2004

Mastering universal transaction management system (UTMS) with software patterns

Ram Goverdhana
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Goverdhana, Ram, "Mastering universal transaction management system (UTMS) with software patterns" (2004). *Master's Theses*. 2659.
DOI: <https://doi.org/10.31979/etd.46bg-a2bc>
https://scholarworks.sjsu.edu/etd_theses/2659

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

MASTERING UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS)
WITH SOFTWARE PATTERNS

A Thesis

Presented to

The Faculty of the Department of Computer Engineering
San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science in Software Engineering

By

Ram Goverdhana

December 2004

UMI Number: 1425461

Copyright 2004 by
Goverdhana, Ram

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1425461

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against
unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

©2004

Ram Goverdhana

ALL RIGHTS RESERVED

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

Dr. Mohamed E. Fayad

Mr. Dan Harkey

Dr. Marcus Fontoura, IBM Corporation

APPROVED FOR THE UNIVERSITY

ABSTRACT

MASTERING UNIVERSAL TRANSACTION MANAGEMENT SYSTEM THROUGH SOFTWARE PATTERNS

by Ram Goverdhana

One of the important challenges within business and research today is the overall management of knowledge-based services through transaction processing systems. When dealing with transactions across different domains, it is evident that the rules (e.g., Atomicity, Consistency, Isolation, Durability) under which the transactions are processed can also vary. Why do we need to duplicate the effort due to the variation in the domain in which they are processed? The proposed solution is to build a Pattern Language for Universal Transaction Management System (UTMS) based on the paradigm of Software Stability Modeling (SSM). This would enhance the overall process involved in providing an efficient and improved integration-based solution for applicability across any domain. This thesis will focus on the development of stable patterns for building the UTMS system.

Acknowledgements

First and foremost, I would like to thank my advisor Dr. Mohamed Fayed for his hearty support during the work of the thesis. He has provided valuable feedback and advice on my work to take it to new levels with enthusiasm and determination. My experience on working at IBM as a Software Engineer for the DB2 Development group has uniquely positioned me in thinking of technological ideas and tackling issues from both the industry and academic aspects. Thanks to my reviewers Dr. Markus Fontura from IBM Almaden Research Center and Mr. Dan Harkey from the Computer Engineering department for providing valuable feedback on my work. Special thanks to my parents, Ranga Chary Goverdhana and Sita Goverdhana, in Chicago for their patience and advice on gaining knowledge, understanding and wisdom on overall avenues of life and reality.

Table of Contents

List of Figures.....	viii
Chapter 1. Introduction	
1.1 Background.....	01
1.2 Motivation	
1.2.1 Problems and Challenges with Current Transaction Management System.....	03
1.2.2 Software Stability Modeling (SSM) Overview.....	05
1.3 Research Methodology.....	07
1.4 Research Goals and Contributions.....	08
1.5 Organization of the Thesis.....	09
Chapter 2. Universal Transaction Management System (UTMS) and Design Rational	
2.1 Introduction.....	11
2.2 Fundamental Concepts of Transaction	
2.2.1 Principals.....	11
2.2.2 Transaction Models and Applicability.....	14
2.3 Pattern Language for Universal Transaction Management System (UTMS)...	18
2.4 Conclusions.....	24
Chapter 3. Universal Transaction Management System (UTMS) Goals	
3.1 UTMS General Goals.....	25
3.2 Goals Analysis and UTMS Core Knowledge.....	26
3.3 Stable Analysis Patterns (SAP).....	39
3.4 Conclusions.....	40
Chapter 4. Capabilities of Universal Transaction Management System (UTMS)	
4.1 UTMS Universal Capabilities.....	41
4.2 Stable Design Patterns (SDP).....	53
4.3 Conclusions.....	55
Chapter 5. UTMS Knowledge Map and Architectural Patterns	
5.1 UTMS Knowledge Map Illustration.....	57
5.2 UTMS Architectural Patterns.....	60
5.3 Conclusions.....	62
Chapter 6. Development of Universal Transaction Management System (UTMS)	
6.1 UTMS Development Process Patterns.....	64
6.2 Sample Applications of UTMS.....	67
6.3 Conclusions.....	70

Chapter 7. Deployment of Universal Transaction Management System (UTMS)	
7.1 Deployment Issues in UTMS.....	71
7.2 Deployment Patterns.....	73
7.3 Conclusions.....	75
Chapter 8. Summary & Future Direction	
8.1 Conclusion.....	76
8.2 Future Direction.....	78
References.....	80
Appendix A. UTMS Stable Patterns List.....	83
Appendix B. Documented UTMS stable Patterns.....	85
Appendix C. Sample Code of Stable Patterns.....	115

List of Figures

1.1	Software Stability Life Cycle	06
2.1	Web Services Technology Stack	15
2.2	Managing Transactions within J2EE Environment	17
2.3	Long Running Transaction	18
3.1	Performance Design Pattern	30
3.2	Stable Pattern for Recovery	37
3.3	Important Goals for the UTMS System	39
4.1	Mapping of EBTs and BOs for Stability Analysis Patterns	42
4.2	Model of Transaction Pattern	46
5.1	Knowledge Map (K-Map) of the UTMS System	59
5.2	Application Scenario I within UTMS	61
5.3	Application Scenario II within UTMS	62
6.1	Mapping of EBTs and BOs and IOs for a Stable System	65
6.2	Travel Agent Application Scenario	69

CHAPTER 1

INTRODUCTION

1.1 Background

Within the Software Engineering arena, the overall development and management of the software systems has been critical to understand the customer core requirements well and to provide a well-managed, cost effective and high-quality based solution for meeting those requirements. As the software enterprise grows extensively, the need to develop a stable system across the enterprise would become more complex and expensive to manage.

With the exponential growth in the amount and type of information available across families of software systems and services, the rate at which the information is being processed, utilized, and maintained is critical to succeed in today's transforming business world. One of the important challenges within business and research today is the overall management and automation of knowledge-based services through the means of transaction processing. The term "transaction" has widely been used within the required context such as relational database systems, service computing, business process and social sciences. As a result, when dealing with any type of data and content that is shared across different domains, it is evident that the rules of the transactions (e.g., Atomicity, Consistency, Isolation, Durability) for processing the data could also vary. Based on the context of the events being executed and the applicability, *some, all or none* of these rules can be relaxed to better manage the overall transactional role of any given service. The existence of various types of transactions such as *flat transactions, nested transactions*

and *time-based transactions* depends on factors such as the criticality of the information being processed, its side effects of failure and the roles of all the events that exist within the transaction. On one hand, legacy based systems such as DB2 for z/OS, CICS, and IMS would offer its their own set of transactional services that are restricted to offer set of functions specific to their own environment. On the other hand, emerging systems based on standards such as XML, web services, semantic web, and business process, offer another set of functions. The integration of the functions and services through the means of transactions across the enterprise is complex to manage.

Every enterprise comprises of its own set of rules based on their business requirements, strategy, and software methodology for development and modeling software solutions. Existing software methodologies such as the Waterfall model, Spiral model (Boehm 1986, 1988), Rational Unified Process (RUP) and Extreme programming (Kent, 2000, 2002) have extensively been used to provide an improved process for developing software systems. However, these models are very different from each other, based on factors such as the modeling of the core requirement specification, implementation, testing, code reusability, maintenance, and the amount of time and money that one can incur to meet their needs. These are limited in scope to a specific set of needs for a particular system in isolation of their applicability across an enterprise for promoting reusability, stability and adaptability.

Service Oriented Computing (Endrei, 2004) is another emerging software technology that aims to provide an improved process for software solutions, particularly from the perspective of services, to better promote reusability and integration of services that

involve transaction processing. We will perform research on the software stability model (Fayad, 2000-2002) and its use in the domain of transaction management to achieve a stable system.

Construction of software patterns is a new process for building software systems and has widely been accepted within education and industry. The aim of patterns is to provide a model by example to help software developers resolve reoccurring problems across a given system. The solution to the problem is provided through the means of a common language to capture the code and knowledge.

1.2 Motivation

1.2.1 Problems and Challenges with Current Transaction Management System

Currently there is no single stable system that can manage any transactional support required to meet the needs of any customer. As more requirements are implemented into the existing system, the system itself can become unstable and the overall cost of maintenance and serviceability rises. Various types of information (structured, semi-structured, unstructured) and the petabytes of information available has resulted in the existence of a wide range of software application integration technologies that are more inclined towards offering a specific set of services for a specific domain. However, these technologies are built without any solid foundation of modeling, to provide a thorough understanding of the functional and non-functional requirements across any enterprise, thus resulting in cost overhead for effective implementation and management of the whole enterprise.

This leaves us to think about how to perceive and model the requirements of any given system, across any business enterprise. Taking a closer look at the functional requirements, we can determine that, to a certain extent, the needs would be based on the overall concepts that exist across different domains.

In meeting the above requirement, the concept of software stability patterns (Fayad, 2001-2002) would be researched with the support of transactions being the focus of the thesis. There are wide array of transactional requirements that also span cross across other domains. This paradigm is briefly discussed in sections 1.2.2 and is based on a new research methodology (software stability life-cycle) for building stable systems.

We will study various types of transactions that exist within the enterprise and determine a set of common requirements that are associated for transactions that can be applied across domains. The notation of transactions needs to be decomposed so that we can clearly understand their underlying infrastructural requirements and behavior through modeling. For instance, the concept of commitment that exists in today's database transaction processing systems can also exist within other contexts of transactions such as business and social contracts. Depending on the participants involved in the system, the concept of commitment in isolation can be seen to be in common between the two domains. There can be slight variations in the terms that are used to establish the same goal.

From the software engineering aspect, this would result in a very stable system and high reusability of the system across domains, with some tradeoffs. The proposed solution presented in this thesis is a ***Universal Transactions Management System***

(UTMS) model based on the Pattern Language that consists of a set of Stable Analysis and Design patterns.

Examples of problems and critical issues with existing software development methodology and related technology will be discussed in more detail in Chapter 2.

1.2.2 Software Stability Modeling (SSM) Overview

The SSM discussed (Fayad, 2001-2002) is based on conceptual thinking for modeling the solution to any given problem, relevant to the needs of the system. Due to the existence of specific concepts across domains, factors such as reusability, complexity of the problem and its context play an important role in generalizing the model for a pattern.

The initial phase would be to establish a set of objectives for the System to accomplish and support, according to the set of customer requirements needed to provide services, and functionality specific to the domain of the system. The established list of goals would be perceived as a set of patterns that exist within everyday life across different domains. Consequently, these patterns would be based on the abstraction and core elements that can exist across domains to solve one or more problems within any given scope. When required, the problem can be decomposed into smaller conceptual stable patterns to achieve better understanding of the problem, problem analysis, its boundaries and to better manage the solution space. As a result, these patterns need to be stable so that they survive for the duration of the existence of the system. Stable Analysis Patterns are commonly known as the Enduring Business Themes (EBTs).

The overall process overview of the software stability model is presented in Figure 1.1. The phases of the life cycle that cater to the development of the UTMS stable system are as follows:

1. Goals of the UTMS System
2. Capabilities of the UTMS System
3. Construction of Knowledge Map for UTMS System
4. Development of the UTMS System
5. Deployment of the UTMS System

Figure 1.1. Software Stability Life Cycle

Conducting Research

Research will be conducted through the following methods:

- Accessing IEEE/ACM Journals for papers on Software Patterns
- Mastering fundamental concepts of Transaction Processing in any context
- Mastering fundamental concepts of modeling software patterns using UML notation

1.3 Research Methodology

Thesis statement is that it is viable to build a stable software system for the overall management of transactions across the enterprise. Over the long term, this would tremendously reduce the time for developing the system and would reduce the business costs for providing various transactional services within the dynamic environment. The software stability life cycle discussed above would be used as a process model to understand and present the core elements for the development of the UTMS. All the possible transactional requirements would be determined and classified as a set of patterns that remain stable over a long period of time. The process through which the goals of the system are implemented can vary, depending on the programming languages used. Over a period of time, a new set of customer requirements is initiated for solving specific problems and to provide new functionality to meet their business needs. These requirements would be satisfied based on the existing stable system so that it is adaptable to the dynamic changes and remains stable.

1.4 Research Goals and Contributions

Our research goals are to:

1. Develop the pattern language that consists of patterns for building the Universal Transaction Management System (UTMS), based on software stability. These patterns can be deployed across any domain that involves support of the transaction processing systems.

2. Model and document the following critical Analysis and Design patterns

- AnyTransaction: This design pattern would provide a generalized solution as to what constitutes a transaction for it to be reused across any domain.
- AnyCompensation: This design pattern would explore the underlying principles as to what defines the semantics of compensation and a generalized solution would be provided for applicability across any domain.
- AnyCommitment: This analysis pattern would model the important requirements for achieving the goal of commitment across any domain. From the database transactions perspective, the notation of multi-phase commit would be studied.

3. Implement several application examples using the above patterns. The core shell of these patterns would be implemented using Java and the corresponding applications would be developed within the J2EE environment. An e-Travel Agent Service Application would be developed where all of the above patterns are used by the

application, to present the utilization of patterns for supporting the transaction based requirements.

Due to the time constraints and the scalability of the number of stable patterns that can exist for supporting all the requirements for transactions, this report would address the above patterns based on the software stability model that can be used to develop and support the requirements of UTMS.

1.5 Organization of the Thesis

Chapter 2 will discuss the fundamental concepts of transactions along with their applicability with an example. We will then compare and contrast the current problems that exist in today's transaction systems and why the existing solutions cannot satisfy the functionality of achieving software stability based on transactional requirements for any given system.

Chapter 3 will study the goals of the UTMS and its core knowledge based on the notation of Software Stability Modeling (SSM). The criteria for determining the Stable Analysis patterns would be discussed. Important Stable Analysis patterns along with examples of their applicability will be provided.

Chapter 4 will provide the capabilities of the UTMS and model the design patterns along with its analysis. We will then compare and contrast the stable design patterns along with the existing design patterns.

Chapter 5 will provide a high-level knowledge map and the architecture for the UTMS system. We will then discuss how this model can be used for constructing various sets of applications to achieve stability and reusability across any domain.

Chapter 6 will discuss the development patterns, implementation details of the pattern along with an application example, based on the underlying infrastructure of the UTMS model. We will also discuss the implementation mechanisms through the use of Industrial Objects and how then can be used within the overall development process.

Chapter 7 will discuss the deployment issues for UTMS along with the deployment patterns within the overall deployment process. We will also discuss the criteria for determining the deployment patterns along with its applicability.

Chapter 8 concludes the thesis with the discussion of the important challenges for utilizing the UTMS system with the constant change in the technology, and on overall strategy for companies to maximize the ROI and expand the paradigm of software stability modeling.

CHAPTER 2

UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS) AND DESIGN RATIONAL

2.1 Introduction

The important concepts of the transactions for their existence and applicability in different contexts of enterprise are discussed in this chapter. This is to provide an understanding of the transformations in the technology and the software processes for meeting any business needs in today's marketplace, with regard to the transactional based requirements.

2.2 Fundamental Concepts of Transactions

2.2.1 *Principals*

According to (Gray, 1993), a *transaction is a collection of operations on the physical and abstract application state*. From the definition, we can see that the existence of the transactions is to solve the problem of grouping a specific set of operations for establishing a new state. The set of operations are based on the Atomicity, Consistency, Isolation, Durability, commonly known as the **ACID** properties and need to be maintained upon establishing a new state. Essentially, all these properties are the core elements required to satisfy the behavior of transactions for any Relational Database Management System (RDBMS). The properties in the order of Atomicity, Consistency, Isolation, and Durability are discussed below in depth.

Atomicity: The transaction is executed fully or not at all (Risnes, 2000). Transaction must either execute completely based on the criteria, from its original state to a new state, or not at all. If every event within the transaction completes successfully, then the whole transaction becomes committed. The new state that is achieved at the time of commit is persistent and is permanent. If there is any type of failure during the execution of *any* of the events within the transaction, then the whole transaction needs to be backed out, to undo the changes to its original state.

Consistency: The transaction takes the accessed data from one consistent state to another (Risnes, 2000). Data consistency based on integrity constraints must be maintained throughout the transaction. The transaction may still complete successfully upon making changes to the database. However, in order to ensure that the correct data changes are made, we would need to ensure that all transactions adhere to the database rules that may have been specified upon creation.

Isolation: No transaction may see or modify the state of a resource accessed by another transaction until the other transaction has finished accessing that resource (Risnes 2000). Any resources that participate within a transaction must not be shared by other actors within the same transaction, or external to the transaction, until the event(s) associated with the resource is completed, committed and successful.

Durability: Once a transaction commits, its modifications are permanent (Risnes 2000). Once the changes to the data have been committed, these changes become permanent and can survive a system failure because they are written into a storage device.

In order to preserve the above properties throughout the transaction, additional mechanisms, some of which are complicated, are involved in controlling the execution of the transaction. These are briefly described below.

Concurrency: Based on the participants of the transactions and access to resource, concurrent execution of transactions needs to be managed well to ensure that the data consistency is not lost. This is critical within a multiple-user system environment. Isolation is one of the means through which the concurrency is achieved. The resources involved in a transaction at any particular time are left in isolation, so that no other transaction is involved in modifying the resource at the same time. In turn, isolation is achieved through the concept of locking. From another perspective, the essential reason behind the notation of isolation is to achieve protection on of the data or any other resource that is involved in the transaction. This can also be perceived as one of the objectives of maintaining concurrency.

Logging: The log manager would write/read the history of the transaction in the form of records. These records would contain information about the phases of transactions along with the data changes being made, based on the constraints specified.

Recovery: This is a mechanism through which transactions are controlled to achieve consistency in data and in the complete execution of the transaction. The concept of COMMIT and ROLLBACK are essential to track various points of consistency across the

system. However, in the web service world, this notation need not be applicable, based on its own specific set of rules.

Security: From a security aspect, traditional transactions can operate beyond a private firewall, while transaction monitors have complete control over the resources involved in the transaction. Every participant involved in the transaction either commits or rolls back the inconsistent changes.

2.2.2 Transaction Models and Applicability

Shown below is a high-level architecture along with an example of where the concept of transactions is applicable within the context of services based requirements. The overall behavior of the events within the transaction can vary tremendously depending on the context of the transaction application. From a different aspect, these can directly be mapped to the requirements and the capabilities of the system. With this in mind, new avenues exist within the subject of Software Stability Modeling and Transaction Management, to provide a better solution. These will be further discussed in the next chapter.

1. *Database Systems:*

The life span of transactions that are used in this context is usually short and events of the transaction are typically executed with *all* the ACID policies. This could range from

a single update query to a complex joint ATM transaction, whose data source is always a database system.

2. *Web Services (J2EE and .NET):*

A web service has been a new breed of Web application development (Nagapan, 2003, et al) to date. They are self-contained, self-describing, modular applications that can be published, located and invoked across the web. Web services perform functions that can be anything from simple requests to complicated business processes of automation. Once the web service is deployed, other applications can discover and invoke the deployed service. It consists of various technologies and standards that are stacked into five layers, shown below in Figure 2.1. These are Network, Transport, Packaging Description and Discovery as illustrated below. Web Service Description Language (WSDL) defines the required behavior to support synchronous or uncorrelated asynchronous interactions. Models for business interactions typically involve peer-peer message exchanges, with long-running interactions between any set of parties.

Figure 2.1. Web Services Technology Stack

As we can see, transactions within the web service environment are complex, involve access to multiple data sources, span across organizations and have long duration. Every service could involve executions of multiple transactions. These transactions are commonly known as the *extended transactions* due to the extension of the scope of the transaction. Rather than limiting the scope of the transaction to a single database system, the scope can be extended to the web that can be perceived as a large database repository that is loosely coupled. Transactions within this environment would involve automation, negotiations, contracts, exception handling and integration across enterprise. Strict ACID properties are not suitable within a loosely coupled web services distributed computing environment. Issues of persistence and concurrency are usually separated within the business transactions. Sometimes it is desirable to cancel a transaction's unit of work without affecting the remainder. Business transactions usually represent a function that is critical to the business; some of which include Supply-Chain Management, Enterprise-Resource Planning and Customer Relationship Management systems. Within the web service environment, they define the XML messaging and communication protocol bindings over other web services. The web services could contain interfaces that would provide access to structured data across systems, so that new services can be composed to support the execution of the business process through multiple transactions.

Figure 2.2. Managing Transactions within J2EE Environment
(JTA is the Java Transaction API)

The transaction that is processed between the Entity Java Bean (EJB) and the database server can be incorporated with a design pattern.

Example: Establishment of Insurance Policy for a New Car

In the Insurance policy scenario illustrated in Figure 2.3, a client application wishes to get the insurance coverage for a new car with four independent services. The rest of the services, T_5-T_n , can be anything regardless of the execution of the previous process. Transaction T_1 is concerned with finding a dealer to purchase a car, transaction T_2 is concerned with checking the customer's credit history, transaction T_3 is concerned with getting a reliable written policy from an insurance company, and transaction T_4 is concerned with adding a new car to an existing policy. If all these operations were performed within a single transactional unit of work (shown by the dotted ellipse), then the resources acquired during T_1 would not be released until the top-level transaction has

terminated. If subsequent activities T2, T3 etc. do not require those resources, then they will needlessly be unavailable to other clients. If there are any failures and concurrent access occurs during the lifetime of these individual transactional activities, issues of transaction compensation and consistency would have to be addressed independently; unlike traditional transactions.

Figure 2.3. Long Running Transaction

As we can see, the long running services and activities can be structured in multiple independent, short or long duration top-level transactions, to form a long running business transaction. The coordination of the transactional services is achieved through the message exchange of the protocols within Web Services specification.

2.3 Pattern Language for Universal Transaction Management System (UTMS)

ACID based transaction models discussed above involve transactions that are tightly coupled within a single task and executed between *trusted systems* over a short period of time. Trusted systems are systems that comprise of a special secure communication channel through which the data and its associated objects are recognized such that they are aware of the expected behavior from the external party. The overall management of data across these streams is important to ensure that the transactions are managed

efficiently. Data is reliable and is recoverable to a consistent save point, to achieve data consistency preservation. These are usually not appropriate for management of business-oriented transactions within web services. These transaction systems usually tie persistence and concurrency control together at the level of the database. With the evolution of the web based systems, Pervasive Computing (Pervasive Computing), Autonomic Computing (Autonomic Computing) and Grid Computing (Grid Computing), I would argue that *all* of the above set of properties are not required for grouping a set of operations, to establish a new state through the means of a transaction. Some or even none of the above properties would be required to satisfy the new notation of transactions. Existing DBMS systems do not support this functionality due to the limitation on their scope of applicability and their underlying architectural design.

From the patterns perspective, we can prove that the body of knowledge for understanding transactions can be captured as a core set of patterns that can be applied across systems to achieve a common goal. These collection patterns, along with their architecture and relationships can be woven together as a “whole” to create a pattern language for transactions in any context. These enduring patterns can be modeled and mapped, along with the relationship on how a specific subset of patterns are involved in meeting the transactional needs. This would maximize the reusability of the architecture and reduce the overall cost of building transactional systems based on patterns.

The ultimate goal of the *Universal Transaction Management System (UTMS)* is to manage and provide transactions based services regardless of the context of applicability within everyday life. These would be discussed in detail in the next chapter. The

architecture of the UTMS would be modeled using the pattern language for transactions. The knowledge map consisting of the architecture with major patterns of the UTMS system would be discussed in depth in Chapter 5.

Business Services and Event Driven Processes within the Transaction Processing Systems is also an important aspect within the overall domain of transactions. With the current transformation in the technology, we are coming to realize that the businesses across any domain are more inclined towards the execution and support of Services Oriented Architecture (Endrei, 2004). Businesses are driven by events that are based on a specific set of services, ranging from simple stand-alone customized enterprises to assimilation of multiple enterprises across the software globe. Examples of some services where business transactions can play a major role are: placing any type of order, receiving a shipment service, dealing with the failure of a system and making infrastructural change to an enterprise organization. The overall management of the business transactions within these business-centric processes is critical and is based on a specific service context.

Existing transaction based systems are inclined towards a specific set of tightly bound transactional properties. Unfortunately, these are not capable of supporting and handling the dynamic emerging customer transactional requirements that are based on services. This is due to the overall stringent design of the existing transaction system technical infrastructure. For example, the traditional concept of Unit-Of-Work (UOW) is restricted only to a set of operations within a transaction. If any of the operation fails, the operations are undone, to a prior point in time through the rollback mechanism.

However, the concept of UOW can also be extended to a specific service or activity, which can consist of clustered operations, to support the needs of business transactions. This can clearly be seen in the example of establishing the car insurance policy, based on a long running transaction. If any of the service requirements are not met, we are not required to undo all the other services that are involved in the single long running transaction. This scenario is described in Figure 2.3. In this context, the overall sequence of events that are executed through various means, such as queuing, can become important to ensure that the overall transactional behavior is maintained across any given enterprise. This can result in a loosely coupled stable system that is highly adaptable and whose components can be reusable across any domain.

So, the question that one should ask is: *how can we build a system using stable patterns to ensure that any type of transaction can be processed, and how to tackle the above problem across any context?* If we were to extend the existing system to support any transactional requirements across any domain, it would result in a very unstable system, more spaghetti code, excessive cost and time for development, maintenance and servicing the system. In order to answer the question, we would need to compare and contrast the existing patterns for transaction processing along with the proposed solution, to ensure its benefit and to provide an improved solution. We would conclude that the best approach is to apply the Software Stability Modeling (SSM) paradigm to develop a pattern language for the Universal Transaction Management System, to develop and manage any type of transaction processing system, independent of the domain, across the software enterprise. We will also briefly discuss the important algorithms within the

existing transaction processing systems along with their limitations in their applicability and reusability. The concept of the SSM has a lot more to offer than just being agile. The process that is involved in modeling the patterns within the pattern language would involve crisp understanding of the goals, along with the capabilities, to build a stable system.

Most of the design patterns that currently exist are not based on a generalized notation of a transaction, independent of its core properties. Rather, they are based on a specific application domain context. The ACID pattern (Grand, 2003) that is discussed by Grand is specific to the context of the database systems and is not applicable within the context of business transactions. On the other hand, the SSM based stable design pattern is capable of being extended, to be applicable within the context of business transactions. This is because the architecture of the stable pattern is based on the core knowledge that constitutes a transaction.

Another tradeoff in using the ACID transaction is that it is tightly coupled to the level of functionality that is offered by the pattern. The pattern heavily relies on the data manager component of the transaction processing system, which in fact is the primary workhorse for achieving the ACID properties. However, the low-level implementation of each of these properties within the data manager is rigid and complex, and is based neither on the concept of the separation of concerns nor the SSM paradigm, but rather on the procedural aspect. This would result in restricting the reusability of the transaction to a specific context. The solution for the modeling, the AnyTransaction stable pattern, is explained in Appendix A of the report.

Another major difference with the existing patterns is the scope in which the solution is provided. For example, the set of data access patterns (Nock, 2003) are very specific to the implementation of a specific database transaction application. These patterns are categorized as follows: Decoupling patterns, Resource patterns, Input/Output patterns, Cache patterns and Concurrency patterns (Nock, 2003). Based on the concept of SSM, these patterns can be perceived as the hooks that can be bound to the business objects, rather than the stable patterns themselves. One of the reasons being that these patterns do not contain all the core knowledge to be applicable across other transactions systems that involve access to data repository, regardless of their location and business requirements. When we have carefully examined these patterns, we would realize that these patterns can be remodeled into stable design patterns to capture the core knowledge at the abstract level of data access and transaction processing.

Within the development of a stable software system, the process patterns are an important element that constitutes the process for developing the transaction-based applications. For example, recovery is an important function that is applicable across different transaction contexts. ARIES (Algorithms for Recovery and Isolation Exploiting Semantics) (Mohan, Haderle, 1992, et al) is a classic example where the algorithms for processing the recovery of transactions are managed. These are very stringent to the specific database transaction context and are very high in complexity for them to be reused and applied across any domain within the on demand environment. The concept of structured, unstructured and semi-structured data was not standardized; rather, the data content was perceived only in the database relational and hierarchical record format.

However, this concept does not fit well with any form and type of content management such as multimedia and application services. As a result, based on the scope, these sets of algorithms are limited in providing the stability for the system as a whole. Various classes of the algorithms as process patterns can be used through the hooks to provide the desired behavior and flexibility, depending on the context and scope of any given application.

2.4 Conclusions

In this chapter we have studied the important concepts of the transactions and the various types of transactions that exist across multiple domains. We have provided an application example of the important type of transactions based on their scope and the rules that constitute the transactional behavior. We have provided the rationale as to why the SSM can be beneficial to develop a pattern language for building a stable transactional system that can be reused across any domain. We have studied the important existing design and the process patterns that are widely used in the marketplace, along with their tradeoffs.

CHAPTER 3

UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS) GOALS

3.1 UTMS General Goals

The initial phase of the Software Stability Model (SSM) would be to establish a set of objectives for the System to accomplish. These are based on a set of customer requirements, to provide services and functionality specific to the domain of any given system. The established list of conceptual goals would be perceived as a set of patterns that exist within everyday life across different domains, to promote reusability and adaptability of patterns. Consequently, these patterns would be based on the abstraction and core elements that can exist across multiple domains to solve one or more problems involving support of transaction management.

At this initial stage of the SSM, we would need to clearly understand the specific problem that is being modeled. If required, the problem can be decomposed into smaller conceptual stable patterns to achieve a better understanding of the problem, analysis of the problem, its boundaries, and to better manage the solution space for any given system. As a result, these patterns need to be stable so that they survive for the duration of the existence of the system. These are commonly known as the Stable Analysis Patterns. These are represented through the Enduring Business Themes (EBTs) and Business Objects (BOs).

3.2 Goals Analysis and UTMS Core Knowledge

In order to model and develop the stable analysis patterns, we would need to determine the Enduring Business Themes of the System. These would consist of the core elements of the problem so that we can clearly understand its necessity. The following question needs to be asked in the context of the transactions, to comprehend their true existence and to determine the EBTs.

1. *What are the reasons for the existence of the transactional systems?* These are basically the “WHATs” of the System. The following criteria should be taken into consideration during the analysis of the goals for the system to ensure that a high-quality stable system is established.

- **Stability:** EBTs are objects that should remain internally and externally stable over time upon existence of the system. They should not be limited to the core knowledge that is applicable *only* to a specific context. Instead, they should be applicable across multiple domains, in accordance with the core elements of the problem. The core elements are the essential features that constitute the EBT. Without the existence of the core elements for a given problem, the concept of endurance and applicability across any system would not exist. In the case of database transactions, the state of the database that is involved during the execution of the transaction can change over time. However, the complex process involved in establishing the new state of the database would need to remain stable.

- **Adaptability:** EBTs should be adaptable without any change to themselves, to match that of the context in which they are applied, across any system. When EBTs are defined to be stable across any system, they are bound to function with other objects, both internal and external, regardless of the changes that the system may undergo.
- **Conceptual:** In traditional software engineering processes, the explicit set of customer requirements and customer specifications that are established are not perceived on conceptual thinking. Most of the time, it is rather based on the methods and processes that are required for the system. As a result, these requirements will not directly map well with the concept of EBTs; therefore, they should not be used as EBTs of the system. EBTs are intangible objects of the system and one must have insight into the abstract representation of the customer requirements.
- **Constancy:** EBTs cannot be physically represented and cannot be replaced with another EBT. If EBTs can be physically represented, they are bound to change over time with the environment and state of the object.
- **Endurance:** The concept of enduring themes should exist for the EBT. The goal here is that the EBT should be permanent so that the same objective is served, regardless of the application domain.
- **Generality:** The EBT should be a generalized representation of a concept so that it is universal and is not bound to a specific context. This is one of the factors that help maximize the applicability.

- **Illustration:** As discussed briefly, these represent the “What’s” of the system, in order to better understand the requirements definition. These would be implemented as the specification’s classes that define the abstraction along with the set of properties that hold for the EBT.
- **Aspects:** The EBTs can be represented as an aspect of the system. The new notation of aspects has widely been utilized within the domain of Aspect Oriented Software Development (AOSD, 2004). These are various parts of a software component that are relevant to a particular concept, goal, task or purpose that are identified. However, there is a difference in the overall process being used in determining the aspects.
- **Quality Factors:** Quality factors such as the reusability, significance of using the EBT, any tradeoffs that may exist upon using the EBT in a specific domain should be carefully evaluated. These are important to ensure that the solution to the problem would remain stable.

Based on the idea of software stability and the concepts of transactions, the ACID properties can be perceived as the goals of the system. However, this is not true with all of these properties and would not meet the properties of the EBTs to achieve stability. These ACID properties are only the set of rules that are defined as a protocol for managing transactions and are needed to serve the existence of the system.

The system consists of multiple components that are specific to a context and these can methodically be modeled as patterns based on the requirements and can be used

across any system. Based on SSM and the overall existence of the system, the important EBTs that can exist for the UTMS system are as follow:

1. *High Availability*: The goal here is to ensure that the required data, service or resource is made available on demand depending on the context of the need. Across any enterprise, systems, business and services, they can function and serve their purpose based on the criticality of their availability needs. This is very critical in financial institutions such as the NYSE or banks to ensure that the systems are available 24/7 to provide services.
2. *Reliability*: The information that is made available must be valid, accurate and dependable with regard to their semantic relationship and association. Dependency is another factor that goes hand in hand with reliability. Systems can depend on specific information that is to date and is reliable to provide a service based on its existence.
3. *Performance*: Quality measurable factors, such as the rate at which the information is processed and retrieved should be met on demand. For businesses such as the stock market, the overall performance of the system is measured by determining the throughput such as the number of transactions that are processed for a given unit. The performance shown below can be modeled as the design pattern (Fayad, Pradeep, 2004). It is defined as a Business Object rather than an EBT because it is a type of measurement. The concept of the business objects and what constitutes a business

object is studied in detail in the next chapter. However from the perspective of transactions, the performance can be a very important factor for most of Real-Time Systems that rely on high throughput through transaction processing. Therefore, it would be perceived as an EBT within this report. The pattern consists of the objects AnyEntity, AnyCriteria, AnyParty and AnyMetrics. This can be mapped to the notation of database transactions. For example, during the stock market trading session, company X would like to ensure that at least 1.8 million database transactions are processed every second to ensure precision in measuring the volatility of the stock market. Here the AnyEntity is the database transactions, AnyCriteria is the 1.8 million transactions/sec, Anyparty can be a Financial Broker and the Anymetrics is the number of transactions/sec.

Figure 3.1. Performance Design Pattern (Fayad-Pradeep 2004)

4. *Scalability*: The System would need to scale well depending or based on the amount of information that exists within the dynamic nature of the business and service needs on demand. The stability pattern can be modeled as an EBT within any system (Fayad, Hamza, Sanchez, 2004).
5. *Federation/Integration*: Information that exists across different repositories is needed and would be processed through the means of transactions processing. Various types of information that can exist across different domains would need to be combined to create a new set of information that is specific to any user's needs. From the data aspect, most of the existing systems are centered towards management of the data in the format of the Relational Database Management System (RDBMS) or the hierarchal database systems. However, with the existence of various heterogeneous systems, we would need to ensure that *any* type of data is managed well. These may include multimedia content or high volume storage systems. The transactional semantics need to be developed to meet the needs within any enterprise. Additionally, from the aspect of services, the overhead in managing stream of service across an enterprise can be reduced through integration.
6. *Trust*: The level of trust that is maintained across the systems at various points in time would need to be achieved and managed well. This is required to ensure that decisions are made based on the minimum expectations from various participants of

the system. The trust pattern can be modeled as an EBT for any system (Fayad,Hamza 2004).

7. *Data Accessibility*: This is required by the system to ensure that the information that is available at any place and at any point in time is accessed and utilized when needed. The main purpose of some of the activities that are involved in the transactions would involve retrieval of the information through gaining access to their location. The Accessibility stable design pattern (Ranganath, Fayad, et al. 2003, Fayad, Ranganath 2004) is defined as the EBT and consists of the objects AnyParty, AnyMechanism, AnyEntity. This can be directly applied within the UTMS system. For example, within the database transactions domain, the user of a system can gain *Accessibility* to the database table information through the means of Querying.
8. *Security*: Access to confidential information across any system must be restricted to a specific user's needs to maintain his or her privacy. The transactions may involve interaction with multiple system resources across the enterprise, and during any system failure, we would need to ensure that the information is maintained securely. The security itself can be modeled as an EBT for any system. (Ranganath, Fayad, 2004).
9. *Intelligence*: This is required by the system to gather existing information and to infer new rules and new information, to make decisions and perform actions on

demand, based on the conditional needs. From the UTMS perspective, the idea of *when* to involve participants in a transaction and *how* to put them into action is important to ensure that the system can be self managed. This is also one of the major goals of the decision support systems that are closely related to the Artificial Intelligence space.

10. *Serviceability/Utilization*: This is required by the system to provide a breed of services based on the availability of information at any point in time. From one aspect, providing services to the customers can be perceived as a set of service based transactions with a specific set of rules. In fact, web services are one of the means through which serviceability is achieved. However, they are limited only to the context of web-based services for a specific set of domain. E.g. The notation of billing is limited to a specific application such as the utility billing, rather than generalizing it to be applied in any context. For the UTMS system and the benefit of SSM, this can be extended to provide services independent of the domain.

11. *Automation*: It would be required by the system to provide high quality throughput and reduce the burden on engineers such as system administrators and database administrators to manage transaction systems. In order for the UTMS system to provide automation, the underlying infrastructure of the whole system needs to be well-designed to ensure that interaction across any level is achieved.

The above requirements can be perceived as Level 1 (Primary) Enduring Business Themes (EBTs) consisting of goals that exist across any enterprise and context. Some of the above goals can also be applicable for a system that is entirely external to transactions. For instance, the concept of availability can also exist within the human system. Availability of energy is required for survival. However, the above EBTs are specifically determined to explore the context and purpose of serving transaction processing within any enterprise.

The next level of EBTs would consist of the important goals of the system that are *very close* in nature to the context of the domain, which in this case is the Universal Transaction Management System (UTMS). These can be perceived as the secondary EBTs of the system. In order to achieve the ACID properties of the system, the following objectives would need to be met, independent of the context. There are many other conceptual goals for the UTMS system. However, due to the limitation in time and scope, we will discuss only the important ones to address the issue of stability.

- *Commitment*: The notion of commitment can exist within any domain with regard to their applicability and existence. For example, marriage is seen as a commitment in a relationship, along with multiple other factors (trust, equilibrium) between a man and woman within the scope of their lifetime. For the UTMS system, the activities or events that are involved within the transactions need to establish a point of commitment, when an agreement has been reached with all the participants involved in the activity. After a commitment has been made, a new current state would be

established for each of the participants and the old state is lost. As we have seen from examples in the previous chapter, the idea of zero-phase commit (0PC), one-phase commit (1PC) and two-phase commit (2PC) (Gray, 1993) does not fit well within the domain of Business Process execution. Whereas, this pattern can be applied to support any required level of commitment, specific to transaction properties and the resources involved during the execution of the transaction. This can be achieved through the relationship with other analysis pattern.

- *Concurrency*: This is another major objective of the UTMS system. This is basically the ability to involve more than one participant in an activity who would need to access the same information at the same time. Within the concepts of transactions, the problem of concurrency needs to be addressed to ensure that the changes that are made to the data is trusted data and is reliable, to complement with another set of related information. The concurrency would need to be controlled to ensure that undesirable behavior is minimized and avoided if possible. This same problem can also exist in other domains. E.g. Management of air traffic control to ensure that the flights are synchronous to their schedules.
- *Recovery*: This is a critical functionality that must be supported by the UTMS system. The goal of the recovery is to establish a new dependable state. From the transaction aspect, any transaction failure can result from severe error or abnormal condition that can cause the system to end abnormally. Partially completed operations within an

activity remains pending and in an inconsistent state. Recovery of the activity is essential to perform any other operations to establish a new consistent state. Upon establishing a new consistent state, we cannot guarantee that the new consistent state would match that of the previous state. For example, in the case of database disaster recovery, we try to minimize the overall loss of the data and try to establish the closest possible consistent state. This should carefully be differentiated from the concept of *Restoration*. The main difference lies in the fact that the restoration would guarantee that the new established state would be identical to that of the previous state. However, this is not the case with recovery. The concept of recovery can also exist in many other contexts. E.g. People are recovered in a Hospital to establish a healthy state after suffering from abnormal conditions such as Flu or any kind of illness. The *Recovery* stable design pattern (Fayad, Rajagopalan 2003), (Fayad, Rajagopalan, Hamza, 2003) shown in Figure 3.2 is defined as a Business Object that is used to achieve applicability across any domain. It consists of the objects *ImparingForce*, *AnyEntity*, and *RecoveryMechanism*. This pattern can directly be applied within the UTMS system. If there is any database system crash the database may undergo recovery through the Log Apply mechanism. This mechanism involves reading the log records and performing operations and activities based on the historic data. The mechanism itself can be perceived as a transaction.

Figure 3.2. Stable Pattern for Recovery (Fayad-Rajagopalan 2003, Fayad et al, 2003)

- Logging:* The goal of logging is to keep a historic record for any activity or entity. This requirement also exists in other domains. Keeping a log of a person's activity is one example. The concept of logging can be modeled as an EBT for any system (Fayad, Ranganath, 2004).
- Monitoring:* This is required by the system to keep track of the status of different activities that are involved in any transaction. The goal here is to establish a report of

an entity that consists of the state of its current information. Monitoring also exists in multiple domains. Monitoring a person's health is one such case, whose goal is to check and make decisions based on his or her health information (Fayad, Battacharya 2004).

- *Persistence*: The preservation of the consistent state is required to ensure that valid information is processed and interpreted by the users at any point in time to provide any function and service information on demand. The criticality of maintaining persistence can vary depending on the context and the applicability. From one aspect of information management, the whole notation of database systems revolves around the concept of persistence.
- *Interoperability*: Due to the existence of a wide array of transactions within any context, all the participants of the existing and new systems and processes must be able to function together to provide the required transactional service. The goal here is to ensure that any layer of any given system has the ability to interact with each other and map the required information, so that they can operate effectively together.

Figure 3.3 shown below is a summary of the important patterns discussed above. From the figure we can contrast the overall high-level views of the important requirements within the space of transactions.

Figure 3.3. Important Goals for the UTMS System

3.3 Stable Analysis Patterns (SAP)

The concept of Stable Analysis Patterns is to apply the above process of analyzing the set of EBTs and BOs for the UTMS System from the perspective of achieving stability. BOs are another set of stable patterns that are semi-conceptual but stable and they define the underlying capabilities of the system. These are closely bound to the EBTs for modeling the solution space with the goal of achieving stability. The final model being

established would consist of the relationship between the EBT and its associated BOs, such that the generalized model would be applicable in any domain. Relationship between multiple EBTs and multiple BOs can exist within the system. The process of determining the BOs within the SSM, along with the example of the stable analysis patterns, will be discussed in depth in the next chapter

3.4 Conclusions

On one hand, the traditional transactions systems are clearly not developed based on stability and applicability across any domain. The requirements of these systems do not map well with goals of the system and its architecture. They are limited to providing specific functionality and transactional services based on the ACID properties with tangled implementation. As a result, these systems are too expensive, rigid and complex to be integrated and managed across any domain. On the other hand, an improved solution is to understand the requirements and determine the goals for the UTMS stable system using SSM model. This is based on conceptual thinking. The knowledge across various domains is taken into consideration, to provide reusability, adaptability and to be a cost effective stable system offering transactional services. The classification of the goals would help in using this solution across domain based on their properties. The goals of the system can be limited to the context of the application level, integration level or the system level. Establishment of goals for any given system is not always an easy task. However, with collaboration and knowledge from various fields is understood, a highly stable system can be modeled.

CHAPTER 4

CAPABILITIES OF UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS)

4.1 UTMS Universal Capabilities

The next step of the Software Stability Model (SSM) (Fayad, Altman, 2001) is to model the solution space design for every Enduring Business Theme (EBT) (Fayad, Altman, 2001). The solution space consists of the set of Stable Analysis Patterns. This is carried out through the means of determining the driving forces required for achieving the specific EBT requirements, with stability in mind. Having been established the WHATs of the UTMS system as the enduring themes; we would need to determine *how* (HOWs) to achieve these objectives for a stable system. As a result, importance is given to the level of abstraction and what additional set of stable patterns specific to the process and mechanisms can interact dynamically. These are commonly known as the Business Objects (BOs), and are, in essence, the workhorses of the system. These partially tangible objects would be mapped to the corresponding EBTs that are intangible. We need to ensure that the boundaries of how and when the participants of the system are engaged in resolving the problem for the associated EBT to be clearly defined, in order to model the solution.

Figure 4.1: Mapping of EBTs and BOs for Stability Analysis Patterns

Figure 4.1 shows the mapping of the EBTs and the BOs within a Stable Analysis Pattern. One or more BOs can be mapped to a single EBT depending on the scope of the applicability. The EBT object remains stable over time, while the BOs are internally stable. The properties are discussed in detail below.

The process for determining the BOs of UTMS system is based on the SSM model (Fayad, Altman, 2001), (Fayad, 2002). The following criteria should be taken into consideration for every EBT that was discussed in section 3.1. This is to ensure that the properties of the Business Objects are maintained and are modeled well within the EBTs of the UTMS system.

- **Stability:** BOs are objects that can be externally adaptable but internally stable.

These objects may interact with other business objects to meet a common goal. In fact some of the BOs may be a requirement of other BOs based on the

relationship within a given context. There may also be BOs that do not have direct relationship with the EBTs; however, these types of BOs are usually very small in number.

- **Applicability:** The scope of the applicability of the business objects should not be limited to a specific context, such that the object itself is replaceable.
- **Consequences/Tradeoffs:** The consequences of using the BOs and the EBTs should be carefully analyzed to ensure that the final stable analysis pattern is applicable across any domain, is adaptable and would not result in the system being unreliable. Any tradeoffs that may exist in using the BOs must be clearly addressed.
- **Granularity:** There can exist patterns where there is a fine distinction between the EBTs and BOs, and between the BOs and the Industrial Objects (IOs), depending on the domain and the problem environment. IOs are the application objects, basically the unstable, tangible objects that can enter and exit the stable system any time. The IOs would be discussed in detail in the next chapter.
- **Separation of Concerns:** Separation of concerns (Dijkstra, 1976) is one of the core elements of understanding the software process, within the domain of Aspect Oriented Software Development (Kiczales, 1997), (AOSD, 2004). This is another means through which the overall components of any software system can be organized and decomposed into parts that can easily be managed and reusable. There are many kinds of concerns that can exist at different points within the analysis, design, and the development phase. Various parts of a software

component that are relevant to a particular concept, goal, task or purpose are identified and can be influenced to meet a specific requirement, commonly known as aspects. This same goal can indeed be accomplished using the Software Stability Model. However, the major difference is that most of the aspects developed so far, are mostly concentrated at the implementation of the solution, rather than directly being engaged in the Design and Analysis process to solve problems across any domain. In the next section we will discuss some of the Business objects relevant to the EBTs discussed in Chapter 3. These will be compared and contrasted, to the existing aspects, where applicable. (Fayad, Ranganathan, et al 2004).

What defines a transaction?

The notation of a transaction can be perceived as a single Unit Of Work (UOW) or an activity that has been completed, to create a new state. In this report we will use the term Activity rather than the unit of work, to represent the work that has been done on behalf of a transaction. This is to generalize the notation of the set of actions being executed under a certain agreement. This is to leverage it on other contexts of work with a high-level of abstraction, such as services that can exist across an enterprise. A set of activities is based on a specific set of transactional properties to create a transaction process. This can be perceived at different levels ranging from a complex business centric action to a simple stand-alone database query.

Transaction can be modeled as a single business object, since it is the essential workhorse consisting of a set of actions that are being executed under the cover of any system, to meet the business themes discussed in section 3.1. The actions taken on behalf of the transaction may not necessarily be local to the system. Some of them can be non-local and can span across any systems. However, in order for the operations and events to be carried out within any system, we would need to clearly understand the processes based on conceptual thinking and stability that are involved in driving the operations within any transaction. These important processes can be represented as a set of patterns and would play a major role for offering greater control and flexibility for managing transactional services across any domain, and to maximize reusability and adaptability of the patterns across the transforming software globe.

Using this process would far exceed the capabilities of the traditional transactions systems such as the Java Transaction Service (JTS) (SunMicroSystems, 2004), Microsoft Transaction Server architecture (MTS) (Microsoft Corp, 2004), CORBA Object Transaction Service (OTS) (Object Management Group Inc, 2004), and the components of the database architecture (Gray, 1993) discussed by Gray.

Figure 4.2: Model of Transaction Pattern

Figure 4.2 describes the general model of a transaction Business Object. It consists of other low-level business objects that are critical to maintain the stable architecture of a transaction: These are as follows:

AnyPolicy: These are the rules under which the operations on behalf of transactions are executed. These would differ based on the context, condition and scope of the transaction and its participants. The participants can be the resource or the party.

AnyResource: Represents the participant of the transaction. In most cases, this would be the data object that is to be modified. E.g. From the database aspect, this can be perceived as the database tables or the tables themselves. This can vary depending on the context and properties of the resource. This can also represent a stand-alone resource that is involved within any other transactions, or the external members that are involved within the scope of a transaction. Depending on the participant, the participant can play various roles within the transaction. Refer to the party object of the model described below.

AnyParty: Represents the participant of the transaction. Any party such as the customer or humans themselves can also be involved within a transaction. E.g. One can refer to the case of the marriage or the case of closing a deal when buying a house. The party would have its own set of rules, within the policy, under which it would perform operations on behalf of the transaction.

AnyTransactionType: This is required to determine the context and the scope of the transaction. Transaction can exist local to the application scope, or local to a system scope, or can also be global to an enterprise scope. This represents the type of the transaction that has been defined based on the participation of the party and the resource within the scope of the transaction. Based on the type of transaction along with its properties, the level of the transaction can be determined. E.g. Existence of one or more transactions on behalf of a single transaction is commonly known as sub-transactions.

AnyTransaction: Can be perceived as an activity that is executed based on the policy, on behalf of the transaction. Based on the condition, if any agreement pertaining to the policy is met at any point in time, various actions can be triggered on behalf of the transaction. The transaction itself can also be represented as a service. One can refer to the case of the Insurance service. This would consist of a set of sub-transactions that can function as a whole, based on all its properties and context. Insurance can involve writing an Insurance policy as a single transaction.

AnyTransactionState: This represents the state of the operations that are executed within the transaction. The state of the transaction itself would also be preserved. These can be the following: {INFLIGHT, PREPARE-TO-COMMIT, INDOUBT, COMMIT, END-OF-COMMIT, ABORT, END-OF-ABORT}. These states are commonly found within the traditional database transaction processing systems such as the DB2. The new state that would need to be established after the transaction is complete would directly be dependent on the participants of the transactions along with their policy. Moreover, the current state of the transaction can govern the operations to be executed on behalf of the transaction. If the execution of any activity within the transaction is not met, it is incomplete. This would result in an incomplete and inconsistent state. As a result, with the assumption that consistency is a primary requirement of the application/system, an action would be taken to ensure that consistency is preserved. These would be defined within the policy.

AnyLog: All the operations or activities that take place under the transaction with the involvement of the participants are recorded into the Log. The log can be current or an historic log.

From the properties that define the existence of the BOs, we can determine additional BOs within the various contexts under which the requirements of the transactions are modeled.

Database Transactions and Business Processes

From the EBTs discussed in the previous chapter, we can see that some of the EBTs such as Data Accessibility, Data Integration, and Data Availability would require management of immense data across enterprises and would heavily rely on the communication across an enterprise, through means of transactions. In reality, this is true within the Distributed Systems environment. Databases can be viewed as one of the ways in which the data is organized and cataloged. From another perspective, databases can be perceived as data centers that reside at any location within the enterprise. These centers can contain structured, semi-structured or unstructured data that can play a major role with any transaction, depending on the environment. A classical example of structured data is the database tables that are represented in an organized manner. Examples of unstructured data are images and multimedia content. As a result, most of

the transaction processing systems that exist are rigid in their overall infrastructure and are restricted to the scope of managing a specific type of data within the database systems.

However, with the convergence and collaboration across the diverse enterprises, workflows and business processes are the vehicles through which the data and services can be managed in today's changing environment. With the above EBTs being some of the common goals, the capabilities provided through the various Business Objects within these contexts would need to be addressed thoroughly, independent of the domain. We need to ensure the interaction with the other Business Objects, including the *Transaction* itself, would be used to achieve the goals of the UTMS. Below are some of the important BOs that play a major role within the UTMS architecture for the level two EBTs defined in the previous chapter. The knowledge map would include a set of the EBTs and the BOs discussed up to now will be presented in the next chapter.

Enduring Business Theme: *Availability, Integration, Accessibility, and Service*

Business Objects:

AnyTransaction: Main workhorse of the UTMS system described above.

Communication with the other objects is essential for it to function as a whole and as a single object entity.

AnyWorkflow: This would represent any process that is involved in the transaction. E.g. Business execution process such as booking a trip to New York can be represented as workflow. This would consist of the elements such as reserving a flight, along with a rental car and a hotel. We would need to ensure that the existence of these elements are interrelated and would be executed based on other BOs such as *Agreement* and *Transaction*, so that none of the elements are left in isolation, resulting in inconsistency.

AnyResource: This is a specific item that is associated with a transaction or the workflow during its execution.

AnyMechanism: Represents the mechanism through which a specific activity is carried out within any process.

Enduring Business Theme: *Management*

Business Objects:

Coordination: Coordination is another important objective of the UTMS system. The coordination between one or more transactions and the participants within the transactions is needed to better manage their execution and make decisions based on the shared information available at any point on demand. As we can see, the concept of coordination can exist in social aspects of life also. E.g., in the workplace, colleagues would need to coordinate their work with each other and their manager to ensure that the

job gets done on time. The same notation also exists in many other everyday applications.

Enduring Business Theme: *Historic Log*

Business Objects:

- *AnyVersioning*: This is important to ensure that the different versions of data are stored. This can be a BO due to it being perceived as a method for storing or keeping historic data.

Enduring Business Theme: *Concurrency*

Business Objects:

- *AnyIsolation*: With regard to Input/Output being done on data, access to provide reliable and consistent information is critical to maintain the properties of transactions.

Enduring Business Theme: *Reliability*

Business Objects:

- *AnyCompensation*: The goal here is to ensure that the compensating activity or operation is done to establish the new state with similar value. E.g. People are insured to ensure that they are compensated for any loss to their assets.

Compensation transactions are transactions that are processed to establish the previous state.

Enduring Business Theme: *Commitment*

Business Objects:

- *AnyProtocol*: A set of rules under which a specific activity is performed. E.g., two-phase commit protocol is used to ensure that an agreement is reached with all the members involved in the transaction.

4.2 Stable Design Patterns (SDP)

The concept of Stable Design Patterns (Fayad, et al, 2004) is to construct the final patterns within the context of the application using the EBTs and the BOs from the perspective of stability. After the underlying stable framework is constructed, we need to determine how this can be used and where it can be used within any domain. Industrial Objects are tangible, unstable objects that can enter and exit the system, depending on the application. The BOs would be mapped to the IO using hooks (Fayad, 1998). These will be discussed in the next chapter.

Distinction between Design Patterns and Stable Design Patterns

One of the fundamental differences between the design patterns and the stable design patterns is in the strategy from which the problem is modeled, within the object-oriented programming space. Design patterns are more inclined towards the architecture and design for a specific context and domain. The popular set of patterns that have widely been used is the Gang Of Four (GoF) (Gamma, et al, 1997) pattern catalog. Every context can have its own set of patterns and it can become a challenge to integrate and reuse multiple patterns within some overlap, within a single system. Moreover, it is tightly coupled to the implementation details, rather than the purpose of the implementation. However, based on the concepts of SSM, this is not precisely the case with the stable design patterns. On the other hand, there is some similarity in the structure and documentation of the pattern. These are based on the objects, classes, aggregation and use-case relationship, which can be defined using the UML notation. Most of the design patterns that currently exist are not based on a generalized notation of a transaction, independent of its core properties. Rather, they are based on a specific application domain context. The ACID pattern (Grand, 2003) that is discussed by Grand is specific to the context of the database systems and is not applicable within the context of business transactions. On the other hand, the SSM based stable design pattern is capable of being extended, to be applicable within the context of business transactions. Another tradeoff in using the ACID transaction is that it is tightly coupled to the level of functionality that is offered by the pattern. The pattern heavily relies on the data manager

component of the transaction processing system, which, in fact, is the workhorse for achieving the ACID properties. However, the low-level implementation of each of these properties within the data manager is rigid and complex, and is based neither on the concept of the separation of concerns nor the SSM, but rather on the procedural aspect. This would result in restricting the reusability of the transaction to a specific context.

The final model being established would consist of the relationship between the EBT, its associated BOs and the IOs, so that a specific application scenario can be modeled. Relationship between multiple EBTs, multiple BOs and multiple IOs can exist within the system. The high-level knowledge map consisting of major EBTs and BOs for the UTMS system would be provided, to better understand the relationships between the EBTs and the BOs. We will provide the example of the Stable Design Patterns consisting of the IOs, to better understand their applicability and reuse, during the process of applying the SSM Model.

4.3 Conclusions

Business Objects are an important piece of the whole construct of building a stable software system. This can be perceived as a backbone of the system and maintain tangibility of the system. The rules that govern the properties of the business objects and the heuristic process through which the BOs can be located are a challenge. Based on the context of the applicability, some of the BOs can also be presented as BOs. Separation of concerns is an important concept that is widely accepted across the software enterprise. The goal for the existence of the separation of concerns within the domain of aspect-

oriented software development can be achieved using the SSM. The EBTs and the BOs can capture the core elements such as the Design, Analysis and Testing that are involved in the software lifecycle. We have discussed some of the important business objects within the transaction-processing domain. However, there are multiple facets within the transaction processing system that can span across other software components; E.g. Query Processing, Caching and Optimization. These would be briefly presented within the knowledge map for transaction processing, discussed in the next chapter.

CHAPTER 5

UTMS KNOWLEDGE MAP AND ARCHITECTURAL PATTERNS

5.1 UTMS Knowledge Map Illustration

In this chapter we will study the modeling aspect of the Pattern Language (Alexander, et al 1979) of the UTMS system. We will study various architectures of the system along with their importance to ensure that the solutions are modeled in a well-formed format and can be applied across systems and to appreciate the benefits of using the SSM modeling. These architectures contrast to some of the existing architectures within the subject of transaction processing. This will be briefly discussed. After the architectures are presented, we will illustrate examples of systems under which these stable patterns can be exploited.

The next phase of the lifecycle is to model the knowledge map of the UTMS system that can be used to generate multiple architectures within UTMS. After the Enduring Business Themes and the Business Objects for the system are determined, we need to define the overall System Architecture (SA) based on a given set of stable patterns that can be used to accomplish the goals of the UTMS, along with their applications in any context. The term architecture is very vague and the semantics of any architecture can vary tremendously, depending on the context and the level of abstraction at which it is modeled and represented. Within the SSM, architecture is perceived from the perspective of the pattern languages for providing a solution model consisting of all the EBTs and BOs that can exist within any system.

The system architecture contains the Knowledge Map (K-Map) for the UTMS system. The map consists of all the possible EBTs and the BOs that can exist within the domain of the transaction processing. This map is a subset of the large universal K-Map that can exist based on the common knowledge that is shared within the Stable Software Development niche, across other domains. Important EBTs and the BOs that have been studied in the previous chapters have their own context of applicability, independent of the domain. These are the core properties that constitute the stability. This ensures that these patterns can be deployed across any system without being tightly bound to a single domain. As a result, these patterns can be interconnected, in such a way that any route or combination of the stable patterns can be formed to establish a solution for a new system based on the capability requirements. Figure 5.1 illustrates the K-Map for the UTMS system. Patterns shown in bold are the ones presented in this report.

5.1.1 Stability patterns based on the SSM

In addition to the EBTs and the BOs, there are two other classes of the patterns. These are known as the development and deployment patterns. The applications for these patterns are specific to the different phases of the lifecycle. These are further discussed in Chapters 6 and 7.

Figure 5.1. Knowledge Map (K-MAP) of the UTMS system

The final system architecture contains the detailed-level view of the specific stable analysis patterns along with its properties. The patterns are described in the standard template format (Fayad, 2004) that was defined to ensure that the pattern is well presented and is easy to understand by the system architects, and to be able to put it to applicability. The list of patterns studied in depth is presented in Appendix A. The detail documentation of the patterns along with their applications is presented in Appendix C of the report.

5.2 UTMS Architectural Patterns

Shown below are two different application scenarios where the stability patterns from the UTMS knowledge map can be reused across different domains.

Application Scenario 1: Patient Undergoing Surgery

Enduring Business Themes: *Recovery*

Business Objects: *AnyParty, Policy, Transaction, Querying*

Description: The EBTs and the BOs involved in this scenario are shown in the shaded part of the K-Map presented in Figure 5.2. A patient is having an emergency surgery for a head injury caused by an accident. Before any surgery is performed, the doctors or the medical staff would need to perform a query on a patient's health records to determine if there are any exceptional circumstances pertaining to the patient. Moreover, they would also perform a query on the patient's insurance policy to determine his coverage and the costs involved in performing the surgery. Performing the surgery can itself be perceived

as a service based transaction. This transaction would consist of multiple sub transactions consisting of operations for retrieving patient information.

Figure 5.2. Application Scenario I within UTMS

Application Scenario 2: Customer Credit Card Rating for purchasing a Car

Enduring Business Themes: *Performance*

Business Objects: *AnyParty, Policy, Transaction, Querying*

Description: The EBTs and the BOs involved in this scenario are shown in the shaded part of the K-Map presented in Figure 5.3. A customer is planning on purchasing a car from a car dealer through a special interest free loan offer. Before the transaction on purchasing the car is complete, the dealer would perform a customer credit card

performance rating, to determine if he is eligible for the special offer. The terms and conditions for the special offer eligibility is part of the policy under which the transaction would be made.

Figure 5.3. Application Scenario II within UTMS

5.3 Conclusions

The overall SSM based architecture plays a major role in providing stability and maximizing reusability across multiple systems. From the above maps, we can see that once the foundations of the Pattern Language (Alexander, et, al 1979) for a specific domain is precisely constructed, the subset of the patterns can be reused across other domains under condition of the same problem occurrence. Moreover, there can be cases where more than one problem can exist together due to the dependency factors and the

overall problem environment. The knowledge map is a means through which we can represent the set of patterns that can share the common knowledge across any domain. From this map, every different route results in constructing a set of mini-architecture patterns that can be applicable across systems.

CHAPTER 6

DEVELOPMENT OF UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS)

6.1 UTMS Development Process Patterns

In this chapter we will discuss the issues that need to be addressed in order for the stability patterns to be put to use. This is performed through the system implementation of the patterns and is the next major step in the overall SSM (Fayad, Altman, 2001). In the last three chapters, we have discussed the core foundations of stable modeling and the properties that would need to be satisfied to construct and model a stable system. These can be perceived as the core component of the system. On top of this layer are another set of patterns that do not remain stable, but are bound to change based on the needs, without affecting the core of the system. The changes that can take place are based on the specifics of the business rules, application requirements and the environment. These are commonly known as the Industrial Objects (IOs) (Fayad, Altman, 2001). One of the essential features of the IOs is that they can have a physical representation. Some of the Business Objects can also be physically represented. However, these are very small in number compared to the physical representation of the IOs. The goal here is to establish a set of industrial objects pertaining to the specific EBTs and the BOs. Due to the existence of a wide array of applications across an enterprise, it is usually easy to determine the IOs for a given stable solution.

Figure 6.1 shows the representation of the stable system using the Industrial Objects (Fayad, Altman, 2001), Business Objects (Fayad, Altman, 2001), and Enduring Business Themes (Fayad, Altman, 2001). This is an extension of Figure 4.1.

Figure 6.1: Mapping of EBTs and BOs and IOs for a Stable System

For instance, some of the rules and mechanisms under which the transactions should execute can vary based on the requirements. For example, within the context of database transactions, a database administrator would make decisions and perform transactions on behalf of the database systems and the users of the system. These factors do not remain the same for every database administrator that can exist for a particular system and can vary by time. A person can play other roles, in addition to the administrator. The roles of the person can be seen as an industrial object because they can change based on the

requirements. Due to the operations performed within the transaction, the overall state of the transactions and the state of the participants of the transactions can constantly change and remain unstable. Therefore, all the possible states that can exist would be regarded as the industrial objects.

Due to the existence of the wide range of tangible objects that do not meet the properties of the EBTs and the BOs across the application enterprise, we need to carefully examine the scale and scope of the tangibility, their properties, and where and how they can be applicable within the periphery of the stable system. These would be essential to ensure that the requirements of the Business Objects based on the Enduring Business Themes are met. With the interaction of the various sets of industrial objects, a specific tangible behavior or mechanism can be achieved. One of the main mechanisms through which the communication between the industrial objects can exist is by the means of a process. These are commonly known as the process patterns and have *some* characteristics of both, the Business Objects and Industrial Objects. The process patterns that constitute the mechanisms can vary vastly depending on the application domain. For example, the concept of data mining can be a process in itself because it consists of various classes of algorithms through which the mining is performed, to achieve the desired results. These process patterns are closely related to the implementation details. In the next section we will study the important implementation details as to how these mechanisms can communicate with the BOs and the IOs.

Implementation Mechanisms of building Stable Analysis and Stable Design Patterns

The classic method of implementation is through using Hooks (Fayad, 1998). These structures would store information as to how and when to bind and reuse the patterns within a framework, depending on the constraints and the pre and post conditions for using the pattern. The IOs and the process patterns can be plugged and played to achieve the desired results, during which the rest of the framework based on the EBTs and BOs remains stable. The major advantage here is that these patterns can be reused with less effort, compared to constructing the patterns from scratch. The important properties of hooks are to provide clean structure, uniformity and adaptability. The three patterns described in Appendix A are implemented along with an application example.

6.2 Sample Applications of UTMS

Application scenarios

Example 1: Modeling the concept of transaction for providing travel service for a customer. This can be seen as a business transaction due to the existence of multiple sub-transactions within the reservation process initiated by the travel agent. These would be based on criteria that are met during the execution of the activities within the transactions. Figure 6.2 models the solution for the travel agent application scenario that involves a set of operations within the transactions. The properties of the IOs pertaining to the BOs are as follows:

Business Object: *AnyTransaction*

Industrial Object: *Reservation* can change based on the transactions such as Hotel or Flight reservations that are made. The main transaction itself can involve multiple other transactions to ensure that the consistency in the reservations is achieved.

Business Object: *AnyTransaction Type*

Industrial Object: *Business* is an instance of the type of transaction that is being processed. The rules are dependent on the participant of the transactions. The type of transactions can vary depending on the context. Multiple sets of transactions of various types can interact with each other, to provide a service.

Business Object: *AnyTransaction State*

Industrial Object: *In-Flight* is the current state of the transaction. This can change based on the transaction operations that are executed. In-Flight implies that the transaction is in the initial stage of modifying the resource that is part of the transaction.

Business Object: *AnyPolicy*

Industrial Object: *Criteria* can be perceived as a set of rules related to the participants of the transaction. Criteria on when to schedule the flight may be specified. Transaction can be executed based on the criteria.

Business Object: AnyParty

Industrial Object: Customer, Travel Agent can be seen as the parties of the transaction because their role can change at any given time.

Business Object: AnyResource

Industrial Object: DB Table can be changed based on the operations and the criteria for executing the transaction. The state of any given database table cannot stay constant throughout the existence of the system.

Business Object: AnyLog

Industrial Object: Receipt is one of the forms through which the execution of the operations within the transaction is complete. Other forms of log such as the system log record information on the operation being performed. However, this may be internal to the system.

Figure 6.2: Travel Agent Application Scenario

6.3 Conclusions

IOs (Fayad, Altman, 2001) also play an important role in providing applicability and reusability of the stable design patterns across the enterprise to meet the requirement of any system. These can be physically represented and cannot remain stable across the existence of the system. The implementation of these patterns is based on the attributes and the properties of the EBTs and the BO. The IOs are basically the class instances specific to the context of the application. These are described in Appendix A of the report. The main purpose of the existence of the transactions is to perform amendments to resources based on a given set of rules. As a result, it is important to understand that there is a constant change to the state of the resources on which the operations are performed. The importance is on *how* these are managed across any system, with stability in mind. The conceptual methods through which these operations are performed can remain stable using this model. Process patterns are also vital for achieving the goals of the SSM, due to the fact that correct types of the process patterns need to be used depending on the criticality of the overall management of information within the context. This is to provide a universal solution, and can be used to support any type of transactions.

CHAPTER 7

DEPLOYMENT OF UNIVERSAL TRANSACTION MANAGEMENT SYSTEM (UTMS)

7.1 Deployment Issues in UTMS

Software stability model (SSM) (Fayad, Altman, 2001) is not over once the development is complete. The next phase of the SSM would involve managing and addressing the problems that arise during the deployment of the stable patterns that have been developed. In fact, it has been proved that the deployment phase itself can take more time than the development phase. Therefore, is important for us to understand how to manage these patterns so that they can be readily plugged and played through automation and put to use into the production systems across the enterprise to meet the original requirements. Some of the questions that need to be addressed within the deployment phase are as follows:

1. How can we validate and verify that the patterns that have been developed would satisfy the requirements of the system when used in production?
2. How can the composition of various patterns be achieved to ensure that, regardless of the route taken within the knowledge map, a stable system is yet achieved?

3. Can the patterns be used simultaneously across multiple existing and new systems, to ensure that no abnormalities are found while in use? Could abnormalities result in additional time and cost for the organization? This issue is especially true for transactions where timing is a critical factor. In Real-Time Systems, such as Health Care systems, any failure within a micro second space can cost the company billions of dollars. People can learn from mistake; however, there is also a level of toleration that is unacceptable. How can we make sure that this is addressed with care?
4. Who should be involved in the deployment of the patterns and what are the roles and responsibilities of the participants within this phase?

Based on the above concerns that can exist within the deployment space of any software component, we would need to determine the Enduring Business themes (EBTs) (Fayad, Altman, 2001) and the Business Objects (Fayad, Altman, 2001) that are based on the goals of the deployment. Some of these EBTs and BOs can also exist across different contexts within the universal knowledge map. Using these patterns can save tremendous cost and time for the organizations to maximize their return of investment. These are commonly known as the quality patterns. The important EBTs and BOs for the deployment of the UTMS system are discussed in the next section.

7.2 Deployment Patterns

- *Testability*: One of the main goals of testing is to ensure that results or behavior of the system would match that of the expected results, based on a set of constraints. In a typical application environment, the tester would be involved in creating various test case scenarios, based on the available knowledge at any given point in time. However, capturing every test combination can become cumbersome and entail high overhead in the overall deployment process.
- *Reusability*: Reusability of the patterns is the core requirement of the deployment. We would need to ensure that the patterns would fit well within the context of the application.
- *Portability*: This is required to ensure that the expected behavior from the system is the same regardless of its location and platform under which these patterns are deployed. Especially, within the transaction systems, the resources and the participants can exist anywhere and it is important to ensure that the behavior from either end is the same. For example, replication is one of the common means through which minimum recoverability of any system is achieved.
- *Fault Tolerance*: This is one of the critical goals of the transaction processing systems. Any abnormal or unexpected failures of the system would need to be handled with care, to ensure that there is no loss in the information about the resources or other participants within the transaction.

- *Coordination*: Coordination between the participants of the deployment phase is needed to ensure that the expected behavior is met. Based on the other goals such as portability, the participants can be local or global. This is directly dependent on the context in which the application is to be used.
- *Maintainability*: This is one of the main goals for the existence of the system. Based on the experience and trend within the software enterprise, every software component that has been developed is not one hundred percent bug free. Till the existence of the system is perfected, we need to ensure that it is maintained well.
- *Adaptability*: When the reusability of the patterns is achieved, we need to ensure that these patterns would be able to adapt to the constant change in the application requirements and rules.
- *Scalability*: The System would need to scale horizontally, vertically, up or down, depending on the amount of information that exists within the dynamic nature of the business and service needs on demand.
- *Reliability*: The information that is made available must be valid, accurate and dependable with regard to their semantic relationship and association. Dependency is another goal that goes hand in hand with reliability. Systems depend on specific information that is reliable to provide a service, based on its existence.

- *Installation:* This is one of the first steps for putting the system to use. The specific set of system requirements are configured by the user of the system, regardless of their domain.

7.3 Conclusions

Determination of the deployment patterns is also an important part of the SSM. This is needed to ensure that all the original system requirements are met. The deployment patterns can be used independent of the domain and deployment centric stable analysis patterns can overlap with the stable patterns that also exist for the UTMS system.

CHAPTER 8

SUMMARY & FUTURE DIRECTION

8.1 Conclusion

In this thesis, we have researched the current problems that exist within the current transaction processing systems and the strategy through which these systems have been implemented within the software engineering arena. We have discussed the current drawbacks and their limitations in functionality, with regard to their application domain, based on the customer requirements, system analysis, system design and implementation. With the concepts of the transaction processing, we are aware that the existence of the software and business services heavily relies on the management of transactions across the heterogeneous environment to maximize their return of investment. These are managed based on the specific set of rules, behavior and the context of the application. Therefore, the transaction systems must be capable of managing any type of transactions across the enterprise. Overall, modeling and developing a single system is a challenging goal due to the complexity and the profundity of the domain knowledge.

The pattern language (Coplien, et al) for the development of the Universal Transaction Management System (UTMS) within the software stability model (Fayad, Altman, 2001) provides a unique approach through which various types of transactions across any domain can be managed. This consists of uncovering the important stable analysis patterns and stable design patterns, to build a solid stable framework (Fayad, Altman, 2001) on which the applications can be built. We have studied the process that is

involved in determining these patterns, along with the properties, to understand how the goals of the UTMS system are achieved. It is important to realize that the overall behavior of the transactions cannot be achieved in isolation of the participants of the transactions. Rather, they function as a whole in harmony and orchestration across the heterogeneous environment. Stable patterns based on these goals can fit well within the UTMS system. The knowledge map that was discussed in chapter 5 is specific to the important patterns that would exist within the transaction processing system. This is a subset of a global knowledge map that is devised for multiple domains.

We have shown the proof of concept of the above solution by implementing and documenting the three important patterns along with two unique sample applications that exploit these patterns. We have studied the importance of using hooks (Fayad, 1998) as a means through which the Business Objects (Fayad, Altman, 2001) and the Industrial Objects can be woven during the implementation phase. We compared and contrasted the existing process pattern such as the ARIES algorithm for implementing the Recovery pattern and to understand the tradeoffs in complexity and stability with the existing solutions. We have also studied the important patterns within the development and the deployment phase of the software stability lifecycle. List of patterns that are developed and documented are as follows:

- Pattern Name: *AnyTransaction*
Pattern Type: *Stable Design Pattern (BO)*
- Pattern Name: *AnyCompensation*
Pattern Type: *Stable Design Pattern (BO)*

- Pattern Name: *Commitment*

Pattern Type: *Stable Analysis Pattern (EBT)*

Lastly, an area of possible improvement is in the ease of use for deploying the UTMS model. The deployment pattern, such as the usability, must be carefully studied and tools can be developed to ensure robust and efficient interfaces with less overhead. Fulfilling these enhancements, we would expect that the Software Stability Modeling paradigm would play a major role in the years to come, to provide fruitful solutions to challenges across the software engineering niche.

8.2 Future Direction

Modeling Stable Analysis and Design Patterns (Gamma, et al, 1997) from a blend of different subjects such as Artificial Intelligence, Sensor Networks, Data Mining and Data Warehousing and using them within the domain of transaction processing, would far surpass the solution space, and would provide a greater advancement in building any type of intelligent transaction applications. Within the Autonomic Computing Environment (Autonomic Computing), the idea of self-healing, self-descriptive and self-recovery can be adapted within the UTMS framework.

With the concept of the SSM, we can see that the majority of systems and their processes depend on the existence of specific core knowledge requirement. This can vary based on the domain of applicability. The UTMS system can be extended in the future to

manage the transactions based on the semantic knowledge that is involved during the transaction processing.

SSM Modeling can be expanded to better model the solution to problems specific to the Service Oriented Computing space and Business Processes management. This can consist of sets of transactions and processes that can remain stable over time.

Moreover, the concept of the Enduring Business Themes (Fayad, Altman, 2001) and Business Objects (Fayad, Altman, 2001) can be extended to fit well within the Service Oriented Computing (Endrei, 2004) arena. The notation of reusability can be extended to the scope of services, so that services that remain stable over time can be modeled as stable service patterns.

REFERENCES

- Alexander, C., Ishikawa, S., Silverstein, M., A Pattern Language: Towns, Buildings, construction (Center for Environmental Structural Series), 1977, Oxford, Oxford University Press.
- AOSD, *Aspect-Oriented Software Development*, information available at <http://www.aosd.net>
- Autonomic Computing, Autonomic Computing, information available at <http://www.research.ibm.com/autonomic/>
- Boehm, B. (1988) A Spiral Model of Software Development and Enhancement, IEEE Computer. 21 (5), 61-72.
- Coplien, O. J., Schmidt, D. *Pattern Languages of Program Design*, 2002
- Dijkstra, E.W., (1976), *A discipline of programming*, Prentice Hall
- Endrei, M. (2004). *Service Oriented Architecture and Web Services*. IBM Redbooks
- Fayad, M. *Accomplishing Software Stability*, Communications of the ACM, Volume. 45, No. 1., 2002
- Fayad, M. (2002), *How to deal with Software Stability*, Communications of the ACM, Volume. 45, No. 4., 2002
- Fayad, M. (2004), *Stable Analysis Patterns: Stable Analysis Pattern Documentation*, Chapter 2.
- Fayad, M., Altman, A. (2001). *An introduction to Software Stability*, Communications of the ACM, Volume. 44, No. 9., 2001
- Fayad, M.E., Battacharjee, R. (2004). Patterns for Real-Time Systems, MS Project, Computer Engineering, San Jose State University.
- Fayad, M.E., Hamza, H., Rajagopalan, J. The Recovery Design Pattern.” The 2003 IEEE International Conference on Information Reuse and Integration, Las Vegas, NV, October 2003
- Fayad, M.E., Hamza, H., Sanchez, H. Toward Scalable Software Architectures. IEEE IRI-04, Oct. 2004

- Fayad, M.E., Pardeep, R. (2004). Pattern Language for Performance Evaluation. Sugerloaf PLoP 2004, Brazil.
- Fayad, M.E., Rajagopalan, J. The Applications of Recovery Pattern. Sixth International Conference on the Unified Modeling Language (UML 2003), Workshop on Stable Analysis Patterns: A true problem understanding with UML, workshop #8, San Francisco, CA, October 20-24, 2003.
- Fayad, M.E., Rajagopalan, J., Ranganath, A. The Accessibility Analysis Pattern. Sixth International Conference on the Unified Modeling Language (UML 2003), Workshop on Stable Analysis Patterns: a true problem understanding with UML, workshop #8, San Francisco, CA, October 20-24, 2003.
- Fayad, M.E., Ranganathan, A. (2004). Stable Design Pattern, MS Project, Computer Engineering, San Jose State University.
- Fayad, M.E., Ranganath, A., Pinto, M., "Two Rationale Driven Approaches to Achieve Stability in Software". *Communications of the ACM* (Submitted March 2004)
- Fayad, M.E., Schmidt, D., Johnson, R., *Building Application Frameworks. Object Oriented Foundations of Framework Design*. Chapter 9 Reusing Hooks, 219-236, Wiley & Sons: New York, 1998.
- Fayad, M.E., Wu, S. (2002, September). Merging Multiple Conventional Models in One Stable Model. *Communications of the ACM*, 45 (9), 102-106.
- Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison Wesley, 1997.
- Grand, M. (2003). *Patterns in Java: Design Patterns for Enterprise and Distributed Applications*, Volume 3, Wiley.
- Gray, J. (1993). *Transaction Processing: Concepts and Techniques*, Morgan Kaufman.
- Grid Computing, *Grid computing*, information available at <http://www.gridcomputing.com/>
- JTS, *Java Transaction Services*, information available at <http://www.java.sun.com/products/jts/>
- JTA, *Java Transaction API*, information available at <http://www.java.sun.com/products/jta/>

- Kiczales, G., et al., (1997), *Aspect-Oriented Programming*, Proceedings of the 1997 European Conference on Object-Oriented Programming, Finland, June 1997, Springer Verlag LNCS 124
- MDA, *Model Driven Architecture*, Object Management Group (OMG), information available at <http://www.omg.org/mda>
- MTS, *Microsoft Transaction Server*, information available at <http://www.windowsitlibrary.com/Content/405/20/toc.html>
- Nagappan, R., Skoczylas, R., Sriganesh, R.P. *Developing Java Web Services: Architecture and Developing Secure Web Service using Java*, 2003.
- Mohan C, Haderle. D, Lindsay. B, Pirahesh. H, Schwarz. P. *ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging*. ACM Transactions on Database Systems, 17(1):94–162, March 1992.
- Nock, C. *Data Access Patterns: Database Interactions in Object-Oriented Applications*, Addison Wesley Publishing, 2003
- OTS, The Object Transaction Service, information available at <http://www.iona.com/support/whitepapers.OrbinxOTS.pdf>
- Pervasive Computing, *Center for Pervasive Computing*, information available at <http://www.pervasive.dk/>
- Ranganath, A., “Software Stability Patterns and Aspects,” *Master Project*, San Jose State University, May 2004
- Risnes, Andre L. *Asynchronous Lock Distribution in the New File Repository*, MS Thesis, Computer Science, University of Tromsø, 2000

Appendix A: UTMS Stable Patterns List

1. Stable Analysis Patterns (EBTs)

- Commitment Pattern

2. Stable Design Patterns (BOs)

- AnyTransaction Pattern
- AnyCompensation Pattern

Appendix B: Documented UTMS Stable Patterns

ANYTRANSACTION STABLE DESIGN PATTERN

Abstract

The traditional concept of a database transaction is purely based on the Atomicity, Consistency, Durability and Consistency (ACID) properties. However, these are properties do not hold in the case of the other transactions which exist in other domains. The concept a transaction can be perceived different based on the context of the application within the enterprise. Problem here is that basic structure of the transaction can be the same, however, the properties under which the transaction exists can vary. Therefore, why do we need to duplicate the effort of building transaction-processing systems that share similar functional and non-functional properties, based on the context? The Transaction stable design pattern would model the solution to the problem based on the concept of Software Stability Model (SSM), such that it can be utilized across any domain, to maximize the reusability and productivity of building stable transaction processing system across any domain.

Index Terms – software stability patterns, stable design pattern, stability analysis patterns, enduring business themes, business objects, transactions and database.

I. INTRODUCTION

According to Gray, a *transaction is a collection of operations on the physical and abstract application state* [1]. From the definition, we can see that the existence of the transactions is to solve the problem of grouping a specific set of operations that can span across an enterprise, for establishing a new state. The set of operations are based on the Atomicity, Consistency, Isolation, Durability, commonly known as the ACID properties, and these properties need to be maintained upon establishing a new state. Essentially, all these properties are the core elements required to satisfy the behavior of transactions for any relational database system (RDBMS). However, these properties do not hold true for the other types of transactions such as business transactions, long running transactions, nested transactions and service-oriented transactions. Each one of these transactions is based upon their own set of rules, ranging from business rules to business intelligence, within the on demand-computing environment. In this paper, we will study the notation of transactions and model a generalized solution of the pattern at an atomic level of transaction; smallest scale possible, so that the important issues can be understood from the SSM aspect. Below, we will briefly compare and contrast some of the existing related transaction patterns that currently exist. In traditional database systems, life span of the transactions is usually short and the operations within the transaction are typically executed with *all* the ACID policies. This could range from a single update query to a complex joint ATM transaction, whose data source is always database systems.

On the other hand, due to the recent shift towards the web services and service oriented computing environment, more and more services are dependent on the overall execution of the business process for achieving automation. These can involve transaction with

loosely coupled properties, such that they can span across multiple enterprises for a longer duration. From carefully examining these services, we can see that underneath these services, a set of operations through multiple sets of transactions can be involved. One of the main differences with the traditional RDBMS is that, these transactions can vary in their granularity and can span across multiple enterprises, based on the application context. For example, a travel agent service can consist of a set of transactions such as a flight reservation, a hotel reservation, etc.

Most of the design patterns that currently exist are not based on a generalized notation of a transaction, independent of its core properties. Rather, they are based on a specific application domain context. The ACID pattern [2] that is discussed by Grand is specific to the context of the database systems and is not applicable within the context of business transactions. On the other hand, the SSM based stable design pattern is capable of being extended, to be applicable within the context of business transactions. Another tradeoff in using the ACID transaction is that is tightly coupled to the level of functionality that is offered by the pattern. The pattern heavily relies on the data manager component of the transaction processing system, which in fact is the workhorse for achieving the ACID properties. However, the low-level implementation of each of these properties within the data manager is rigid and complex, and is based neither on the concept of the separation of concerns nor the SSM, but rather on the procedural aspect. This would result in restricting the reusability of the transaction to a specific context.

II. SOFTWARE STABILITY & STABLE DESIGN PATTERNS: BACKGROUND

Software Stability Model (SSM) [3] would be to establish a set of objectives for the System to accomplish. These are based on a set of customer requirements, to provide services and functionality specific to the domain of any given system. The established list of conceptual goals would be perceived as a set of patterns that exist within every day life across different domains, to promote reusability and adaptability of patterns.

Consequently, these patterns would be based on the abstraction and core elements that can exist across multiple domains, to solve one or more problems involving support of transaction management.

These patterns need to be stable such that they survive for the duration of the existence of the system. These are commonly known as the Stable Analysis Patterns. These are represented through the Enduring Business Themes (EBTs) and Business Objects within the core stable system and they represent the knowledge logic. Another set of patterns that do not remain stable, but are bound to change based on the needs, without affecting the core of the system. The changes that can take place are based on the specifics of the business rules, application requirements and the environment. These are commonly known as the Industrial Objects (IOs) [3]. One of the essential features of the IOs is that they can have a physical representation and represent the application logic. The goal here is to establish a set of industrial objects pertaining to the specific EBTs and the BOs. Transactions would be the main focus of the domain in this paper.

III. PATTERN DEVELOPMENT

PATTERN NAME: AnyTransaction

PROBLEM: How can we model and represent the notation of a transaction so that it can be used across any domain based on the properties that constitute its core knowledge? This problem can exist in isolation, and can well be plugged into any application, to function as a universal solution, independent of the domain. The main factors that are involved within notation of any transaction are as follows:

1. *Persistence:* Regardless of the domain of applicability, we need to ensure that the overall transformation of the state of the transaction itself and the state of the resources that are involved in the transaction is managed.
2. *Types of operations:* The operations or events that are executed or involved within transaction need to be understood based on their scope and the state changes that are made within the transaction. This activity or operation that is performed is directly based on the participant of the transaction. Especially, within the distributed environment, the participants within the transaction can be local or remote with relation to their environmental scope.
3. *Rules & Properties:* Rules can be defined based on the specific operations within the transaction. For traditional database systems, we can perceive this as the ACID properties that determine the overall behavior of the transaction. These can greatly vary, depending on the context of the application and the available knowledge at any point in time. Rules can also be defined based on the participants of the transaction.
4. *Timing:* Based on the transaction policy, some the application would heavily rely on the timing factor to ensure that the transaction throughput is met. This would be based on the priority and the sequence of the operations that are executed, based on the available information at any point in time. This is required to ensure that the transaction can complete successful.

CONTEXT: The concept of transactions is very important and is required to ensure that the needs of any system based on their requirements are met. Transactions are also critical within the domain of the Information Integration, and Data Management. A set of transactions can be composed into a single service and various types of services can exist within an enterprise. With reference to the problem context, we will perceive transactions as a set of operations to manage their state at any point in time.

FORCES: Transaction pattern should resolve the following forces:

- The overall required behavior of any transaction is dependent on the properties under the operation within the transaction are to be executed. The design patterns must be able to manage the behavior of the transactions based on these properties because these can vary depending on the application context.
- Based on the application requirement, a transaction itself can consist of multiple sub-transactions. This is especially true for the case of the long-running transactions. The

- If there are any failures during the execution of any operations within the transaction, we need to ensure these cases are taken into consideration so that the final result can still be accomplished. Additional information can be referenced in the Compensation Pattern within the appendix section.
- Greater flexibility in the overall management of the data and services is required to ensure that correct decisions are made based on the overall execution various types of transactions within any domain.

```

classDiagram
    class EBTs {
        <<Pattern-EBT>> Service
    }
    class BOs {
        <<Pattern-BO>> AnyTransaction Type
        <<Pattern-BO>> AnyParty
        <<Pattern-BO>> AnyTransaction
        <<Pattern-BO>> AnyPolicy
        <<Pattern-BO>> AnyTransaction State
        <<Pattern-BO>> AnyResource
        <<Pattern-BO>> AnyLog
    }
    EBTs --> BOs : based on
    EBTs --> BOs : part of
    BOs --> BOs : preserves
    BOs --> BOs : engages in
    BOs --> BOs : engages in
    BOs --> BOs : engages in
    BOs --> BOs : preserves
    BOs --> BOs : engages in
    BOs --> BOs : composes
  
```

Figure 1: AnyTransaction Design Pattern

1. Patterns

- *Service*: This represents the activities that are performed on behalf of one or more transactions.
- *AnyTransaction*: Represents the set of operations that are executed within the transaction.
- *AnyTransaction Type*: Represents the type of the transaction that has been defined based on the participation of the party and the resource within the scope of the

transaction. Based on the type of transaction along with its properties, the level of the transaction can be determined.

- *AnyTransaction State*: The current state of the operations that are executed within the transaction. These can be the following: {INFLIGHT, PREPARE-TO-COMMIT, INDOUBT, COMMIT, END-OF-COMMIT, ABORT, END-OF-ABORT}. Any of these states can exist across any type of a transaction. The new state that would need to be established after the transaction is complete would directly be dependent on the participants of the transactions along with their policy.
- *AnyPolicy*: Represents the specific set of rules for the party and the resource that is involved in the transactions so that the required behavior can be achieved, based on the policy defined by the application.
- *AnyParty*: Represents the participant of the transaction based on its policy
- *AnyResource*: Represents the participant of the transaction based on its policy
- *AnyLog*: All the operations or activities that take place under the transaction with the involvement of the participants are recorded into the Log. The log can be current or an historic log. More information on this pattern can be found in [4].

Figure 2: Sequence Diagram for the AnyTransaction Design Pattern

CRC-Cards

Service – EBT (Provider)		
Responsibility	Collaboration	
Describes the transaction based service requirements	Clients	Server
	<i>AnyTransaction</i>	requestService() provideService() serviceInfo() serviceproperties() terminateService()

AnyTransaction– BO (Activity)		
Responsibility	Collaboration	
Execute operations or events within the Activity on be-half of the participants and resources.	Clients	Server
	<i>Transaction Type</i>	defineTranxType()
	<i>AnyTransaction State</i>	defineTranxState()
	<i>AnyParty</i>	createTranx()
		startTranx()
		endTranx()
		cancelTranx()
		performActivity()
		backoutTranx()
		modifyResource()
	<i>AnyResource</i>	grant()
		lock()
		isConsistent()
	<i>AnyLog</i>	logTranx()
		provideTranxLog()

AnyTransaction Type- BO (Classifier)		
Responsibility	Collaboration	
Describe the type of the transaction	Clients	Server
	<i>AnyTransaction</i>	determineType() modifyType() classify()

AnyTransaction State – BO (Reporter)		
Responsibility	Collaboration	
Indicates the state of the transaction	Clients	Server
	<i>AnyTransaction</i>	modifyState() currentState() previousState()

AnyPolicy– BO (Policy Provider)		
Responsibility	Collaboration	
Define rules and criteria for executing operations based on the resource and the party	Clients	Server
	<i>AnyParty</i>	definePolicy() verifyRules() validateRules() checkConsistency() defineResourceConstrantInfo()
	<i>AnyResource</i>	

AnyParty– BO (Participant)		
Responsibility	Collaboration	
Participant of the Transaction	Clients	Server
	<i>AnyTransaction</i>	startTranx() joinTranx() closeTranx() modifyTranx monitorTranx() permit()
	<i>AnyPolicy</i>	

AnyResource– BO (DataStore)		
Responsibility	Collaboration	
Participant of the transaction	Clients	Server
	<i>AnyTransaction</i>	resourceTranxInfo() locateTranxResource() modifyResource() verifyResource()
	<i>AnyPolicy</i>	

AnyLog– BO (Database)		
Responsibility	Collaboration	
Keep a record of the operations along with the transaction state information	Clients	Server
	<i>AnyTransaction</i>	writeLog() readLog() startLogTime() endLogTime() currentPointOfLog()

Figure 3: CRC Card Representation of the AnyTransaction Design Pattern

Consequences

The use of the Transaction pattern has the following benefits:

1. *Generalization*: The notation of transaction can be generalized using this model so that it can be used in any application scenario with variation in the properties. Multiple classes of transactions can exist within any business process and using this transaction model would allow greater flexibility in gaining control on the

execution of multiple types of operations within any transaction, across the enterprise.

2. *Adaptable*: The stable design pattern model can be used along with the other stability patterns so that it can fit well within any context.
3. *Simplicity*: This model is presented in a simple form so that it can be well understood and can be stable over time and is independent of the external factors that can be intangible. In the traditional database transaction processing systems, many factors are taken into consideration at any given point in time, resulting in a complex system.

The use of the Transaction pattern has the following limitations:

1. *Scope and Control of the operations*: There can be cases where some transactions are irreversible. As a result transactional-based services such as restoration can be limited. As a result, these operations cannot be controlled after any unknown failures or abnormal behavior.

IV. APPLICABILITY

Case Study 1: Travel Agent Services

Modeling the concept of Transaction for providing Travel service for a customer. This can be seen as a business transaction due to existence of multiple sub-transactions within the reservation, initiated by the travel agent. These would be based on the criteria that met during the execution of the activities within the transactions. Figure 6.2 models the solution for the Travel agent application scenario that involves a set of operations within the transactions. The properties of the IOs pertaining to the Business Objects are as follows:

Figure 4: Stable model for Travel Agent Service

Business Object: *AnyTransaction*

Industrial Object: *Reservation* can change based on the transactions such as Hotel or Flight reservations that are made. The main transaction itself can involve multiple other transactions to ensure that the consistency in the reservations is achieved.

Business Object: *AnyTransaction Type*

Industrial Object: *Business* is an instance of the type of transaction that is being processed. The rules are dependent on the participant of the transactions. The type of transactions can vary depending on the context. Multiple sets of transactions of various types can interact with each other, to provide a service.

Business Object: *AnyTransaction State*

Industrial Object: *In-Flight* is the current state of the transaction. This can change based on the transaction operations that are executed. Other possible states with accordance to time can be PREPARE-TO-COMMIT, INDOUBT, COMMIT, END-OF-COMMIT, ABORT, END-OF-ABORT.

Business Object: *AnyPolicy*

Industrial Object: *Criteria* can be perceived as a set rules related to the participants of the transaction. Specific criteria on when to schedule the flight may be specified.

Business Object: *AnyParty*

Industrial Object: *Customer, Travel Agent* can be seen as the party of the transaction because their role can change at any given time.

Business Object: *AnyResource*

Industrial Object: *DB Table* can be changed based on the operations and the criteria for executing the transaction. The state of any given database table cannot stay constant throughout the existing of the system.

Business Object: *AnyLog*

Industrial Object: *Receipt* is one of the forms through which the execution of the operations within the transaction is complete. Other forms of Log such as the system log within information on the operation being performed. However, this may be internal to the system.

Case Study 2: Multi-Database Update

Modeling the concept of Transaction for making performing a multi-database update query that involves access to multiple data sources across an enterprise. These are the partitioned dataset, a word file and the XML file. The operations within the transaction would take place based on the criteria set between the marketing analyst and the associated rules for performing the update.

Figure 5: Stable model for Multi-Database Update Transaction

Business Object: *AnyTransaction*

Industrial Object: *Update Query* statement can change based on the data source that is currently being modified. E.g., There can be multiple statements such as SELECT, JOIN, MERGE etc within the main transaction. Any these transactions span across the distributed environment. E.g., Applications of information integration.

Business Object: *AnyTransaction Type*

Industrial Object: *Multi-Database* is an instance of the type of transaction that is being processed. The rules are dependent on the participant of the transactions. The type of transactions can vary depending on the context and the current type of transaction that is processed. Multiple sets of transactions of various types can interact with each other, to provide a service based on the Update Query.

Business Object: *AnyTransaction State*

Industrial Object: *In-Commit* is the current state of the transaction. This can change based on the transaction operations that are executed. Other possible states with accordance to time can be PREPARE-TO-COMMIT, INDOUBT, COMMIT, END-OF-COMMIT, ABORT, END-OF-ABORT.

Business Object: *AnyPolicy*

Industrial Object: *Criteria* can be perceived as a set rules related to the participants of the transaction. Specific criteria or condition under which the update would take place.

Business Object: *AnyParty*

Industrial Object: *Marketing Analyst*, can be seen as the party of the transaction because their role can change at any given time. Any party can initiate the transactional request.

Business Object: *AnyResource*

Industrial Object: *Word File, XML File* can be changed based on the operations and the criteria for executing the transaction.

Business Object: *AnyLog*

Industrial Object: *Log Dataset* is a type of file that is generated for keeping track of the changes that have been made to the files within the scope of the transactions.

RELATED PATTERNS

AnyEvent

AnyMedia

AnyCompensation

Commitment

IV. CONCLUSIONS

In this paper we have provided a generalized model of AnyTransaction to maximize their applicability across any domain to maximize stability and reusability. With this mode, we understand the main existence and the behavior of what constitutes a transaction based on multiple factors across different domains. This pattern can be used with other patterns within the context of any domain and any transaction processing system through UTMS. We have provided two general applicability case studies where this pattern can be applied.

References:

- [1] Gray, J. (1993). *Transaction Processing: Concepts and Techniques*, Morgan Kaufman.
- [2] Grand, M. (2003). *Patterns in Java: Design Patterns for Enterprise and Distributed Applications*, Volume 3, Wiley.
- [3] M.E. Fayad and A. Altman. *Introduction to Software Stability*, *Communications of the ACM*, Vol. 44, No. 9, Sept. 2001, pp. 95-98.
- [4] M.E. Fayad, J. Rajagopalan, A.Ranganath. *AnyLogStable Design Pattern*, June 2003

ANYCOMPENSATION STABLE DESIGN PATTERN

Abstract

The act of compensating or the state of being compensated is a problem that exists across multiple transaction processing application domains within any enterprise. The Compensation pattern would model the solution to the problem based on the concept of Software Stability Modeling (SSM), such that it can be utilized across any domain, to maximize the reusability and productivity of building stable transaction processing system across any domain.

Index Terms – software stability patterns, stable design pattern, stability analysis patterns, enduring business themes, business objects, transactions, database and compensation.

I. INTRODUCTION

Compensation is a main concept that is commonly used within the context of business and social aspect. However, it can also be applicable across multiple other domains such as the transaction processing. Compensation can be perceived as a process in which as tendency for a change in a given direction if counteracted by another change so that the original change is not apparent. Three different cases will be studied below so that we can understand the notation and can model is as a single solution.

One of the context under which compensation can come to mind is the worker's compensation or health compensation. These are types of compensations where if there is any injury or failure while the person is involved in doing their job, some compensating action is taken to make up for the loss or equivalence in the value. These can be services such as money given or received in the form of payment or reparation. The services can consist of one or more different types of transactions. The transaction pattern is discussed in depth in next section of the report.

On the other hand, the same idea can also be seen in the area of transaction processing systems. If there is any failure in any of the operations within a transaction, we need to ensure that the changes made to the point of the failure are not visible and would not be left in inconsistent state. As a result, those changes are carried out through compensating activities such as undoing the operation to the previous point of consistency. This is the classical method that is widely used across different type of transaction processing systems. Other than through relying on the undoing operations, there can also be other methods where the original state can be restored.

Finally, the concept of compensation can be applicable regardless of the failure or abnormalities at any point in time. E.g., Person may be a hard worker or has done something that is positively extraordinary. In this case, the person would be rewarded with compensation in the form of cash incentives or bonus. This is not perceived as a loss

or a makeup; rather, it is a mechanism through the original state of being significant is maintained.

The three cases presented here are entirely different in their types. However, we can see that the goal based on the concept of compensation along with their separation of concerns here are common. This can be modeled as a generalized solution such that it can be reused across the software service oriented computing domain that heavily rely on transactions processing systems. In this paper we would focus on the context of transaction processing.

II. SOFTWARE STABILITY & STABLE DESIGN PATTERNS: BACKGROUND

Software Stability Model (SSM) [1] would be to establish a set of objectives for the System to accomplish. These are based on a set of customer requirements, to provide services and functionality specific to the domain of any given system. The established list of conceptual goals would be perceived as a set of patterns that exist within every day life across different domains, to promote reusability and adaptability of patterns. Consequently, these patterns would be based on the abstraction and core elements that can exist across multiple domains, to solve one or more problems involving support of transaction management.

These patterns need to be stable such that they survive for the duration of the existence of the system. These are commonly known as the Stable Analysis Patterns. These are represented through the Enduring Business Themes (EBTs) and Business Objects within the core stable system and they represent the knowledge logic. Another set of patterns that do not remain stable, but are bound to change based on the needs, without affecting the core of the system. The changes that can take place are based on the specifics of the business rules, application requirements and the environment. These are commonly known as the Industrial Objects (IOs). One of the essential features of the IOs is that they can have a physical representation and represent the application logic. The goal here is to establish a set of industrial objects pertaining to the specific EBTs and the BOs. Transactions would be the main focus of the domain in this paper

III. PATTERN DEVELOPMENT

PATTERN NAME: AnyCompensation

PROBLEM: How can we model and represent the concept of compensation so that it can be used across any domain based on the variables that constitute its core knowledge? This problem can exist in isolation and can well be applied for it to function as a universal solution. The main factors that are involved within notation of compensation are as follows:

1. *Persistent State Management:* Independent of the domain of applicability, we need to ensure that the overall transformation of the state is managed and known upon performing the compensating activities.

2. *Restore to a previous point of consistency*: Restoration can be seen as one of the main goals of the compensation. The data changes that are involved within the transaction should not be left in inconsistent state and should not violate the Atomicity property of the transaction. Moreover, rather than limiting the consistency within the scope of the transaction processing and database systems, we can extend it to services.
3. *Policy*: Rules can be defined such that the scope of the compensation that is known and can be limited if necessary. E.g., In the case of compensation through payment, we need examine the equivalence in value of the payment.

CONTEXT: Compensation is a very important concept that is required to ensure that the needs of any system are met, to establish the goals of restoration along with additional factors such as trust, reliability and quality of services on behalf of the transaction processing. This can involve multiple business process transactions under which the act of compensation can be performed. This itself can be perceived as a type of transaction.

Transactions are also critical within the domain of the Information Integration, Data Management and Service Oriented Computing. A set of transactions can be composed into a single service and various types of services can exist within an enterprise. If there is any failure during any of the operations within any of the transactions, we need to ensure the correct compensating action is taken.

Any set of services across multiple domains would need to interact with each other, such that a single solution is modeled, to maintain consistency through the goal of restoration. E.g., Simple case of booking a trip from Chicago to London would consist of services such as booking a flight along with booking a hotel and a rental car. If the flight is cancelled, the latter two services should also be cancelled or undone, along with booking a new set of reservations, so that consistency is maintained. All these operations can be within the compensating scope and can consist of one or more sub transactions.

FORCES: Compensation pattern should resolve the following forces:

- Even though there can be difference in the compensating actions that are taken to perform the act of compensation or the state of being compensated, we need to ensure that all these cases can be modeled within the solution.
- With the emerging SOC paradigm, we need to ensure that the compensation model can fit well based on the concept of services. E.g., There can be scenarios where compensation can involve services such as Insurances, Restoration Services.
- If there are any failures during the act of compensation, we need to ensure these cases are taken into consideration so that the final result can still be accomplished.
- The compensation mechanisms that may be involved in the compensation process need to be generalized so that the overall timing of performing the compensating

operations does not exceed that of the initial transaction under which the failure might have occurred.

- Due to the existence of various types of participants within the transaction, we need to ensure that the mechanisms have the capability to process any of type of transaction that can be compensated. E.g., From the database aspect, some of the transactions that exist would involve multiple modification to heterogeneous types of data. For cases such as these, we need to ensure that the compensating mechanism would be able to handle heterogeneous data types.

PATTERN STRUCTURE AND PARTICIPANTS:

Figure 1: AnyCompensation Design Pattern

Participants

1. Patterns

- *Restoration*: This is to represent that the restoration is achieved through the existence of the transaction compensation. It would consist of the policy under which the consistency would need to be maintained.
- *AnyCompensation*: Represents the act of compensation that is performed along with its properties.
- *AnyCompensation Mechanism*: These are the various mechanisms under which the compensating activities can take place.
- *AnyTransaction*: The compensating activities can be composed as a transaction itself. At the atomic level, these would consist of a set of operations within the transaction.

Figure 2: Sequence Diagram for the AnyCompensation Design Pattern

CRC-Cards

AnyCompensation– BO (Processor)		
Responsibility	Collaboration	
Describes the context and the compensation properties	Clients	Server
	<i>AnyCompensation Mechanism</i> <i>Transaction</i>	execute() describeComp() process() checkConsistency()
AnyCompensation Mechanism – BO (Compensating Service Provider)		
Responsibility	Collaboration	
Provide means though which the compensation can take place.	Clients	Server
	<i>AnyCompensation</i>	executeMechanism() findMechanism()
AnyTransaction– BO (Service)		
Responsibility	Collaboration	
Composed of the compensating operations within the transaction.	Clients	Server
	<i>AnyCompensation</i>	startCompTranx() endCompTranx() cancelCompTranx() joinCompTranx() performActivity()
Restoration – EBT (State Manager)		
Responsibility	Collaboration	
Defines the consistency rules under which the compensation is achieved.	Clients	Server
	<i>AnyCompensation</i>	compensate() agreement()

Figure 3: CRC Card Representation of the AnyCompensation Design Pattern

Consequences

The use of the Compensation pattern has the following benefits:

1. *Generalization*: The notation of compensation can be generalized using this model so that any application scenario can meet be modeled through the stable design pattern. Especially in the context of the web services and business process, there can be multiple transactions where the compensation would need to be managed, where applicable.

The use of the Compensation pattern has the following limitations:

1. *Non-Compensating Activities*: There can be cases where some transactions are irreversible. There is no path through which the transaction can be compensated. E.g., Within the Real-Time Systems environments domain, the launch of the Space shuttle to Mars. This undoing of the operation is not applicable in this scenario.

IV. APPLICABILITY

Case Study 1: Modeling the concept of compensation for undoing a database operation through a ROLLBACK mechanism. A new compensation transaction for generating a new database record would ensure that the original state of consistency is maintained.

Figure 4: Stable model for undoing a database operation

Business Object: *AnyTransaction*

Industrial Object: *New DB Record* consists of the new entry that has been created for the user based on the transactional properties.

Business Object: *AnyPolicy*

Industrial Object: *State DMV Law* can be perceived as a set rules pertaining under which the activity has taken place and an entry is created in the record.

Business Object: *AnyCompensation Mechanism*

Industrial Object: *ROLLBACK* is a mechanism through which the undo operations are carried out on behalf of the transaction.

Example 2: Modeling the concept of compensation for making a payment to make up or recompense for loss or injury. A new compensation transaction for generating a new database record would ensure that the original state of consistency in the service that is offered is achieved. From these two examples we can see that one of the main factors that meet the requirements for the compensation is the compensating mechanism. This can tremendously vary based on the application context in which it is used.

Figure 5: Stable model for Payment compensation

Business Object: *AnyTransaction*

Industrial Object: *New DB Record* consists of the new entry that has been created for the user based on the transactional properties.

Business Object: *AnyCompensation Mechanism*

Industrial Object: *Payment* is a compensating mechanism through which compensation is processed on behalf of the transaction. In the context of the compensation, the notation of payment can be perceived for making up the loss or injury.

RELATED PATTERNS

AnyEvent

AnyMedia

AnyTransaction

Commitment

IV. CONCLUSIONS

In this paper we have provided a generalized model of AnyCompensation to maximize their applicability across any domain to maximize stability and reusability. With this mode, we understand the main existence and the behavior of what are the underlying goals of compensation based on multiple factors across different domains. This pattern can be used with other patterns within the context of any domain and any transaction processing system through UTMS. We have provided two general applicability case studies where this pattern can be applied.

References:

[1] M.E. Fayad and A. Altman. *Introduction to Software Stability*, *Communications of the ACM*, Vol. 44, No. 9, Sept. 2001, pp. 95-98.

THE COMMITMENT STABLE ANALYSIS PATTERN

Abstract

The act of commitment to move from one consistent state to another is a reoccurring problem that exists across multiple domains, with regard to the perspective from which it is perceived. The Commitment analysis pattern would model the solution to the problem based on the concept of SSM, such that it can be utilized across any domain, to maximize the reusability and productivity of building stable transaction processing system.

Index Terms – software stability patterns, stable design pattern, stability analysis patterns, enduring business themes, business objects, transactions, database and commitment.

I. INTRODUCTION

The act of commitment can exist within a wide array of domains. This is basically perceived as being agreed to take an action and abide to a set of rules within the required scope of time, between any two or more parties. The parties that are involved would result in a new consistent state. E.g., From the social aspect, marriage is seen as a commitment in the relationship, along with multiple other factors (e.g., trust, equilibrium) between a man and women within the scope of their lifetime. The scope within which the actions are committed, can also very depending on the constant disparity in the environment and the ambiance of applicability. Following the example of the marriage, if there are situations where the wife and husband realize that they are not made for each other, they may conclude to file for a divorce. This would result in taking action and being committed to the court of Law for the termination in the relationship between the two parties. From the psychological aspect, this is seen as a breakup of Marriage. However, these semantics can again vary depending on the context of applicability. If you look carefully, the concept of commitment, it can exist anywhere within everyday applications.

We would use the Software Stability Modeling (SSM) that involves construction of the Analysis and Design Patterns to establish a stable model for addressing the notation of commitment within the Social and technical aspect. In this paper, we would concentrate on the technical aspect and would in-depth study the notation of transactions and how the model of commitment can fit well within the concept of transactions. The transactions can range from traditional short-lived transactions to business long running transactions. From the fundamentals of the transaction processing, we can see that the concepts of transactions can exist at different levels within the enterprise. E.g., Within the area of Business Process Management, Workflow Management. As a result, the notation of commitment can also apply to different levels ranging from commitment of a single transaction to commitment of a specific service, within any given stable system. Later in the paper, we would investigate how other analysis patterns can be utilized to support the theme of meeting any transactional requirements across any domain.

II. SOFTWARE STABILITY & STABLE DESIGN PATTERNS: BACKGROUND

Software Stability Model (SSM) [1] would be to establish a set of objectives for the System to accomplish. These are based on a set of customer requirements, to provide services and functionality specific to the domain of any given system. The established list of conceptual goals would be perceived as a set of patterns that exist within every day life across different domains, to promote reusability and adaptability of patterns.

Consequently, these patterns would be based on the abstraction and core elements that can exist across multiple domains, to solve one or more problems involving support of transaction management.

These patterns need to be stable such that they survive for the duration of the existence of the system. These are commonly known as the Stable Analysis Patterns. These are represented through the Enduring Business Themes (EBTs) and Business Objects within the core stable system and they represent the knowledge logic. Another set of patterns that do not remain stable, but are bound to change based on the needs, without affecting the core of the system. The changes that can take place are based on the specifics of the business rules, application requirements and the environment. These are commonly known as the Industrial Objects (IOs). One of the essential features of the IOs is that they can have a physical representation and represent the application logic. The goal here is to establish a set of industrial objects pertaining to the specific EBTs and the BOs.

Transactions would be the main focus of the domain in this paper.

III. PATTERN DEVELOPMENT

PATTERN NAME: Commitment

PROBLEM: We can build a model based on the Stability Analysis Patterns that can be applied within any given field of application. This is to provide a crisp understanding of the core domain knowledge and to exploit it as a reusable stable pattern within any transaction based application. The true problem is *how* to achieve a generalized solution for the commitment problem based on the level of abstraction.

The main factors under which the notation of fulfilling a commitment can revolve under are as follows:

1. *Timing:* Depending on the domain and the application environment, some transactions can be time depended and timing can continually play a critical role. E.g., Within the Air Traffic Control or stock market where the exchange of information is established within a short time frame.
2. *Criteria/Rules:* Commitment can be made based on a set of rules for defining the criteria under which an agreement can be reached.
3. *Agreement:* An agreement can be perceived as the condition for preparing to fulfill a commitment. This would directly be dependent on process or a protocol through which an agreement can be reached.

4. *Scope*: The fulfillment of the commitment can exist within the local body of any system or can also exist on behalf of multiple systems during any process, across different domains.

CONTEXT: Businesses cannot survive if they are not committed to providing quality services to satisfy the customer needs. This can involve multiple business process transaction under which the commitment between many parties can take place. This itself can be perceived as a type of transaction.

Transactions are also critical within the domain of the Information Integration and Data Management. A set of transactions can be composed into a single service and various types of services can exist within an enterprise. Here, the concept of commitment is important because these services would need to interact with each other, such that a single solution is modeled. E.g., Booking a trip to London would consist of services such as booking a flight along with booking a hotel and a rental car. These need to co exist for the final solution to be provided as a single solution. Other factors such as automation, reliability, trust and consistency also go hand in hand with commitment.

Evidently, from the high-level view, there are many existing social and technical applications where the commitment pattern can be assimilated with other analysis patterns to maximize the overall stability of the application solution. Application of the patterns would be presented later in the paper.

FORCES: Commitment pattern should resolve the following forces:

- From the transaction aspect, the idea of zero-phase commit (0PC), one-phase commit(1PC) and two-phase commit(2PC) does not fit well within the domain of Business Process execution. The pattern can be applied to support any required level of commitment, specific to transaction properties and the resources involved during the execution of the transaction. This can be achieved through the relationship with other analysis pattern.
- Keeping a record of the activities being committed between two or more parties would be necessary to make further decisions.
- Successful automation of business process can cost valuable time and money, due to manual labor and awareness of domain knowledge where timing can play an important role. As a result, the act of when to fulfill commitment can become critical to various applications.
- An individual party may make commitment for themselves to get work done, based on their own needs and requirements. The pattern model should be able to model these scenarios.

PATTERN STRUCTURE AND PARTICIPANTS:

Figure 1: Commitment Analysis Pattern

Participants

1. Classes

- *Commitment*: Implies that an agreement has been fulfilled between two or more parties. The agreement is based on the set of rules that are defined within the policy.

2. Patterns

- *AnyTransaction*: Represents a set of operations being performed within a single unit of work, with the involvement of the party. This is a mechanism through which the party has decided to make the commitment. The operations within this time frame are committed. The type of protocol or process that is used to fulfill the commitment can itself be a transaction.
- *AnyPolicy*: This is a set of rules that the party would be committed to during their engagement in the Transaction.

- *AnyLog*: All the operations or activities that take place under the transaction are recorded into the Log. The log can be current or an historic log.
- *AnyParty*: Represents the participants that are involved in the transactions for fulfilling the commitment based on the agreement of the policy.

Figure 2: Sequence Diagram for the Commitment Analysis Pattern

CRC-Cards

AnyPolicy – BO (Policy Provider)		
Responsibility	Collaboration	
Define rules and criteria under which the commitment is fulfilled	Clients	Server
	<i>AnyParty</i>	rules() verifyRules() checkConsistency()

AnyParty – BO (Transaction Participant)		
Responsibility	Collaboration	
Participant involved in the Transaction	Clients	Server
	<i>AnyPolicy</i> <i>AnyTransaction</i>	defineRules() engageInTranx() preparedToCommit()

AnyTransaction– BO (Service)		
Responsibility	Collaboration	
Execute events of the Activity on the behalf of the party involved in the Activity	Clients	Server
	<i>AnyParty</i> <i>AnyLog</i>	startTranx() endTranx() cancelTranx() joinTranx() performActivity() backoutTranx() logged() loggedTranx()

Commitment – EBT (Signature)		
Responsibility	Collaboration	
Establish a commitment upon agreement of the rules specified in the policy	Clients	Server
	<i>AnyParty</i> <i>AnyPolicy</i> <i>AnyTransaction</i>	fulfilledCommitment() agreement() commitTranx()

AnyLog– BO (Database)		
Responsibility	Collaboration	
Keep a record of the operations along with the state of the transaction	Clients	Server
	<i>AnyTransaction</i>	writeLog() readLog() startLogTime() endLogTime() currentPointOfLog()

Figure 3: CRC Card Representation of the Commitment Analysis Pattern

Consequences

The use of the Commitment Analysis has the following benefits:

1. *Generalization*: The notation of commit can be generalized using this model so that any application scenario can meet be modeled through the analysis pattern. Especially in the context of the web services and business process, there can be multiple levels of commitment that needed to be fulfilled by one or more transactions. The notation of multi-phase commit can be accommodated through this model.
2. *Representation of the Participants of the Transaction*: The participants of the transaction can be anything ranging from a single resource, a transaction itself or any human that is involved in an activity.

The use of the Commitment Analysis has the following limitations:

1. *Multiplicity*: Multiplicity can exist for cases there are multiple sub transactions within a main transaction. In this case there is a limitation in the representation of the commitment of different transactions, especially if there are any failures within any of the sub transactions.

IV. APPLICABILITY

Case Study 1: Car Driver License

Modeling the concept of commitment in getting Car Driver License from any DMV office. This can also be seen as a business transaction been committed on behalf of a student.

Figure 4: Stable model for Car Driver License

Business Object: *AnyTransaction*

Industrial Object: *Taken Exams* can be perceived as an activity that is performed as a transaction to ensure that the student is committed to the rules of the state DMV law.

Business Object: *AnyLog*

Industrial Object: *DB Record* is historic record of the database that contains the information about the student driving test information and the DMV license information.

Business Object: *AnyPolicy*

Industrial Object: *State DMV Law* can be perceived as a set rules pertaining under which the activity has taken place.

Business Object: *AnyParty*

Industrial Object: *Student* can be seen as the party of the transaction because their role can change at any given time.

Case Study 2: Self-Commitment for performing exercise to keep fit

A student is committed to performing exercise based on their requirement for keeping themselves fit, with regard to weight conscious. The act of performing exercise can be perceived as an activity performed on behalf of the transaction. The student keeps track of their exercise program in a diary to achieve their commitments of being weight conscious.

Figure 5: Stable model of a student committed to performing exercise

Business Object: *AnyTransaction*

Industrial Object: *Exercise* can be perceived as an activity that is performed as a transaction to ensure that the student is committed to their goals pertaining to being weight consciousness.

Business Object: *AnyLog*

Industrial Object: *Diary* is historic record of the exercise performed by the students for establishing their goals.

Business Object: *AnyPolicy*

Industrial Object: *Weight Conscious* can be perceived as a student's perception of the right weight to ensure that they are fit and healthy.

Business Object: *AnyParty*

Industrial Object: *Student* can be seen as the party of the transaction because their role can change at any given time.

RELATED PATTERNS

AnyEvent

AnyTransaction

AnyCompensation

IV. CONCLUSIONS

In this paper we have provided a generalized model of Commitment to maximize their applicability across any domain to maximize stability and reusability. The notation of commitment can exits ranging from a single person being committed to achieving their goal, to program that would perform a set of operations on behalf of a transaction to ensure consistency. With this model, we understand the main existence and the behavior of what are the underlying goals of achieving commitment across different domains. The notation of phase0, pahse1 and phase2 commitment can be achieved when the pattern is applied and integrated across multiple types of transactions. This pattern can be used with other patterns within the context of any domain and any transaction processing system through UTMS. We have provided two general applicability case studies where this pattern can be applied.

References:

[1] M.E. Fayad and A. Altman. *Introduction to Software Stability, Communications of the ACM*, Vol. 44, No. 9, Sept. 2001, pp. 95-98.

Appendix C: Sample Code of Stable Patterns

1. Sample Demo of the AnyTransaction Service pattern

Travel Agent

Enterprise Travel Center

Agent Name: Ram Goverdhana

Customer Name: Ram Goverdhana

From	Chicago	Date	10/05/04
To	London	Date	11/15/04

☒ E-Receipt

☒ Hotel

Make Reservations

Reservation Confirmation

Are you sure?

Message

Travel Confirmed!

2. Sample demo of the AnyCompensation pattern

Shown below is the class diagram of the AnyTransaction patterns and the AnyCompensation pattern

- AnyTransaction Pattern:

- AnyCompensation Pattern:

Shown below are the sample code snippets with abstraction of the analysis and business objects.

```

package TranxPat;

/**
 * @author: ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */

/*****
 * Author: Ram Goverdhana
 * Date: 09/01/04
 * Description: Implementation of AnyCompensatingMechanism
 * Design Pattern.
 * Relationships: AnyCompensation (BO), AnyPolicy(BO)
 * Sterotype: Business Object
 *****/
  
```

```

// *****
// Name: <ANYCOMPENSATINGMECHANISM>
// Description: The <AnyCompensatingMechanism> function would used be driver for
// initiating the compensating actions based on the policy.
// Dependencies: <AnyCompensatingMechanism> <AnyTrasaction>
// *****
public abstract class AnycompensatingMechanism{
/**
 * Constructor for AnycompensatingMechanism.
 */
public AnycompensatingMechanism() {
    super();
}

// *****
// Name: <FINDMECHANISM>
// Description: The <AnyCompensatingMechanism> function would used be driver for
// initiating the compensating actions based on the policy.
// Dependencies: <AnyCompensatingMechanism> <AnyTrasaction>
// *****
public abstract String findMechanism(AnyPolicy p);

// *****
// Name: <EXECUREMECHANISM>
// Description: The <AnyCompensatingMechanism> function would used be driver for
// initiating the compensating actions based on the policy.
// Dependencies: <AnyCompensatingMechanism> <AnyTrasaction>
// *****
public abstract boolean executeMechanism();

// *****
// Name: <PERFORMUNDOOPERATION>
// Description: The <AnyCompensatingMechanism.performUNDOOperation> is a mechanism through
// which the compensation takes place. This would consist of one or more transaction
// based on the goven policy.
// Dependencies: <AnyCompensatingMechanism> <AnyTrasaction>
// *****

public abstract boolean performUNDOOperation();

}
package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */

/*****
 * Author: Ram Goverdhana
 * Date: 09/01/04
 * Description: Implementation of AnyCompensation
 * Design Pattern.
 * Relationships: AnyCompensatingMechanism (BO), AnyTransaction(BO)
 * Stereotype: Business Object
 *****/

// *****
// Name: <ANYCOMPENSATION>
// Description: The <AnyCompensation.Compensate> function would used be driver for
// initiating the compensating actions based on the policy.
// Dependencies: <AnyCompensatingMechanism> <AnyTrasaction>
// *****
public abstract class AnyCompensation {
/**
 * Constructor for AnyCompensation.
 */
public AnyCompensation() {

```

```

        super();
    }

    /**
     * *****
     * Name: <COMPENSATE>
     * Description: This function would trigger the compensating transactions involved for
     * processing the AnyCompensation based on the given policy.
     * Dependencies: <AnyPolicy>
     * *****
     */
    public abstract boolean Compensate(AnyPolicy p);

    /**
     * *****
     * Name: <CHECKCONSISTENCY>
     * Description: This is a verification process to ensure that the resource that is involved
     * in the transaction is consistent after the compensation takes place based on the policy.
     * Dependencies: <AnyResource>
     * *****
     */
    public abstract boolean checkConsistency(AnyResource ar);
}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyLog {

    private String LogRec;
    String blank = "";

    /**
     * Constructor for AnyLog.
     */
    public AnyLog() { // CONSTRUCTOR
        this.LogRec = " "; // Default to BEGIN phase of the transaction
    }
    /**
     * *****
     */
    public abstract void UpdateLogRec(String logrec);
    public abstract String GetLogRec();
    public abstract void WriteLog(AnyTransaction t);

}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyParty {

    private String PartyName;
    private int pid;

    /**
     * Constructor for AnyParty.
     */
    public AnyParty() {
        super();
    }
}

```

```

    /**
     * Constructor for AnyParty.
     */
    public AnyParty(String name) {
        this.PartyName = name;
        this.pid = 0;
    }

    //public boolean EngageInTranz(AnyTransaction atl) {
    //    int tid = 1;
    //    //
    //    return (atl.RunTranz(PartyName));
    //}

    //public abstract boolean permit(AnyPolicy ap);

    public abstract boolean EligibleParty(String criteria);

    public abstract boolean GetMeService();

    public abstract void Playrole(String name);

    public abstract String GetName();
    public abstract void SetName(String name);

    public abstract int GetPid();
    public abstract void SetPid(int pid);

}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyPolicy {

    private String PolicyName;
    private String PolicyInfo;
    private String PolicyQuery;

    /**
     * Constructor for AnyPolicy.
     */
    public AnyPolicy() {
        super();
    }

    /**
     * Constructor for AnyPolicy.
     */
    public AnyPolicy(String name) {
        this.PolicyName = name;
        this.PolicyInfo = "";
        this.PolicyQuery = "";
    }

    public abstract boolean sat(int PID, String PDesc);
    public abstract int CreatePolicy(String name);
    public abstract boolean VerifyPolicy(int pid);

```

```

        public abstract boolean MetPolicy(AnyResource r);

    }

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyResource {

    private String ResourceName;
    private int ResourceID;

    public AnyResource() {
        super();
    }

    public AnyResource(String name, int rid) {        // CONSTRUCTOR
        this.ResourceName = name;                    // Default to BEGIN phase of the transaction
        this.ResourceID = rid;
    }
    /**/
    public abstract boolean request(AnyPolicy ap, AnyTransaction at);

    public abstract int ResourceState(String name);
    public abstract String GetResourceName();
    public abstract void SetResourceName(String name);
    public abstract int GetResourceID();
    public abstract void SetResourceID(int rid);

    public abstract int getMeasure();

    public abstract boolean ModifyResource(String criteria);
    public abstract boolean GetResourceAccess(int rid);
    public abstract boolean LockResource(int rid);
    public abstract boolean ReleaseResource(int rid);

    public abstract void ResourceConstraiant(String condition);
    public abstract boolean ComplyResourceConstraiant(String condition);
    public abstract void LogResourceChange();

}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyTransaction {

    private int TID;
    private int Ttype;
    private String ttype;

    private String Tstate;
    private String Tlog;

```

```

        private tranx K = new tranx();

        /**
         * Constructor for AnyTransaction.
         */

        public AnyTransaction(){
            this.TID = TID ++;
        }

        public boolean RunTranx(int polyid, String polyinfo) {

        }

        public class tranx{
            private int TID;
            private String TSTATE;
            private int TTYPE;
            private boolean ISACTIVE;
            private int TPHASE;
            private boolean TCANCEL;
        }

        public void SetTranxProperty(int tid) {
            K.TID = tid;
            K.TSTATE = "";
            K.TTYPE = 1;
            K.ISACTIVE = true;
            K.TPHASE = 1;
            K.TCANCEL = false;
        }

        public tranx GetTranxProperty() {
            return K;
        }

        protected abstract int CT();
        protected abstract boolean ST(AnyPolicy p);
        protected abstract int GenerateTID();
        protected abstract void LogTranx();
        public abstract int getTID();
        public abstract String getTState();
    } //*****

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */

/** Stereotype: Business Object(Pattern-BO)
 */

public abstract class AnyTransactionState {

    private String TranxState;
    private int stateid;

    public AnyTransactionState(String state) { // CONSTRUCTOR
        this.TranxState = state; // Default to BEGIN phase of the transaction
        this.stateid = this.stateid+1;
    }

```

```

    }

    /**
     * *****
     * public abstract void SetState(String state);
     * public abstract String GetState(int id);
     * public abstract int GetStateID();
     */
}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public abstract class AnyTransactionType {

    private int TranxType;
    private String TranxTypeName;

    public AnyTransactionType() { // CONSTRUCTOR
        this.TranxType = 0; // Default to BEGIN phase of the transaction
        this.TranxTypeName = "BUSINESS";
    }
    /**
     * *****

    public abstract void SetTranxType(String type);

}

package TranxPat;

/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
public class Restoration {

    /**
     * Constructor for Restoration.
     */
    public Restoration() {
        super();
    }

    /**
     * Author: Ram Goverdhana
     * Date: 09/01/04
     * Description: Implementation of Restoration Analysis Pattern.
     * Relationships: AnyCompensation (BO), AnyPolicy (BO)
     *
     * Sterotype: Enduring Business Theme
     */

    /**
     * *****
     * // Name: <RESTORE>
     * // Description: Restoration is achieved through the restore function. The Restore function
     * // would be driven by the Compensation activity that takes place under the covers based on
     * // the given policy.
     * // Dependencies: <AnyCompensation> <AnyPolicy>
     * // *****
     * public boolean Restore(AnyCompensation c, AnyPolicy p) {
     * System.out.println("Entry into {Restoration.Restore}");
     * if (c.Compensate(p))

```

```

        return true;
    else
        return false;
}

// *****

}

package TranxPat;
import TranxPat.*;
/**
 * @author ramgove
 *
 * To change this generated comment edit the template variable "typecomment":
 * Window>Preferences>Java>Templates.
 * To enable and disable the creation of type comments go to
 * Window>Preferences>Java>Code Generation.
 */
/**
 * Author: Ram Goverdhana
 * Date: 09/01/04
 * Description: Implementation of Service centric functions
 *              of the analysis pattern.
 * Relationships: AnyTransaction (BO), AnyPolicy (BO)
 * Sterotype: Enduring Business Theme
 */
public class Service {

    String ServiceName;
    int ServiceID = 0;

    AnyPolicy ServicePolicy;

    /**
     * Constructor for Service.
     */
    public Service() {
        super();
    }

    /*******/
    public Service(int sid, String sname) {
        this.ServiceID = sid + sid;
        this.ServiceName = sname;
    }

    /*******/

    public int ProcessService(AnyTransaction t, AnyPolicy p) {
        boolean strc;

        t.CT(); // Create a Transaction Process task
        strc = t.ST(p); // Start a Transaction Process on the created task

        if (strc) return (1); // Return 1 as a valid int value
        else return (0); // Return 0 as an invalid int value
    }
    // *****

```