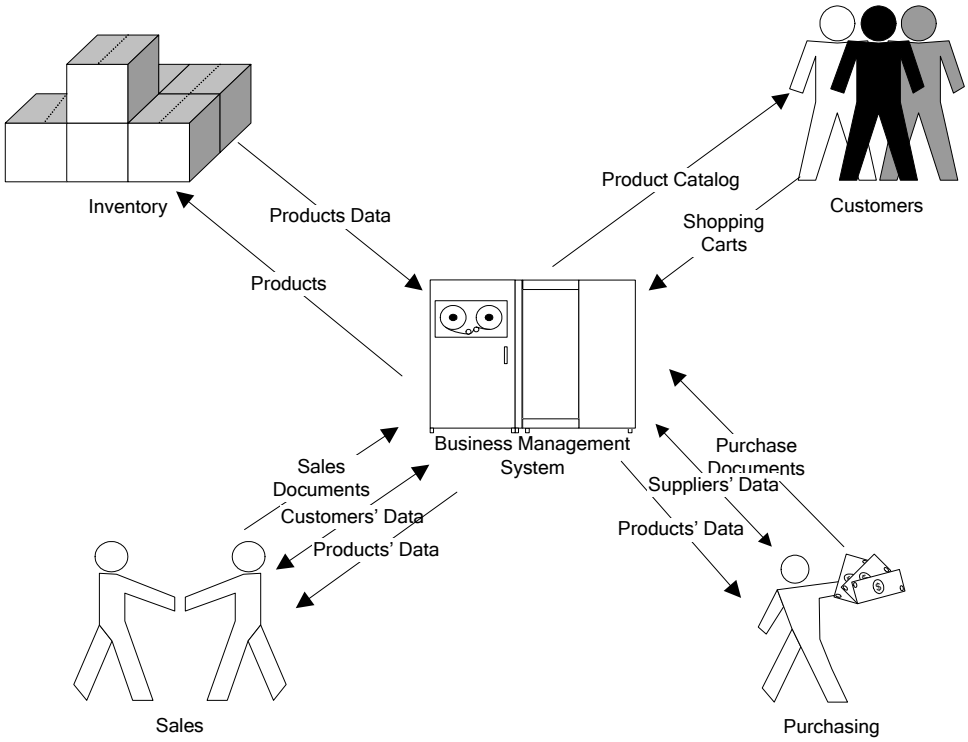


Figure 6-1 Context Diagram of BMS

As mentioned in Section 3.1.2, there are tons of documents a company creates and preserves. The types of the documents that are managed by BMS are shown in Figure 6-2. The arrowed lines represent the processing flow.

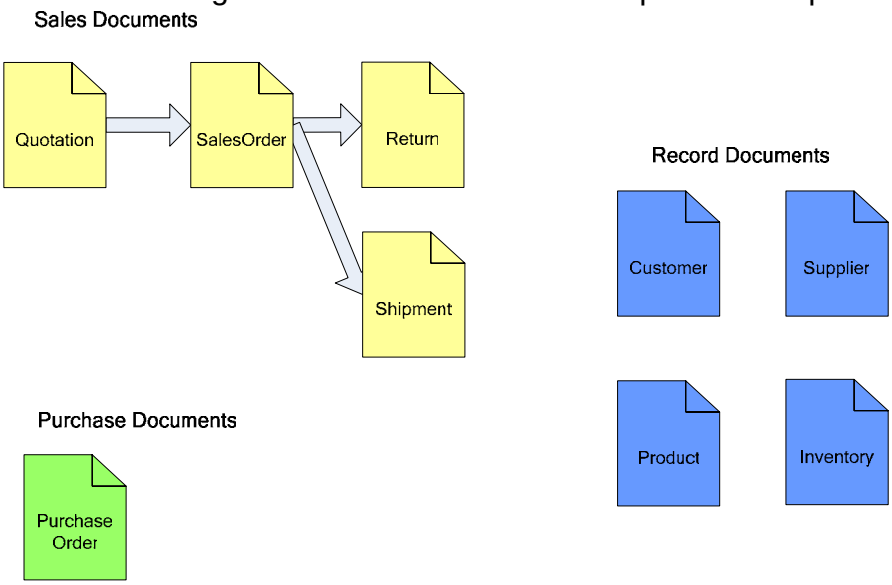


Figure 6-2 Types of Documents Managed by BMS

In the following sections, the evaluation is performed against software architecture and object model of the hand-coded BMS and the ECSDF-generated BMS. The criteria use to evaluate are list as the following:

1. How “well” the desired quality attributes of both systems are achieved.
2. Productivity of both systems

Different software system has different focused quality attributes [28]. The hand-coded BMS focuses on maintainability and flexibility. The ECSDF-generated BMS focuses on flexibility and reliability. It makes no sense using only one set of quality attributes to evaluate both systems. Thus, the evaluation is not carried out by using the same set of quality attributes to evaluate both systems. Instead, the evaluation is carried out to see how “well” the desired quality attributes of each system is achieved. The software quality attributes are not only concerned with the choices of architectural styles but shall be considered throughout design, implementation, and deployment [28]. My evaluation focuses on the architecture and the object model. There is no any implementation and deployment artifact created, thus implementation and deployment aspects are not considered. There is no any standard way, quantified approach to evaluate quality attributes, thus a narrative style is used.

For further discussion, the definitions of the quality attributes are shown in Table 6-1

Table 6-1 Definitions of Quality Attributes

Quality Attributes	Definition
Maintainability	The quality of being modifiable without affecting other parts of the system
Flexibility	The quality of being adaptable to other situations
Reliability	The quality of being dependable by the users of the system
Productivity	The quality of being effective and speedy completion of the system without compromising other desired quality attributes

6.2. Architecture

6.2.1. Hand-Coded BMS

The hand-coded BMS follows “traditional” three-layered architecture

which consists of presentation layer, domain layer, and data source layer [29][30][31]. The presentation layer handles interaction between the user and the software system. The domain layer contains the application-specific or domain-specific logic that work for current problem domain. The data source layer handles communication with other systems [30]. This three-layered architecture has following benefits:

- The dependencies between layers are minimized. For example, the change of external database system only affects the data source layer. The presentation layer and the domain layer are not affected. Maintainability is achieved.
- Each layer has specific responsibilities. New services can be added to a layer easily. For example, if a new service to access data stored in LDAP (Lightweight Directory Access Protocol) must be added, the only necessary place to extend it is the data source layer. Flexibility is achieved.
- The developing works can be divided according to the separation of the layers. Each layer follows a set of well-known patterns to speed up design works. Productivity is achieved.

Although the three-layered architecture helps to achieved desired quality attributes, any hand-coded system suffers from the ripple effect of any requirement change. It requires the developers to track the system around to make modification. It does not only concerns with design and implementation; it is the limitation of human brains.

6.2.2. ECSDF-generated BMS

The ECSDF-generated BMS consists of multiple collaboration groups. Each collaboration group takes the responsibilities of the Business Model. Changes to the Business Model can be easily handled because the only work is a creation of a new collaboration group. Flexibility is achieved.

A collaboration group realizes the responsibilities of the Business Model by using the language the customers understand. The specifications described by the language are later directly used to generate software system. There are no misunderstanding or misinterpreted of the requirements. Reliability is achieved.

ECSDF-DE provides a “magic” menu item to generate the object model (see Figure 6-3). All the works of software development are simplified by ECSDF-DE with a click on the menu item. Any change to the Business Model can be easily handled by re-generating of collaboration groups. Productivity is

achieved.

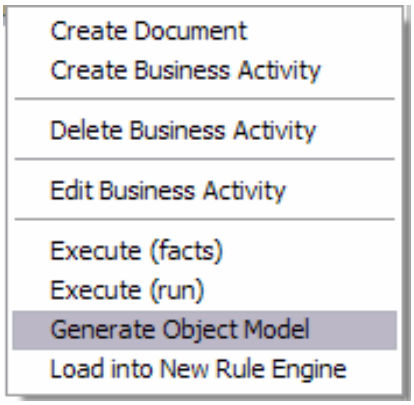


Figure 6-3 “Magic” Menu Item

6.2.3. BMS versus BMS

As discussed in Section 6.2.1 and Section 6.2.2, both BMSes achieves their own desired quality attributes, but the criteria of the evaluation are how “well” they are. The following comparison discussed both BMSes:

With regard to the hand-coded BMS, maintainability is a big issue. Although layered-architecture and well-modularized design and implementation have some remedies, human brains are limited.

Both BMSes are productive. With a good design, implementation can be simplified to a set of the coding patterns. The developers only have to follow the set of the coding patterns. Undoubtedly, ECSDF provides much higher productivity than the hand-coded BMS. Human are inferior to machines in this kind of recursively occurred works.

Both BMSes are flexible. The layered-architecture and the well-modularized design and implementation are helpful. But again, ECSDF provides much superior performance of flexibility than human.

With regard to reliability of ECSDF-generated BMS, there are issues about verification and validation of the Business Model. It shall be the future work.

6.3. Object Model

6.3.1. Hand-Coded BMS

Framework and patterns are two keys to achieve the desired quality attributes in the hand-coded BMS. Struts [12], a framework for web applications, is used in the presentation layer. The hand-coded BMS also uses a wealth of the patterns. A pattern consists of one or more classes. The patterns are used as a

convenient design and communication tool for designers [30][31]. The patterns used in the hand-coded BMS and the relationships among them are shown in Figure 6-4. The design of the hand-coded BMS provides the following benefits:

Maintainability is achieved by the framework and the patterns. Struts provides a clean separation of models, views, and controllers. Different pattern takes different responsibilities. For example, ApplicationController in the presentation layer has the responsibilities to manage views and commands. The views are used to show information to the users. The commands represent requests of the user to the hand-coded BMS. Any change to the views or the commands is limited to this pattern. Modification is easier.

Flexibility is achieved by the patterns. For example, if a new database is added to support data storage, a new class that implements Data Access Object pattern [30][31] can be added easily to support this requirement.

Productivity can be achieved by the patterns. The patterns provide well-known solutions to well-defined problems [22][30][32]. The developers comprehend the design of the hand-coded BMS easily by pattern names. Learning curve is flatted. Communication path is shortened. And coding time is compressed.

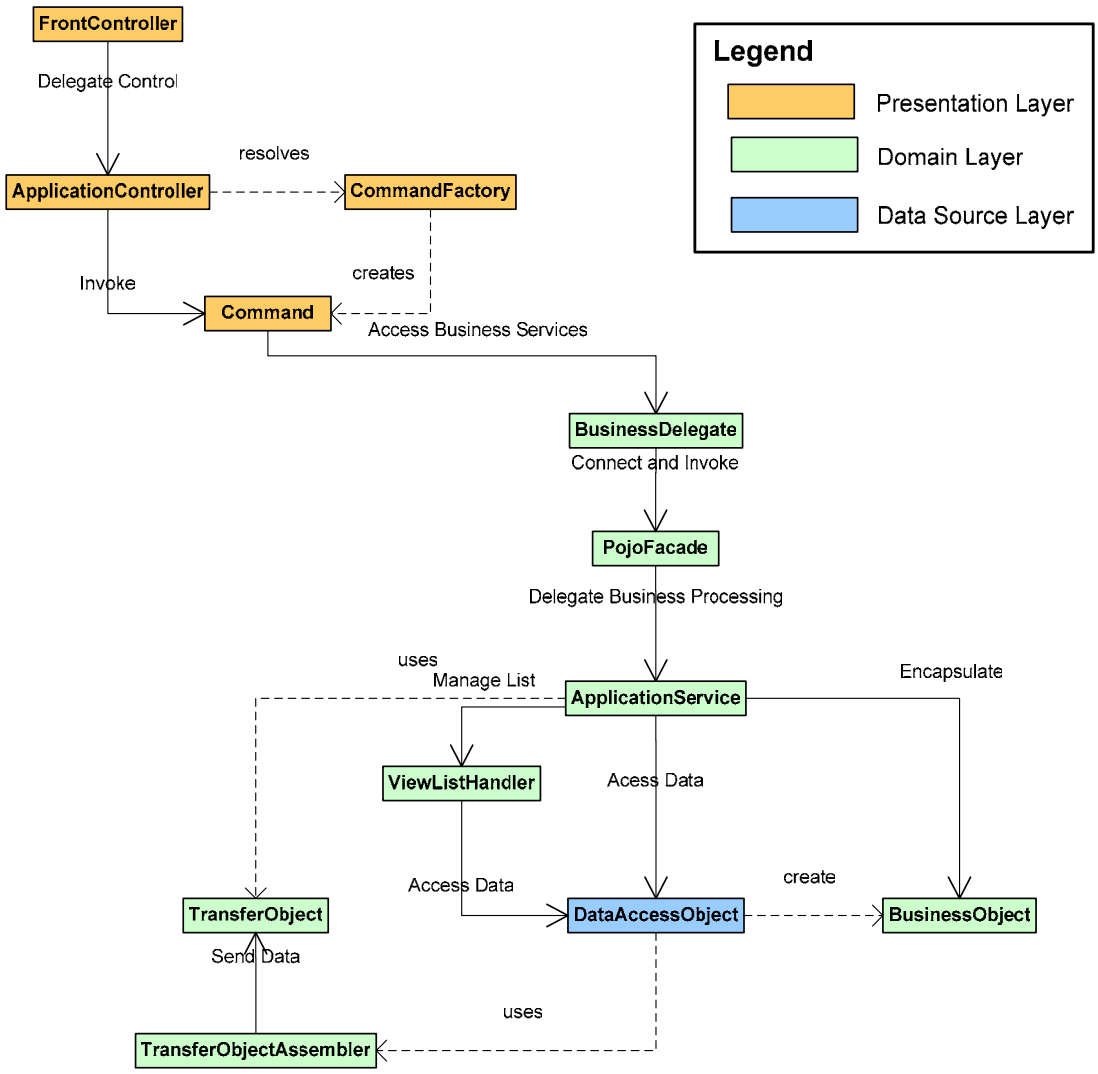


Figure 6-4 Pattern Relationships of the Hand-Coded BMS

6.3.2. ECSDF-Generated BMS

ECSDF-generated BMS consists of a set of collaboration groups. A collaboration group is created from a collaboration template. A collaboration template consists of various role stereotypes (see Figure 4-5). Usually, a role stereotype is mapped to a class. The object model of the ECSDF-generated BMS provides the following benefits:

Flexibility is achieved by collaborations. Collaboration patterns define possible message sending paths between role stereotypes. In Figure 4-5, a collaboration template is defined for enterprise computing.

Other domains may have different collaboration template which results from different collaboration patterns.

Reliability is achieved by collaborations. Each collaboration group realizes the responsibilities of the Business Model. The users can dependent on the ECSDF-generated BMS by testing each collaboration group separately.

There is not question that ECSD provides superior productivity. Tedious and recursively occurred works are best done by machines.

6.3.3. BMS versus BMS

How “well” both BMSes are? The following comparisons discuss both BMSes:

With regard to the hand-coded BMS, the framework and the patterns are used to alleviate issues of maintainability. But any change to the requirements still lead to a long journey of modification.

Both BMSes provide productivity. ECSDF is superior to the hand-coded BMS. With a cleanly responsibilities sharing among collaborative software objects, the object model can be easily generated and expanded.

Both BMSes are flexible. But the ECSDF-generated BMS is inferior to the hand-coded BMS. Although the collaboration patterns and the collaboration templates provide flexibility, there is no magic that machines can generate code for new technology they have not known yet. New rules shall be written by the developers.

As mentioned in 6.2.3, more studies on reliability of ECSDF are ncessary. I will focus on how to ensure the users can dependent on the ECSDF generated software system.

7. CONCLUSION

This chapter describes the conclusions of the thesis. It briefly states what I have represented about my works. It also introduces if I have accomplished the purpose of my work, the importance and the effects of the current research works, and the possibilities for the future work.

7.1. Summary

Programming from higher abstraction level provides many benefits. Developing software for enterprise computing is especially beneficial. A software developing framework for enterprise computing is conceived. The framework uses only one language for the users and the developers. Misunderstanding and misinterpretation are avoided. The object model generator and the source code generator of ECSDF are developed for building flexible and reliable software system for enterprise computing. The implementations of ECSDF, based on Eclipse, are also developed. They are the combination of Eclipse plug-in for graphical interactions, the rule-based engine for object model and source code generation, and the database for the Business Model repository. An evaluation of two software systems, one is developed by hand-coded and the other is developed by ECSDF, is carried out. The results of the evaluation show that the ECSDF-generated software system provides flexibility, reliability, and productivity

7.2. Significance of the Current Research

The research is successful. A software developing framework, ECSDF, that can be used to describe business activities of enterprise computing and to generate proper software architecture, object models, and source code is conceived. The implementation of ECSDF, the development environment based on Eclipse, is built. The hand-coded business management system is developed for evaluation purpose. The gap is filled.

The Business Model of ECSDF uses only one language for the users and the developers. It is understood by the users and the users get away with mysterious jargon. It also helps to avoid misunderstanding and misinterpretation of requirements. It is expressive to describe essential business activities. It is also rigorous and unambiguous to be transformed by machines.

The software systems generated by ECSDF are reliable because ECSDF uses the language customers can understand. They are also flexible due to the idea of roles, responsibilities, collaborations. ECSDF brings business benefits of time to market and customer satisfaction to application users and application developers.

ECSDF is inspired by the ideas of roles, responsibilities, and collaborations. With the well-defined responsibilities in the Business Model, and the ideas of the collaboration patterns and the collaboration groups, the generation mechanism bridges the gap between the real world facts and the cognizant of the real world, and the gap between the cognizant of the real world and software world cleanly. Although many software developing methodologies refer to responsibilities, ECSDF is a vanguard to use the ideas in the study of theoretical and practical model-driven approach.

7.3. Future Work

The conceived framework for enterprise computing leaves out room for improvements and opens many possibilities.

Currently, responsibility assignments transform to rules manually. It is better to edit the assignments directly and have machines do the transformation of assignments to rules. The source code generator needs more discussions and considerations. The generated software systems are proved to be flexible. To be a truly reliable, more studies are needed. There should be a way that the correctness of the Business Model can be validated and verified. Currently, the d/v model is used as a database schema definition. If the d/v model can be defined directly by “the documents” of the real world and have machines to generate the computational database schema, productivity can be raised. The benefits of object model optimization are unclear. It needs more studies. With the progress of the improvements, the implementations of ECSDF, ECSDF-DE will be modified simultaneously.

The ultimate goal of the ECSDF is the virtual machine. The virtual machine is an autonomous execution environment for the Business Model. With only the definition of the Business Model and the desired quality attributes, the virtual machine can have itself adapt to the desired quality attributes. For example, if the desired quality attributes of software system are performance and security, the virtual machine can make a trade-off between these two quality attributes by adding specific implementations to the software system without human interference. To achieve this functionality, the study of executing

characteristics of ECSDF-generated software shall be done.

APPENDIX-A. SPECIFICATION OF BMS

A-1. Overview

Mount Tea Store is a mail order store. Its main business is selling Taiwan tea and tea jar. Almost all of its business processes and business information is done manually by using spreadsheet application. Customers' information, product records, supplier information, and purchase records are recorded in the spreadsheet application. When a data update is needed its staff have to use Find and Replace in the spreadsheet application. Although spreadsheet application provides easy calculation and reporting function, there are many inconsistent found in monthly account settlement. Since data are spread in different sheets, copy and paste is needed when exchange information among sheets.

Because the growing business in recent months, the boss of *Mount Tea Store* decided to adopt a software system, to integrate and to automate its selling, purchasing, and stocking business information processing. Such system would save staff time and provide precise and timely business information and meet the *Mount Tea Store*'s future growing business needs. It would also provide a web-based ordering system, allowing existing and new customers to order products directly on Internet.

A-2. Vision Statement

For *Mount Tea Store* which wishes to maintain selling, purchasing, and stocking information, the **Business Management System** is an Internet-based application that will store all selling, purchasing, and stocking related information safely, automate business process and provide easy to use interface to retrieve and maintain these data. Unlike the current spreadsheet program, staffs that use the **Business Management System** will not have to record the business information manually, which will save them time, will provide precisely and timely information, and will increase the business opportunity.

A-3. Features

FE-1: Maintain selling data (quote to cash)
FE-2: Maintain purchasing data (purchase to pay)
FE-3: Maintain stocking data (inventory management)

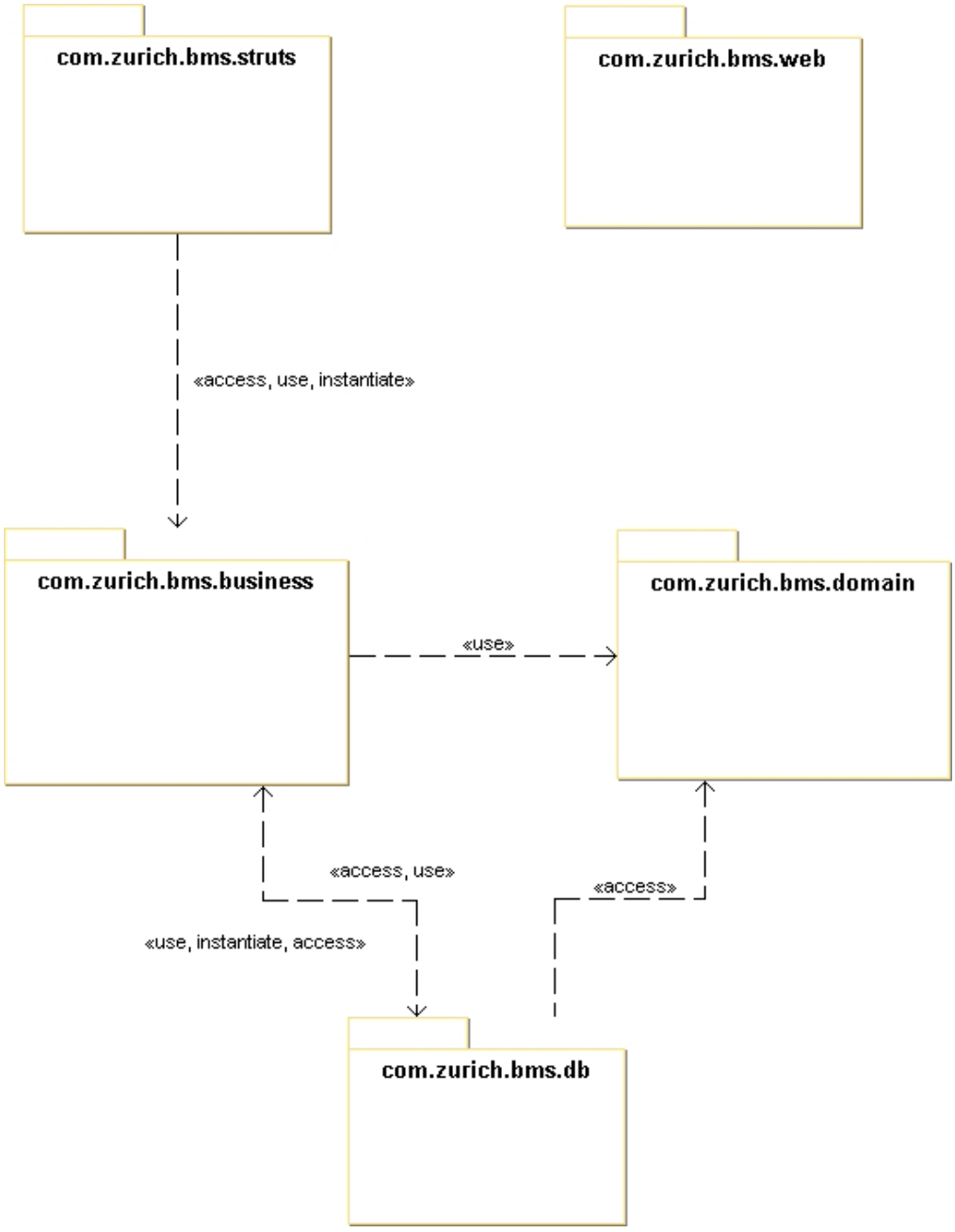
FE-4: Customer can order product on Internet (web storefront)

A-4. Actor-Goal-Use Cases List

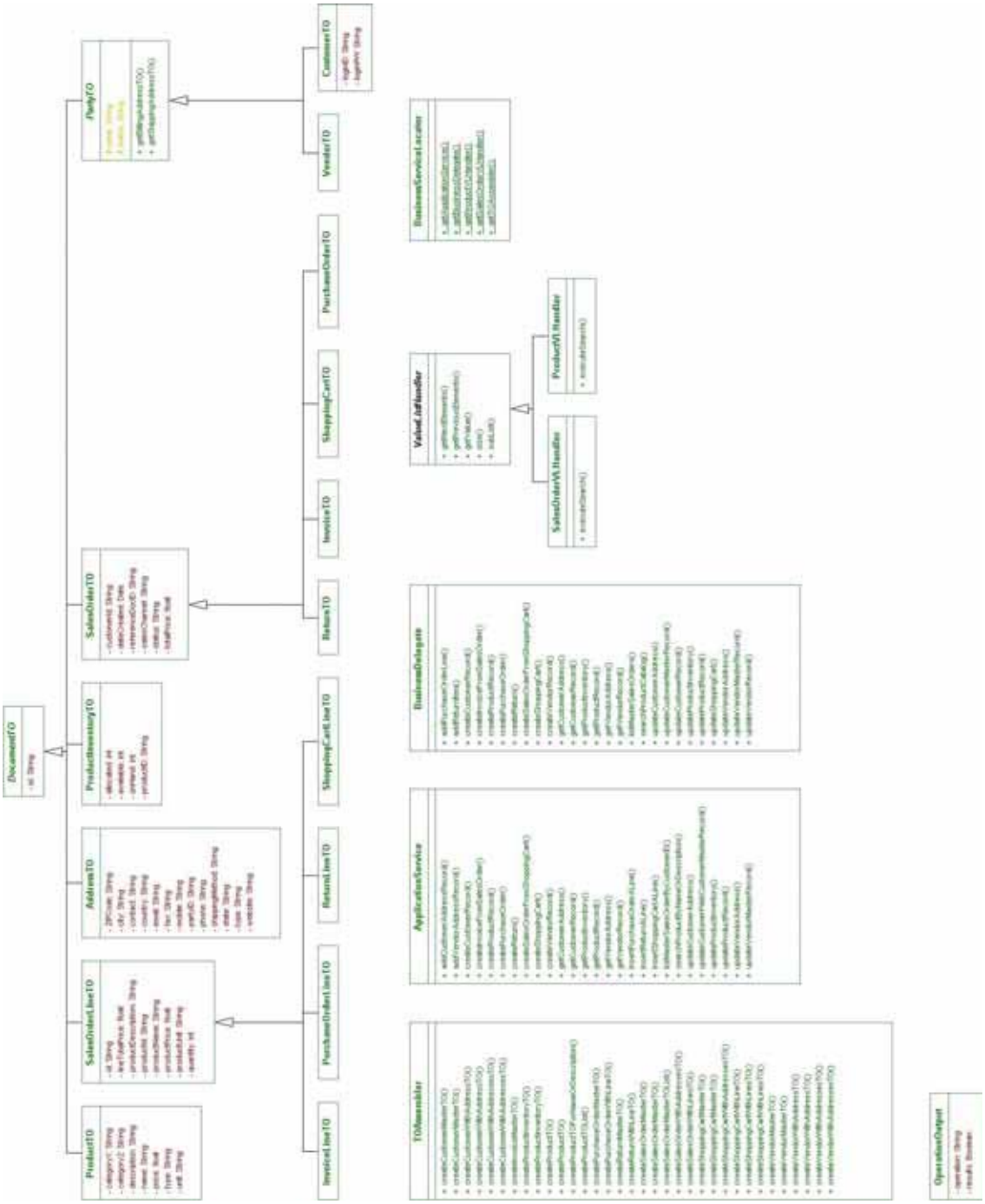
Actor	Goal	Use Case
Web Customer	Search product catalog	Search product catalog
	Buy products	Buy products
	Track order history	Track order history
	Join membership	Join membership
	Get product recommendations	Get product recommendations
	Update member account	Update member account
Sales Staff	Process sales documents	Process sales documents
	Process payment	Process invoice documents
	Process returns	Process return documents
	Create members	Manage members
	Modify members	
	Delete members	
System Administrator	Create users	Manage user
	Update users	
	Delete users	
	Create items in product catalog	Manage product catalog
	Update items of product catalog (change product description, supplier etc.)	
	Delete items from product catalog	
Inventory Staff	Add items to inventory	Manage inventory
	Modify items in inventory (location, stocking amount)	
	Delete items from inventory	
Procurement Staff	Process purchases	Process purchase documents

APPENDIX-B. DESIGN OF BMS

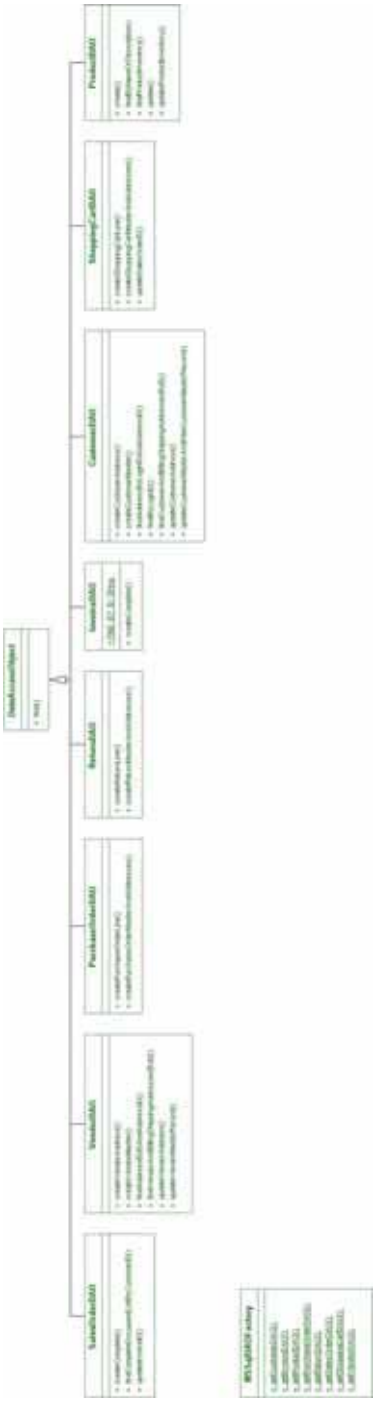
B-1. Package Diagram of the Hand-Coded BMS



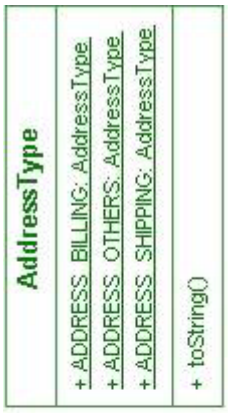
B-2. Class Diagram of the Package com.zurich.bms.business of the Hand-Code BMS



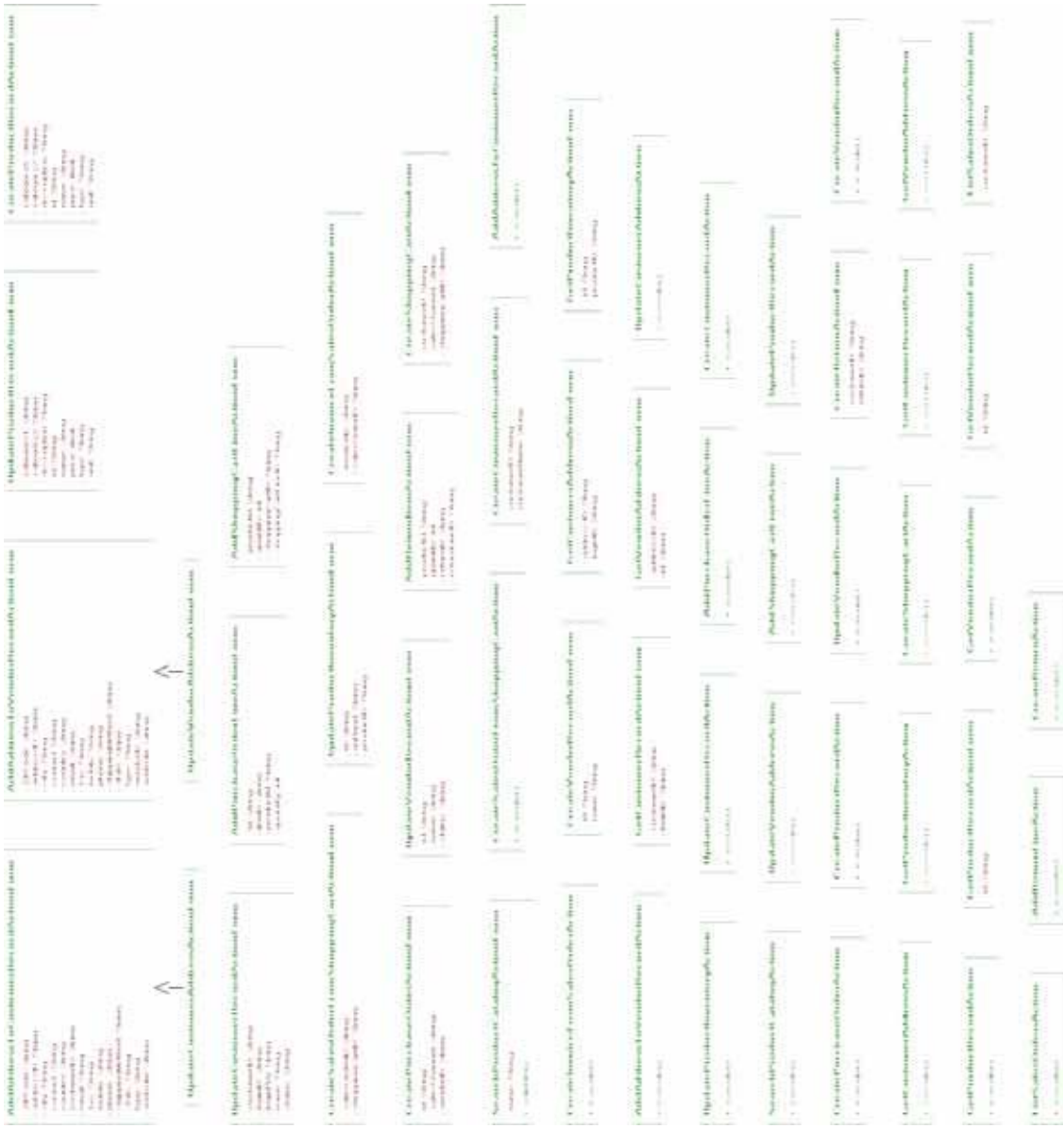
B-3. Class Diagram of the Package com.zurich.bms.db of the Hand-Code BMS



B-4. Class Diagram of the Package com.zurich.bms.domain of the Hand-Code BMS



B-5. Class Diagram of the Package com.zurich.bms.struts of the Hand-Code BMS



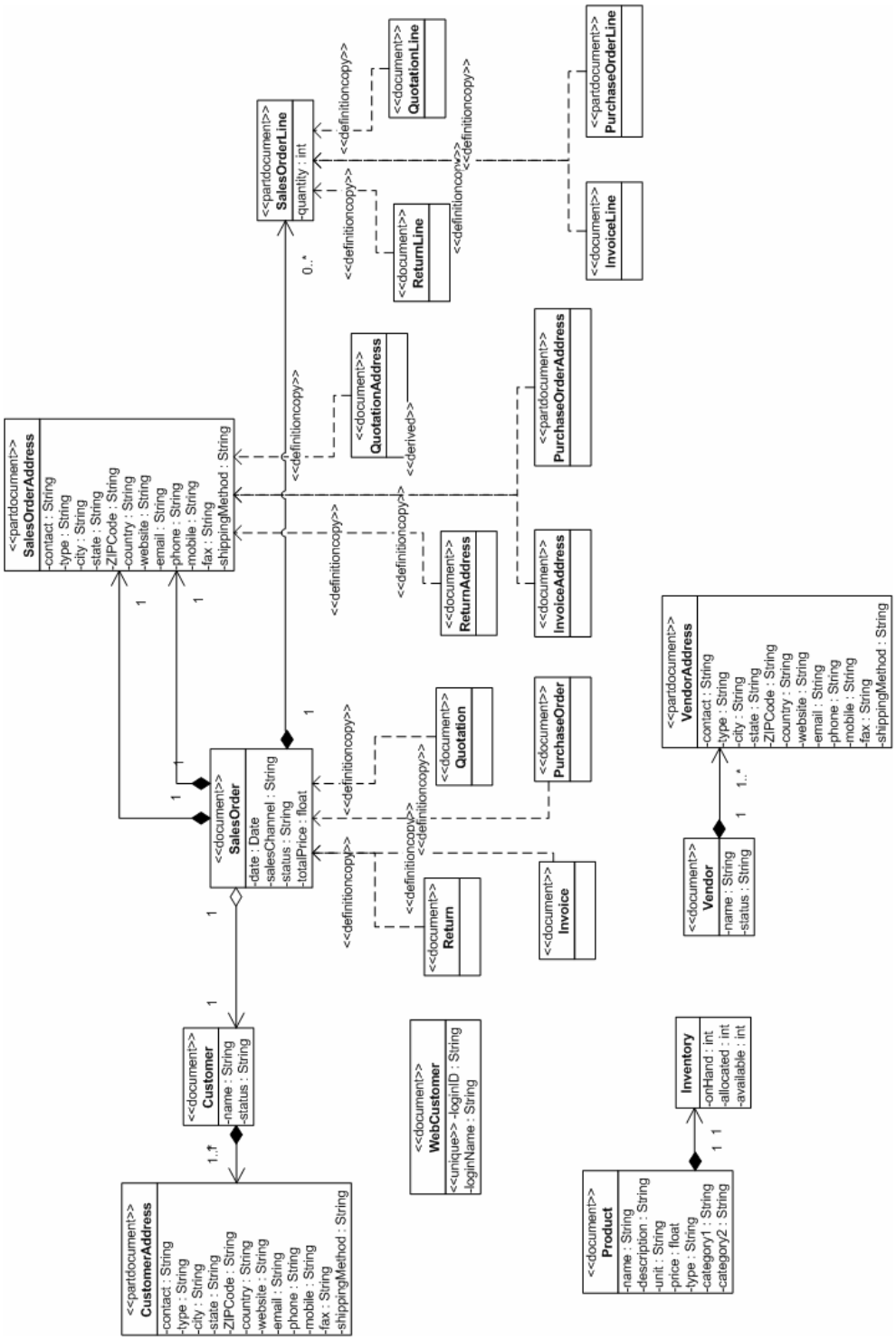
B-6. Class Diagram of the Package com.zurich.bms.struts of the Hand-Code BMS

B-7. List of Business Activities of the ECSDF-Generated BMS

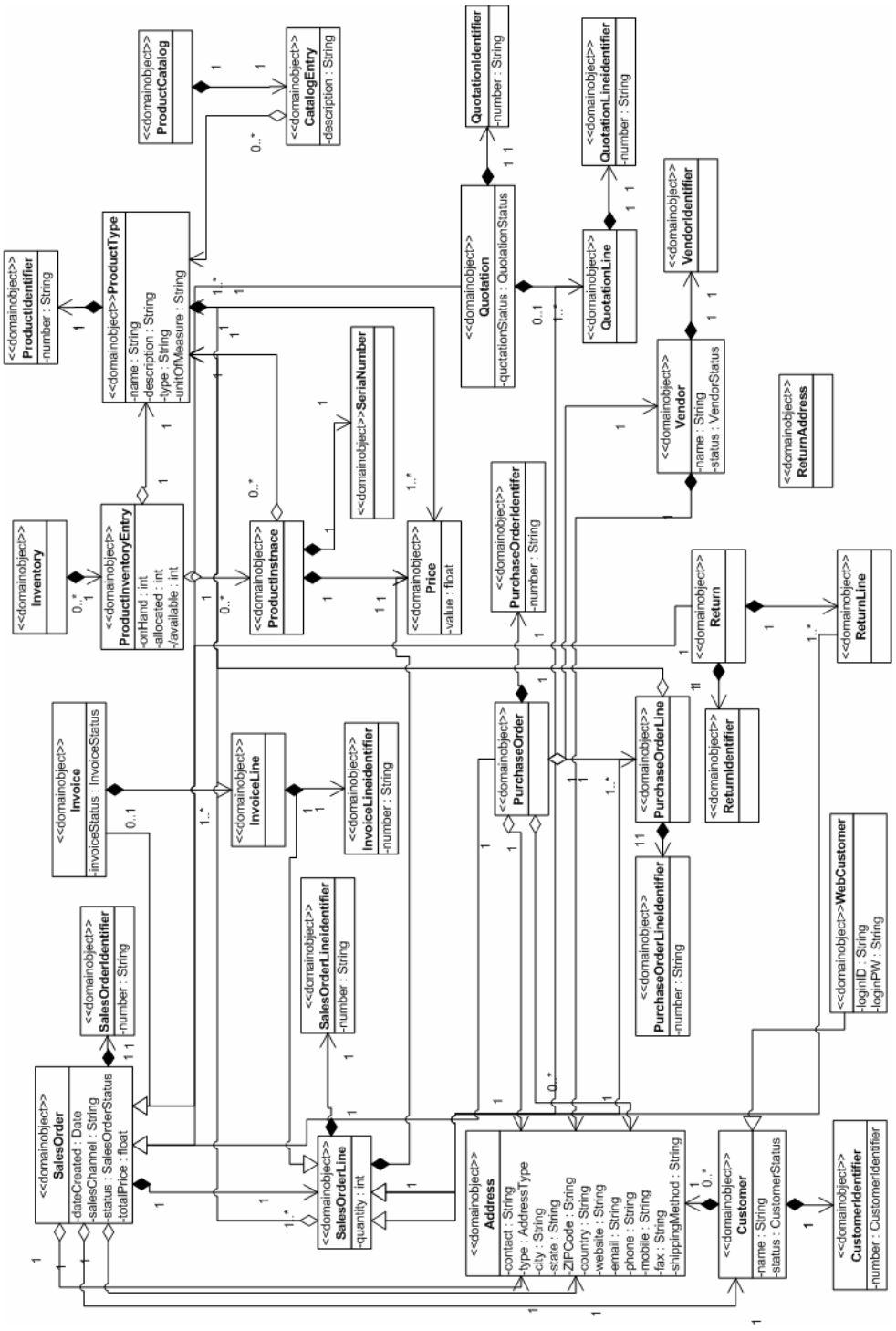
Use Case	No.	Name
Search Product Catalog	BA-1	Search products
Buy Products	BA-2	Create shopping cart (Quotation)
	BA-3	Add shopping item to shopping cart
	BA-4	Create SalesOrder from shopping cart
Track Orders	BA-5	List SalesOrders
Join Membership	BA-6	Create Customer record
	BA-7	Add address to Customer record
Get Product Recommendations	BA-8	Get product recommendation
Update Member Account	BA-9	Get Web Customer master record
	BA-10	Update Web Customer master record
	BA-11	Get Web Customer address
	BA-12	Update Web Customer address
Process Sales Documents	BA-13	Create Quotation
	BA-14	Add quotation item to Quotation
	BA-15	Create SalesOrder from Quotation
Process Invoice Documents	BA-16	Create Invoice from SalesOrder
Process Return Document	BA-17	Create Return
	BA-18	Add return item to Return
Manage Customers	BA-19	Create Customer record
	BA-20	Add address to Customer record

	BA-21	Get Customer master record
	BA-22	Update Customer master record
	BA-23	Get Customer address
	BA-24	Update Customer address
Manage Vendors	BA-25	Create Vendor record
	BA-26	Add address to Vendor record
	BA-27	Get Vendor master record
	BA-28	Update Vendor master record
	BA-29	Get Vendor address
	BA-30	Update Vendor address
Process Purchase Document	BA-31	Create PurchaseOrder
	BA-32	Add purchase item to PurchaseOrder
Manage Product Catalog	BA-33	Create Product Record
	BA-34	Get Product Record
	BA-35	Update Product Record
Manage Inventory	BA-36	Get Inventory record
	BA-37	Update Inventory record

B-8. The Document/View Model of the ECSDF-Generated BMS



B-9. The Domain Model of the ECSDF-Generated BMS



Acknowledgments

Many people helped me in this thesis, and I wish to acknowledge and thank them.

First, I would like to thank Professor Takuya Katayama for his kindness encouragement and guidance. I would also like to thank all the members of Foundations of Software Laboratory, especially, Mr. Toshiaki Aoki, Mr. Mitsutaka Okazaki, and Mr. Naohiro Hayashihara. Thank you for giving me precious comments on my research.

I would like to thank my family in Taichuang, mom, my elder sister, my elder bother-in-law, my lovely nephew Chi-Yo, and my lovely nieces Chi-Jun and Chi-An. Thank you for your endless support. I would like to thank my family in Tainan, my mother-in-law, my younger sister-in-law Marilyn, my younger bother-in-law In-Bin, and my younger bother-in-law Jamy. Thank you for your kindness to me. Especially, I would like to thank my father-in-law. Thank you for brining me Lisa and your family to my life. I would also like to thank Mr. Toshio Hamada and Mrs. Tomoko Hamada, and their family. Thank you for treating me like one of your family. You are my parents in Japan. So many thanks to you all. All of you support me go such far. And last, my dear Lisa, you know I do not have to say no more words here. You are my whole life.

References

[1] Wirfts-Brock, Rebecca. *Object Design, Roles, Responsibilities, and Collaborations*. Addison-Wesley, 2003.

[2] OMG. *MDA Guide Version 1.0.1*. http://www.omg.org/docs/omg/03-06-01.pdf.

[3] Frankel, David. *Model Driven Architecture, Applying MDA to Enterprise Computing: Practice and Promise.* Wiely, 2003

[4] Wirfs-Brock, Rebecca, and et al. *Object-Oriented Design: A Responsibility-Driven Approach*. OOPSLA'89

[5] Beck, Kent, et al. *A Laboratory for Teaching Object-Oriented Thinking*, OOPSLA'89.

[6] Wirfs-Brock, Rebecca, *Characterizing Your Objects*. The Small Talk Report, Vol. 2, No. 5, 1992.

[7] OMG. *Unified Modeling Language, version 1.5*, http://www.omg.org/cgi-bin/apps/doc?formal/03-03-01.pdf

[8] OMG. *Meta-Object Facility (MOF™), version 1.4*, http://www.omg.org/cgi-bin/apps/doc?formal/02-04-03.pdf

[9] Thomas, Dave. *MDA: Revenge of the Modelers or UML Utopia?* IEEE Software, Vol 21, No. 3, 2004.

[10] Sun Microsystems, *Java 2 Platform, Standard Edition, Version 1.4.2*, http://java.sun.com/j2se/1.4.2/docs/api/index.html

[11] Sun Microsystems, *Java 2 Platform, Enterprise Edition, Version 1.3*, http://java.sun.com/j2ee/sdk_1.3/techdocs/api/index.html

[12] Apache Software Foundation, *Apache Struts Framework Javadoc*, http://struts.apache.org/api/index.html

[13] Fowler, Martin. *UML Distilled, 3rd Ed.* Addison-Wesley, 2004.

[14] Cook, Steve. *Domain-Specific Modeling and Model Driven Architecture.* MDA Journal, January 2004.

[15] Wiegers, Karl. *Software Requirements*. Microsoft Press, 2003

[16] Evans, Eric. *Domain-Driven Design: Tackling Complexity in the Heart of Software.* Addison-Wesley, 2004.

[17] Thomas Dave, *UML- Unified or Universal Modeling Language: UML2, OCL, MOF, ECSDF – The Emperor Has Too Many Clothes*, Journal of Object Technology, Vol 2, No. 1.

[18] Fowler, Martin. *PlatformIndependentMalapropism.* http://martinfowler.com/bliki/PlatformIndependentMalapropism.html

[19] Martin, James, et al. *Object-Oriented Methods: A Foundation.* Pretence Hall PTR, 1995.

[20] Business Process Group. *What is a Business Rule?* http://www.businessrulesgroup.org/brgdefn.htm

[21] Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd ed.* Prentice Hall PTR,

[22] Gamma, Erich, et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[23] Sun Microsystems. *JavaServer Pages 1.2 Specifications*. http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html

[24] Sun Microsystems. *Java Servlet 2.3 Specifications*. http://jcp.org/aboutJava/communityprocess/first/jsr053/index.html

[25] Fowler, Martin. *Inversion of Control Containers and the Dependency Injection pattern*. http://martinfowler.com/articles/injection.html

[26] Shavor, Sherry, et al. *The Java Developer's Guide to Eclipse*. Addison-Wesley, 2003.

[27] Friedman-Hill, Ernest. *Jess in Action*. Manning, 2003.

[28] Bass, Len, et al. *Software Architecture in Practices.* Addison-Wesley, 2003.

[29] Buschmann, Frank, et al. *A system of Patterns, Pattern-Oriented Software Architecture*, Wiley. 1996.

[30] Fowler, Martin, et al. *Patterns of Enterprise Application Architecture.* Addison-Wesley, 2003.

[31] Alur, Deepak, et al. *Core J2EE Patterns, Best Practices and Design Strategies.* Prentice Hall PTR, 2003.

[32] Alexander, Christopher, et al. *A Pattern Language: Towns, Buildings, Construction.* Oxford, 1977.