

A COMPARATIVE ANALYSIS OF FEATURE-ORIENTED DEVELOPMENT (FOD)
WITH OBJECT-ORIENTED DEVELOPMENT (OOD) IN SOFTWARE ENGINEERING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
ÇANKAYA UNIVERSITY

BY

İLKER SAPAN

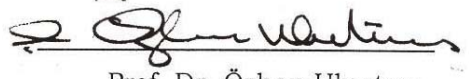
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER, 2007

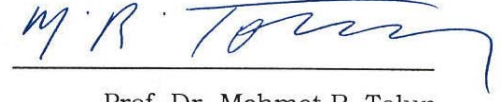
Title of the Thesis : **A Comparative Analysis of Feature-Oriented Development (FOD) with Object-Oriented Development (OOD) in Software Engineering**

Submitted by **İlker Sapan**


Approval of the Graduate School of Natural and Applied Sciences, Çankaya University

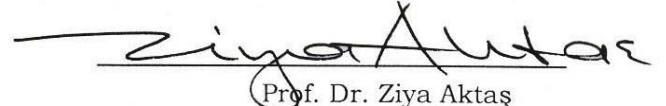

Prof. Dr. Özhan Uluatam
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.


Prof. Dr. Mehmet R. Tolun
Head of Department


This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.



Dr. Semih Çetin
Co-Supervisor

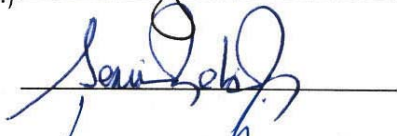

Prof. Dr. Ziya Aktaş
Supervisor

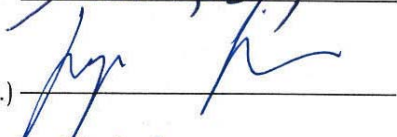
Examination Date : September 12, 2007

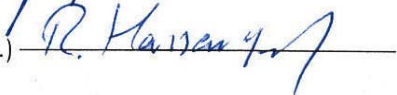
Examining Committee Members

Prof. Dr. Mehmet R. Tolun (Çankaya Univ.) 

Prof. Dr. Ziya Aktaş (Çankaya Univ.) 

Dr. Semih Çetin (Cybersoft) 

Prof. Dr. Hayri Sever (Çankaya Univ.) 

Asst. Prof. Dr. Reza Hassanpour (Çankaya Univ.) 

STATEMENT OF NON PLAGIARISM

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name : İlker Sapan

Signature



Date

: 12.09.2007

ABSTRACT

A COMPARATIVE ANALYSIS OF FEATURE-ORIENTED DEVELOPMENT (FOD) WITH OBJECT-ORIENTED DEVELOPMENT (OOD) IN SOFTWARE ENGINEERING

Sapan, İlker

M.Sc., Department of Computer Engineering

Supervisor : Prof. Dr. Ziya Aktaş

Co-Supervisor : Dr. Semih Çetin

September 2007, 78 pages

The objective of this thesis is to compare the Feature-Oriented Development (FOD) with Object-Oriented Development (OOD) on a case problem. Employing the features in software engineering has become quite popular recently with the emerging tools and techniques. FOD is used within a context in this thesis as partitioning an application domain in terms of “features” yielded by Feature-Oriented Domain Analysis, and then managing them through a relevant software process model known as Feature-Driven Development (FDD).

The conventional FDD approach makes use of classical objects to implement features. Whereas, features can also be implemented by means of a dedicated programming model, i.e. Feature-Oriented Programming, to treat the features as first class entities. However, the FOD vision in this study proposes another model for expressing and implementing the features in terms of “business processes”, “business rules”, and “business services”.

The thesis will examine and evaluate the processes, analysis of the performance, time management, and other relevant issues of FOD in comparison specifically

with OOD. Eventually, the advantages and disadvantages of FOD will be summarized with respect to other related development methodologies.

Keywords: Features, Feature-Driven Development, Feature-Oriented Development, Object-Oriented Development, Service-Oriented Architecture

ÖZ

YAZILIM MÜHENDİSLİĞİNDE ÖZELLİK YÖNELİMLİ GELİŞTİRME METODU İLE NESNEYE YÖNELİK GELİŞTİRME METODUNUN KARŞILAŞTIRMALI ANALİZİ

Sapan, İlker

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı

Tez Yöneticisi : Prof. Dr. Ziya Aktaş

Ortak Tez Yöneticisi : Dr. Semih Çetin

Eylül 2007, 78 sayfa

Bu tez çalışmasının amacı Özellik Yönelimli Geliştirme (ÖYG) metodu ile Nesneye Yönelik Geliştirme (NYG) metodunun karşılaştırmalı analizini bir örnek problem üzerinde gerçekleştirmektir. Yazılım mühendisliğinde “özellik” kavramı; yeni araç ve teknikler sayesinde oldukça popüler olmaya başlamıştır. Burada ÖYG kavramı; Özellik Yönelimli Alan Analizi yaklaşımı kullanılarak bir uygulama kümesinin özellikler bazında ayrıştırılması ve sonrasında bu özelliklerin bir yazılım süreç yaklaşımı olan Özellik Gündümlü Geliştirme (ÖGG) ile yönetilmesi anlamında kullanılmaktadır.

Klasik anlamda ÖGG yaklaşımı; özelliklerin modellenmesinden sonra bilinen nesnelere yardımcı ile gerçekleştirilmesini öngörmektedir. Bununla birlikte, Özellik Yönelimli Programlama yaklaşımı ise özelliklerin kendi başlarına ifade edilebildiği tamamen konuya özgü bir programlama modeli ortaya koymaktadır. Ancak bu tez çalışması; ÖYG vizyonu bünyesinde özelliklerin ifade edilebilmesi ve gerçekleştirilmesi için “iş süreçleri”, “iş kuralları” ve “iş servisleri”nden oluşan yeni bir model önermektedir.

Bu çalışma; ÖYG metodunun özellikle NYG metodu ile karşılaştırılması adına yazılım süreçlerini irdelenecek ve değerlendirecek, her iki metodun

performanslarını analiz edecek ve diđer uygun nitelikleri gözönünde tutacaktır. Sonuçta, ÖYG'nin avantaj ve dezavantajları diđer uygulama geliştirme yöntemleri ile de karşılaştırılacaktır.

Anahtar Kelimeler: Nesneye Yönelik Geliştirme, Özellikler, Özellik Güdümlü Geliştirme, Özellik Yönelimli Geliştirme, Servis Odaklı Mimari

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor Prof. Dr. Ziya Aktaş and co-supervisor Dr. Semih Çetin for their guidance, advice, criticism, encouragements, and insight throughout the research.

TABLE OF CONTENTS

STATEMENT OF NON PLAGIARISM	iii
ABSTRACT.....	iv
ÖZ	vi
ACKNOWLEDGMENTS.....	viii
TABLE OF CONTENTS	ix
LIST OF TABLES.....	xiii
LIST OF FIGURES.....	xiv
LIST OF ABBREVIATIONS.....	xvi
CHAPTERS:	
1. INTRODUCTION	1
1.1 Statement of the Problem	1
1.2 Objective of the Study	3
1.3 Organization of the Thesis.....	4
2. AN OVERVIEW OF SOFTWARE DEVELOPMENT METHODOLOGIES.....	6
2.1 Software Development Processes.....	6
2.2 Heavy Software Development Methodologies.....	7
2.2.1 Object-Oriented Development.....	8
2.2.2 Model-Driven Development.....	8
2.2.3 Software Product Line Engineering.....	9
2.3 Agile Software Development Methodologies	10
2.3.1 Scrum.....	10
2.3.2 Dynamic Systems Development Method	11
2.3.3 Crystal Methods	11
2.3.4 Lean Development.....	11

2.3.5	Extreme Programming.....	12
2.3.6	Adaptive Software Development.....	12
2.3.7	Feature-Driven Development.....	12
3.	USING FEATURES IN SOFTWARE DEVELOPMENT.....	13
3.1	Features in Software Development	13
3.2	Feature-Oriented Domain and Application Engineering.....	14
3.2.1	Feature-Driven Domain Engineering	15
3.2.1.1	Domain Analysis	16
3.2.1.2	Domain Design	17
3.2.1.3	Domain Implementation.....	18
3.2.2	Feature-Driven Application Engineering.....	19
3.2.2.1	Feature-Driven Development.....	19
3.2.2.2	Milestones.....	26
3.2.2.3	Best Practices	27
3.2.2.4	Role-Playing in Feature-Driven Development.....	28
3.3	The Relationship between Domain and Application Engineering.....	29
4.	THE PROPOSED APPROACH: FEATURE-ORIENTED DEVELOPMENT USING SERVICE-ORIENTED ARCHITECTURE	33
4.1	From Object-Orientation to Service-Orientation.....	34
4.2	Business Process Execution Language for Web Services	34
4.3	Business Process Modeling	36
4.3.1	Business Workflow Model (BWM).....	36
4.3.2	Business Rule Model (BRM).....	36
4.3.3	Business Computation Model (BCM)	38
4.4	Service-Oriented Architecture	40
4.5	The Proposed Approach	43
5.	THE IMPLEMENTATION WITH OBJECT-ORIENTED DEVELOPMENT	45
5.1	A Case Study: BIONET	45
5.2	Implementation with OOD Approach	46

5.2.1 Project Schedule	46
5.2.2 Object-Oriented Analysis and Design	46
5.2.3 Object-Oriented Model.....	48
5.2.4 Use Case Diagram.....	48
5.2.5 Activity Diagram.....	49
5.2.6 Class Diagram.....	49
6. THE IMPLEMENTATION WITH FEATURE-ORIENTED DEVELOPMENT	52
6.1 Implementation with FOD on SOA	52
6.2 Implementation Tools	53
6.2.1 Microsoft BizTalk Server 2006	53
6.2.2 What BizTalk Server 2006 Provides	54
6.3 Analysis using FODA	55
6.3.1 Context Diagram	55
6.3.2 Feature Model	56
6.4 FDD Practices	57
6.4.1 Domain Service Modeling	57
6.4.2 Build a Feature List.....	59
6.4.3 Plan by Feature	61
6.4.4 Design and Build by Feature List	61
7. COMPARATIVE ANALYSIS	64
7.1 Fundamentals	64
7.2 Requirements Analysis and System Behavior Issues	68
7.3 Architectural Issues	69
7.4 Analysis Issues	69
7.5 Design Issues.....	70
7.6 Implementation Issues	70
7.7 Testing Issues	71
7.8 Maintenance Issues.....	72
7.9 Administrative Issues	72

7.10 Tool Support	73
8. SUMMARY AND CONCLUSIONS	74
8.1 Summary	74
8.2 Conclusions	75
8.3 Extensions of the Study	77
REFERENCES.....	R1
APPENDICES:	
A. Desktop Application of BioNET Administration Module	A1
B. BioNET Credit Management Module.....	A2
C. BioNET Shopping Management Module.....	A3
D. BioNET Shopping Management Module Sale Processing.....	A4
E. BioNET Transition Management Module	A5
F. Sample BioNET Orchestrations, XML Schemas and Rule Compositions....	A6
F1: Read a Fingerprint.....	A6
F2: Verify a Fingerprint.....	A7
F3: Validate an Account.....	A9
F4: Calculate a Sale.....	A11
G. BioNET Web Services	A12
G1: ReadFingerprint Web Service.....	A13
G2: VerifyFingerprint Web Service	A14
G3: CalculateOfTheSale Web Service	A15
H. BioNET Web Service: VB.NET Code.....	A17

LIST OF TABLES

Table 1. A Sample of Feature List	24
Table 2. Milestones	26
Table 3. Features Database Table.....	60-61
Table 4. The Comparison of UP, FDD and XP	65
Table 5. The Comparison of OOD using RUP with FOD using FDD.....	66
Table 6. The Comparison of BioNET with FOD and BioNET with OOD	67-68

LIST OF FIGURES

Figure 1.	Product-Line Engineering Phases	15
Figure 2.	Domain Engineering Process	16
Figure 3.	A Sample Feature Model	18
Figure 4.	The Domain Neutral Component or a Class Diagram of UML	20
Figure 5.	The FDD Project Lifecycle	21
Figure 6.	The Design By Feature and Build By Feature Processes of FDD	22
Figure 7.	A Sample Overall Model	23
Figure 8.	Component Assembly in FDD	25
Figure 9.	Application Engineering Process	30
Figure 10.	The Relationship between Domain and Application Engineering	31
Figure 11.	Domain Engineering vs. Application Engineering	32
Figure 12.	The Proposed Approach: Feature-Oriented Development	33
Figure 13.	BPEL4WS Logical View	35
Figure 14.	A Sample Sequential Workflow	37
Figure 15.	A Sample Business Rule	38
Figure 16.	A Finite State Machine	38
Figure 17.	Object-Oriented Computation	39
Figure 18.	Conceptual Structure of SOA	41
Figure 19.	Web Services with Protocols	42
Figure 20.	Orchestration of Features in Business Process Modeling	43
Figure 21.	Project Schedule of BioNET	47
Figure 22.	Use Case Diagram of Shopping Module	48
Figure 23.	Activity Diagram of Shopping Module	50
Figure 24.	Class Diagram of Shopping Module	51
Figure 25.	Communication via Web Services	53
Figure 26.	BizTalk Server 2006 Engine and Its Interactions	54
Figure 27.	Context Diagram of Shopping Module	56
Figure 28.	The Feature Model of BioNET Shopping Module	57
Figure 29.	Business Workflow Model of VerifyFingerPrint	58
Figure 30.	Domain Service Model	59
Figure 31.	Business Orchestration in BizTalk Server 2006	62

Figure 32. A Sample XML Schema File.....	63
Figure 33. A Sample Business Rule Composition	63

LIST OF ABBREVIATIONS

AHEAD	: Algebraic Hierarchical Equations for Application Design
AM	: Agile Modeling
AOP	: Aspect-Oriented Programming
ASD	: Adaptive Software Development
ASDE	: Agile Software Development Ecosystem
BAM	: Business Activity Monitoring
BCM	: Business Computation Model
BPEL	: Business Process Execution Language
BPEL4WS	: Business Process Execution Language for Web Services
BPM	: Business Process Management
BPMN	: Business Process Modeling Notation
BRM	: Business Rule Model
BWM	: Business Workflow Model
CORDET	: Component-Oriented Development Techniques
DNC	: Domain Neutral Component
DSDM	: Dynamic Systems Development Method
FDD	: Feature-Driven Development
FOD	: Feature-Oriented Development
FODA	: Feature-Oriented Domain Analysis
FOP	: Feature-Oriented Programming
LD	: Lean Development
MDA	: Model-Driven Architecture
MDD	: Model-Driven Development
OOA	: Object-Oriented Analysis
OOAD	: Object-Oriented Architecture Design
OOADM	: Object-Oriented Analysis and Design Method
OOD	: Object-Oriented Development
OOP	: Object-Oriented Programming
OORA	: Object-Oriented Requirements Analysis
OOSDM	: Object-Oriented Software Development Methodology
PIM	: Platform Independent Model

PSM	: Platform Specific Model
RAD	: Rapid Application Development
RSEB	: Reuse-Driven Software Engineering Business
SDK	: Software Development Kit
SDLC	: Software Development Life Cycle
SEI	: Software Engineering Institute
SOA	: Service-Oriented Architecture
SOAP	: Simple Object Access Protocol
SPL	: Software Product Line
SPLE	: Software Product Line Engineering
SRS	: System Requirements Specification
SSD	: System Sequence Diagram
TDD	: Test-Driven Development
UDDI	: Universal Description, Discovery and Integration
UML	: Unified Modeling Language
USDP	: Unified Software Development Process
XML	: Extensible Markup Language
XP	: Extreme Programming
WSDL	: Web Services Description Language
WSIL	: Web Services Inspection Language

CHAPTER 1

INTRODUCTION

1.1 Statement of the Problem

During the last decade, several agile software development methodologies have emerged, which position themselves as an alternative to the traditional “waterfall” development model. Waterfall model divides the whole software development lifecycle into a number of stages, and it also assumes that each stage is 100% complete before the next stage starts. One of the main weaknesses of this approach is the fact that the design errors are often not discovered till deployment [Khramtchenko, 2004].

Object-Oriented Development (OOD), as a paradigm based on the basic building block of “objects”, has been used increasingly as an approach to cure the weaknesses of the waterfall approach. The use of object-oriented programming languages, object-oriented analysis and design methodologies, distributed object computing techniques, and object-oriented domain modeling languages have come to the scene for better quality software and improved reuse. During the decades, it had been sternly advocated that OO paradigm encompassed the complete view of software engineering without the loss of communication between the stages [Booch, 1993].

However, more detailed research on object technologies has also revealed that OOD, on its own, has some drawbacks in achieving better quality software especially in terms of the right level of abstraction and reuse [Rothenberger, 1999] [Fichman, 1997] [Pancake, 1995]. This mainly stems from the fact that OOD has been originally designed for completely meeting the functional requirements, whereas it has lack of design in mind to manage the crosscutting concerns based

on non-functional issues such as security, performance, and reliability in software development.

Managing these non-functional concerns crosscutting the objects has yielded another programming model known as Aspect-Oriented Programming (AOP), which proposes that applications are better structured by separately specifying the various concerns that can be weaved together into a coherent program [Kiczales, 1996] [Elrad et al., 2001]. These related concerns are grouped as “aspects”, and AOP provides appropriate isolation, composition and reuse of the code used to implement them. This is useful when these concerns are crosscutting design decisions that have many objects leading to different places in the code doing the same thing like logging [Kiczales et al., 1997].

While OOD tries to manage functional requirements by means of “objects” whereas AOP handles the non-functional issues crosscutting the objects by means of “aspects”, another school of thought has approached to the problem by means of encapsulating the interrelated functional and non-functional issues in a single building block, known as “feature”, to better model the software systems [Kang et al., 1990] [Jadhav et al.]. A feature model, including feature definitions and composition rules, describes a domain that not only includes the standard terms/concepts and their definitions, but also describes how they are related structurally and compositionally [Kang et al., 1998].

Current research and practical experience suggest that achieving significant progress with respect to the software reuse requires a paradigm shift towards modeling and developing software system families rather than individual systems. System Family Engineering (i.e. Product Line Engineering) seeks to exploit the commonalities among systems from a given problem domain while managing the variability among them in a systematic way [Kuloor and Eberlein, 2002] [Czarnecki, 2005]. The construction of product line architectures is divided into corresponding two phases: “domain engineering” that takes care of producing the common core architecture, and “application engineering” that derives the individual application from core architecture [Harsu, 2003].

1.2 Objective of the Study

Stemmed from the fact that features might abstract more cohesive building blocks in domain and application engineering rather than objects or aspects either individually or both together, they demand on more loosely coupled architectures. An emerging approach, known as Service-Oriented Architecture (SOA), can provide such a baseline to manage the component interaction where the components may be executed by business services, governed by business rules, and directed by business flows.

This thesis work proposes a Feature-Oriented Development (FOD) model where features can be expressed in terms of “business services”, “business rules”, and “business workflows”, and can be executed by proper frameworks based on Service-Oriented Architecture. The FOD model proposed here has two distinct stages: it starts first with a domain engineering activity where features are determined, modeled and represented by Feature-Oriented Domain Analysis (FODA) [Kang et al., 1990], and then the features are built in an agile and iterative manner according to the well-known Feature-Driven Development (FDD) method [Coad et al., 1999].

FDD has become quite popular recently with the tools and techniques for managing a software project. FDD is an agile and adaptive approach for developing systems within the context of separated software “features”. It is a reality of software development projects is that application requirements change for many reasons. The problem is worse in development projects using the traditional waterfall approach where there could be months between the initial requirements gathering and the construction, and testing of an application [Morrison]. FDD is designed to address such difficulties in software development.

The conventional FDD approach employs basic objects to build features, but it has already been realized that pure object-oriented models suffers from the crosscutting concerns that may be better expressed with aspects. However, using two different paradigms, OOD and AOP respectively, might complicate the development process, and [Batory et al., 2003] has introduced a step-wise refinement model accordingly to design and develop features by using a dedicated programming approach known as Feature-Oriented Programming (FOP) in order to deal with such difficulties. However, the AHEAD (Algebraic Hierarchical Equations

for Application Design) model proposed by this approach has also suffered from the lack of complete design and development environments, which complicates the development process as well.

Consequently, the FOD model proposed here uses a different programming model where features can be expressed in terms of “business services”, “business rules”, and “business workflows” all of which are the basic building blocks of today’s modern Service-Oriented Architecture and Business Process Management (BPM) paradigms. Moreover, proper frameworks and development environments directly support these building blocks as of today. As an example, the case study carried out within the context of FOD in the thesis has been completely modeled, designed, developed, and executed by Microsoft BizTalk Server 2006 environment. Apart from proposing an FOD model, this thesis work compares this Feature-Oriented Development model with classical Object-Oriented Development on a case study in order to experiment and validate the proposed model. This thesis work will also examine and evaluate the processes, analysis of the performance, time management and the advantages of FOD comparing it with OOD specifically, and also with other agile development methodologies and traditional incremental iterative approach. Eventually, the advantages and disadvantages of FOD will be summarized with respect to OOD methodology.

The expected gain from the planned studies will be specified how to tackle the recent core problems in software development, that of constructing the software correctly and delivering on time, using the FDD approach and FOD method.

1.3 Organization of the Thesis

The rest of the manuscript has been organized as follows: in Chapter 2, an overview of software development methodologies has been given where these methods have been mainly classified according to being formal or agile in nature.

Chapter 3 discusses the use of features in software development. This chapter introduces the feature concept as well as the use of features both in domain and application engineering. Regarding the domain engineering and especially domain analysis, Feature-Oriented Domain Analysis method has been introduced. For application engineering, the agile software development lifecycle of Feature-Driven Development has been given in detail.

The proposed approach for Feature-Oriented Development is introduced in Chapter 4. Basic constituents of the proposed approach, namely business services, business rules and business workflows have been given together with the adequate principles of service orientation and business process management.

Chapter 5 and 6 are dedicated to the implementation of a case problem with Object-Oriented Development and Feature-Oriented Development, respectively.

Chapter 7 provides the comparative analysis of the results obtained from Chapter 5 and 6, and Chapter 8 finally concludes the thesis.

CHAPTER 2

AN OVERVIEW OF SOFTWARE DEVELOPMENT METHODOLOGIES

2.1 Software Development Processes

A software development process defines the values, principles and practices used to achieve the goal of the software project. It aims, to promote best practices, to reduce the risks, to increase the productivity, try to satisfy customers' real needs, and to infuse a common vision and culture in a team [Hayes and Andrews].

Software engineers are highly skilled individuals and software development processes define how they work as a team. A development process is the way one organizes the creation of software systems. A software development process deals with the people, technologies, tools and organizational patterns. People defines a wide-range of people, with different skill-sets, are involved. Technologies are infrastructure upon which software will be based. Tools are software development and project management tools. Organizational Patterns define how team members interact [Hayes and Andrews].

There has been a lot of software development processes created over the years. [Hayes and Andrews] identifies a number of categories of processes, which are most real-life projects employ a blend of these:

1. Pure waterfall
2. Code-and-fix
3. Spiral
4. Modified Waterfalls
5. Evolutionary Prototyping
6. Staged Delivery

7. Evolutionary Delivery
8. Design-to-Schedule
9. Design-to-Tools
10. Commercial Off-the-Shelf Software

With the exception of code-and-fix, these processes have a few things in common – they assume that software development is analogous to a defined industrial process; they are based on physical engineering processes; they are predictive; and they assume that people can be treated as abstract resources.

As noted by Pressman [2005] software engineering methods can be categorized on a “formality” spectrum that is loosely tied to the degree of mathematical rigor applied during analysis and design. One can place the above stated methods into the informal end of the spectrum. A combination of diagrams, text, tables, and simple notation is used to create analysis and design models, but little mathematical rigor has been applied.

On the other end of the formality spectrum, say formal methods, a specification and design are described using a formal syntax and semantics that specify system function and behavior.

Informal methods may be divided into two as Heavy Software Development Methodologies and Agile Software Development Methodologies.

2.2 Heavy Software Development Methodologies

Regardless of the software development process lifecycle a methodology follows, every methodology has some certain characteristics to identify it from the others. As an example, each one uses different abstractions for software building blocks such as objects in Object-Oriented Development, models in Model-Driven Development, and assets in Software Product Line Engineering. These three development methodologies will be introduced briefly in the following subsections.

2.2.1 Object-Oriented Development (OOD)

Object Orientation (OO), as a paradigm, has been used increasingly as an approach to facilitate the reuse. The use of object-oriented programming languages, object-oriented analysis and design methodologies, distributed object computing techniques, and object-oriented domain modeling languages have come to scene for better quality software and improved reuse. During the last decades, it had been sternly advocated that OO paradigm encompassed the complete view of software engineering without the loss of communication [Booch, 1993].

The idea behind object orientation assumes that we have been living in a world of objects. Modeling, understanding, and developing objects are easier since they constitute a common vocabulary. The objects take place in nature, in human made entities, in businesses, and in the products that we use. Both data and the processing applied to that data have been encapsulated by objects. The practice of defining data structures and code in the same class keeps the elements that need to be reused as a unit within one framework, and encapsulation forces to clearly define the interfaces of each class to the outside world [Aktas and Cetin, 2006]. OOD has a common modeling language (Unified Modeling Language – UML) where all stakeholders of the OOD are expected to speak the same language to overcome the communication barriers.

The object-oriented paradigm has been attractive to many software development organizations with the expectation that it yields reusable classes and objects. While, at the same time, the software components derived using the object-oriented paradigm exhibit design characteristics (e.g. proper decomposition, functional independence, information hiding etc.) that are associated with high-quality software [Coad and Yourdon, 1991].

2.2.2 Model-Driven Development (MDD)

Model-Driven Development (MDD) is a development practice where high-level and iterative software models (often domain-specific) are created and evolved as software design and implementation takes place. The key characteristic of MDD is that the model literally becomes part of the development process. Contrast this with an approach such as the waterfall development process where modeling

appears as a separate step in the process and tends to get left behind once the development proceeds to the next phase [Schwaderer, 2006].

MDD is a model-centric software engineering approach, which aims at improving the quality and lifespan of software artifacts by focusing on models instead of code [Gitzel and Korthaus, 2004]. Models are considered as first class entities. A system is described by a family of models, each representing the system from a specific perspective and at a specific level of abstraction. Thus, working with models by means of refinement and transformation provides traceability between elements in different models.

The most important realization of MDD is definitely OMG's MDA [OMG]. The MDA approach comprises the creation of a Platform Independent Model (PIM), which is based on a suitable UML profile and represents business functionality and behavior and, subsequently, the semi-automatic or fully automatic transformation of the PIM into a Platform Specific Model (PSM). In the next step, code can then be generated from the PSM.

2.2.3 Software Product Line Engineering (SPLE)

A Software Product Line (SPL) is a set of software-intensive systems sharing a common, managed set of features that satisfies the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [Clements and Northrop, 2001]. A product line's scope is a description of the products that constitute the product line or what the product line is capable of producing. Within that scope, the disciplined reuse of core assets, such as requirements, designs, test cases, and other software development artifacts greatly reduces the cost of development.

The key objectives of SPLs are to capitalize on commonality and manage variation thus reduce the time, effort, cost, and complexity of creating and maintaining a different product line of similar software systems. Therefore, with the disciplined reuse of core assets and commonalities, SPLs can address problems such as dissatisfaction with current project performance, reduce cost and schedule, decrease complexity of managing and maintaining product variants, and quickly respond to customer / marketplace demands.

The key component enabling the effective resolution of these problems is the use of a product line architecture that allows an organization to identify and reuse software artifacts for the efficient creation of products sharing some commonality, but varying in known and managed ways. The architecture, in a sense, is the glue that holds the product line together [Zubrow and Chastek, 2003].

2.3 Agile Software Development Methodologies

Agile methods are approaches to managing the development of Internet products and services based on principles of flexible manufacturing and lean development. Agile methods have been a reaction to the rise of traditional software development methods, which were too large, expensive, rigid, and fraught with failure. Downsizing was the norm and large corporations in decline, rather than young, energetic firms on the rise were using traditional methods. Millions of websites were created overnight by anyone with a computer and a modicum of curiosity. Agile methods marked the end of traditional methods in the mind of their creators [Rico, 2006].

Agile methods emerged with a focus on early customer involvement, iterative development, self-organizing teams, and flexibility. Internet technologies such as HTML and Java were powerful new prototyping languages, enabling smaller teams to build bigger software products in record time. Because they could be built faster, customers could see the finished product sooner and provide earlier feedback, and developers could rapidly refine their products. This gave rise to closed-loop, circular, highly recursive, and tightly knit processes for rapidly creating Internet products, leading to increased customer satisfaction and firm performance [Hayes and Andrews].

Jim Highsmith has explored and compared the major agile methodologies. The following synopses are taken from the introduction to his book [2002].

2.3.1 Scrum

Ken Schwaber and Jeff Sutherland initially developed scrum, named for the scrum in Rugby, with later collaborations with Mike Beedle. Scrum provides a project management framework that focuses development into 30-day Sprint cycles in

which a specified set of Backlog features are delivered. The core practice in Scrum is the use of daily 15-minute team meetings for coordination and integration. Scrum has been in use for nearly ten years and has been used to successfully deliver a wide range of products.

2.3.2 Dynamic Systems Development Method (DSDM)

The Dynamic Systems Development Method (DSDM) was developed in the U.K. in the mid-1990s. It is an outgrowth of, and extension to, rapid application development (RAD) practices. DSDM boasts the best-supported training and documentation of any Agile Software Development Ecosystem (ASDE), at least in Europe. DSDM's nine principles include active user involvement, frequent delivery, team decision-making, integrated testing throughout the project life cycle, and reversible changes in development.

2.3.3 Crystal Methods

Alistair Cockburn is the author of the "Crystal" family of people-centered methods. Alistair is a "methodology archaeologist" who has interviewed dozens of project teams worldwide trying to separate what actually works from what people say should work. Alistair and Crystal focus on the people aspects of development – collaboration, good citizenship, and cooperation. Alistair uses project size, criticality, and objectives to craft appropriately configured practices for each member of the Crystal family of methodologies.

2.3.4 Lean Development (LD)

The most strategic-oriented development methodology might also be the least known: Bob Charette's Lean Development (LD), which is derived from the principles of lean production, the restructuring of the Japanese automobile industry that occurred in the 1980's. In LD, traditional methodology's view of change as a risk of loss to be controlled with restrictive management practices is extended to a view of change producing "opportunities" to be pursued during "risk entrepreneurship". LD has been used successfully on a number of large telecommunications projects in Europe.

2.3.5 Extreme Programming (XP)

Extreme Programming (XP) was developed by Kent Beck, Ward Cunningham, and Ron Jeffries. XP preaches the values of community, simplicity, feedback and courage. Important aspects of XP are its contribution to altering the view of the cost of change and its emphasis on technical excellence through refractory and test-first development. XP provides a system of dynamic practices, whose integrity as a holistic unit has been proven. XP has clearly garnered the most interest of any of the agile approaches.

2.3.6 Adaptive Software Development (ASD)

Adaptive Software Development (ASD) is Jim Highsmith's contribution to the Agile movement, and it provides a philosophical background for Agile methods, showing how software development organizations can respond to the turbulence of the current business climate by harnessing rather than avoiding change. ASD contains both practices – iterative development, feature-based planning, customer focus group reviews – and an “Agile” management philosophy called Leadership-Collaboration management.

2.3.7 Feature-Driven Development (FDD)

Feature-Driven Development (FDD) is a client-centric, architecture-centric and pragmatic software process. In FDD, the term client represents what Agile Modeling (AM) refers to as project stakeholders and Extreme Programming calls customers. Significantly, FDD contains just enough process to ensure scalability and repeatability, all the while encouraging creativity and innovation [Ambler, 1].

FDD was first introduced in 1999 [Coad et al., 1999]. It is a combination of the software process followed by De Luca's company and Coad's concept of features. FDD was first applied on a 15-month, 50-person project for a large Singapore bank in 1997, immediately followed by a second, 18-month, 250-person project [Coad et al., 1999]. In Chapter 3, FDD will be discussed with its processes in detail.

CHAPTER 3

USING FEATURES IN SOFTWARE DEVELOPMENT

Feature modeling originating from the Feature-Oriented Domain Analysis method has been commonly used in literature to represent the basic building blocks of modern software development of product line engineering. As part of the Domain Analysis method, feature models are used to describe and hierarchically structure common and variable features for product line-members. Features represent product capabilities and characteristics that are important to the user (stakeholder or external system). A feature indicating variability corresponds to a variation point [Berg, 2005].

This chapter examines the use of “features” in software development, modeling them in domain analysis, implementing them within the context of the agile Feature-Driven Development in application engineering, and the best practices of employing features.

3.1 Features in Software Development

A feature is a small, client-valued function expressed in the form of **<action><result><object>** [Palmer and Felsing, 2002].

Examples of features are [Palmer]:

- Calculate the *total* of a sale.
- Assess the *performance* of a salesman.
- Validate the *password* of a user.
- Retrieve the *balance* of a bank account.
- Authorize a *credit card transaction* of a card-holder.
- Perform a *scheduled service* on a car.

The explicit template in the form of <action> <result> <object> provides some strong clues to the **operations** required in the system and the **classes** to which they should be applied. For example:

- "Calculate the *total* of a sale" suggests a calculateTotal() operation in a Sale class.
- "Assess the *performance* of a salesman" suggests an assessPerformance() operation in a Salesman class.
- "Determine the *validity of the password* of a user" suggests a determinePasswordValidity() operation on a User class that can then be simplified into a validatePassword() operation on the User class.

The use of a natural language, such as English, means that the technique is far from foolproof. However, after a little practice, it becomes a powerful source of clues to use in discovering or verifying operations and classes [Palmer].

Features are to **FDD** as **use cases** are to the **Rational Unified Process (RUP)** and **user stories** are to **Extreme Programming (XP)** – they're a primary source of requirements and the primary input into your planning efforts [Ambler, 2].

3.2 Feature-Oriented Domain and Application Engineering

Using features in software development is not quite new but it has been widely anticipated by the vision of Software Product Line Engineering (SPLE). SPLE demands on a product line architecture where features can be modeled, implemented and deployed accordingly. Product-line architectures emphasize software reuse among several closely related applications. Concerning product-line architectures, the requirements analysis and design of such applications are carried out together. These applications form a family sharing the same core architecture. Each application typically has a variant part the design of which is also supported by product-line architectures, for example, via parameterization. It is essential to find out the common features and components of the applications belonging to the same family. Thus, the requirements analysis and design of a family of applications are more complicated than those of a single system [Harsu, 2003].

Product-line software architectures consist of two parts: the common (application-independent) core architecture and the variant (application-specific) architecture.

The former part includes those components that are common for (at least almost) the whole family of applications sharing the same architecture. The latter part includes (specialized) components that are specific for an individual application. In the same way, the construction of product-line architectures is divided into corresponding two phases. The first phase, called **domain engineering**, takes care of producing the common core architecture, while the second phase, called **application engineering**, derives individual application from the core architecture. The second phase consists typically of composing and specializing the components by parameterization. This whole product-line architecting process is depicted in Figure 1 [Harsu, 2003].

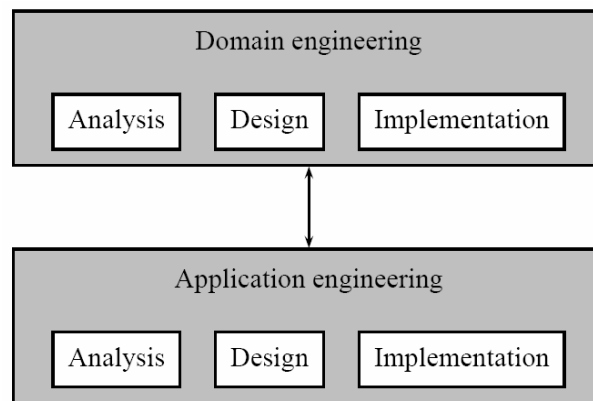


Figure 1. Product-Line Engineering Phases

In the following subsections domain engineering and application engineering are briefly summarized.

3.2.1 Feature-Driven Domain Engineering

Domain engineering is a process for creating a competence in application engineering for a family of similar systems. Domain engineering covers all the activities for building software core assets. These activities include identifying one or more domains, capturing the variation within a domain (domain analysis), constructing an adaptable design (domain design), and defining the mechanisms for translating requirements into systems with reusable components (domain implementation). The products (or software assets) of these activities are domain

model(s), design model(s), domain-specific languages, code generators, and code components [SEI].

The Domain Engineering Process is divided into three phases: Domain analysis, Domain Design and Domain Implementation as shown in Figure 2 [Olivier].

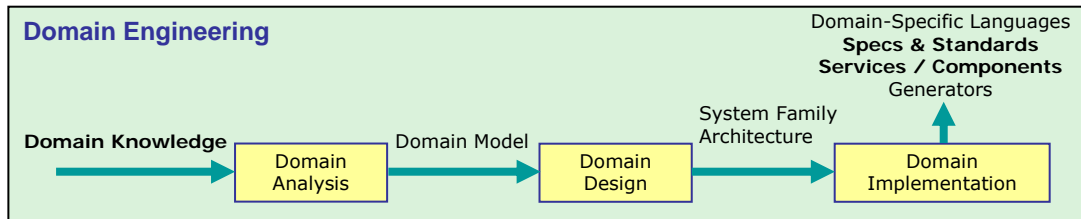


Figure 2. Domain Engineering Process

3.2.1.1 Domain Analysis

Feature-Oriented Domain Analysis (FODA) is a domain analysis and engineering technique, which focuses on developing reusable core assets for multiple products in the domain [Kollu, 2005]. Domain analysis is “the process of identifying, collecting, organizing and representing relevant information in a domain based on the study of existing systems and their development history” [Kang et al., 1990].

Domain Analysis is the activity that discovers and formally describes the commonalities and variability within a domain. The domain engineer captures and organizes this information in a set of domain models with the end of making it reusable when new systems. The output of domain analysis is a domain model: an explicit representation of knowledge about the domain [GMV, 2007].

The Domain Model [GMV, 2007] will consist of:

- Domain dictionary (domain lexicon)
- Context model (using e.g. diagrams, formalisms,) and
- Feature models

The Domain dictionary provides and defines the terms concerning the domain. Its purpose is to make communication among developers and other stakeholders easier and more precise.

The Context model specifies the boundaries of the domain. The model considers both the commonalities and variabilities of the application in the domain.

The Feature model is a hierarchical decomposition of features. Feature model that also tell which combinations of features are meaningful can depict features. Feature models provide notations for different kinds of features such as the FODA-like features.

FODA feature models describe mandatory, optional and alternative properties of concepts within domain. A filled circle at the top of the feature identifies a mandatory feature. A mandatory feature must be selected in all the systems of a domain. An empty circle at the top of the feature identifies as optional feature. Optional features are only present in the application if the customer has chosen them. An arc spanning two or more edges of the feature nodes depicts as set of alternative features. The term alternative feature indicates that a system can possess only one sub-feature at a time for main feature [Griss et al., 1998].

As an example of the requirements of representation of a Feature Model [Benavides, 2006], the features are from the automotive industry where features are used to specify and build software for configurable cars. In order to clarify the subject, a simple example is used where one considers only the features of transmission type (automatic or manual), engine type (electric or gasoline), and the option of cruise control. See Figure 3 from [Benavides, 2006] for this widely used example.

3.2.1.2 Domain Design

The Domain Design takes a Domain Model as input and applies Partitioning Strategy Architecture as a control model to produce a Generic Design. According to the domain models, it should also be selected which components or items (such as requirements) are provided in the core architecture and which items are implemented as variations in individual applications [GMV, 2007].

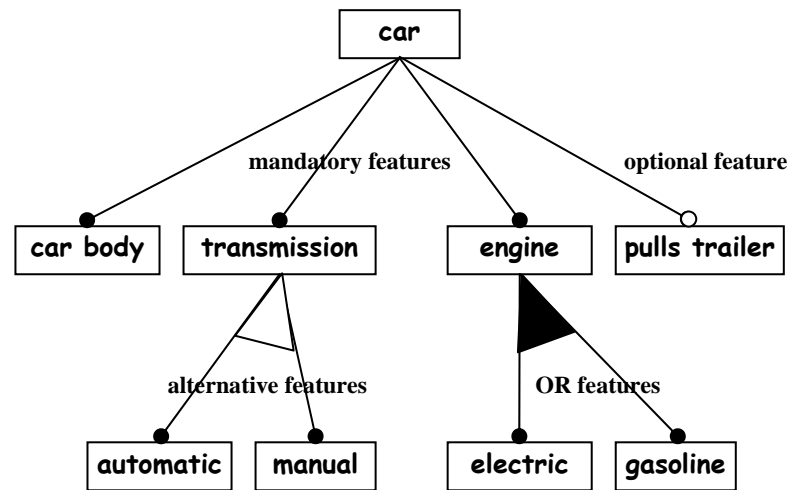


Figure 3.a) A Sample Feature Model

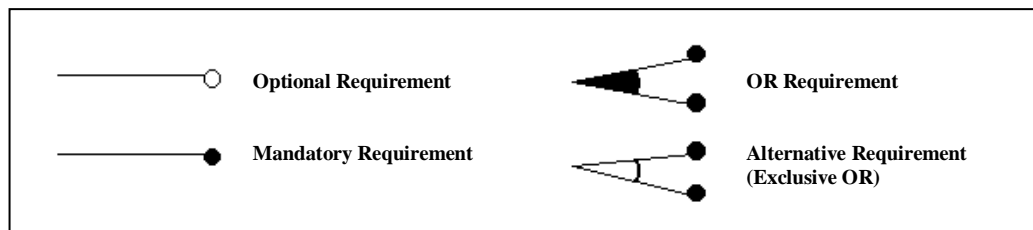


Figure 3.b) Requirements Representation of a Feature Model

The partitioning strategy defines the elements (e.g. subsystem, objects, data types etc.) and how the domain features are allocated to them. Selection of a strategy in part depends on the major factors of change identified in the domain models. In advance to the domain design, the domain implementation takes as inputs the design models and the generic architectures designed to identify and create reusable assets. The main outputs are these reusable and also application generators and domain languages [GMV, 2007].

3.2.1.3 Domain Implementation

The domain implementation takes as inputs the design models and the generic architectures designed to identify and create reusable assets. The main outputs are these reusable and also application generators and domain languages [GMV, 2007].

3.2.2 Feature-Driven Application Engineering

An application engineering activity conducted in parallel to feature-oriented domain engineering is highly probable either “feature-oriented” or “feature-driven”. The basic difference between being “feature-oriented” and “feature-driven” is whether using the “features” as first class entities or not throughout the stages. A “feature-oriented” one treats the features as first class entities whereas a “feature-driven” one makes use of features but they do not treated them as first class entities. The proposed approach in this thesis does not treat the features as first class entities in application engineering and it anticipates the Feature-Driven Development process model as the application engineering methodology.

3.2.2.1 Feature-Driven Development (FDD)

Feature-Driven Development (FDD) is a client-centric, architecture-centric and pragmatic software process [Ambler, 2], which follows the belief that a strong design will create a process that is better managed and thus more efficient. The project is divided into "features," which are small pieces of the project that possess some customer value. FDD creates design, code, and code inspection schedules that lack the depth and mounds of paperwork associated with a system completely specified in the requirements phase, instead relying on people and their roles to address the details as needed [Palmer and Felsing, 2002].

As the name suggests, FDD is "feature-driven", which means that it makes use of the concept of a **feature**". A feature in FDD is [Cause, 2004]:

- a. **Small:** 1-10 days of effort are required to complete it, mostly 1-3 days. They are designed and built in batches. The batch is the work package. A work package cannot take more than 10 days.
- b. **Client valued:** it is relevant and has a meaning to the business; in business systems this usually relates to a step within some business activity within a business process.
- c. **Named:** <action><result><object> naming template has proper prepositions between them <**action**> **the** <**result**> <**by**|**for**|**of**|**to**> **a(n)** <**object**>.

In the Coad Method [FDD], definition of a feature is based on the following template:

<action>[a | the]<result> [of | to | for | from | ...]<object>[with | for | of | ...]<parameters>

Features are defined based on a domain model. The domain model is created using a technique called the Unified Modeling Language (UML) class diagram and Peter Coad's enhanced technique called the Domain Neutral Component (DNC) and class archetypes, as shown in Figure 4 [Coad, 1999].

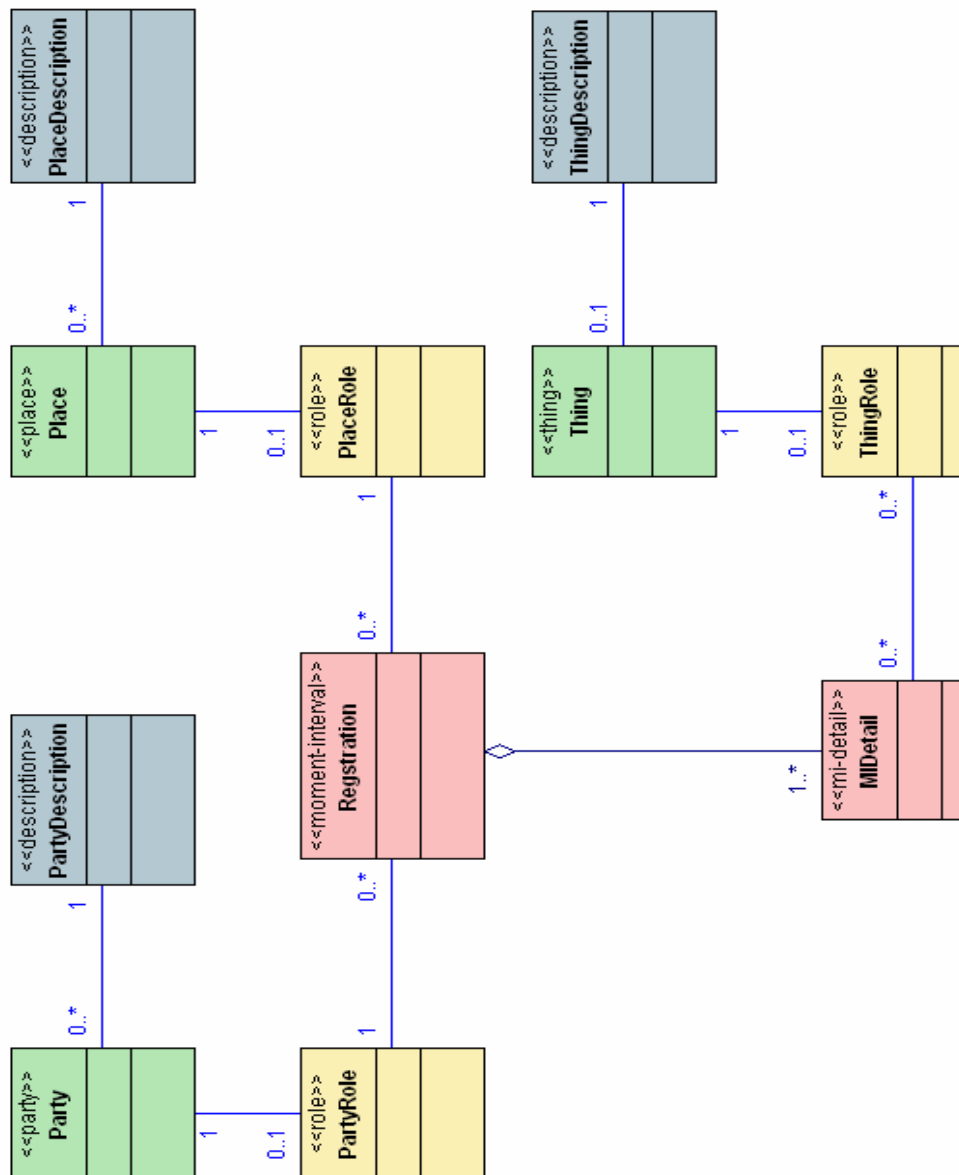


Figure 4. The Domain Neutral Component or a Class Diagram of UML

As Figure 5 from [Palmer and Felsing, 2002] depicts, there are five main activities in FDD that are performed iteratively:

- a) Develop an Overall Model
- b) Build a Feature List
- c) Plan by Feature
- d) Design by Feature
- e) Build by Feature

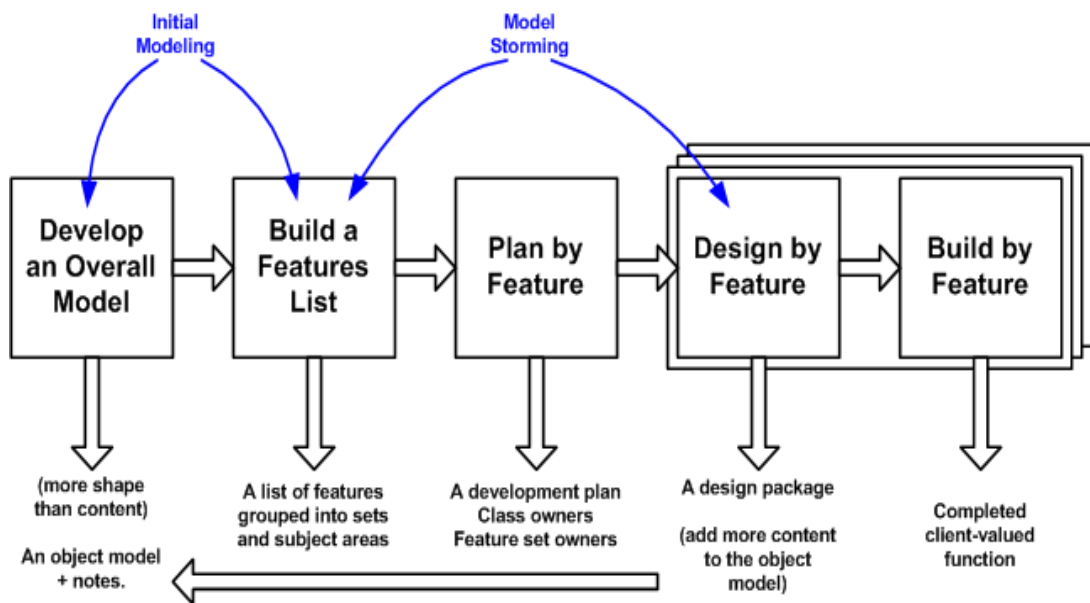


Figure 5. The FDD Project Lifecycle

The first is **Develop an Overall Model**, the initial result being a high-level object model and notes. At the start of a project your goal is to identify and understand the fundamentals of the domain that your system is addressing, and throughout the project you will flesh this model out to reflect what you're building [Ambler, 2].

The second step is **Build a Features List**, grouping them into **related sets and subject areas**. Next you **Plan By Feature**, the end result being a development, the identification of class owners and the identification of feature set owners [Ambler, 2].

The majority of the effort on an FDD project, roughly 75%, is comprised of the fourth and fifth steps namely, **Design by Feature** and **Build by Feature**. These two activities are exactly what you would expect; they include tasks such as detailed modeling, programming, testing, and packaging of the system. During the first three sequential activities an overall model shape is established. The final two activities are iterated for each feature [Ambler, 2].

This iterative process includes such tasks as design inspection, coding, unit testing, integration, and code inspection. After a successful iteration, the completed features are promoted to the main build while the iteration of designing and building begins with a new group of features taken from the feature set in Figure 6 [Abrahamsson, 2002].

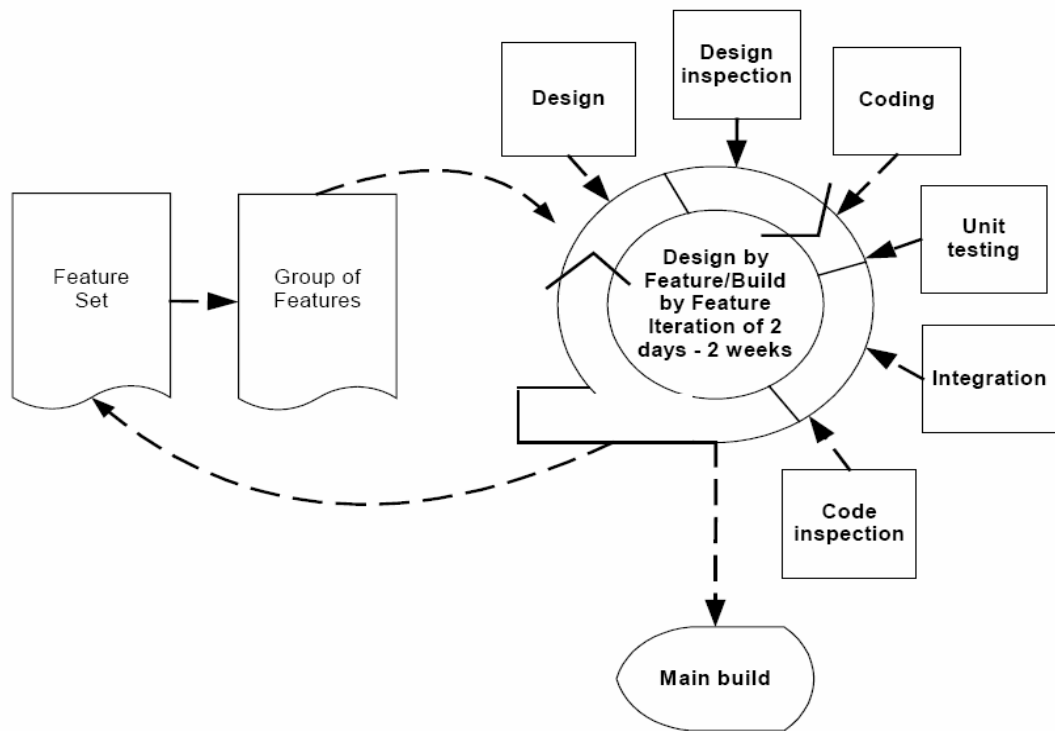


Figure 6. The Design By Feature and Build by Feature Processes of FDD

The following subsections are excerpted from [Palmer and Felsing, 2002] unless specified or referenced otherwise.

a) Develop Overall Model

The project starts with a high-level walkthrough of the scope of the system and its context. Next, detailed domain walkthroughs are held for each modeling area. In support of each domain, walkthrough models are then composed by small groups, which are presented for peer review and discussion. One of the proposed models or a merge of them is selected which becomes the model for that particular domain area. Domain area models are merged into an overall model, the overall model shape being adjusted along the way. Figure 7 represents order process as a link of a Feature Set to one or more objects in the object model [Morrison].

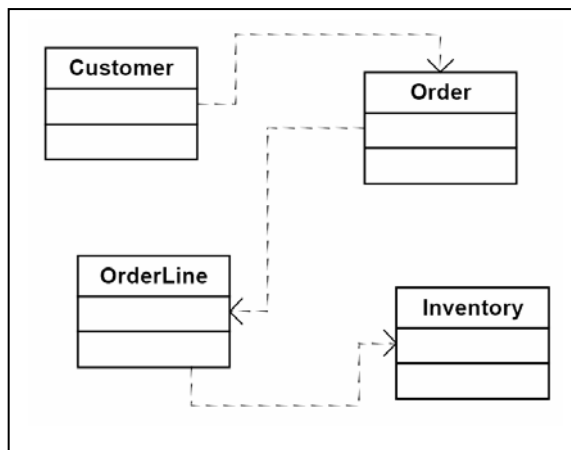


Figure 7. A Sample Overall Model

b) Build Feature List

The knowledge that is gathered during the initial modeling is used to identify a list of features. This is done by functionally decomposing the domain into subject areas. Subject areas each contain business activities; the steps within each business activity form the categorized feature list. As noted earlier, features in this respect are small pieces of client-valued functions expressed in the form **<action>** **<result>** **<object>**. Features should not take more than two weeks to complete; else they should be broken down into smaller pieces [FDD]. A sample feature list is shown in Table 1 [Morrison].

Table 1. A Sample of Feature List

Feature List				
ID	Feature Group	Feature Set	Feature Name	Object
2.1.1.1	Maintenance	Customer Maintenance	Add a new customer to customer list	Customer
2.1.1.2	Maintenance	Display Customers	Display list of customers	Customer
....				
2.1.2.1	Maintenance	Inventory Maintenance	Add Items to Inventory	Inventory
....				
2.2.1.1	Order Entry	Inventory Control	Reduce total inventory for product Inventory	OrderDetail
2.2.1.2	Order Entry	Inventory Control	Check available inventory for product Inventory	OrderDetail
....				
2.2.2.1	Order Entry	Create Order	Add a new order for customer Order	Customer
2.2.2.2	Order Entry	Create Order	Add a product to order Order	OrderDetail

Thus, FDD offers a mechanism for defining value in a fine-grained manner and for tracking the flow of the value through a set of transformative steps [Anderson, 2004].

Features are grouped into collections known simply as “feature sets”. Each set of features is associated with a single <<Moment-Interval>> archetype class (link) on the domain model. In turn, features sets are grouped into collections known as “subject areas”. Each subject area is associated with a sequence of <<Moment-Interval>> archetype classes on the domain model. In this respect FDD is

analogous to a V-Plant where very small components, called features, are constructed from a raw material, a feature description and subject matter expertise. They are, then, assembled into larger components of greater value called feature sets and, then, yet larger components of even greater value called subject areas, as shown in Figure 8. The flow of value in FDD happens in a V-Plant model [Anderson, 2004].

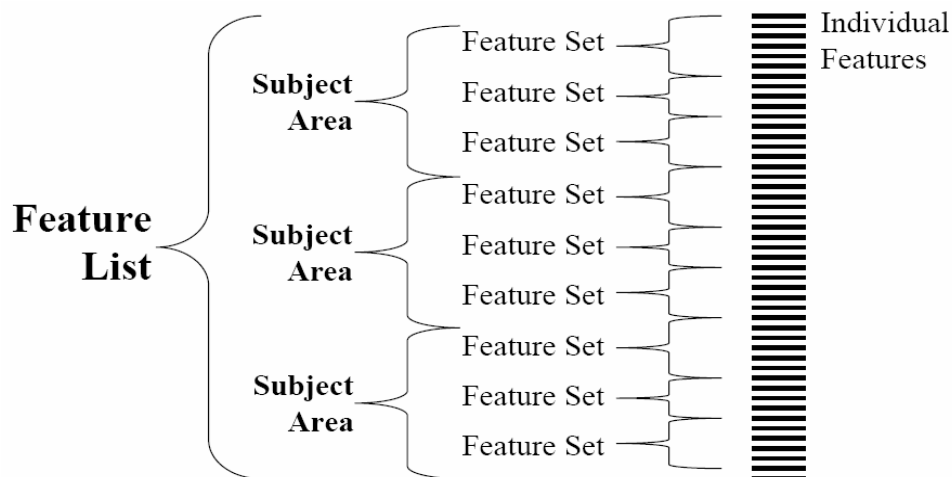


Figure 8. Component Assembly in FDD in a V-Plant Model

c) Design by Feature

A design package is produced for each feature. A chief programmer selects a small group of features that are to be developed within two weeks. Together with the corresponding class owners, the chief programmer works out detailed sequence diagrams for each feature and refines the overall model. Next, the class and method prologues are written and finally a design inspection is held.

d) Plan by Feature

Now that the feature list is complete, the next step is to produce the development plan. Class ownership is accomplished by ordering and assigning features (or feature sets) as classes to chief programmers.

e) Build by Feature

After a successful design inspection for each feature activity, a complete client-valued function (feature) is being produced. The class owners develop the actual code for their classes. After a unit test and a successful code inspection, the completed feature is promoted to the main build.

3.2.2.2 Milestones

Since features are small, completing a feature is a relatively small task. For accurate state reporting and keeping track of the software development project it is, however, important to mark the progress made on each feature. FDD therefore defines six milestones per feature that are to be completed sequentially. The first three milestones are completed during the “Design by Feature” activity; the last three are completed during the “Build by Feature” activity. To help with tracking progress, a percentage completed is assigned to each milestone. In Table 2, the milestones and their completion percentage are shown where the feature that is still being coded is 44% complete (Domain Walkthrough 1%, Design 40% and Design Inspection 3% = 44%) [Anderson, 2004].

Table 2. Milestones

Domain Walkthrough	1%
Design	40%
Design Inspection	3%
Code	45%
Code Inspection and Unit Test	10%
Promote To Build	1%

The terms in Table 2 are defined as follows [Anderson, 2004]:

1. **Domain Walkthrough** – explanation of the requirement to the developers (face-to-face)
2. **Design** – creation of the sequence diagram
3. **Design Inspection** – peer review to check the design meets the requirements
4. **Code** – methods are written in class files to deliver the design
5. **Code Inspection and Unit Test** – test & peer review to check that code does what was specified in the design
6. **Promote To Build** – into the integrated build for system / product testing

3.2.2.3 Best practices

Feature-Driven Development is built around a core set of industry-recognized best practices, derived from software engineering. These practices are all driven from a client-valued feature perspective. It is the combination of these practices and techniques that makes FDD so compelling. The best practices that make up FDD are shortly described below. For each best practice, a short description will be given from [Palmer and Felsing, 2002].

a) Domain Object Modeling: Domain Object Modeling consists of exploring and explaining the domain of the problem to be solved. The resulting domain object model provides an overall framework in which to add features.

b) Developing by Feature: Any function that is too complex to be implemented within two weeks is further decomposed into smaller functions until each sub-problem is small enough to be called a feature. This makes it easier to deliver correct functions and to extend or modify the system.

c) Individual Class (Code) Ownership: Individual class ownership means that distinct pieces or grouping of code are assigned to a single owner. The owner is responsible for the consistency, performance, and conceptual integrity of the class.

d) Feature Teams: A feature team is a small, dynamically formed team that develops a small activity. By doing so, multiple minds are always applied to each design decision and also multiple design options are always evaluated before one is chosen.

e) Inspections: Inspections are carried out to ensure good quality design and code, primarily by detection of defects.

f) Configuration Management: Configuration management helps with identifying the source code for all features that have been completed to date and to maintain a history of changes to classes as feature teams enhance them.

g) Regular Builds: Regular builds ensure there is always an up to date system that can be demonstrated to the client and helps highlighting integration errors of source code for the features early.

h) Visibility of progress and results: By frequent, appropriate, and accurate progress reporting at all levels inside and outside the project, based on completed work, managers are helped at steering a project correctly.

3.2.2.4 Role-Playing in Feature-Driven Development

There are several roles recommended for every FDD project. Besides the Project Manager who maintains the expected project responsibilities, there is the Chief Architect who is responsible for the overall technical design of the solution. The Chief Architect is primarily involved early in the project when the problem domain is being documented. One or more Chief Programmers are assigned to a project as senior developers. A Chief Programmer will be responsible for the day-to-day tracking of development progress. Class Owners are developers who are given responsibility for a particular piece of the application [Ambler, 1].

Whether or not these roles are held by individual people depends on the size of the organization and project. In many cases a Project Manager may also be a Chief Architect and Chief Programmers are often Class Owners as well [Morrison]. The final role required in FDD is the Domain Expert(s). They are the subject matter experts that will be working with the project team to explain and document the problem domain, as well as identify and prioritize the features required [Ambler, 2].

FDD also defines a collection of supporting roles, including:

- Domain Manager
- Release Manager

- Language Guru
- Build Engineer
- Toolsmith
- System Administrator
- Tester
- Deployer
- Technical Writer

3.3 The Relationship between Domain and Application Engineering

Application engineering uses the production facilities provided during domain engineering to produce applications of the family quickly. However, as shown already in Figure 1, application engineering can be performed in parallel to the domain engineering. Application engineering exploits those parts (tools, components) that are already available and implemented in the context of domain engineering [Harsu, 2003].

The applications should satisfy customer requirements, and thus, application engineering is connected to customers either directly or via other people. Application engineering is an iterative process, because the customers are not necessarily satisfied with the application for the first time, and they may suggest improvements. Application engineering is also iterative with domain engineering, because the custom suggestions may have effect on the core architecture, and thus, on domain engineering [Harsu, 2003].

Application Engineering Process also has three elements as shown in Figure 9 [Olivier]:

- Requirements Analysis
- Product Configuration
- Integration and Testing

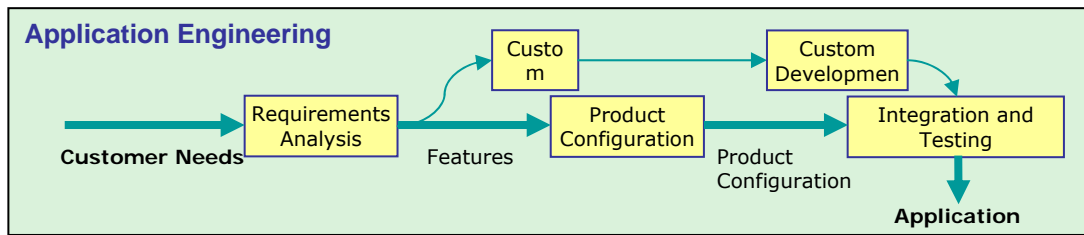


Figure 9. Application Engineering Process

Application engineering (also referred to as product development) is “development with reuse”, where concrete applications are built using the reusable assets. Just as traditional system engineering, it starts with requirements elicitation, analysis, and specification; however, the requirements are specified as a configuration of some generic system requirements produced in domain engineering. The requirements specification is the main input for system derivation, which is the manual or automated construction of the system from the reusable assets [Czarnecki, 2005].

As seen in Figure 10 [Olivier], domain and application engineering feed on each other: domain-engineering supplies application engineering with the reusable assets, whereas application engineering feeds back new requirements to domain engineering. This is so because application engineers identify the requirements for each given system to be built and may be faced with requirements that are not covered by the existing reusable assets. Therefore, some amount of application-specific development or tailoring is often required in order to quickly respond to the customer’s needs. However, the new requirements should be fed back into domain engineering in order to keep the reusable assets in sync with the product needs [Czarnecki, 2005].

Domain engineering can be applied at different levels of maturity. At minimum, domain analysis activities can be used to establish a common terminology among different product-development teams. The next level is to introduce a common architecture for a set of systems. Further advancement is to provide a set of components covering parts or all of the systems in the system family. Finally, the assembly of these components can be partially or fully automated using generators and/or configurators. The last level represents the focus of generative software development. In general, the generated products may also contain non-software

artifacts, such as test plans, manuals, tutorials, maintenance guidelines, etc. [Czarnecki, 2005].

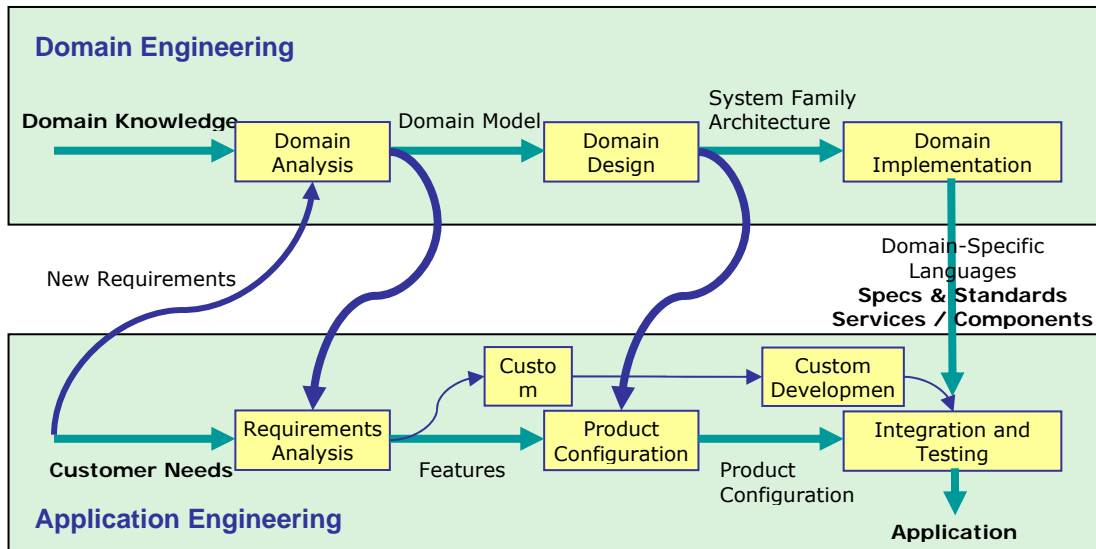


Figure 10. The Relationship between Domain and Application Engineering

Domain engineering is a process that aims at identifying, representing and implementing reusable artifacts. Examples of methods that apply the “domain engineering” principles are FODA and Reuse-Driven Software Engineering Business (RSEB) [Griss et al. 1998]. Figure 11 shows the possible interactions between domain engineering and application engineering [Bragança and Machado, 2004].

The artifacts produced by domain engineering can be reused in application engineering. The arrows from application engineering to domain engineering show that application engineering can supply input to domain engineering, usually in the form of domain knowledge. Each new application that is built within the domain will gain from reusing domain artifacts but will also provide knowledge to refine the domain artifacts or build new ones.

The artifacts that result from the domain engineering process consist on common components to be used in applications but also on components implementing the variants of a variability point that will be used only in specific applications. This is

also what is needed when building flexible and extensible applications based on variability. So, instead of using domain engineering as a process to support the development of diverse applications in a domain, it may be used to develop the variants of variability points in single applications.

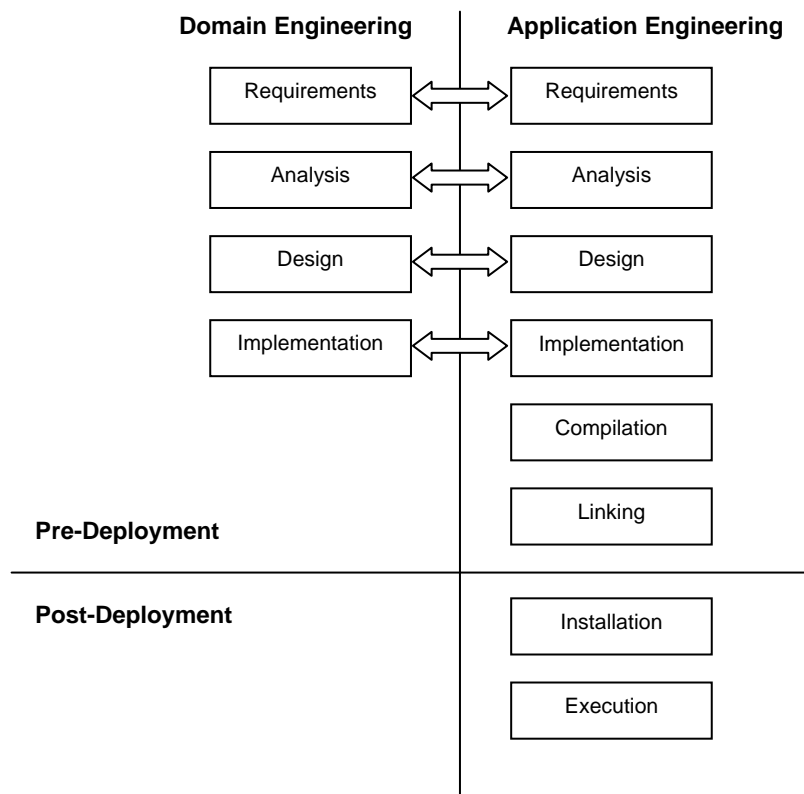


Figure 11. Domain Engineering vs. Application Engineering

Domain engineering methods can be adopted to support the design of variability points in single application development processes. In fact, if we consider each variability point a domain, then domain engineering methods can be applied for each variability point. Since domain engineering methods require extra effort in the engineering process, care must be taken in the selection of variability points that should be designed using domain engineering methods [Bragança and Machado, 2004].

CHAPTER 4

THE PROPOSED APPROACH: FEATURE-ORIENTED DEVELOPMENT USING SERVICE-ORIENTED ARCHITECTURE

In this chapter, the proposed approach will be explained in detail. The basic concept of the proposed Feature-Oriented Development approach, which is shown in Figure 12, is to provide adequate design models, software components, and techniques in domain engineering by Feature-Oriented Domain Analysis method to support the actual development process in application engineering through Feature-Driven Development. FOD specifies and implements the “features” in an application within the domain artifact of Web services based on an emerging paradigm of Service-Oriented Architecture (SOA). Hence, the proposed approach has been technically identified as **“Feature-Oriented Development using Service-Oriented Architecture”**.

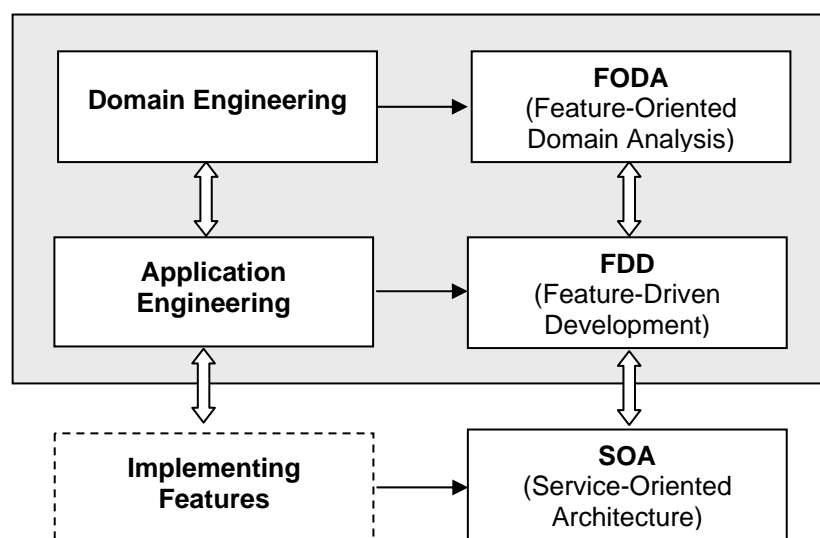


Figure 12. The Proposed Approach: Feature-Oriented Development

4.1 From Object Orientation to Service Orientation

This section is based on [Fancey]. The problem of coordinating and managing messages as they are routed from service to service comes onto the stage with the ability to easily integrate with and invoke Web services. As time passes, it will become increasingly unlikely that any interesting business processes will execute in a single message exchange. Traditionally, implementing business logic meant writing code (C++ or C#, for example). With some system, however, many common requirements can be realized without coding a single line. It seems reasonable to code manually when dealing with a few services, but this becomes more problematic as the number and complexity of services increases. Business Process Execution Language (BPEL) is based on XML, WSDL, and XML Schema.

Successful SOA implementations require reusable logic and service autonomy. In order to achieve them, you need to start thinking of applications as the collaboration of message exchanges. Of course, when you begin exposing application functionality as services, you give up a certain amount of control because you no longer know how your services are being employed or even who is using them.

Designing an application architecture that fits this description using existing standards in such a way that the specifications themselves do not intrude on the process is the great challenge in Web services today.

4.2 Business Process Execution Language for Web Services (BPEL4WS)

In July 2002, BEA, IBM, and Microsoft released a trio of specifications designed to support business transactions over Web services. These specifications, **BPEL4WS** (Business Process Execution Language for Web Services), WS-Transaction, and WS-Coordination, together form the bedrock for reliably choreographing Web services based applications, providing business process management, transactional integrity, and generic coordination facilities respectively [Webber].

The value of BPEL4WS is that if a business is the sum of its processes, the orchestration and refinement of those processes is critical to an enterprise's continued viability in the marketplace. Those businesses whose processes are agile

and flexible will be able to adapt rapidly to and exploit new market conditions. This section introduces the key features of “Business Process Execution Language for Web Services”, and shows how it builds on the features offered by WS-Coordination and WS-Transaction to support the reliable orchestration of business processes [Webber].

The BPEL4WS model is built upon a number of layers, with each layer building on the facilities of the previous one [Webber]. Figure 13 shows the fundamental components of the BPEL4WS architecture [Webber].

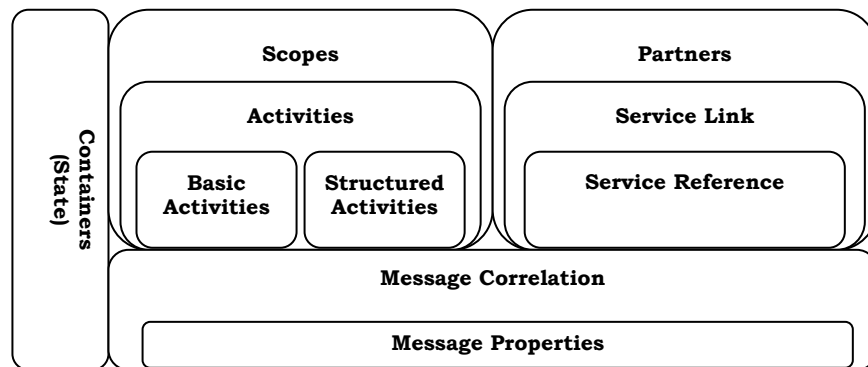


Figure 13. BPEL4WS Logical View

The fundamental components are:

- a means of capturing enterprise interdependencies with partners and associated service links;
- a message correlation layer which ties together messages and specific workflow instances
- state management features to maintain, update and interrogate parts of process state as a workflow progresses;
- scopes where individual activities (workflow stages) are composed to form actual algorithmic workflows.

4.3 Business Process Modeling (BPM)

Business Process Modeling (BPM) is the discipline of defining and outlining business practices, processes, information flows, data stores and systems. BPM often involves using a notation such as UML to capture graphical representations of the major processes, flows and stores [Sparx Systems].

“Business Process Modeling is an important part of understanding and restructuring the activities and information a typical enterprise uses to achieve its business goals. With a particular modeling tool, you can model, document and restructure those processes and information flows using industry standard UML and the Business Process Modeling Notation (BPMN). Best of all, the process designs and models can be used to drive process re-structuring and software development.” [Sparx Systems]

4.3.1 Business Workflow Model (BWM)

An example of sequential workflow is shown in Figure 14 [Schapiro]. Workflow systems are currently the leading technology for supporting business processes. This technology manages the execution of the tasks involved in a business activity, the scheduling of resources and the control of the flow of the associated information required by performers to execute the tasks. Typically the tasks involved in the business process are interdependent in that the execution of one task is conditional upon the execution of one or a number of other tasks [Mangan and Sadiq, 2003].

4.3.2 Business Rule Model (BRM)

A **business rules model** is a software system that helps manage and automate business rules. The rules a business follows may come from legal regulation ("An employee can be fired for any reason or no reason but not for an illegal reason"), company policy ("All customers that buy more than \$100 at one time will receive a 10% discount") or other sources. The Rule Engine software, among other functions, may help to register, classify and manage all these rules; verify consistency of formal rules ("Flooring material must be flattish to ease cleaning" is

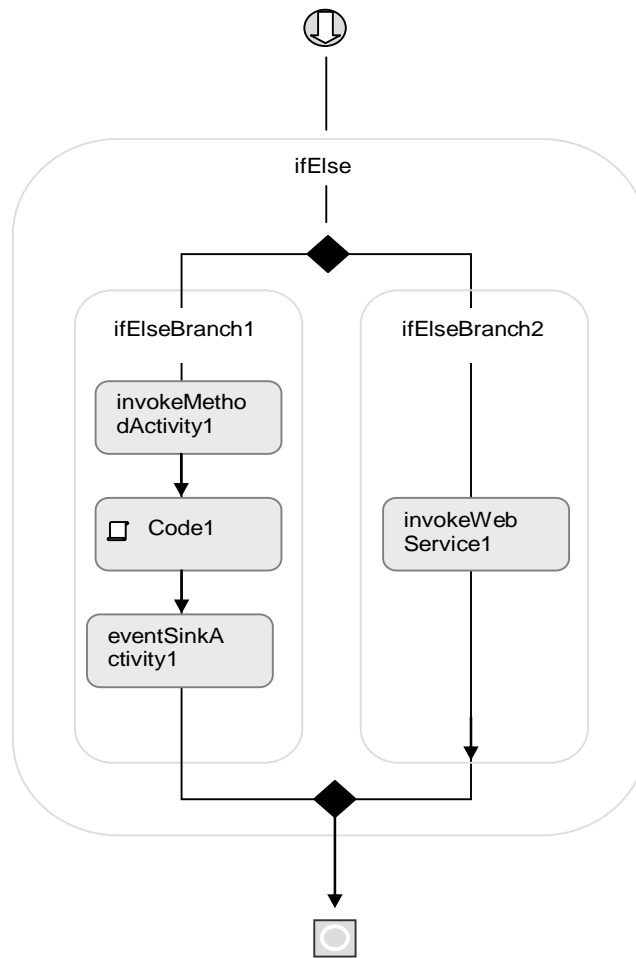


Figure 14. A Sample Sequential Workflow

inconsistent with "flooring material must be rough to avoid slipping"); infer some rules based on other rules; and relate some of these rules to Information Technology applications that are affected or need to enforce one or more of the rules. Rules can also be used to detect interesting business situations automatically. For example, "notify sales when inventory is lower than 10 and we have more than 5 pending orders on a Monday". In Figure 15, a simple business rule is represented. The conditions may be collected either from a database tables or a defined schema property or a combination of them.



Figure 15. A Sample Business Rule

4.3.3 Business Computation Model (BCM)

“Model of computation” is an academic term for a particular way of representing the behavior of a system, such as the finite state machine seen in Figure 16 [Garcia], or a data flow representation. A particular model of computation may better reveal execution characteristics of the system [Garcia].

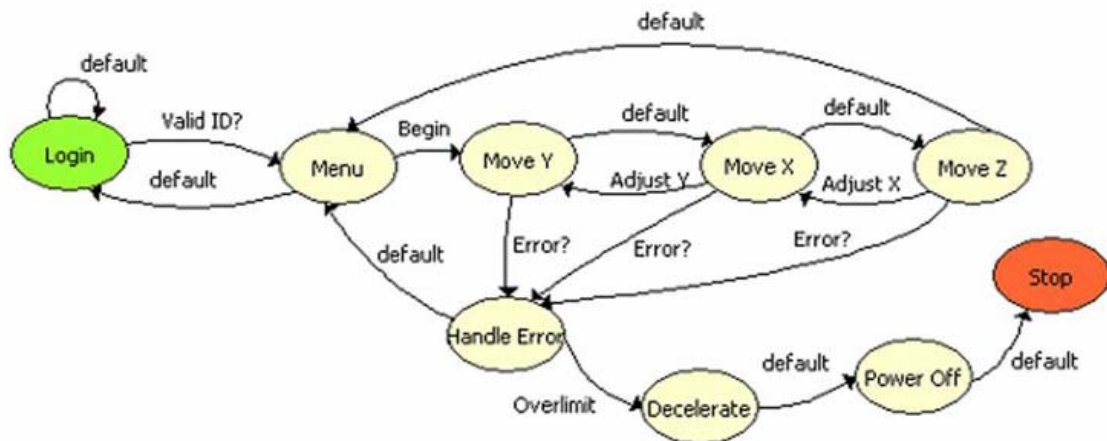


Figure 16. A Finite State Machine

A finite state machine offers a valuable representation of the system. The designer can define exactly what state the system is in depending on various inputs to the system and previous state information.

Consider the definition of a model of computation as containing aesthetic properties (i.e., how it looks) and execution properties (i.e., how it runs) [Garcia].

A model named SCOOP (Simple Concurrent Object-Oriented Programming) and developed by Arslan et al. [2003] offers a comprehensive approach to building high-quality concurrent and distributed systems. The model takes advantage of the inherent concurrency implicit in object-oriented programming to provide programmers with a simple extension enabling them to produce concurrent applications with little more effort than sequential ones.

The basic idea of OO computation shown in Figure 17 [Arslan et al., 2003] offers for performing computation is to apply certain actions, to certain objects and to using certain processors.

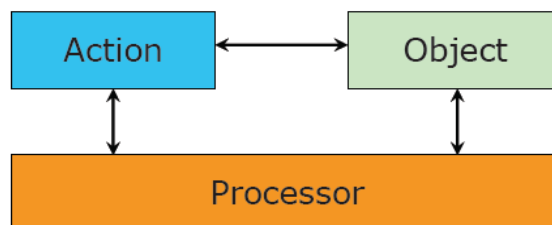


Figure 17. Object-Oriented Computation

The properties of the model are

- Processor is a thread of control supporting **sequential execution** of instructions on one or several objects.
- Processor is an abstract concept
- A processor can be implemented as; process, thread, web service, .NET application domain etc.

- All actions on a given object are executed by its handling processor. No shared access to objects.
- The object is handled by its processor.
- This relationship is fixed, i.e. not considered migration of objects between processors.
- Each processor, together with all objects it owns, can be seen as a sequential subsystem.
- A (concurrent) software system is composed of such subsystems.

4.4 Service-Oriented Architecture

A Service Oriented Architecture (SOA) [MSDN] is an approach that partially overlaps building distributed systems. A service-oriented approach has several characteristics:

- **Loosely coupled.** The application's business logic is separate from the logic of handling the service.
- **Discoverable.** There should be a mechanism for applications to find the service.
- **Contractual.** The interface to the service implements the contract between users and the service.

Although the literature often treats service-oriented approaches as synonymous with web services, they are not necessarily synonyms. Web services present an attractive way to implement service-oriented solutions, but you can use other technologies, such as .NET remoting, to create services [MSDN].

Web services can be viewed as an implementation of a SOA. The SOA promotes the ability to use of Web services from anywhere on the network. SOA is a natural evolution from the world of procedural and object-oriented programming. In order to realize the full benefit of the SOA, we should not use our traditional systems to implement a solution that requires a dynamic, adaptable infrastructure [Dearing].

As shown in Figure 18 [Dearing] [Gardner], any service-oriented architecture contains three roles:

- Service provider
- Service registry
- Service requestor

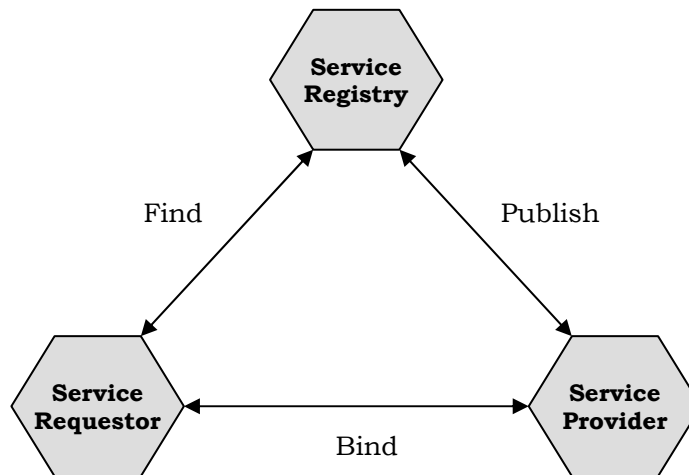


Figure 18. Conceptual Structure of a SOA

a) Service Provider

The service provider is responsible for publishing a description of the service to the service registry. Normally, the service provider hosts the web service.

b) Service Registry

The service registry is a repository that provides the capability of discovering services by the service requestors.

c) Service Requestor

A software application that is responsible for discovering and invoking the service. The service requestor binds to the service obtained from the service registry.

Service Oriented Architecture also includes three operations:

- Publish
- Find
- Bind

Services discover and communicate with each other using the publish, find, bind triad of operations. A service **publishes** its interface definition to the network, a service consumer **finds** the definition and, using the information in the definition, is able to **bind** (resolve the address and send messages), to the service [Harrison and Taylor].

These operations define the contracts between the Service Oriented Architecture roles.

Web Services and SOA

Web services are reusable components with standard interfaces and communication protocols. They are based on a set of standards as shown in Figure 19 [Zhang, 2003].

They are namely,

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)
- Web Services Inspection Language (WSIL)

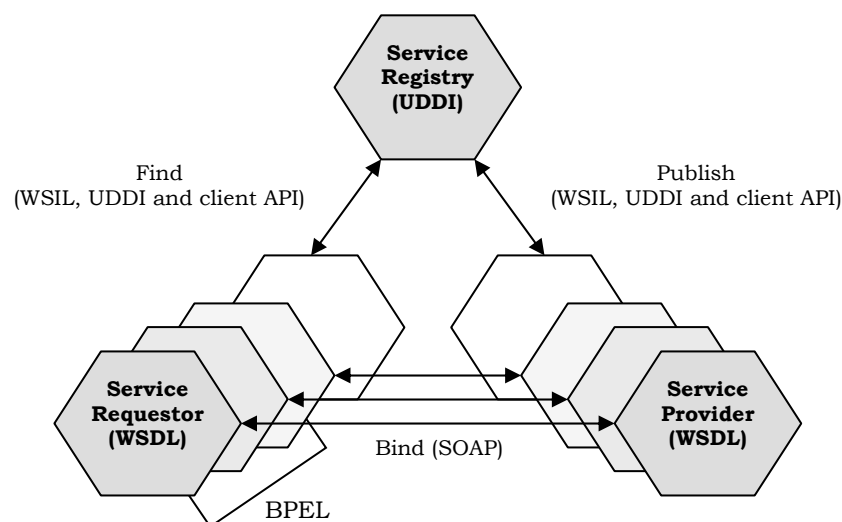


Figure 19. Web Services with Protocols

4.5 The Proposed Approach

In the proposed approach, each feature can be represented with the Business Workflow Model (BWM) to follow business activities, the Business Rule Model (BRM) to classify and manage all its rules, and the Business Computation Model (BCM) to execute the business activities. So, the term “feature” can be simply defined in this form (4.1) as a combination of business process models:

$$\text{Feature} = \text{BWM} + \text{BRM} + \text{BCM} \quad (4.1)$$

Figure 20 shows the orchestration of business processes of features. One or more than one feature combination might be implemented as a web service; each one must be defined and designed in a business process.

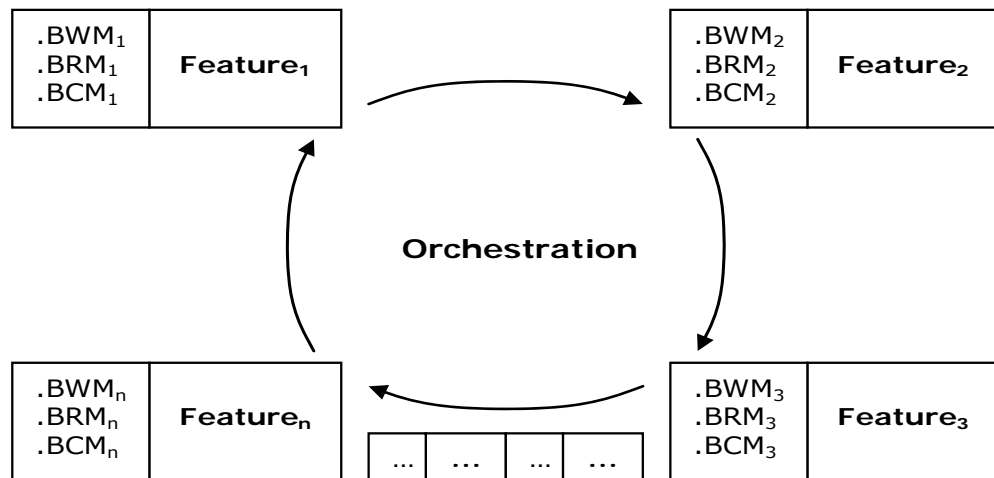


Figure 20. Orchestration of Features in Business Process Modeling

In the following feature term (4.2), each feature may not have workflow models (4.3), and also rule models (4.4). But each one should have at least one computational model (4.5). A feature can be composed with more than one workflow models. For example, to register a person in a system firstly needs to

check its record in the system if it exists or not. Secondly, after verifying the record, desired information of the person can be loaded to the system. So just to check and load the person information needs 2 different workflow models separately. Each model can be represented as BWM_1 , BWM_2 . A Web service may happen from one or more than one workflow model or combination of business models.

Thus, one can express the feature as

$$\text{Feature} = \sum_{i=1}^L BWM_i + \sum_{j=0}^M BRM_j + \sum_{k=1}^N BCM_k \quad (4.2)$$

where

$$BWM = \sum_{i=1}^L OS_i \quad (OS: \text{Orchestration Service}) \rightarrow \text{Web Service} \quad (4.3)$$

$$BRM = \sum_{j=0}^M RBS_j \quad (RBS: \text{Rule-Based Service}) \rightarrow \text{Web Service} \quad (4.4)$$

$$BCM = \sum_{k=1}^N CS_k \quad (CS: \text{Computational Service}) \rightarrow \text{Web Service} \quad (4.5)$$

In summary, we can simply express the feature in proposed FOD model as:

$$\text{Feature} = BWM + BRM + BCM$$

In the next section, the proposed new feature term will be explained on service-oriented architecture with a case study that is a real life commercial application. This application has already been implemented by using the Rational Unified Process (RUP) and object-oriented architecture design (OOAD) by UML.

CHAPTER 5

THE IMPLEMENTATION WITH OBJECT-ORIENTED DEVELOPMENT

The case study given below has already been implemented using .NET Framework with Rational Unified Process (RUP) and UML in the company that author works.

In the analysis and design stages of the system RUP methodology had been used. In addition to use cases and object modeling, web service technology to communicate different applications servers was also used.

5.1 A Case Study: BioNET

BioNET is a security controlling, product and shopping management system using fingerprint validation system.



As noted above this application has been developed using .NET Framework with Rational Unified Process and UML. It consists of seven modules as System Management, Transition Management, Product Management, Credit Management, Shopping Management, Policy Management and KIOSK.

Nowadays, automatic fingerprint matching is becoming increasingly popular in systems that control access to physical locations (such as doors and entry gates), which register employee attendance time in enterprises.

Human fingerprints are unique to each person and can be regarded as a sort of signature, certifying the person's identity.

Alternative access control systems (such as Password entrance, ID Cards and

TOM-Touch on Memory) have some disadvantages. These systems are vulnerable to misuses when the password data or ID cards are shared between users. And also physical devices like Cards and TOM are easy to lose and the reproduction of these items are costly. Because of disadvantages of other access control systems, fingerprint identification is adopted as a better solution for access control.

BioNET Fingerprint System is developed by YUCE Information Systems and it is a fingerprint identification system. The basic property of BioNET Fingerprint System is that it is a centralized system meaning that administration and security policy of the system is easy to control and the system can store unlimited fingerprint samples.

5.2 Implementation with OOAD Approach

5.2.1 Project Schedule

In Figure 21, the project schedule has been defined after designed uses cases and activity diagrams of the system using Microsoft Project. The time period for whole the project is 3 months with 4 persons.

5.2.2 Object-Oriented (OO) Analysis and Design

The OO process begins with definition of a Use Case diagram and a Concept Model. The Use Case Diagram shows external actors (users and systems) and their interaction with Use Cases. The concept model is an initial class model based on domain concepts, their attributes and associations [White, 2004].

Each Use Case is further detailed as required. A Use Case is a scenario that identifies the interaction between an actor (human or external system) and the system under consideration. The scenario includes normal flow of events and system actions, followed by alternate events that may take place, and the resulting system actions. Use Cases can be represented in System Sequence Diagrams (SSDs). An SSD is a trace of messages from the actor to the system. Each message represents a high level operation that the system shall perform. Input and Output parameters are included on the SSD for each system operation. Each System Operation is further defined in a Contract. The Contract pre-conditions and post-

conditions specify the system state prior to Operation execution and the system state after Operation execution [White, 2004].

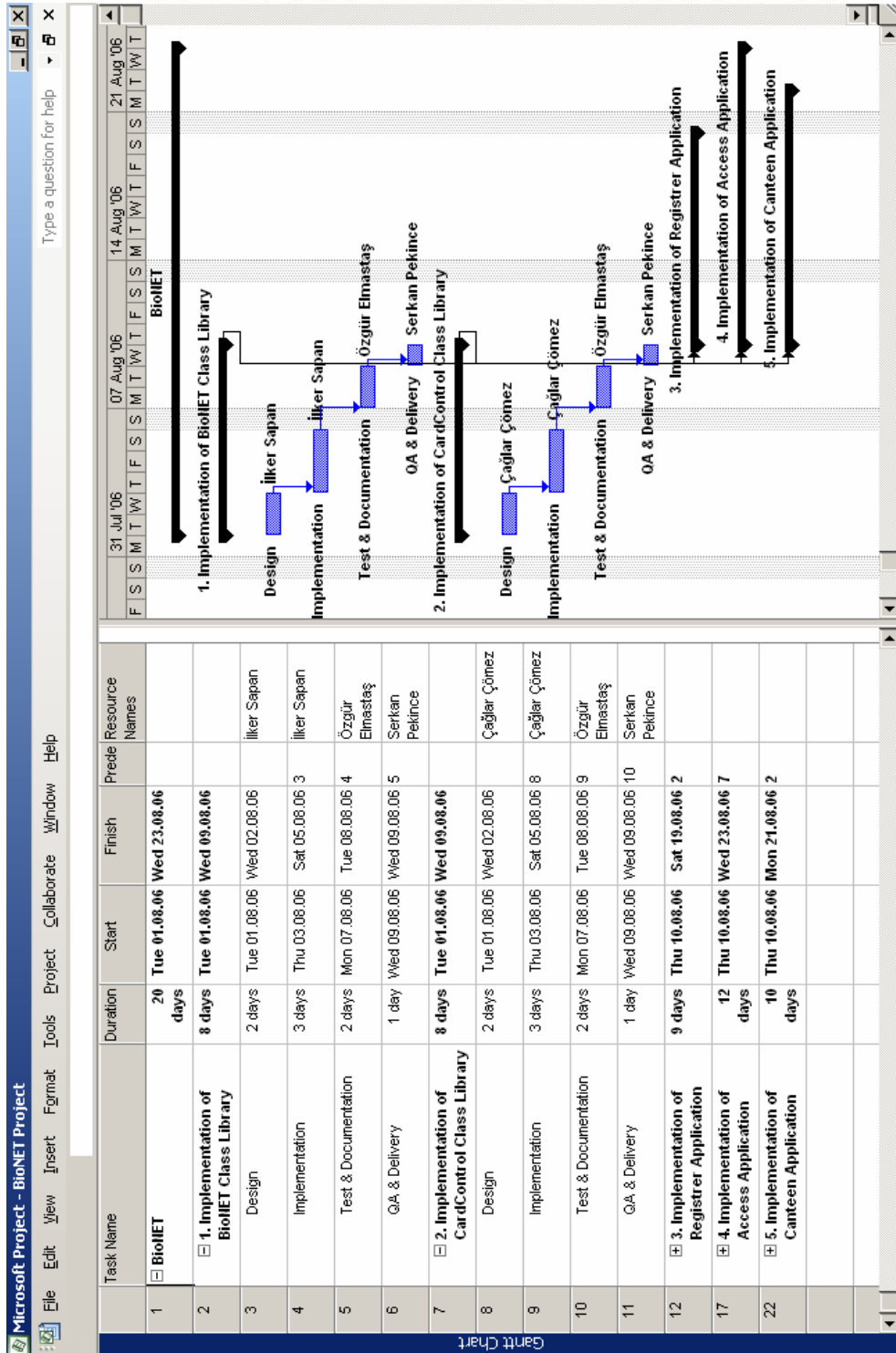


Figure 21. Project Schedule of BioNET

5.2.3 Object-Oriented Model

Use Case Diagrams define context & high-level system functions. Use Cases can be formalized using extended abstract machines (state transition diagrams with guards) or System Sequence Diagram traces. Class Diagrams define class attributes, methods, association, and visibility. Package Diagrams define system composition. Interaction Diagrams define collaboration among cooperating objects. Each object can also be defined as an extended abstract machine. Deployment Diagrams specify allocation of components to nodes [White, 2004].

5.2.4 Use Case Diagram

In Figure 22, the Use Case Diagram of BioNET Shopping Module is shown. Two actors are participated to the system as employee and person. In the sale processing use case Add Item is always needed, but Remove Item depends on if the total of the sale exceeds the daily or total credit limits.

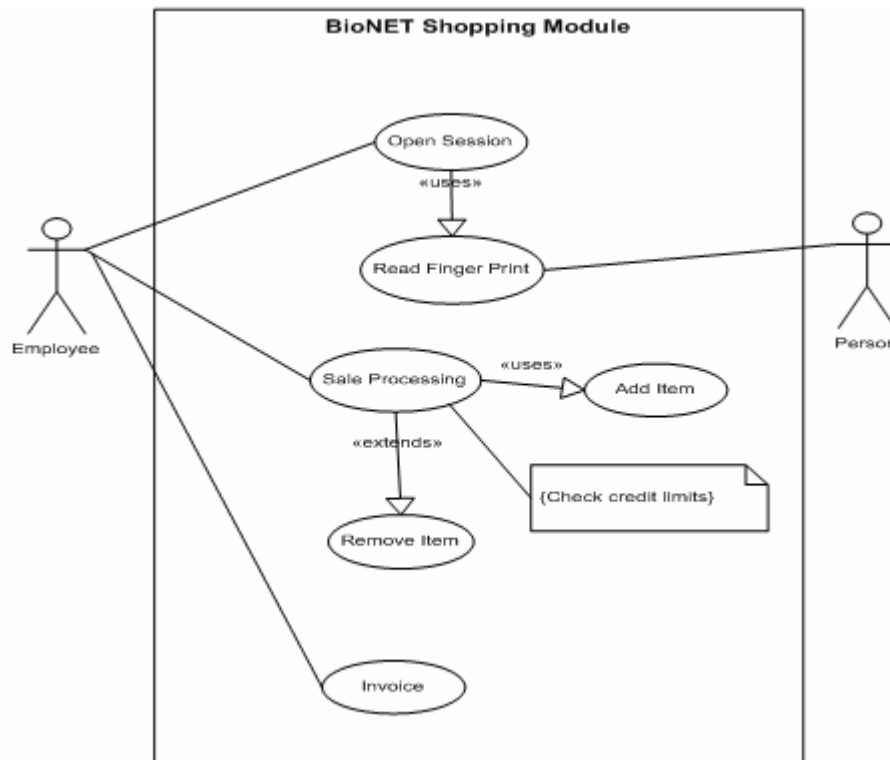


Figure 22. Use Case Diagram of Shopping Module

5.2.5 Activity Diagram

The Activity Diagram of BioNET Shopping Module is shown in Figure 23. As well as verifying the fingerprint, controlling the restrictions and updating person's and product account are represented in detail. If the added new item is defined as restricted for the person, the item is removed and also the total of the sale exceeds any limit either daily or credit limit, an item is removed from the sale. Otherwise, the sale is not realized.

5.2.6 Class Diagram

The class diagram of BioNET Shopping Module is shown in Figure 24. The classes of Application, UserLogin, UserSearch and Shopping are represented.

BioNET desktop application components are shown in Appendix A, B, C, D, and E.

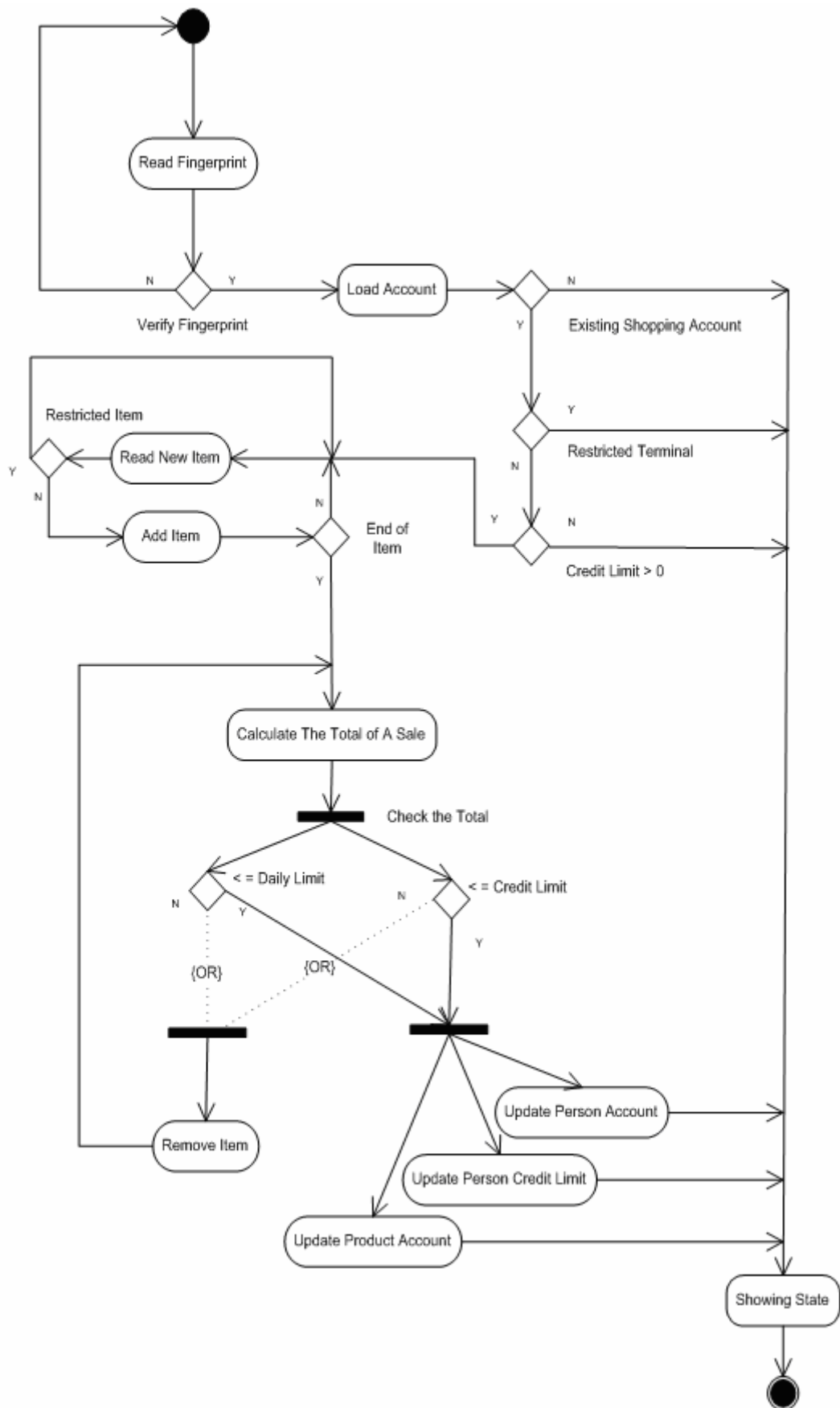


Figure 23. Activity Diagram of Shopping Module

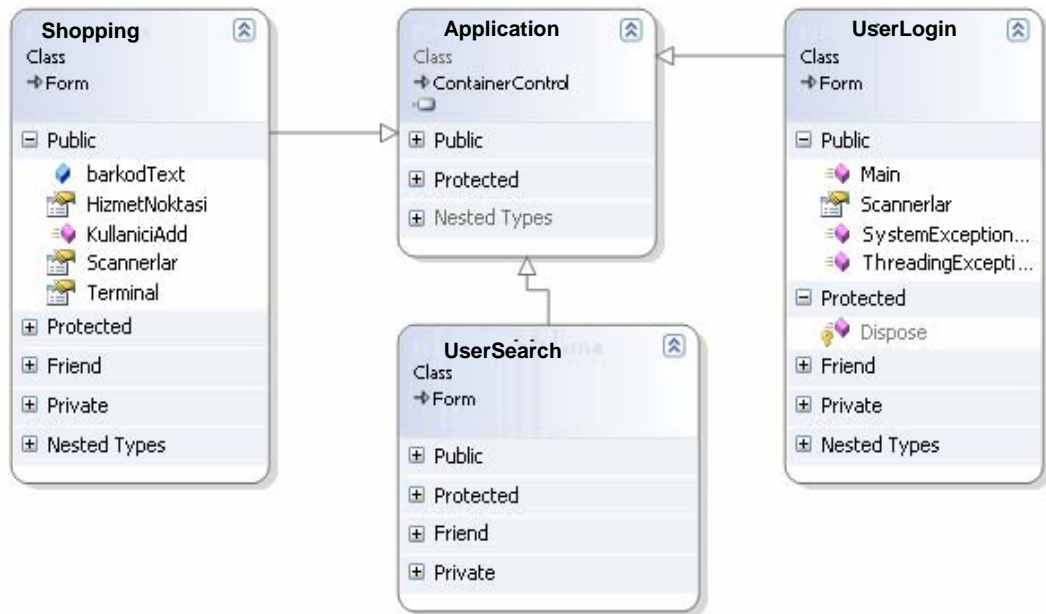


Figure 24. Class Diagram of Shopping Module

CHAPTER 6

THE IMPLEMENTATION WITH FEATURE-ORIENTED DEVELOPMENT

In this chapter, the proposed approach will be applied on the case study as noted in the previous chapter; the proposed approach is a Feature-Oriented Development using Service-Oriented Architecture. The proposed approach will deal with the Shopping Management Module of BioNET.

6.1 Implementation with FOD using SOA

The case study, Shopping Management Module of BioNET, has been analyzed, designed and implemented with Feature-Oriented Development using Service-Oriented Architecture.

Feature-Oriented Development methodology is used for the general design and planning and analysis of the system process. Eventually, FOD suggests us a software management system, but does not offer construction architecture. For implementation of the system designed by FOD, SOA and FODA, Microsoft BizTalk Server 2006 tools and techniques have been used.

The created BioNET Web Services (in Figure 25) provide the relation between clients and database server. Each client makes request to web service. Web service transmits the request to database server and finally web service response to clients. BioNET UpdateService also provides the software updates to clients and databases.

6.2 Implementation Tools

The business workflows of the system have been created using BizTalk Server 2006 environment.

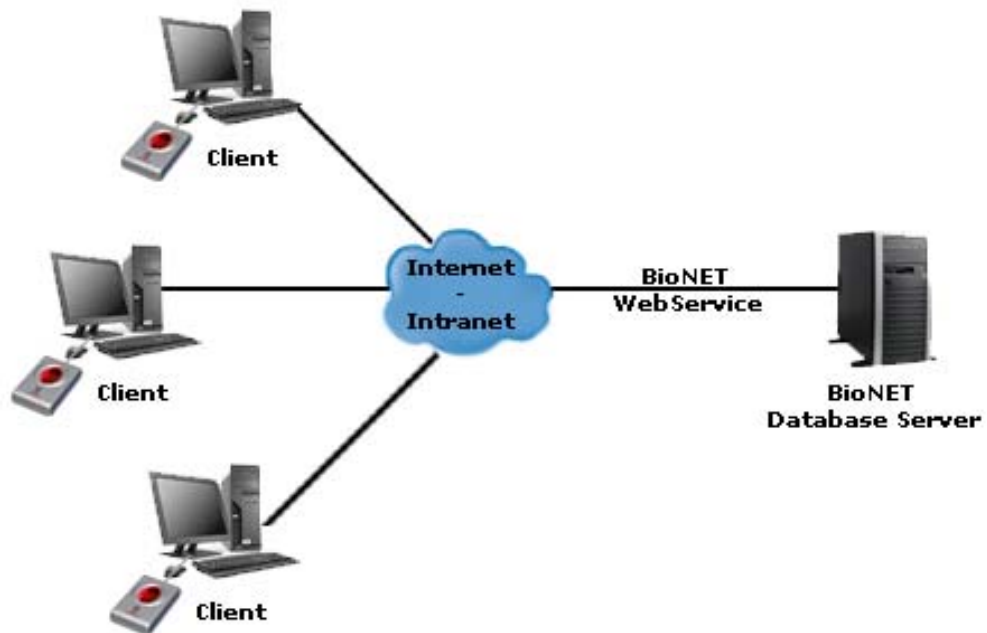


Figure 25. Communication via Web Services

6.2.1 Microsoft BizTalk Server 2006

As organizations move toward a service-oriented world, BizTalk Server 2006 [Chappel, 2005] supports creating effective business processes that unite separate systems into a coherent whole. Like its predecessors, this latest release allows connecting diverse software, then graphically creating and modifying process logic that uses that software. The product also lets information workers monitor running processes, interact with trading partners, and perform other business-oriented tasks.

BizTalk Server 2006 is built on the foundation of its predecessors, BizTalk Server 2000, 2002 and 2004. The most important new additions in BizTalk Server 2006 are [Chappel, 2005]:

- Better support for deploying, monitoring, and managing applications.
- Significantly simpler installation.
- Improved capabilities for Business Activity Monitoring (BAM).

BizTalk Server 2006 is built on version 2.0 of the .NET Framework, and its developer tools are hosted in Visual Studio 2005. For storage, the product can use SQL Server 2005, the latest version of Microsoft's flagship database product, or SQL Server 2000, the previous release. BizTalk Server 2006 can also run on 64-bit Windows, taking advantage of the larger memory and other benefits this new generation of hardware offers [Chappel, 2005].

6.2.2 What BizTalk Server 2006 Provides

Combining different systems into effective business processes is a challenging problem. Accordingly, BizTalk Server 2006 includes a range of technologies. Figure 26 illustrates the product's major components [Chappel, 2005].

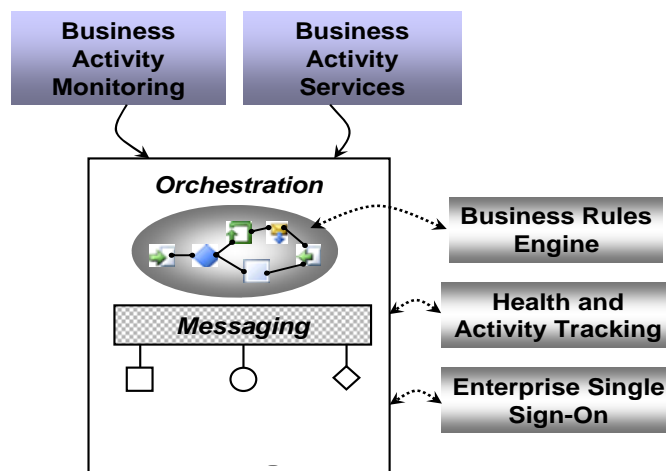


Figure 26. BizTalk Server 2006 Engine and Its Interactions

As the Figure 26 suggests, the heart of the product is the *BizTalk Server 2006 Engine*. The engine has two main parts:

- A **messaging** component that provides the ability to communicate with a range of other software. By relying on pluggable adapters for different kinds of communication, the engine can support a variety of protocols and data formats, including Web services and many others.
- Support for creating and running graphically defined processes called **orchestrations**. Built on top of the engine's messaging components, orchestrations implement the logic that drives all or part of a business process.

Several other technologies can also be used in concert with the engine, which is in interactions, including:

- A *Business Rules Engine* that allows evaluating complex sets of rules.
- A *Health and Activity Tracking* tool that lets developers and administrators monitor and manage the engine and the orchestrations it runs.
- An *Enterprise Single Sign-on* facility, providing the ability to map authentication information between Windows and non-Windows systems.
- On top of this foundation, BizTalk Server 2006 provides a group of technologies that address the more business-oriented needs of information workers such as Business Activity Monitoring and Business Activity Services.

6.3 Analysis Using FODA

6.3.1 Context Diagram

The highest-level graphic representation is a context diagram of the system, as shown in Figure 27.

The external entities such as Person's Account Registration, Person's Account Verification, Terminal and Product Constraints, Sale and Invoice Processing are represented according to request and response type.

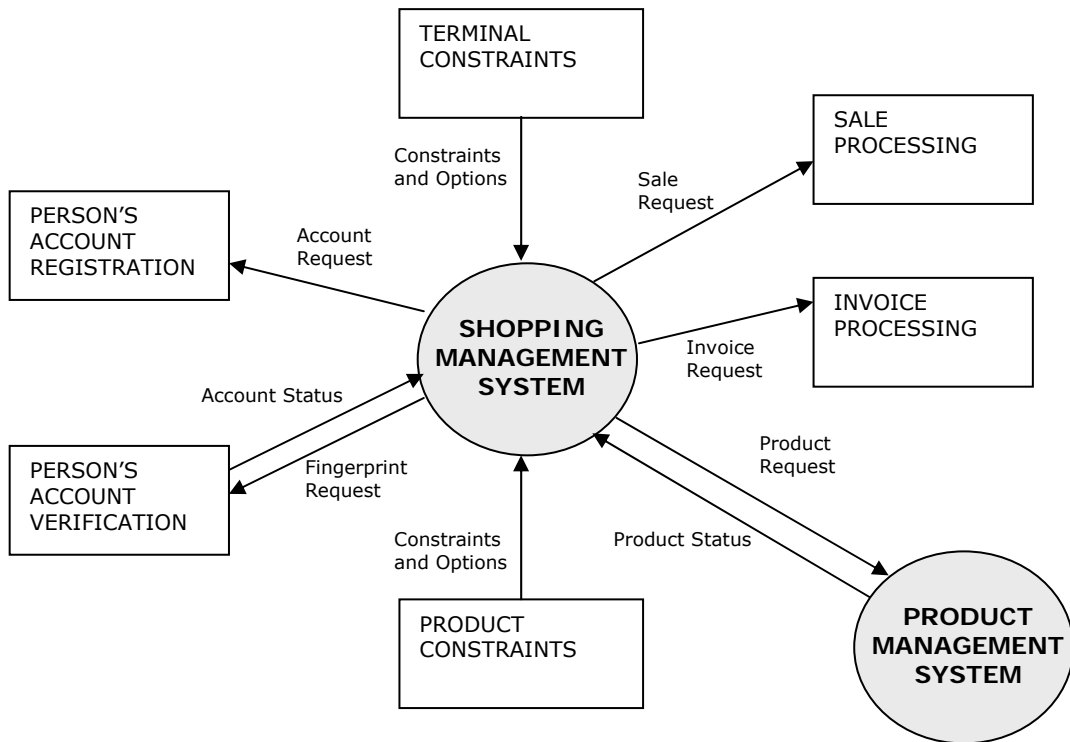


Figure 27. Context Diagram of Shopping Module

Shopping Management System starts with reading person's fingerprint from the fingerprint scanner and verifying the record on the database according to person's fingerprint images. External entity Person's Account Verification returns account's status as true or false. According to status system either load account information of the person to the session or direct to Person's Account Registration to create a new account. If the account verified, system opens a new session and load account information with its constraints, which are restricted terminals or products and daily-account credit limit. Shopping Management System is in collaboration with Product Management System to control product accounts. If added items to a sale processing are met the conditions, system starts sale and invoice processing.

6.3.2 Feature Model

All the components of the system decomposed as features are shown in FODA feature model in Figure 28, using the notation already given in Figure 3.

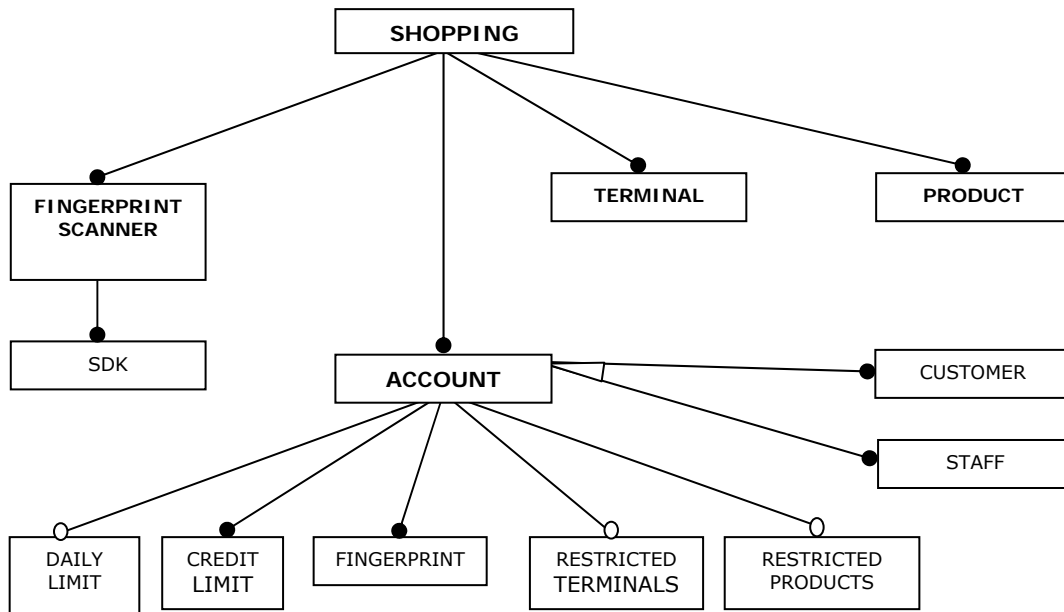


Figure 28. The Feature Model of BioNET Shopping Module

Fingerprint Scanner with its Software Development Kit's (SDK) of the scanners, Account, and Terminal and Product are mandatory features of the system. Meanwhile, sub features of an Account are Credit Limit and Fingerprint. Daily Limit, Restricted Terminals and Restricted Products are optional features. In the system, staff may not be a customer.

6.4 FDD Practices

This section follows the FDD practices from Domain Service Modeling to Build Feature step on only one feature of Shopping Management Module, which is Verify Fingerprint feature. The business workflow model of **VerifyFingerPrint** is shown in Figure 29.

6.4.1 Domain Service Modeling

Domain Service Modeling has been adapted from Domain Object Modeling for the case study. The domain object model provides an overall framework to which to add function, feature by feature. It helps to maintain the conceptual integrity of

the system. Using it to guide them, feature teams produce better initial designs for each group of features [Palmer and Felsing, 2002].

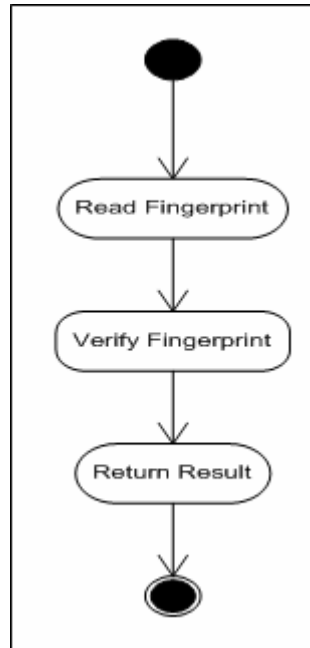


Figure 29. Business Workflow Model of VerifyFingerPrint

Domain Object Modeling is a form of object decomposition [Palmer and Felsing, 2002]. The business services with their functional requirements are decomposed without classes and methods in the modeling representation as in Figure 30. The problem is broken down into the significant web objects involved. The design and implementation of each object or inside class identified in the model is smaller problem to solve. When the completed classes are combined, they form the solution to the larger problem [Palmer and Felsing, 2002].

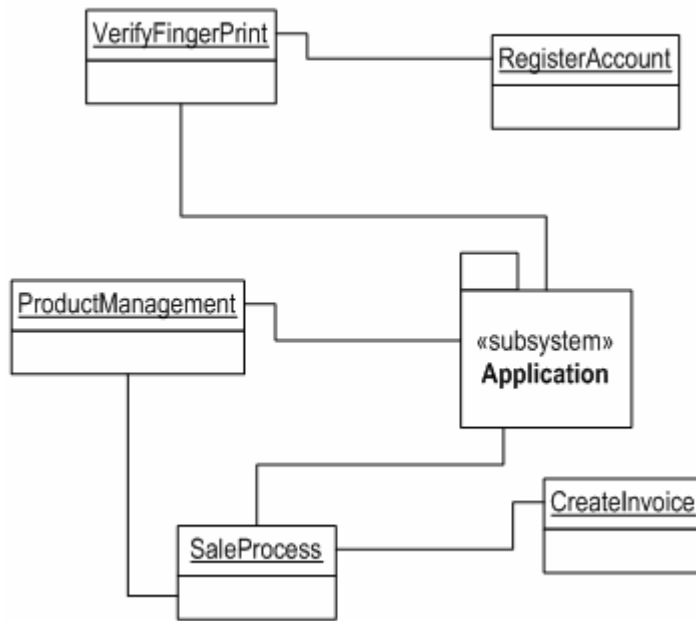


Figure 30. Domain Service Model

6.4.2 Build a Feature List

Now considering the seven modules of the BioNET given in Section 5.1 they are listed as follows.

a) Feature Group

1. SYSTEM MANAGEMENT
2. TRANSITION MANAGEMENT
3. PRODUCT MANAGEMENT
4. CREDIT MANAGEMENT,
- 5. SHOPPING MANAGEMENT**
6. POLICY MANAGEMENT
7. KIOSK

The following Feature Sets are composed according to V-Plant model [Anderson, 2004] which is given in subsection 3.2.2.1.

b) Feature Sets of the Shopping Management Feature Group

- 5.1. Reading a Fingerprint
- 5.2. Validating an Account
- 5.3. Registering an Account
- 5.4. Opening a Session
- 5.5. Processing a Sale
- 5.6. Creating an Invoice
- 5.7. Performing a Shopping

Features Database Table of Shopping Management Feature Group is shown in Table 3. In the table, for each feature set's feature specifications are defined in the feature format at **<action><result><object>** with its prerequisite feature ID.

Table 3. Features Database Table

Feature ID	Feature Group	Feature Set	Feature Specification	Prerequisite Feature (set)
5.	Shopping Management			
5.1		Reading a Fingerprint		
5.1.1			Read the fingerprint by fingerprint scanner	-
5.1.2			Verify the fingerprint for the person	5.1.1
5.2		Validating an Account		
5.2.1			Validate the account for the person	5.1.2
5.3		Registering an Account		
5.3.1			Add a new record for the person	5.2.1
5.4		Opening a Session		
5.4.1			Load the account information for the session owner	5.2.1
5.4.2			Load the shopping information for the session owner	5.2.1
5.4.3			Load the allowed items for the session owner	5.2.1

Table 3. Features Database Table (continued)

Feature ID	Feature Group	Feature Set	Feature Specification	Prerequisite Feature (set)
5.4.4			Load the allowed terminals for the session owner	5.2.1
5.5		Processing a Sale		
5.5.1			Create a space for a sale	5.4
5.5.2			Read an item for sale	5.5.1
5.5.3			Add an item to sale	5.5.1
5.5.4			Calculate the total of the sale	5.5.1
5.5.5			Validate the total sale by the account	5.5.1
5.6		Creating an Invoice		
5.6.1			Create an invoice for the sale	5.5.1
5.7		Performing a Shopping		
5.7.1			Update the product account for the sale	5.6.1
5.7.2			Update the person account for the sale	5.6.1

6.4.3 Plan by Feature

In this step, all the feature sets have been sequenced, depending on priority and dependencies. Also, the Web services – its classes- identified in the modeling activity have been assigned to individual developers; the owner of a web service or class is responsible for its development.

6.4.4 Design and Build by Feature

A number of features have been scheduled for development by assigning to programmer. The defined and assigned features have been implemented on Microsoft BizTalk Server 2006 environment to create schema and orchestration of the processes of a feature and to generate business rules respectively.

Figure 31 shows an orchestration to receive fingerprint image and validate the image id and then to return a result. In this orchestration, evaluating data has been done via messages. Each orchestration should have input and output ports to receive and send data. In order to receive and send data from a source to a destination for each functional operational in feature schemas (XML files) has been used. As an example of VerifyFingerPrint schema file is shown in Figure 32.

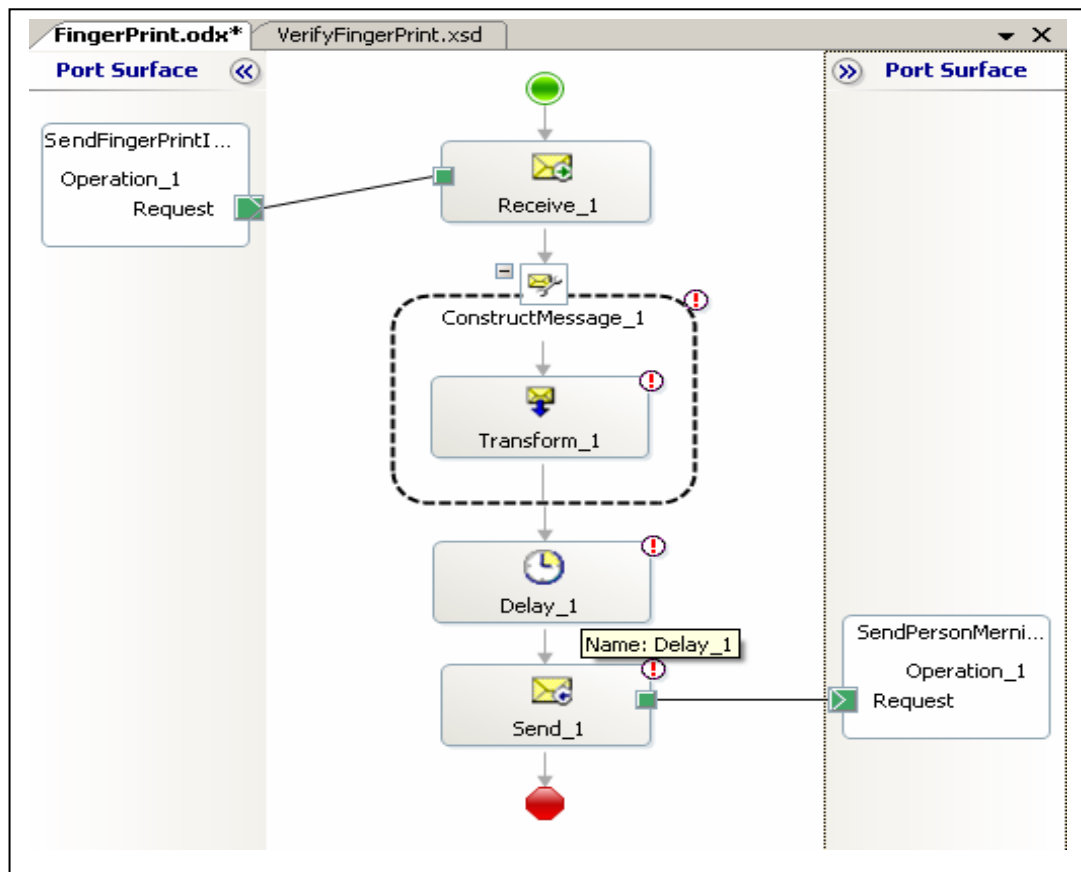


Figure 31. Business Orchestration in BizTalk Server 2006

Finally, Business Rule Composer may be used to match sending and storing data. It needs to create a schema, which is defined at the previous step. However, some feature may not need to create business rules; they might be just functional business activities. As an example of VerifyFingerPrint rule sets is shown in Figure 33.

```

<?xml version="1.0" encoding="utf-16" ?>
- <xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003"
xmlns="http://FingerPrint.VerifyFingerPrint" targetNamespace="http://FingerPrint.VerifyFingerPrint"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
- <xs:element name="PersonRecord">
- <xs:complexType>
- <xs:sequence>
<xs:element name="MemisID" type="xs:unsignedInt" />
<xs:element name="FingerPrintImageID" type="xs:unsignedByte" />
<xs:element name="Result" type="xs:boolean" />
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>

```

Figure 32. A Sample XML Schema File

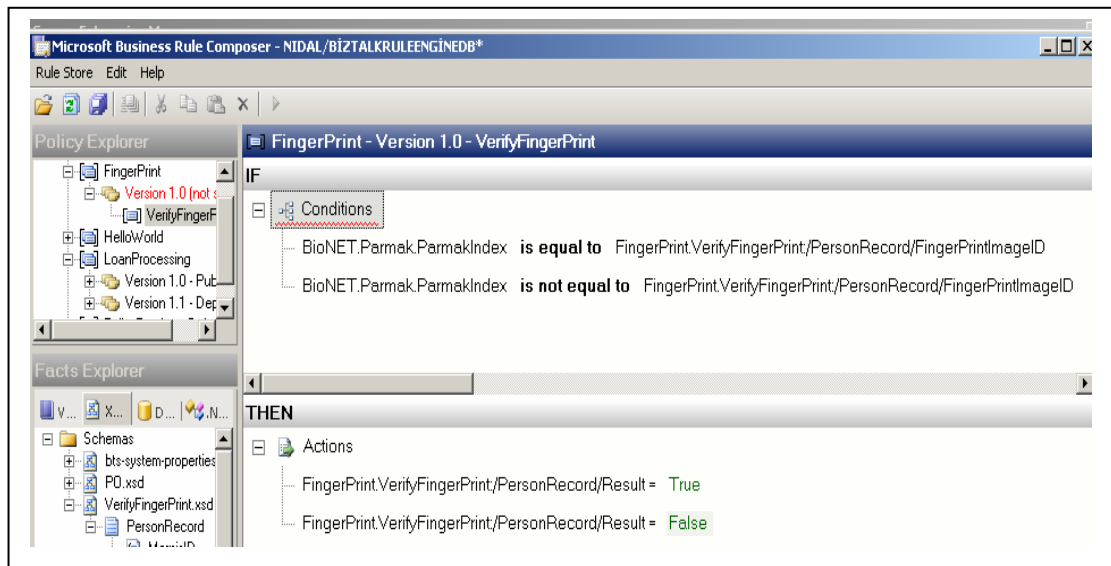


Figure 33. A Sample Business Rule Composition

The created sample BioNET orchestrations, XML schemas, rule compositions, web services with their SOAP messages and Visual Basic.NET code are shown in Appendix F, G and H.

CHAPTER 7

COMPARATIVE ANALYSIS

The case problem, BioNET, has been implemented according to Object-Oriented Development and Feature-Oriented Development in Chapters 5 and 6 respectively. In this chapter, results of the proposed approach and object-oriented approach will be compared.

7.1 Fundamentals

Every popular method or process contains some form of functional decomposition activity that breaks down this high-level statement into more manageable problems. Functional specification documents, use case models and use case descriptions, and user stories and features all represent functional requirements, and each representation has its own advantages and disadvantages [Palmer].

An important feature of Object-Oriented Development (OOD) is that software objects represent the real-world objects, which are derived from classes, and a class hierarchy allows objects to inherit characteristics from parent classes. This allows software object reuse, less coding, encapsulation of functionality, and many other advantages. A major problem that arose with OO programming is that if the class hierarchy is not properly designed, almost all of the OO advantages might disappear. The OO model attempts to properly define and document the class hierarchy from which all the system objects are created and object interactions are defined [Palmer and Felsing, 2002].

Commonly, a statement of purpose has been taken and broken it down into a number of smaller problems and defined a set of subsystems (or modules) to solve those smaller problems using Feature Oriented Development. Then, each

subsystem has been broken its problem into a hierarchical list of functional requirements. When getting the requirements granular enough that decomposition has been stopped and started to implementation respectively [Palmer and Felsing, 2002].

In Feature-Oriented Development, the software features represented in feature-set models and every feature-set is divided into “features’ in development process. [Coad, 2003] compares some software development phases, which are define, design, build and test, among UP, FDD and XP in Table 4 [Coad, 2003].

Table 4. The Comparison of UP, FDD and XP

	UP	FDD	XP
Define	Uses cases and class diagrams	Feature list and class diagrams (sub teams then teams)	User stories
Design	Sequence diagram	Sequence diagram	Refactor
Build	Code	Feature teams (chief programmers and class owners)	Pair programming (team ownership)
Test	Code, test, inspect	Code, test, inspect	Write test, code, test-and continuously inspect

Accordingly, Table 5 shows the comparison of “FOD using FDD” with “OOD using RUP”, which is basically adapted from the Table 4. Both the common phases of software development life cycles (analysis, design, build, and test) and external properties (activities and roles) have been compared for FOD and OOD. The case study comparison of FOD with OOD, shown in Table 5, is based on software development activities, which are software architecture, overall model, process of development and implementation.

The case study comparison of BioNET with FOD and BioNET with OOD, shown in Table 6, is based on issues, which are Requirements Analysis and System Behavior Issues, Architectural Issues, Analysis Issues, Design Issues, Implementation Issues, Testing Issues, Maintenance Issues, Administrative Issues, and Tool Support. The following subsections will discuss the items of Table 5 in detail.

Table 5. The Comparison of OOD using RUP with FOD using FDD

	OOD using RUP	FOD using FDD
Activities	Inception, Elaboration, Construction, Transition	Develop an Overall Model, Build a Feature List, Plan by Feature, Design by Feature, Build by Feature
Roles	Requirements Analyst, Designer, Project Manager, Software Quality Assurance (SQA) Analyst, Domain Analyst, Tester, Project Librarian, Other Roles	Project Manager, Chief Architect, Chief Programmers, Class Owners, Domain Manager, Release Manager, Language Guru, Build Engineer, Toolsmith, System Administrator, Tester, Deployer, Technical Writer, Sub Feature Teams, Feature Teams
Based on	Object Orientation	Service Orientation
Define	Software Requirements Specification (SRS), Uses Cases and Class Diagrams	Software Requirements Specification (SRS), Feature List and Service Hierarchy Diagrams (Sub teams and teams)
Analysis	UML Activity Diagrams, Object Diagrams	Context Diagrams, Feature-Oriented Domain Analysis (FODA) Feature Model, Domain Service Modeling
Design	Sequence Diagram	Sequence Diagram, Business Workflow Model, Business Rule Model, Business Computation Model, Business Orchestration Model
Build	Code	Code, Deploy and Publish Web Services
Test	Code, Test, Inspect	Model, Code, Test, Inspect

Table 6. The Comparison of BioNET with FOD and BioNET with OOD

Issues	BioNET with OOD	BioNET with FOD
Requirements Analysis and System Behavior Issues	<ul style="list-style-type: none"> • System Requirements • Functional Requirements • Based on system requirements captured as objects and use cases 	<ul style="list-style-type: none"> • System Requirements • Functional Requirements • Based on system requirements captured as features
Architectural Issues	<ul style="list-style-type: none"> • Components (Objects) • Connectors (Object Method Calls) • Configurations (Tightly Coupled) • Client-server architecture • Based on functional, behavioral, and structural modeling of the problem domain and the system 	<ul style="list-style-type: none"> • Components (Services) • Connectors (Service Calls) • Configurations (Loosely Coupled) • Service-Oriented architecture • Based on structural and behavioral modeling of the problem domain
Analysis Issues	<ul style="list-style-type: none"> • Define activity diagram • Identify domain objects • Express relationships among objects • Describe classes with their relationships 	<ul style="list-style-type: none"> • Design context diagram • Identify services • Define feature model • Define the relationship among features
Design Issues	<ul style="list-style-type: none"> • Define software objects • Define relationships among objects • Design class diagrams • Inheritance and Overloading • User Interface Design 	<ul style="list-style-type: none"> • Design feature model • Design domain service modeling • Build a feature list • Design business modeling of feature(s)(sets)-workflow, rule and computational model
Implementation Issues	<ul style="list-style-type: none"> • Object-oriented implementation • Coding • Object interactions 	<ul style="list-style-type: none"> • Service-oriented implementation • Build by feature list • Create orchestration, rule modeling for web services by feature list • Web services deploying and publishing

Table 6. The Comparison of BioNET with FOD and BioNET with OOD (continued)

Issues	BioNET with OOD	BioNET with FOD
Testing Issues	<ul style="list-style-type: none"> • Error handling • Defect testing • Functional testing • Integration testing (tightly coupled) 	<ul style="list-style-type: none"> • Error handling • Defect testing • Functional testing • Model testing • Integration testing (loosely coupled)
Maintenance Issues	<ul style="list-style-type: none"> • Tightly coupled refactoring • Tracing • Debugging • Monitoring (by developer) 	<ul style="list-style-type: none"> • Loosely coupled refactoring • Tracing • Debugging • Monitoring (by environment)
Administrative Issues	<ul style="list-style-type: none"> • Unified Modeling Language (UML) • Rational Unified Process (RUP) 	<ul style="list-style-type: none"> • Domain Engineering • Feature-Driven Development (FDD) • Service-Oriented Architecture (SOA)
Tool Support	<ul style="list-style-type: none"> • .NET Framework 2.0, MSSQL Server 2000 	<ul style="list-style-type: none"> • .NET Framework 2.0, MSSQL Server 2000, BizTalk Server 2006

7.2 Requirements Analysis and System Behavior Issues

The system and functional requirements of BioNET are fingerprint scanner with its Software Development Kit (SDK), modules, clients, and server and their behaviors with actors in the system. For both development methodologies, Software Requirements Specification (SRS) has been used to specify requirements and behavioral issues. For these issues, use case diagrams have been used in OOD; features have been used in FOD to capture software components and requirements. The difficulties of use case diagrams were in design and implementation phases caused to dive into an inappropriate level of detail. The use of features provided the facility for user and developer ensuring to understand that requirements in a business value.

7.3 Architectural Issues

First application of BioNET has been implemented using object-oriented methodology as a client-server application. So, the components and connectors have been planned with objects. The communication between clients and server has been realized via local area network. The basic building blocks in OOD implementation are the objects and they have been interconnected with object method calls, which created a very tightly coupled interaction model, which complicated the maintenance issues.

BioNET in FOD was implemented as a Web-based application in Service-Oriented Architecture. The basic building blocks were Web Services, rules and workflows. They are defined and deployed onto BizTalk Server 2006, which provided a loosely coupled environment for and facilitated the further maintenance with a great deal of ease and effectiveness.

7.4 Analysis Issues

System analysis of BioNET with OOD has been detailed with activity diagrams derived from the previous uses case diagrams. Domain objects and their relationships are revealed for each module, but they couldn't lead to a thorough domain object model. During the increments of systems analysis, majority of the diagrams should have been revisited and modified accordingly. Such dynamism through the increments makes the use of Case tools a must for OOD. Otherwise, it would be quite complicated to manage the complete model with complex interdependencies of the object model.

For the FOD case, Feature-Oriented Domain Analysis techniques have been used. The highest-level graphic representation was drawn as a context diagram that used to obtain the business domain. All components of the system decomposed as features (mandatory, optional, alternative etc.) in a feature model. The feature model of BioNET provided the capturing of commonalities and differences between features as reusable domain and application artifacts. For example read fingerprint and validate account are stressed in the feature model for this purpose.

7.5 Design Issues

Major differences between the design of BioNET with FOD and OOD are the focused items. In FOD approach, the system is designed with “features” to yield the “domain service” model, and a feature list is built to feed the implementation phase. Main strategy is to define the elements, which are subsystems, services, etc., through which domain features are allocated to achieve them. The feature lists have been built using FDD practices facilitated to decompose the application. In the proposed approach, this is as a combination of business workflows, business services and business rules to create a business process model for a feature.

In OOD approach, the software objects, class diagrams, and relationships such as inheritance and aggregation have been mainly used to design the system. Commonly, this approach provided many advantages in BioNET desktop application such as code reuse and fast implementation, but it has brought some disadvantages such as lack of changeability and lack of testability in isolation, since such monolithic nature of object models does usually end up with very tightly coupled designs.

7.6 Implementation Issues

The implementation of BioNET with OOD has been realized by using object-oriented programming in 3-tiers, which are presentation, business and data tiers. Each of these tiers has been accompanied by a set of components comprising dedicated class hierarchies, and they have been linked at the middle tier in .NET 2.0 Component Framework. Implementing an object-oriented model in N-tier architecture complicates the coding efforts since class hierarchies should be aligned accordingly to manage the transactions. Another drawback was to manage the crosscutting issues such as security and logging, as identified by Aspect-Oriented paradigm.

The implementation of BioNET with FOD has been realized as business process modeling using Microsoft BizTalk Server 2006 environment. In the defined business process of features, each feature or feature set(s) has been designed in service orchestration model, which expects the manipulation of data via messages.

Each orchestration should have input and output ports to receive and send data. In order to receive and send data from a source to a destination for each functional operation, dedicated feature schemas (XML files) have been used. In Business Rule Composer tool of BizTalk Server 2006, the created schema or storage data in database tables may establish connection as individually or mutual for defined conditions. For each defined feature in orchestration or rule composer has been deployed and published as Web services. This technique provides implementation without single line coding thought the declarative and generative nature of BizTalk Server 2006.

The FOD implementation has ended up with a loosely coupled design model in which every building block (business rule, business service and business workflow) has been managed on its own and composed through the orchestration capability of the BizTalk Server 2006. This directed and facilitated the implementation roadmap without any hassles. Another advantage of this approach is to provide the reuse and sharing model. For example, a created rule model of a feature or feature set(s) might be used for other feature or feature set(s). However, the first insight of the BPM approach required the understanding of roadmap in advance. Consequently, a certain time and dedication are needed upfront for the proposed approach.

7.7 Testing Issues

Error handling and defect recovery have been applied for both methodologies. Integration testing has been applied to check the complete modules of BioNET and their interactions. Functional testing has been attempted to determine whether a component's implementation provides the behavior described in its specification, which is SRS and defined use case diagrams or feature set.

Model testing has been applied for implementation with FOD distinctly. Using model testing with Web services for “workflow model”, “rule model”, “business process” has been applied in cases where the models are used and attached with a kind of executability.

It has been observed that loosely coupled designs of FOD has yielded more facilitated environments for unit and integration testing when compared to more monolithic and tightly coupled designs of OOD. Moreover, model testing was

available in FOD by default but OOD needs extra effort to put this on the table unless a pure Model-Driven Development approach has been anticipated.

7.8 Maintenance Issues

The maintenance issues of continuously checking against desired functionality, tracing, debugging, monitoring have been applied for both of the development methodologies for the measurements of performance, serviceability, usability, and reliability.

Debugging and monitoring in FOD method follow easily rather than OOD method during the implementation of BioNET. Because of each feature can be implemented in a single application as Web service, FOD provided a better environment for separation and composition of concerns when compared to the monolithic building blocks of objects. Managing crosscutting issues is another fact to consider. Pragmatically, OOD should be accompanied with separate techniques like Aspect-Oriented Programming, but since the proposed FOD approach has been implemented on a proper application framework (BizTalk Server 2006), which has managed majority of these issues on its own.

7.9 Administrative Issues

In the development of BioNET with OOD, both the functional and non-functional requirements were all realized in terms of active and running objects. This approach necessitates the coding or re-coding of all issues in order to manage the change of requirements. Another administrative issue for OOD to keep in mind that almost every requirement should be delegated to IT personnel to model, design, develop and deploy, which lengthens and complicates the process as well as making it more error prone.

On the other hand, the development of BioNET with FOD has embraced a more declarative and generative approach that facilitates some minor changes even through the standard tools and abilities BizTalk Server 2006 provides. In that sense, the loosely coupled nature of Service-Oriented Architecture and the more agile nature of Feature-Driven Development have helped much to leverage a more systematic administrative environment for BioNET with FOD implementation.

7.10 Tool Support

The implementation of BioNET with OOD has been facilitated with Visio as the UML Case tool, .NET 2.0 Framework as the composing environment, Visual Studio 2005 as the development environment, and MS-SQL Server 2000 as the database management system. Even though the same vendor has provided the complete toolset, a lack of integrity has been observed during the development of the application. For example, model changes in Visio could not be reflected automatically to the actual class implementations in Visual Studio. This complicated the debugging and monitoring efforts as well.

Microsoft BizTalk Server 2006, rather, provides a more complete environment to model, design, implement, test, and deploy the application within the context of FOD approach. The development environment empowers more declarative and generative abilities, which lessens down the development and maintenance efforts dramatically. The most important drawback of FOD approach in tool support has been faced in drawing the FODA-compliant feature diagrams since this part is completely an add-on to BizTalk Server 2006, which was not originally designed to support neither features, nor Feature-Oriented Domain Analysis.

CHAPTER 8

SUMMARY AND CONCLUSIONS

This chapter summarizes the achievements and concludes the issues based on applying Feature-Oriented Development and Object-Oriented Development approaches on a case study and comparing the results accordingly. Extensions of the study have also been provided at the end.

8.1 Summary

A software development process defines the values, principles and practices used to achieve the goal of the software project. It aims, to promote best practices, try to satisfy customers' real needs, and to infuse a common vision and culture in a team [Hayes and Andrews]. There has been a lot of software development processes created over the years [Hayes and Andrews]. An overview of software development methodologies reported in Chapter 2, where these methods have been mainly classified as formal and informal. Informal methods were classified as heavy or agile in nature.

Each heavy methodology, such as Object-Oriented Development using objects, Model-Driven Development using models and Software Product Line Engineering using assets, has some certain characteristics to identify it from the others.

On the other hand, agile methods are approaches to managing the development of Internet products and services based on principles of flexible manufacturing and lean development [Rico, 2006]. Agile methods emerged with a focus on early customer involvement, iterative development, self-organizing teams, and flexibility [Hayes and Andrews]. The fundamental agile methods are Scrum, Dynamic Systems Development Method, Crystal Methods, Lean Development, Extreme

Programming, Adaptive Software Development, and Feature-Driven Development [Highsmith, 2002].

The use of features in software development has been discussed in Chapter 3. The feature concept and the use of features both in domain and application engineering have been introduced. For domain engineering, Feature-Oriented Domain Analysis is commonly used in a case study to represent the basic building blocks of the software development. For application engineering, the agile software development lifecycle of Feature-Driven Development introduced and applied practices whole the software development also in a case study. Application engineering uses the production facilities provided during domain engineering to produce applications of the family quickly. However, application engineering can be performed in parallel to the domain engineering [Harsu, 2003].

Chapter 4 has introduced the proposed Feature-Oriented Development approach. FOD uses a FODA-based feature domain modeling approach for domain engineering, which is supported by an agile Feature-Driven Development process for the application engineering. Moreover, the proposed FOD approach uses a different implementation strategy rather than conventional FDD is proposing the classical objects of OO, which is based on a loosely coupled combination model composing “business services”, “business rules”, and “business workflows” on a Service-Oriented Architecture.

Chapter 5 and Chapter 6 have demonstrated the way to apply OOD and FOD on the case study BioNET, respectively. BioNET is a security-based controlled, product and shopping management system using fingerprint validation system. The comparative analysis of both methodologies has been explained in detailed in Chapter 7.

8.2 Conclusions

An Object-Oriented Software Development Methodology (OOSDM) is specifically aimed at viewing, modeling and implementing the system as a collection of interacting objects, using specialized modeling languages, activities and techniques needed to address the specific issues of the object-oriented paradigm [Ramsin, 2006]. An important feature of Object-Oriented Development (OOD) is that software objects represent real-world objects [Palmer and Felsing, 2002].

The Object-Oriented Software Development Life Cycle [Purcell, 2007] model has these phases that roughly correspond to the traditional SDLC phases noted in brackets:

1. **Object-Oriented Requirements Analysis (OORA) [System Analysis]:** This is where classes of objects and the interaction between them are defined.
2. **Object-Oriented Analysis (OOA) [Analysis]:** In terms of object-oriented concepts, understanding, and modeling a particular problem within a problem domain.
3. **Object-Oriented Design (OOD) [System Design Specification]:** The object is the basic unit of modularity; objects are instantiations of a class.
4. **Object-Oriented Programming (OOP) [Programming and Testing]:** Emphasizes the employment of objects and methods rather than types or transformations, as in other programming approaches.

The proposed approach, **Feature-Oriented Development using Service-Oriented Architecture**, is defined with models, methodology and techniques. The concept of the proposed approach is analyzing the domain by **FODA** as a “feature model”, and applying the software development processes with **FDD** based on “feature lists”, and implementing the features in terms of “business services”, “business rules”, and “business workflows” all having their roots on Web services of Service-Oriented Architecture.

The followings are the main results and contributions of this thesis:

In the proposed approach, each feature can be represented with the Business Workflow Model (BWM) to follow business activities, the Business Rule Model (BRM) to classify and manage all its rules, and the Business Computation Model (BCM) to execute the business activities. So, the term “feature” can be simply defined in this form as a combination of business process models:

$$\mathbf{Feature = BWM + BRM + BCM}$$

This thesis work either formed or exploited the FOD approach with:

1. Using features facilitates the user and developer ensuring to understand the requirements in terms of business-valued “features”.

2. Main strategy is to define the elements, which are subsystem, services, etc., and how the domain features are allocated to them. The feature lists have been built using FDD practices, which provided a facility to decompose the application acts.
3. The business process model is created with feature or feature set(s) and the feature lists are then realized by FDD practices.
4. The features are modeled in terms of services, rules, and workflows with a “separation of concerns” principle in mind. They are later composed and orchestrated by a Business Process Modeling framework.
5. FOD approach provides the communication in terms of loosely coupled Web services.
6. Business modeling technique provides implementation with less effort of coding.
7. The FOD approach facilitates the reuse and sharing of feature models.
8. Debugging and monitoring in FOD follow easily rather than OOD method in implementation BioNET. Because of each feature can be implemented in a single application as Web service. Consequently, the maintenance of a single application or service is rather easy to achieve.

8.3 Extensions of the Study

As an extension, a seamless integration of a FODA modeling tool with BizTalk Server 2006 will make the FOD a “roundtrip engineering” approach, where reverse and forward engineering might be facilitated.

Another extension of the study might be proposing a more detailed feature-oriented design technique to manage the commonality and variability of features in terms of “business services”, “business rules”, and “business workflows”.

A future work might be another comparative analysis of OOD with FOD, but this time features can be implemented with AHEAD model and thorough Feature-Oriented Programming (FOP). Even, the results of such a study will be further cross-analyzed with the results of this thesis study where a three column (OOD, FOD using SOA, and FOD using FOP) matrix can be formed.

REFERENCES

- [1] **ABRAHAMSSON, P., et al.**, [2002], *Agile Software Development Methods Review Analysis*, VTT Electronics, University of Oulu, Finland.
- [2] **AKTAS, Z., and CETIN, S.**, [2006], *We Envisage the Next Big Thing*, In Integrated Design and Process Technology, IDPT-2006, Society for Design and Process Science, San Diego, CA, USA, June.
- [3] **AMBLER, S.W.**, [1], URL: <http://www.ddj.com/dept/architect/184415799>.
- [4] **AMBLER, S.W.**, [2], URL: <http://www.agilemodeling.com/essays/fdd.htm>.
- [5] **ANDERSON, D.J.**, [2004], *Feature-Driven Development: Towards a TOC, Lean and Six Sigma Solution for Software Engineering*, Microsoft Corporation, October.
- [6] **ANDERSON, D.J.**, URL: <http://www.agilemanagement.net/Articles/Weblog/Archives/February2007.html>.
- [7] **ARSLAN, V., et al.**, [2003], *Concurrent Object-Oriented Programming on .NET*, IEE Proceedings Software, Special Issue on ROTOR, October.
- [8] **BATORY, D., et al.**, [2003], *Scaling Step-Wise Refinement*, International Conference on Software Engineering, Portland, Oregon.
- [9] **BENAVIDES, D., et al.**, [2006], *A Survey on the Automated Analyses of Features Models*, XV Jornadas de Ingeniería del Software y Bases de Datos JISBD, Barcelona.
- [10] **BERG, K., and BISHOP, J.**, [2005], *Tracing Software Product Line Variability – From Problem to Solution Space*, University of Pretoria and DIRK MUTHIG Fraunhofer IESE.
- [11] **BOOCH, G.**, [1993], *Object-Oriented Analysis and Design with Applications* (2nd ed.). Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.
- [12] **BRAGANÇA, A., and MACHADO, R.J.**, [2004], *A Methodological Approach to Domain Engineering for Software Variability Enhancement*, Proceedings of the 2nd Workshop on Method Engineering for Object Oriented and Component Based Development - ME'04 (OOPSLA'04), Vancouver, Canada, pp. 39-50, COTAR Edition, Sydney, Australia, [ISBN-0-9581915-3-0], October.

- [13] **CAUSE, G.**, [2004], *Delivering Real Business Value using FDD*, Methods & Tools, Global knowledge source for software development professionals ISSN 102-4918, Winter 2004 (Volume 12 – number 4).
- [14] **CHAPPEL, D.**, [2005], *Understanding BizTalk Server 2006*, Microsoft Corporation.
- [15] **CLEMENTS, P., and NORTHROP, L.**, [2001], *Software Product Lines: Practices and Patterns*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [16] **COAD, P., and YOURDON, E.**, [1991], *Object-Oriented Analysis (2nd ed.)*, Yourdon Press, Upper Saddle River, NJ, USA.
- [17] **COAD, P., et al.**, [1999], *Java Modeling in Color with UML*, Prentice Hall.
- [18] **COAD, P.**, [2003], *Agile Processes: Developing Your Own Secret Recipes*, Borland.
- [19] **COHEN, S.G., et al.**, [1992], *Application of Feature-Oriented Domain Analysis to the Army Movement Control Domain*, Technical Report CMU/SEI-91-TR-028 ESD-91-TR-028, Carnegie Mellon University, Pittsburgh, Pennsylvania.
- [20] **CZARNECKI, K.**, [2005], *Overview of Generative Software Development*. In J.-P. Banâtre et al. (Eds.): *Unconventional Programming Paradigms (UPP) 2004*, Mont Saint-Michel, France, LNCS 3566, pp. 313–328.
- [21] **DE OLIVEIRA, T.C., et al.**, [2001], *Using XML and Frameworks to Develop Information Systems*, Pontificia Universidade Católica do Rio de Janeiro, Brazil.
- [22] **DEARING, R.**, URL:http://searchwebservices.techtarget.com/originalContent/0,289142,sid26_gci871817,00.html.
- [23] **ELRAD, T., et al.**, [2001], *Discussing Aspects of AOP*, Communications of the ACM 44, pp. 33–38.
- [24] **FANCEY, J.**, URL:<http://msdn.microsoft.com/msdnmag/issues/05/03/BPEL4WS>.
- [25] **FDD**, URL:<http://www.featuredrivendevelopment.com/>.
- [26] **FICHMAN, R. G., and KEMERER, C. F.**, [1997] *Object Technology and Reuse: Lessons from Early Adopters*. Computer, 30(10):47–59.
- [27] **GARCIA, G.**, URL:http://www.eetasia.com/ARTICLES/2006JUN/PDF/EEOL_2006JUN16_EMS_EDA_TA_01.pdf.
- [28] **GARDNER, T.**, URL: <http://www.ariadne.ac.uk/issue29/gardner/>.
- [29] **GITZEL, R., and KORTHAUS, A.**, [2004], *The Role of Metamodeling in Model-Driven Development*. In Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI2004) , 19-21 July, 2004, Orlando, USA, July.

- [30] **GMV**, [2007], *Domain Engineering Methodologies Survey*, Version: Issue 1 Draft C, Date: June 20, 2007, GMV Innovating Solutions, Madrid.
- [31] **GRISS, M.L., et al.**, [1998], *Integrating Feature Modeling with the RSEB*, Software Reuse, 1998. Proceedings of Fifth International Conference on Volume, Issue, 2-5 Jun 1998 Page(s): 76 – 85.
- [32] **HARRISON, A., and TAYLOR, I.**, URL: <http://www.cl.cam.ac.uk/~ey204/ADPUC/ADPUCSCHEDULE/Papers/a2-harrison.pdf>.
- [33] **HARSU M.**, [2003], *From Architectural Requirements to Architectural Design*. Report 34, Institute of Software Systems, Tampere University of Technology, May, 44 pp.
- [34] **HAYES, S., and ANDREWS, M.**, URL: <http://www.wrytradesman.com/articles/>.
- [35] **HIGHSMITH, J.** [2002], *Agile Software Development Ecosystems*, Pearson Education.
- [36] **HIGHSMITH, J.M., and Cutter Consortium**, [2002], *What Is Agile Software Development?*, CrosTalk The Journal of Defense Software Engineering.
- [37] **JADHAV, A., et al.**, URL: http://www.wagse.informatik.unikl.de/teaching/re/ss2007/downloads/Student%20Talks/FODA_Talk.pdf.
- [38] **KANG K., et al.**, [1990], *Feature Oriented Domain Analysis (FODA) Feasibility Study*, Technical report CMU/SEI -90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburg, PA.
- [39] **KANG K., et al.**, [1998], *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*. Annals of Software Engineering, 5:143–168.
- [40] **KHRAMTCHENKO, S.**, [2004], *Comparing eXtreme Programming and Feature Driven Development in Academic and Regulated Environments*, Final paper for CSCIE-275: Software Architecture and Engineering, Harvard University.
- [41] **KICZALES, G.**, [1996], *Aspect-Oriented Programming*, ACM Computing Survey, Volume 4: 157.
- [42] **KICZALES, G., et al.**, [1997] *Aspect-Oriented Programming*, in ECOOP'97, LNCS 1241, pp. 220–242.
- [43] **KOLLU, K.R.**, [2005], *Evaluating the PLUSS Domain Modeling Approach by Modeling the Arcade Game Maker Product Line*, Master's Thesis in Computing Science June 21st, 2005, Umeå University, Sweden.
- [44] **KULOOR, C., and EBERLEIN, A.**, (2002), *Requirements Engineering for Software Product Lines*, Proceedings of the 15th International Conference on Software & Systems Engineering and their Applications (ICSSEA'02), Paris, France.

- [45] **LEICH, T. et al.**, URL: <http://www.cs.uvic.ca/~mstorey/etx2005/papers-1/12%20Tool%20Support%20for%20Feature%20Oriented%20Software%20Development%20FeatureIDE.pdf>.
- [46] **MANGAN, P.J., and SADIQ, S.**, [2003], *A Constraint Specification Approach to Building Flexible Workflows*, Journal of Research and Practice in Information Technology, Vol. 35, No. 1, February.
- [47] **MICROSOFT**, URL: <http://www.microsoft.com/>.
- [48] **MORRISON, K.**, URL: http://www.nysia.org/events/past/docs/20031021_softEngProjMgt_fddMorrison.pdf.
- [49] **MSDN**, URL: <http://msdn2.microsoft.com/en-us/library/aa560249.aspx>.
- [50] **OLIVIER, B.**, URL: <http://www.jisc.ac.uk>.
- [51] **OMG**, URL: <http://www.omg.org/mda>.
- [52] **PALMER, S.R.**, URL: <http://www.phptr.com/articles/article.asp?p=26059&rl=1>.
- [53] **PALMER, S.R., and FELSING, J.M.**, [2002], *A Practical Guide to Feature-Driven Development*, Prentice Hall, (ISBN 0-13-067615-2).
- [54] **PANCAKE, C. M.**, [1995], *The Promise and the Cost of Object Technology: A Five-Year Forecast*. Communications of ACM, 38(10):32–49.
- [55] **PINTO, M., et al.**, [2002], *Separation of Coordination in a Dynamic Aspect Oriented Framework*, AOSD 2002, Enschede, The Netherlands.
- [56] **PRESSMAN, M.**, [2005], *Software Engineering*, 6th Ed., McGraw Hill, International Edition.
- [57] **RAMSIN, R.**, [2006], *The Engineering of an Object-Oriented Software Development Methodology*, Department of Computer Science The University of York, UK, April.
- [58] **RICO, D.F.**, [2006], *Agile Methods and the Links to Customer Satisfaction and Firm Performance*, Strawman, Version 0.0, August 21.
- [59] **ROTHENBERGER, M. A., et al.**, [2003], *Strategies for Software Reuse: A Principal Component Analysis of Reuse Practices*. IEEE Transactions on Software Engineering, 29(9):825–837.
- [60] **SCHAPIRO, A.**, URL: <http://blogs.southworks.net/blogs/ariel/default.aspx?p=2>.

- [61] **SCHWADERER, C.**, [2006], *Pioneering Model Driven Development*, CompactPCI and AdvancedTCA Systems, US.
- [62] **SEI**, URL: http://www.sei.cmu.edu/domain-engineering/domain_eng.html.
- [63] **SPARX SYSTEMS**, URL: http://www.sparxsystems.com/platforms/business_process_modeling.html.
- [64] **SYED-ABDULLAH, S.L.**, [2005], *Empirical Study on Extreme Programming*, Department of Computer Science University of Sheffield, January.
- [65] **WEBBER, J.**, URL: <http://soa.sys-con.com/read/39830.htm>.
- [66] **WHITE, S.M.**, [2004], *A Comparative Analysis of Object-Oriented and Other Methods For Modeling Computer Based Systems*, Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04).
- [67] **ZHANG, L.J.**, [2003], *On Demand Business Collaboration with Services Computing, E-Business Solutions*, IBM T.J. Watson Research Center.
- [68] **ZUBROW, D., and CHASTEK, G.**, [2003], *Measures for Software Product Lines*. Technical Report CMU/SEI-2003-TN-031, Carnegie Mellon University, Software Engineering Institute.

APPENDIX A

Desktop Application of BioNET: ADMINISTRATION MODULE

Employees fingerprint administration;

- Registering new fingerprint samples
- Deleting fingerprint samples

Fingerprint registration process

Parmak İzi Kayıt Ekranı

Employee Searching:

- by Status
- by Name and Surname

Kisi Arama

Alta

Administrator
 Öğrenci
 Öğretmen
 Okul İdareci
 Kurum İdareci
 Veli
 Personel
 Stajyer Öğretmen
 Bilim Kurulu Üyesi
 Aile Fardi

Tümünü Seç

Adı Kamil
Soyadı ZEN
T.C. Kimlik No 12212113121

TCKimlikNo	Adı	Soyadı
654656565646	Ülkü	
122344565656	Nil	
54545454212	Hasan	
12235454446	Z.Deniz	
11227144316		
13458797442		
12457878545		
12313154841		
12494420196		
12212113121		
12121211212		
13214261808		
32135465165		
54654565456	Ümer	

APPENDIX B

BioNET CREDIT MANAGEMENT MODULE

Parmak izi Kaydı Yemekhane ve Kredi İşlemleri Güvenlik İlkeleri

Yemekhane Kaydı **Lunch registration**

Yemekhane Kaydı Onayla

Kredi İşlemleri **Employee shopping history**

[Kullanıcı Ekstresi](#)

[Kullanıcı Ürün İndirim Denetimi](#)

[Kullanıcı Hizmet Noktası Denetimi](#)

[Kredi Geçmişi](#)

Daily Limit

[Günlük Limit](#) 20

Kalan Kredi Miktarı 150

Employee service point restrictions

Credit History

Kaydet

Employee shopping discount management

FaturaNo	Tarih	Tutar
123123	13.10.2006 14:11	150,00 TL

Employee credits

APPENDIX C

BioNET SHOPPING MANAGEMENT MODULE

Ürün Listesi

Ekle	Ad	Grubu
▶	Hamburger	Fast food
	Sosizli Sandwich	Fast food
	Soguk Sandwich	Fast food
	Toot kayali	Fast food
	Toot salinli	Fast food
	Toot sucuklu	Fast food
	Toot karigak	Fast food
	Gozleme	Fast food
	Patales kuzatma	Fast food
	Inegol kofte	Fast food
	Izgara kofte	Fast food
	Pizza	Fast food
	Kumpir sade	Fast food
	Kumpir soszli	Fast food
	Kumpir karigak	Fast food
	Pogaca	Fast food
	Ekler Pasta	Fast food
	Profitorol	Fast food
	Alman Pastasi	Fast food
	CHEESEBURGER	Fast food
	TON BALIKLI SAND.	Fast food
	SALATA	Fast food
	YAS PASTA	Fast food
	ÇORBA	Fast food
	ET DONER	Fast food

Sistem Yöneticisi Harcamaları

Çıkar	Ürün	Miktar	Tutar
▶	MAGNUM	1	2,00 TL
	COPNETTO	1	1,50 TL

Toplam Tutar : 3,50 TL **Kalan Kredi Miktarı : 1,40 TL**
Günlük Limit : 15,50 TL

Yapılan Son Alışveriş
 Adı : x
 Soyadı : x
 Son Harcama Tutarı : x
[Stok Durumu](#)

Kullanıcı Listesi

Adı Soyadı	Customer List and credit information
Sistem Yöneticisi	1,40 TL
AYŞE KIZILDEMİR	0,15 TL
MELİKE SÜLÜZÜN	0,00 TL

Ürün Grupları

Yan ürünler	Product Groups
İçecekler	
Fast food	
Bisküvi	
Şekerleme	
Çikolata	
Kırtasiye	

APPENDIX D

BioNET SHOPPING MANAGEMENT MODULE

SALE PROCESSING

BioNET Ürün Yönetim Modülü [Sistem Yöneticisi]

Product list

Ürünler	Fiyat	ID	Adı	On/Demeli	Grup	BarKod	Combo
2.5	86	Hamburger	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.75	87	Soslu Sandwich	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.5	88	Soğuk Sandwich	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
1.5	89	Tost kaşarlı	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
1.5	90	Tost salanlı	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2	91	Tost sucuklu	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.5	92	Tost karnaj	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2	93	Göbeme	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
1.5	94	Palates kızartma	<input checked="" type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.5	96	İnegöl köfte	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
3	97	İzgara köfte	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.5	98	Pizza	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
2.5	100	Kumpir sade	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	
3	101	Kumpir soslu	<input type="checkbox"/>	Fast food		<input type="checkbox"/>	

Product management

Ürün ID: 161
 Ürün Adı: YAŞ PASTA
 Ürün Grubu: Yan ürünler
 BarKod No:
 Fiyat: 1.5
 On/Demeli:
 Aktif:

Ürün Grubu Dizenleme
 Ürün Grubu:
 Ürün Grubu Seç: Yan ürünler
 Güncelle

Products discount management

Adı	BarKod	Fiyat
Hamburger		2.5
Coca cola bardak		1.5
Palates kızartma		1.5

Combo product management

Adı	BarKod	Fiyat
Hamburger		2.5
Coca cola bardak		1.5
Palates kızartma		1.5

Products inventory management

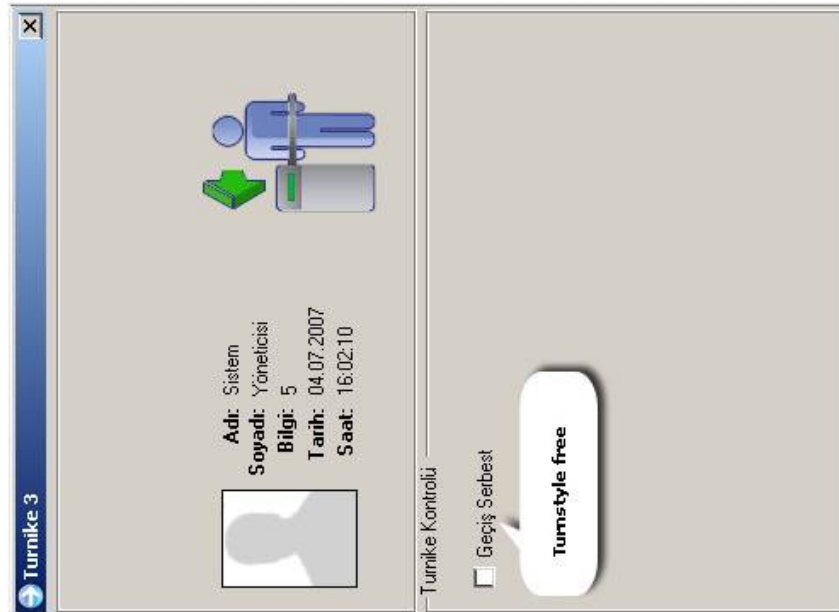
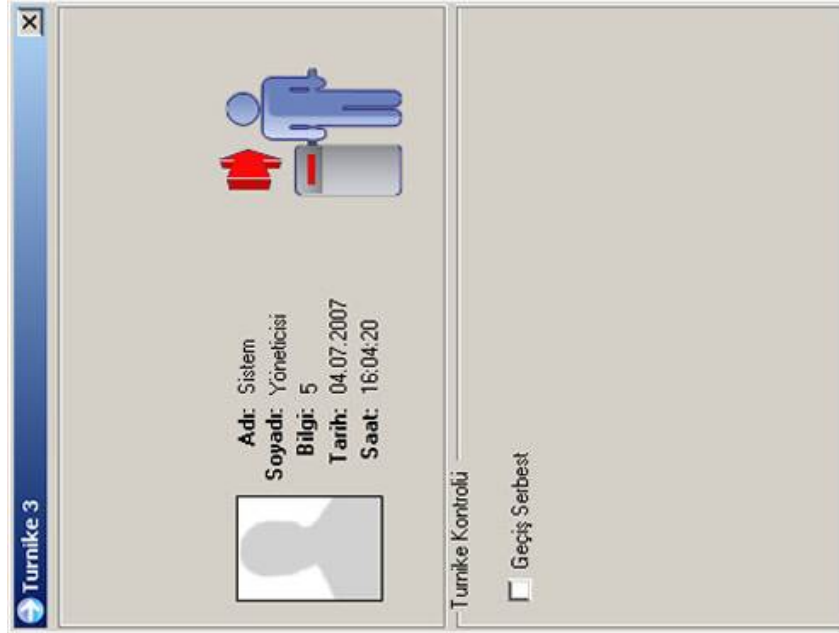
Hizmet Noktasında Kalan Ürün Miktarı	Fiyat	ID	Adı	On/Demeli	Miktar	Grup	BarKod	Hizmet Noktası
6440	2.5	86	Hamburger	<input type="checkbox"/>	6440	Fast food		Fast Food
9353	2.75	87	Soslu Sandwich	<input type="checkbox"/>	9353	Fast food		Fast Food
8498	2.5	88	Soğuk Sandwich	<input type="checkbox"/>	8498	Fast food		Fast Food
3158	1.5	89	Tost kaşarlı	<input type="checkbox"/>	3158	Fast food		Fast Food
8904	1.5	90	Tost salanlı	<input type="checkbox"/>	8904	Fast food		Fast Food
9277	2	91	Tost sucuklu	<input type="checkbox"/>	9277	Fast food		Fast Food
7203	2.5	92	Tost karnaj	<input type="checkbox"/>	7203	Fast food		Fast Food
9528	2	93	Göbeme	<input type="checkbox"/>	9528	Fast food		Fast Food
3177	1.5	94	Palates kızartma	<input type="checkbox"/>	3177	Fast food		Fast Food
7684	2.5	96	İnegöl köfte	<input type="checkbox"/>	7684	Fast food		Fast Food
9323	3	97	İzgara köfte	<input type="checkbox"/>	9323	Fast food		Fast Food
836	2.5	98	Pizza	<input type="checkbox"/>	836	Fast food		Fast Food
9035	2.5	100	Kumpir sade	<input type="checkbox"/>	9035	Fast food		Fast Food

Stok
 Hizmet Noktası: Fast Food
 Ürün Adı: YAŞ PASTA
 Tarih: 04.07.2007
 Miktar:
 Gönder

Hizmet Noktası Stok Tanıması
 Hizmet Noktası Seç: Fast Food
 Göster

APPENDIX E

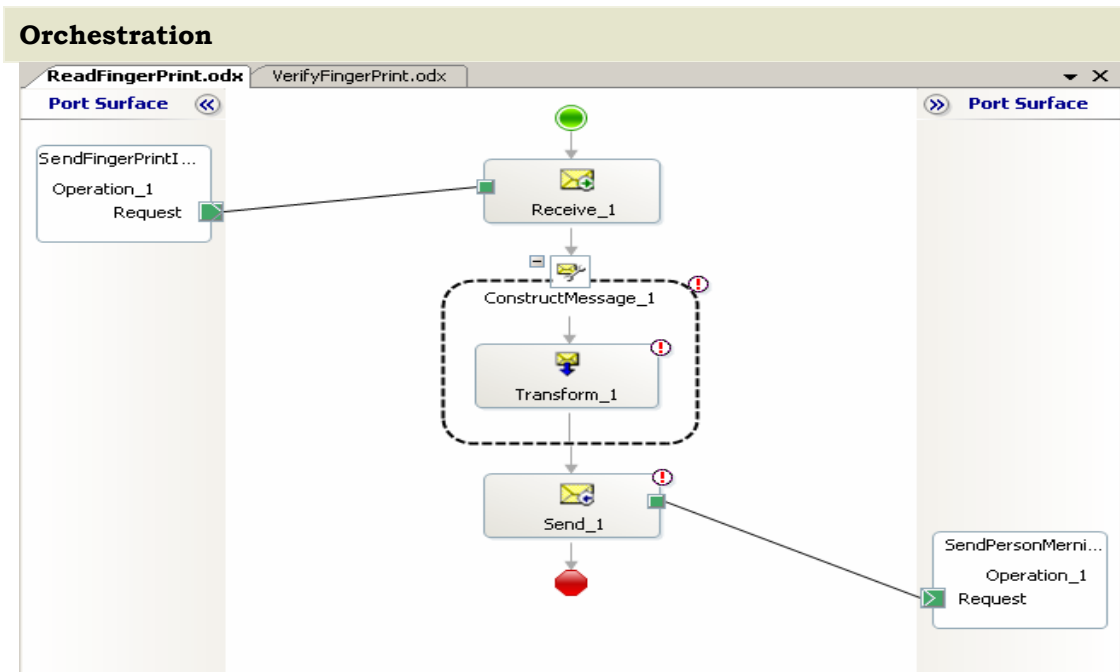
BioNET TRANSITION MANAGEMENT MODULE



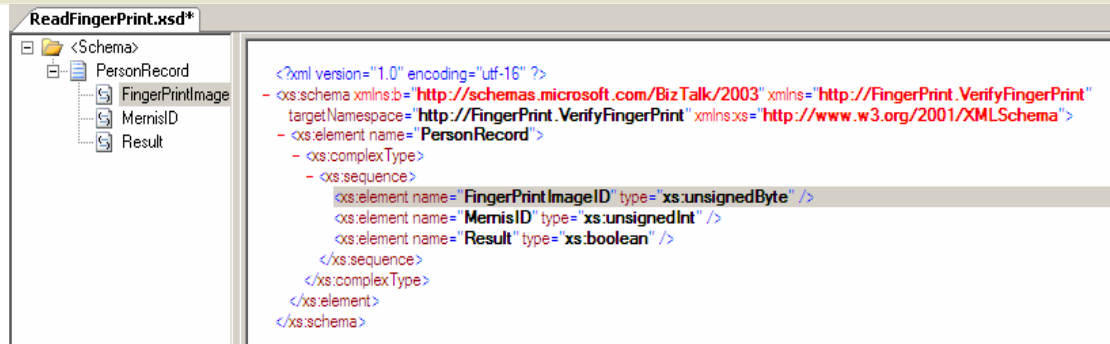
APPENDIX F

SAMPLE BioNET ORCHESTRATIONS, XML SCHEMAS AND RULE COMPOSITIONS

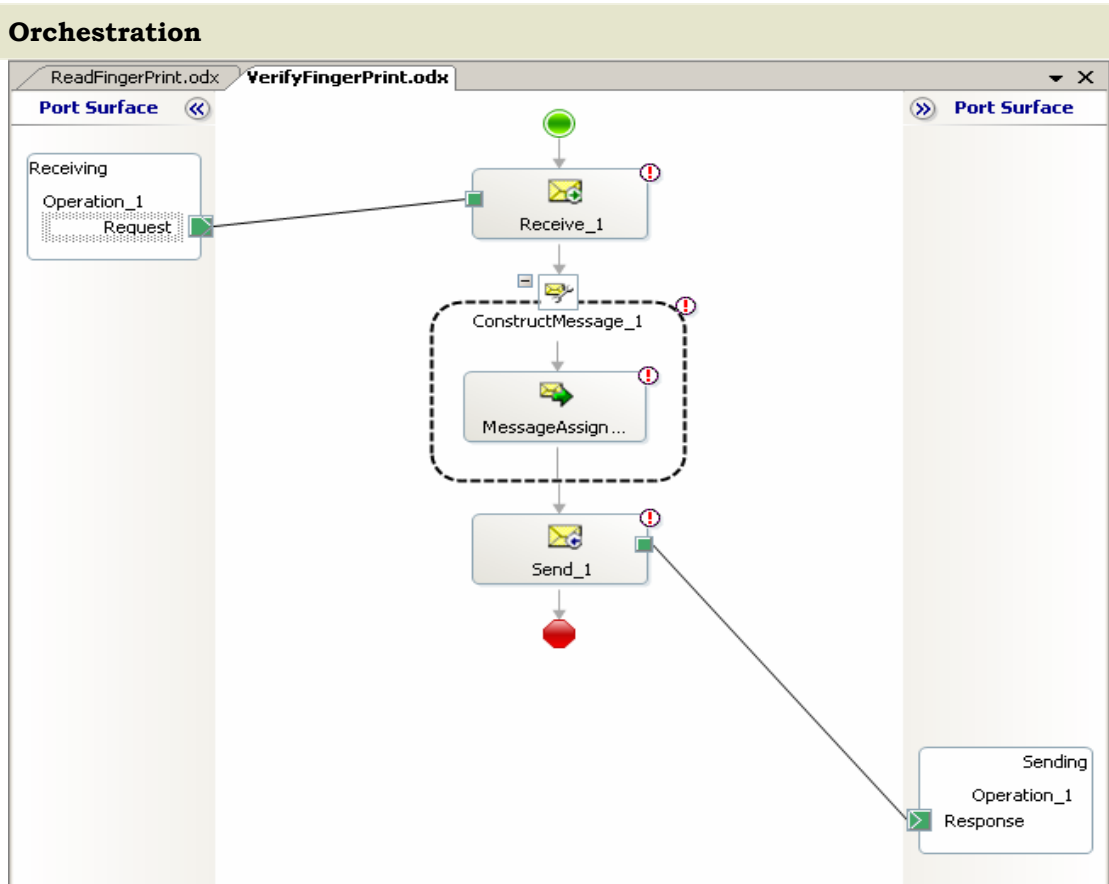
F1: READ A FINGERPRINT



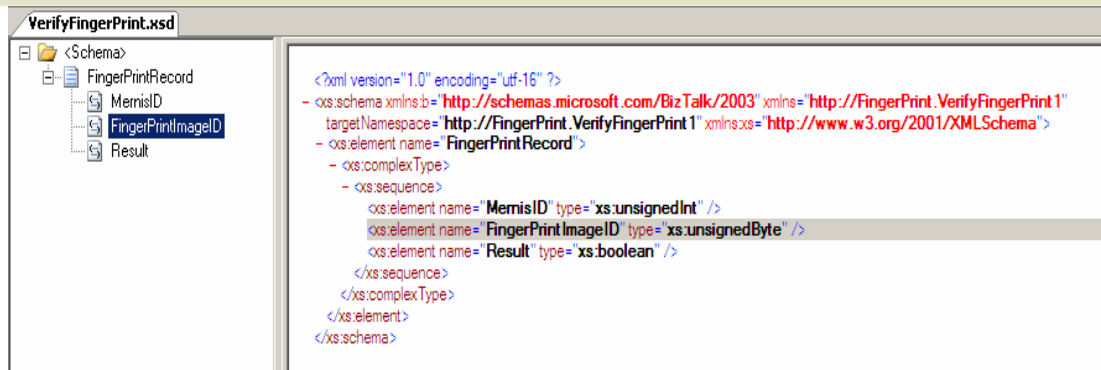
XML Schema



F2: VERIFY A FINGERPRINT



XML Schema



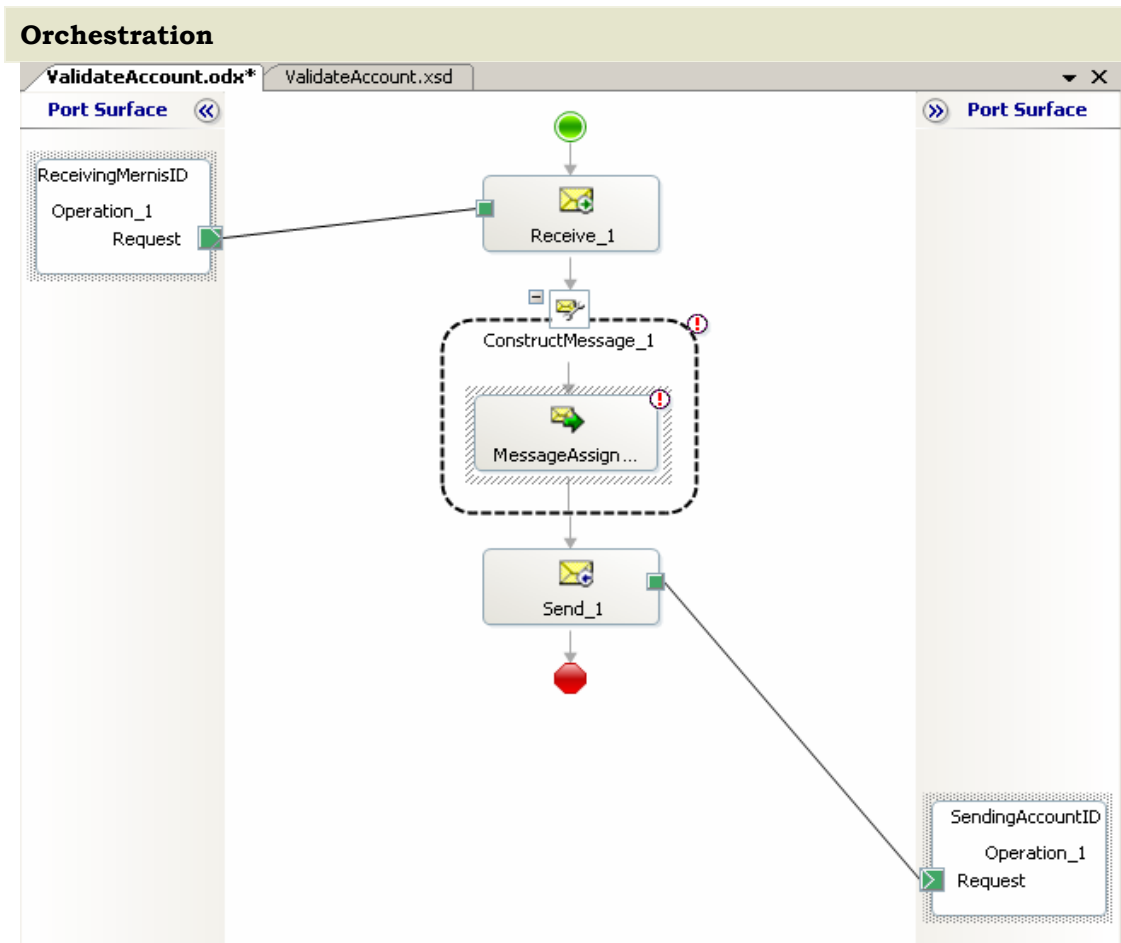
F2: (Cont'd)

Rule Composition

The screenshot displays the Microsoft Business Rule Composer interface for a rule named "VerifyFingerPrint - Version 1.0 - Verifying". The interface is divided into several panes:

- Policy Explorer:** Shows a tree view of policies including LoanProcessing, PolicyPurchaseOrder, VerifyFingerPrint, and its sub-policies Version 1.0 and Verifying.
- Facts Explorer:** Shows a tree view of schemas including VerifyFingerPrint.xsd, FingerPrintRecord, MemrisD, FingerPrintImage, and Result.
- IF Conditions:** Contains an AND condition with two sub-conditions:
 - FingerPrint.VerifyFingerPrint1;/FingerPrintRecord/MemrisID is equal to BioNET.Kullanici.MemrisID
 - FingerPrint.VerifyFingerPrint1;/FingerPrintRecord/FingerPrintImageID is equal to BioNET.Parmak.ParmakIndex
- THEN Actions:** Contains one action:
 - FingerPrint.VerifyFingerPrint1;/FingerPrintRecord/Result = True

F3: VALIDATE AN ACCOUNT



XML Schema

```
ValidateAccount.xsd ValidateAccount.odx
```

<Schema>
 Root
 MerrisID
 AccountID
 Status

```
<?xml version="1.0" encoding="utf-16" ?>  
- <xs:schema xmlns:b="http://schemas.microsoft.com/BizTalk/2003" xmlns="http://FingerPrint.ValidateAccount"  
  targetNamespace="http://FingerPrint.ValidateAccount" xmlns:xs="http://www.w3.org/2001/XMLSchema">  
- <xs:element name="Root">  
- <xs:complexType>  
- <xs:sequence>  
  <xs:element name="MerrisID" type="xs:unsignedInt" />  
  <xs:element name="AccountID" type="xs:unsignedInt" />  
  <xs:element name="Status" type="xs:boolean" />  
</xs:sequence>  
</xs:complexType>  
</xs:element>  
</xs:schema>
```

F3: (Cont'd)

Rule Composition

Microsoft Business Rule Composer - NIDAL/BIZTALKRULEENGINE.DB

Rule Store Edit Help

Policy Explorer

- PolicyPurchaseOrder
- ValidateAccount
 - Version 1.0
 - Validating
- VerifyFingerPrint

Facts Explorer

- Voc...
- XML...
- Dat...
- NE...
- Schemas
 - ValidateAccount.xsd
 - Root
 - MemisID
 - AccountID
 - Status

ValidateAccount - Version 1.0 - Validating

IF

Conditions

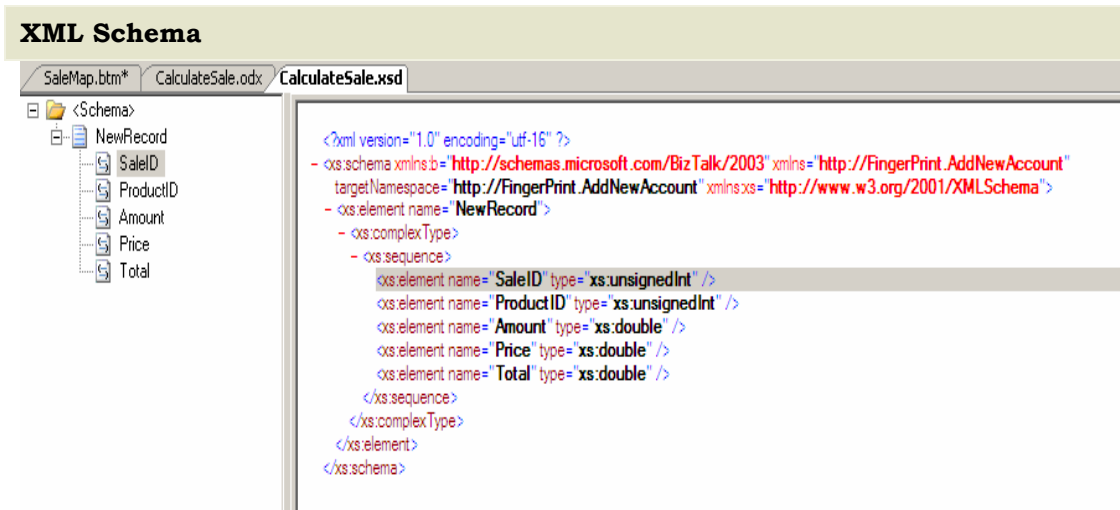
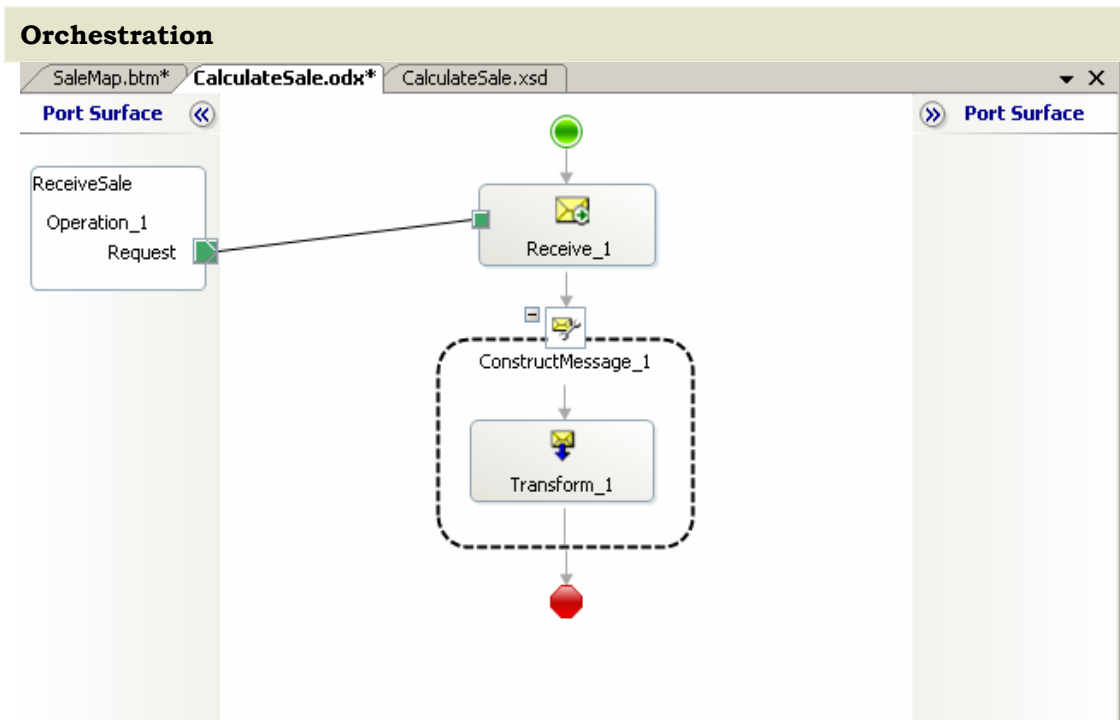
- AND
 - FingerPrint.ValidateAccount/Root/MemisID is equal to BioNET.Kullanici.MemisID
 - FingerPrint.ValidateAccount/Root/AccountID is equal to BioNET.Kredi.ID

THEN

Actions

- FingerPrint.ValidateAccount/Root/Status = True

F4: CALCULATE A SALE



APPENDIX G

BioNET WEB SERVICES

BioNETWebService

The following operations are supported. For a formal definition, please review the [Service Description](#).

- [ParmakDelete](#)
- [RunProcedure\(String,String\)](#)
- [Search](#)
- [Load\(IDList\)](#)

G1: READFINGERPRINT WEB SERVICE

ReadFingerPrint

Click [here](#) for a complete list of operations.

Reading

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
FingerPrintImageID:	<input type="text"/>

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
Produced by Yuce Information System, May 2007
POST /ReadFinger/ReadFingerPrint.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Reading"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Reading xmlns="http://tempuri.org/">
      <FingerPrintImageID>string</FingerPrintImageID>
    </Reading>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ReadingResponse xmlns="http://tempuri.org/">
      <ReadingResult>long</ReadingResult>
    </ReadingResponse>
  </soap:Body>
</soap:Envelope>
```

G2: VERIFYFINGERPRINT WEB SERVICE

VerifyFingerPrint

Click [here](#) for a complete list of operations.

Verifying

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
MernisID:	<input type="text"/>
FingerPrintImageID:	<input type="text"/>

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
Produced by Yuce Information System, May 2007
POST /VerifyFinger/VerifyFingerPrint.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Verifying"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Verifying xmlns="http://tempuri.org/">
      <MernisID>long</MernisID>
      <FingerPrintImageID>string</FingerPrintImageID>
    </Verifying>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <VerifyingResponse xmlns="http://tempuri.org/">
      <VerifyingResult>boolean</VerifyingResult>
    </VerifyingResponse>
  </soap:Body>
</soap:Envelope>
```

G3: CALCULATEOFTHESALE WEB SERVICE

CalculateOfTheSale

Click [here](#) for a complete list of operations.

Calculating

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

Parameter	Value
SaleID:	<input type="text"/>
ProductID:	<input type="text"/>
Amount:	<input type="text"/>
Price:	<input type="text"/>
Total:	<input type="text"/>

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
Produced by Yuce Information System, May 2007
POST /CalculateSale/CalculateOfTheSale.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/Calculating"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Calculating xmlns="http://tempuri.org/">
      <SaleID>int</SaleID>
      <ProductID>int</ProductID>
      <Amount>double</Amount>
      <Price>double</Price>
      <Total>double</Total>
    </Calculating>
  </soap:Body>
</soap:Envelope>
HTTP/1.1 200 OK
```

G3: (Cont'd)

```
Content-Type: text/xml; charset=utf-8
Content-Length: length<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CalculatingResponse xmlns="http://tempuri.org/">
      <CalculatingResult>boolean</CalculatingResult>
    </CalculatingResponse>
  </soap:Body>
</soap:Envelope
```

APPENDIX H

BioNET WEB SERVICE: VB.NET CODE

```
'Produced by Yuce Information System, May 2007
Imports System.Web.Services
<System.Web.Services.WebService(Namespace :=
"http://tempuri.org/BioNETWebService/BioNETWebService")> _
Public Class BioNETWebService
    Inherits System.Web.Services.WebService
    #Region " Web Services Designer Generated Code "
    Public Sub New()
        MyBase.New()
        'This call is required by the Web Services Designer.
        InitializeComponent()
        'Add your own initialization code after the
InitializeComponent() call
    End Sub
    .....
    <WebMethod(MessageName:="Identify(Byte(), Integer)")> _
    Public Function Identify(ByVal fingerprint As Byte(), ByVal
threshold As Integer) As Long
        Dim G As Byte
        Dim parmak As Parmak
        Dim contextH As Integer = VFCreateContext()
        Dim errorID As Integer = VFIdentifyStart(fingerprint,
contextH)
        SetDefault()
        CheckError()
    .....
        If errorID <> 0 Then
            VFIdentifyEnd(contextH)
            VFFreeContext(contextH)
            Throw New Exception(VFErrorToString(errorID))
        End If
    .....
    End Function

    <WebMethod(MessageName:="Identify(String, Integer)")> _
    Public Function Identify(ByVal fingerprint As String, ByVal
threshold As Integer) As Long
    .....
    End Function
```

```

    <WebMethod(MessageName:="RunProcedure(String)")
    Public Function RunProcedure(ByVal sql As String) As Integer
    .....
    End Function

    <WebMethod(MessageName:="RunProcedure(String,Integer)")> _
    Public Function RunProcedure(ByVal sql As String, ByVal ID As
Integer) As Integer

        Dim db As New dbObject
        Dim value As Integer
        value = db.RunProcedure(sql, ID)
        db.Close()
        Return value
    End Function

    <WebMethod(MessageName:="RunProcedure(String,String)")> _
    Public Function RunProcedure(ByVal sql As String, ByVal
tableName As String) As DataSet
    .....
    End Function

<WebMethod(MessageName:="RunProcedure(String,DataSet,tableName)")>
-
    Public Function RunProcedure(ByVal sql As String, ByRef dataset
As DataSet, ByVal tableName As String) As DataSet
        Dim db As New dbObject
        Dim ds As DataSet
    .....
        Return ds
    End Function

    <WebMethod(MessageName:="Load(mernisID)")> _
    Public Function Load(ByVal mernisID As Long) As Kullanici
    .....
    End Function

    <WebMethod(MessageName:="Load(userName,password)")> _
    Public Function Load(ByVal userName As String, ByVal password
As String) As Kullanici
    .....
    End Function

    <WebMethod()> _
    Public Sub ParmakSave(ByVal parmak As Parmak)
        Dim db As New dbObject
        Dim ds As DataSet
        Dim parameters(4) As SqlClient.SqlParameter
        Dim gruplar As ParmakGruplari =
CType(Application("ParmakGruplari"), ParmakGruplari)
    .....
        parameters(0) = New SqlClient.SqlParameter
        parameters(0).ParameterName = "@Prefix"
        parameters(0).SqlDbType = SqlDbType.Char

        If ds.Tables(0).Rows.Count = 0 Then

```

```

        parameters(0).Value = "I"
    Else
        parameters(0).Value = "U"
    End If

Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.ComponentModel

<System.Web.Services.WebService(Namespace:="http://tempuri.org/")>
_
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.Basi
cProfile1_1)> _
<ToolboxItem(False)> _
Public Class CalculateOfTheSale
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function Calculating(ByVal SaleID As Integer, ByVal
ProductID As Integer, ByVal Amount As Double, ByVal Price As
Double, ByVal Total As Double) As Boolean
        Return True
    End Function

    End If
.....
End Sub

    <WebMethod()> _
    Public Sub ParmakDelete(ByVal parmak As Parmak)
        Dim gruplar As ParmakGruplari =
CType(Application("ParmakGruplari"), ParmakGruplari)
        Dim grup As ParmakGrubu
        Dim rParmak As Parmak
        Dim db As New dbObject

.....

        For Each grup In gruplar
            For index As Integer = 0 To grup.Parmaklar.Count - 1
                If index < grup.Parmaklar.Count Then
                    rParmak = grup.Parmaklar(index)
                    If rParmak.Kullanici.MernisID =
parmak.Kullanici.MernisID AndAlso rParmak.Index = parmak.Index Then
                        grup.Parmaklar.Remove(rParmak)
                        index -= 1
                    End If
                End If
            Next
        Next
    End Sub

    <WebMethod()> _
    Public Function CheckIsRegistered() As String
        Return CStr(Application("ErrorString"))
    End Function

    Private Sub CheckError()

```

```
        If Not CStr(Application("ErrorString")) = "Registered" Then
            Throw New Exception(CStr(Application("ErrorString")))
        End If
    End Sub
End Class
```

```
Public Class ReadFingerPrint
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function Reading(ByVal FingerPrintImageID As String) As
Long
        Dim mernisID As Long
    .....
    End Function
```

```
Public Class VerifyFingerPrint
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function Verifying(ByVal MernisID As Long, ByVal
FingerPrintImageID As String) As Boolean
    .....
    End Function
```

```
Public Class CalculateOfTheSale
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function Calculating(ByVal SaleID As Integer, ByVal
ProductID As Integer, ByVal Amount As Double, ByVal Price As
Double, ByVal Total As Double) As Boolean
    .....
    End Function
```