Improving Software Product Line Engineering using AOP, LEL and UML

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Megha
(801031019)

Under Supervision of
Mrs. Shivani Goel
Assistant Professor,CSED

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2012

Certificate

I hereby certify that the work which is being presented in the thesis report titled, **"Improving Software Product Line Engineering using AOP, LEL and UML"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mrs. Shivani Goel and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

Megha

(Megha)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

Shivani Goel.

(Mrs. Shivani Goel)
Assistant Professor,
Computer Science & Engineering Department,
Thapar University,Patiala.

Countersigned by

(Dr. Maninder Singh)
Associate Professor & H.O.D.
Computer Science & Engineering Department.
Thapar University,
Patiala.

(Dr. S.K. Mohapatra)
Dean (Academic Affairs),
Thapar University,
Patiala.

Acknowledgement

First of all, I thank God almighty for guiding me the right direction. With his mercy, it has been made possible for me to reach so far.

It is a great privilege to express my gratitude and admiration towards my respected supervisor Mrs. Shivani Goel. She has been an esteemed guide and a great support behind achieving the task. Without her able guidance, kind efforts and encouragement, the work wouldn't have been what it is. I am truly grateful to her for extending her total co-operation and understanding whenever I needed help and guidance from her.

I am extremely grateful to Dr. Maninder Singh, Head, Department of Computer Science & Engineering, Thapar University, Patiala for providing best facilities and atmosphere for the creative work guidance and encouragement.

I would like to thank all those who have contributed to the completion of my thesis work and helped me with valuable suggestions for improvement. Above all I am grateful to my parents and grandparents who soulfully provided me their constant support, and encouraging attitude to undertake the challenge of this proportion.

I am also very thankful to the entire faculty and staff members of Computer Science & Engineering Department for their direct and indirect help, cooperation, love and affection which made my stay at Thapar University memorable.

Megha

(801031019)

Abstract

Software product lines are a booming approach to software reuse. Software product lines enable modular and huge scale reuse. The success of a software product line approach directly depends on how well feature variability within the group is implemented and managed all through the development lifecycle. In order to obtain the profit of software product lines, their designs need to be stable. To an important scope, the development of product lines is difficult by crosscutting concerns. It is generally assumed that aspect-oriented programming promotes better modularity and changeability of product lines than conventional variability mechanisms. Aspect-oriented programming may grow the design level of software, the reusability of components and the implementation of separation of concerns. Modeling variability is a center problem in software product line engineering. So modeling variability with aspect-oriented techniques is a clear idea which has been exploited before to some scope. Report discusses a new approach aspect oriented technique for the modularization of concerns that facilitates variability implementation, management and tracing by aspect-oriented technique. The relationship between variability, commonality and variation point in a software product line bears similarities to the relationship between crosscutting concerns, core concerns and joint point in aspect oriented modeling respectively. The overall goal of this research is to improve software product line using aspects so as to enable software reuse, design functionality and flexible extensibility of requirements.

Domain analysis has been recommended by many researchers in the reuse research as an essential process for achieving successful reuse. Reuse of products, processes and all kinds of knowledge has been recognized as an objective in software engineering, in order to build up reliable and high quality software systems on schedule and within budget. This duality of issues calls for methods as domain analysis to systematically build reusable elements. In this report a description of domain analysis including domain analysis phases, domain analysis methods and quality attributes of domain analysis methods are given. There are many methods for domain analysis during reuse. These include FODA, JODA, ODM, DSSA and DADP methods. They all differ in the quality attributes they support. The quality attributes supported by two domain

analysis methods namely joint object oriented domain analysis and domain analysis and design process method have been identified and verified using a case study of ATM.

Requirements of a software product line are organized into features in feature- oriented domain analysis approach. Feature models are widely used to model the information gathered during domain analysis. These are generally used as a means for capturing commonality and managing variability of a software product line. Natural language oriented models (language extended lexicon) are useful during the first stages of software development as it is in general understandable by stakeholders encouraging their participation. Here the recommendation is to define natural language oriented requirement model, known as language extended lexicon and derive a UML class diagram and object diagram from them.

Abbreviations

AOP	Aspect Oriented Programming
ATM	Automated Teller Machine
DA	Domain Analysis
DADP	Domain Analysis and Design Process
DISA	Defense Information Systems Agency
DSSA	Domain Specific Software Architecture
FODA	Feature Oriented Domain Analysis
JIAWG	Joint Integrated Avionics Working Group
JODA	Joint Object Oriented Domain Analysis
LEL	Language Extended Lexicon
ODM	Organizational Domain Modelling
SPL	Software Product Line
SPLE	Software Product Line Engineering
SPLs	Software Product Line
STARS	Software Technology for Adaptable and Reliable Systems Program
UofD	Universe of Discourse

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Software Product Line

Software reuse is the use of already existing software to build new software. Software reuse is important because people want to construct systems that are bigger, more complex, more reliable, and less expensive and which must be delivered on time. The traditional software engineering methods are found inadequate for these demands of software engineering. The purpose of a software product line is to decrease the engineering effort required to develop a group of related systems by capitalizing on the commonality between the systems and by officially managing the variation between the systems. Software product line support the set of products which is determined by the specified product line variability [32]. Modeling variability is a core problem in software product line engineering. Cross cutting concerns are a tedious job which the traditional methods have failed to implement.

Aspect orientation provides better modularization of crosscutting concerns. One of the major promises of aspect-oriented programming is to advance software maintainability during new modularization mechanisms for encapsulating crosscutting concerns. So modeling variability with aspect- oriented techniques is a clear idea which has been exploited before to some extent.

Object oriented approach, however, encounters the separation of concerns that leads to the code-tangling and code-scattering. Aspect oriented programming (AOP) has been considered a promising abstraction principle to decrease the problem of code tangling and make software structure clean and configurable.

Aspect oriented programming (AOP) separates crosscutting concerns that are usually tough to do in object-oriented programming. Some concerns can be simply encapsulated within modules, according to the selected implementation language; however, others whose functionality affects numerous modules, are called crosscutting concerns and they

are difficult to separate. Before encapsulating crosscutting behavior into an aspect, the developer must initially identify it in the requirements. This is difficult, because, by their nature, they produce an entangled or scattered code, which is difficult to understand and maintain and are likely to be described in multiple parts of the requirements document. They cannot be easily encapsulated into new functional units as implicit functionality because they crosscut the entire system and are implemented in various classes.

1.2 Domain Analysis

Domain analysis is used in software product line to identify the main characteristics of applications in a particular domain. Domain analysis considers commonalities and differences across the group and provides a systematic approach for dealing with commonalities from reuse point of view [1]. Domain analysis is part of the domain engineering process that uses the domain model to define reusable software architecture, to design reusable code and to define the structure of the domain. The objects in domain are defined by their service and their attributes, while domain structure is defined through whole-part concepts, variation in objects, services, attributes, and concepts, and the relationships between these objects and concepts [2].

Domain analysis is an object-oriented analysis with emphasis on a group of applications. Domain analysis produces models for architecture and implementation as well [3, 4]. Thus domain analysis approach can support a mapping from the problem space to appropriate objects and classes, while considering the design context for patterns and frameworks. The goal of domain analysis is to define a domain model that can be used to produce reusable software objects, especially reusable requirements [5]. There are a number of domain analysis methods. These include FODA, JODA, ODM, DSSA and DADP methods. These all have different steps for performing domain analysis. All of these support different quality attributes like reusability, stability, coupling and cohesion, etc. [1].

1.3 Language Extended Lexicon

Software product line engineering is a procedural frame for engineering software product lines (SPLs). SPLs are developed as a entire and share many assets, together with this increasing reusability. The problem space in SPLs is captured with feature models. A feature model, first introduced by Kang [6], is represented with a feature diagram. It includes actions for identifying and modeling the commonalities and variabilities of a product line in terms of features and for analyzing dependencies between these features. Existing feature models frequently tend to produce very large feature models in size. To treat such large feature models as monolithic entities makes them very hard to understand, develop, manage and evolve. Usually every change performed on a feature model must be verified by a group of experts with different expertise [7], which is time consuming as well as costly.

Particularly, feature model distinguish between non-crosscutting concerns called base concerns and crosscutting concerns called aspects. However, a feature model is not simply comprehensible to stakeholders. During the early stages of software development the interaction with stakeholders is mainly inconvenient. For this explanation, natural language is still widely used to model requirements information. In this report, case studies are used to demonstrate the viability of the approach on a real world scenario and to analyze guidelines for the constructions of the feature model of the various case studies. Natural language (language extended lexicon) is the only notation that can be read and understood by the stakeholders, hence encouraging their active participation crucial in first steps of software development.UML is used to represent design for the developers. This report presents transformation processes, using natural language oriented requirements models (LEL) to derive UML class diagrams and object diagrams from them.

Organization of the thesis: Chapter 1 has introduction. Literature survey is presented in chapter 2. Chapter 3 contains the problem statements taken up in the thesis. Proposed solutions are presented in chapter 4. Chapter 5 has conclusion and future scope.

Chapter 2

Literature Survey

2.1 Software Product Line Engineering

Software product line engineering is the discipline of engineering a set of software-intensive systems which are sharing a common, managed set of features that fulfill the specific needs of a particular market segment and that are developed from a common set of core assets in a given way[8] e.g., car engine management systems and so many. The main purpose of a software product line is to decrease the overall engineering effort which is required to produce a group of similar systems by capitalizing on the commonalities between the systems and by managing the variabilities between the systems. The commonalities consist of the artifacts and the properties that are common to all product line applications. The variabilities define how a variety of applications derived from the product line can differ. Though not all product line engineering approaches explicitly address a product line requirements document, it is beneficial to have one [9]. Such a document gives details of the variability by specifying variation points and product specific variants that can be bound to the variation points [10].

A variation point defines what does vary? It documents a variable item or a variable property of an item. A variant defines how does it vary? It documents the possible instances of a variation point. It is a paradigm that systematizes reuse**.** By adopting SPL practices, organizations are able to achieve significant improvement in the time-to-market, cost, efficiency and product quality.

Software product line engineering is generally composed of two engineering processes:
Domain Engineering: In software product line engineering, reusable assets are planned and formed during domain engineering process. It defines the commonality and the variability of the applications of the software product line [11]. It includes activities such as domain analysis and definition etc.

Application Engineering: During application engineering, products are either automatically or manually assembled, using the assets formed through the domain engineering process and completed with product-specific artifacts. It includes application requirements analysis and instantiation of the reference architecture etc.

Fig 2.1 [12] shows the correlation between domain engineering and application engineering phases. One of the key aspects of product lines is variability and its management. Variability management is the activity which includes the identification, designing, implementing, and tracing flexibility in software product lines (SPLs).

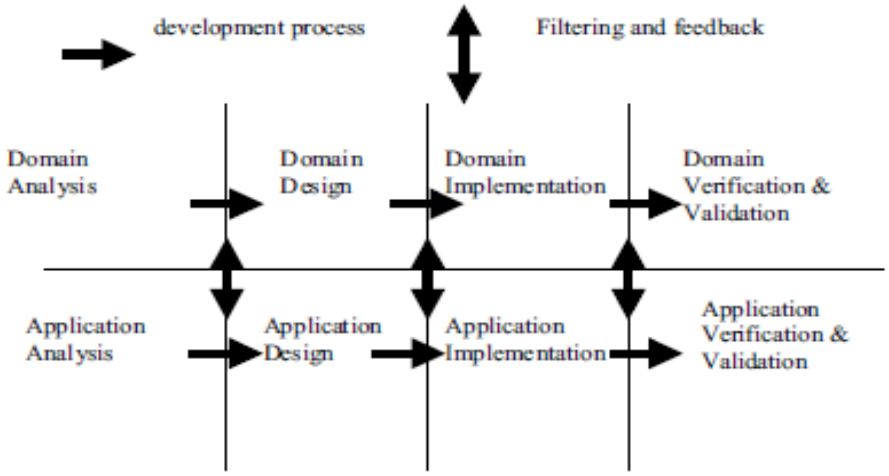

Fig 2.1: Software product line life cycle :domain and application engineering phases [12].

2.2 Aspect Oriented Programming

AOP is a new technology for separating crosscutting concerns into single units known as aspects. Aspects should have the following properties [13] :

a. Robust (it means modification to one aspect should have a partial impact on other aspects)

b. Cross-cutting each other (in the target program information concerning one aspect is mixed with information about other aspects),

c. Systemic (they affect the target program at several different places),
d. Loosely coupled (an aspect must not know the information of other aspects),
e. Contain join points (which are used to join the aspects).

Aspects are behaviors that are tangled and scattered across a system .An aspect is a modular unit of crosscutting implementation. These are the descriptions of behaviors that are interwined, and woven throughout. It encapsulates behaviors that influence multiple classes into reusable modules. Behavior that cannot be encapsulated because of its impact across the whole system is called crosscutting behavior. AOP provides the capability of separating several concerns through development, the effects of the crosscuts must be included back into the solution space. The goal of the separation of concerns is to develop the conceptual ability of programmers throughout the development so that the end result at run-time will have crosscutting concerns that are transparent. Separation of concerns is a generic concept that can be used to decrease software complexity by dividing the software into modules such that each module is accountable for different general concerns.

Aspect oriented programming uses abstraction and decomposition in order to break large problems into manageable sub-problems without losing its nature. In AOP, large problems are broken down in a different way, using aspectual decomposition rather than functional decomposition. An important difference between aspects and functional units is that aspects primarily cross-cut each other and the resulting executable code. Aspects interfere with other artifacts at so called join points. These are the well defined points in the structure of an artifact. Point cut expressions quantify over the join points to choose the set of actual composition points for an exact aspect. An aspect weaver mechanically composes aspects with the rest of the system, either statically during compilation, or at load-time or dynamically at runtime [14]. Aspect oriented programming is a promising paradigm to support improved separation of concerns, leading to the production of software systems that are easier to sustain and reuse. It is centered on the aspect notion as an abstraction proposed to modularize such crosscutting concerns and develop the system maintainability and reusability. Recent research trends suggest using AOP notions and

mechanisms also at an early on stages of software development to reduce development costs.

2.2.1 Terminology

The typical terminology used in AOP helps us to understand the concepts.

Concern: A concern is defined as those interests belonging to the system and its operation, or other aspects which are critical or important for one or more stakeholders. Concern is a feature, a responsibility, or a sub-problem.

Crosscutting Concern: Some concerns can be simply encapsulated within classes or modules; though others, whose functionality affects several modules, are called crosscutting concerns and they are difficult to separate. As a result, the elements of crosscutting concerns are scattered and tangled within elements of other concerns.

For example, we may want to insert logging to classes within the data-access layer and also to classes in the UI layer whenever a thread enters or exits a method. Even though the main functionality of each class is very different, the code needed to achieve the secondary functionality is often identical [13].

Advice: An advice is defined as the behavior or code that is executed at a join point.

a. Before advice: Runs as a join point is reached, before the program proceeds with the join point and executes prior to the join point.

b. After advice: Executes following the join point. On a particular join point runs after the program proceeds with that join point.

c. Around advice: It surrounds the join point's execution. On a join point runs as the join point is reached, and has explicit control over whether the program proceeds with the join point.

It can continue original execution, bypass execution or cause execution with an altered context. It can cause execution of the join point multiple times. In the given example, this is the logging code that we wish to relate whenever the thread enters or exits a method [13].

Aspect: The combination of the point-cut and the advice is termed an aspect. It is the unit of modularity for crosscutting concerns. In the example, we add a logging aspect to our application by defining a point-cut and give the correct advice [13].

Point-cut: Declaration that selects join points and collects contexts at that point. It is a group of join points. In the given example, a point-cut is reached when the thread enters a method, and another point-cut is reached when the thread exits the method [13].

Composition: Composition is bringing together separately created software elements [15].

Weaving: Weaving is defined as the process of composing core functionality modules with aspects, there by yielding an operational system. It tells how to integrate the final system [13].

Join Point: It places where crosscutting concerns can be woven in. It is a well-defined point in the program flow. Examples: field access, object creation etc.

2.2.2 Various Examples of AOP

The various examples of AOP are as follows:

Table 2.1: Examples of AOP

• Tracing (determining what methods are called when) • Session tracking, session expiration • Logging	• Synchronization • Verifying correctness • Visualization • Special security management

2.2.3 Major Advantages of using AOP

The major advantages of using AOP are as follows:

Table 2.2: Advantages of AOP

• It can be used for developing simpler and cleaner code which can be reusable and therefore reduce the redundancy of code • It provides well defined responsibilities for individual modules.	• It provides ease of maintenance, evolution, customization, integration • It makes adding crosscutting functionality easier

2.2.4 Disadvantages of using AOP

Although AOP has several advantages it has some disadvantages also as discussed below:

Table 2.3: Disadvantages of AOP

• It is difficult to understand software because of invisibly injected aspects • Difficult to change because it have to find all the code involved and be sure to change it consistently	• It has a complicated control flow breakage • There may reduce quality of software if aspects are not appropriately managed.

2.3 Domain Analysis (DA)

Software development cannot possibly become an engineering discipline if it has not worked upon a technology for developing products from reusable assets. As a discipline software reuse must define and promote the managerial, organizational, and technical standards which are required to achieve this goal [16]. Whether the organizational structure is domain centered or application centered, it is one of the most important organizational decisions which is having the greatest impact on reuse operations. In domain-centered organization, domain-engineering team take decision about the

development of reusable assets and then leaving it to application engineering team to adjust their design discipline to take the best advantage of available reusable assets. In application centered organization, the application engineering team assigns development tasks to the domain engineering team, in order to serve the goals of their application development activity.

In domain engineering, the activity domain analysis or product line analysis, describes the common and variable parts within a domain. The term was coined by James Neighbors in 1980s Domain analysis is the first phase of domain engineering. It is a key method for realizing systematic software reuse [17].

The key to reusable software in domain analysis is that it focuses on the reusability of analysis and design, not code. In simple words, domain analysis is similar as systems analysis but instead of being applied on a single system it is made for multiple systems related to each other.

Domain analysis is done by reengineering techniques and domain analysis methods. Domain analysis is the method of identifying, collecting, organizing and representing the relevant information in a domain which is based upon the study of existing systems and their development histories, knowledge captured from domain experts and emerging technology within a domain.

One of the objectives of domain analysis is to make all that information readily available for reuse. In making a reusability decision, that is, to decide whether or not to reuse a component, a software engineer has to understand the context, which prompted the original designer to build the component the way it is. The chain of design decisions used in the development process is absent in the source code. By making this development information available, reuse becomes more effective.

A domain analysis is the system engineering of a family of systems in an application domain. The last goal of a domain analysis is to build and maintain a reuse library asset which is used to develop instances of systems in the domain family.

Domain analysis research should try to reuse existing research from other disciplines. To make reuse possible, software engineering is divided into two parts: domain engineering to find and implement the common features and application engineering to produce the individual applications. Domain engineering is then divided into three phases: domain analysis, domain design, and domain implementation. All the products of the domain analysis activities are reusable in future system developments in the family, and also in the analysis of related domains.

2.3.1 Definition of Domain Analysis

It is the analysis of systems within a domain to discover commonalities and differences among them and in which a reusable software architecture and reusable code are defined. It is the process of identifying, organizing and representing the relevant information of a domain.

2.3.2 Domain analysis as the Foundation for Reusability

Domain analysis is the systems engineering of a family of systems in an application domain through development and application of reusable assets.

The domain analysis must be focused to address a family of systems in a particular application domain. The more definitive these families of systems can be defined, the more focused and precise the domain analysis activity can be. In fact, domain analysis is a system engineering activity [18].

Reusability is the central concept of domain analysis. Thus, the conclusion of a domain analysis activity is to develop a reuse library asset that will be used in the implementation of system instances in the domain family. These assets in the library will comprise of software components and generators, documentation, interface specifications, test plans, procedures and data. In addition, the domain model and the generic architecture are themselves valuable reusable assets.

2.3.3 Domain Analysis Methods

By means of domain analysis methods, a more general perception of domain analysis is taken by looking at the existing methodologies to analyze a domain. In general a domain analysis method can be characterized by the process (steps to conduct the analysis), the

product (usually domain models), and supporting tools. There are a number of domain analysis methods but nobody has ever tried to categorize them. Although some authors have compared different methods according to different criteria.

A domain analysis method should have two things [19]:

a. A domain theory, along with an anatomy of this theory both of which will appear in domain model.

b. A process, which, if strictly held, will allow construction of the generic domain model.

The series of steps to be described in the process will be clearly influenced by the type of the domain model. The question is, however, what should be the domain model state; domain model should set out the information required (now and in the future) to allow reuse in the domain. Traditionally, it has been acknowledged that the information involved for modelling consisted of identifying entities, operations, events and relationships in the domain.

However, if DA is applied to other software elements, apart from software products (architectures, code, etc.), the modelling and the way models are built will have to be changed. It means that both the domain model and the model construction process will be very much influenced by the type of element to be reused.

2.3.4 Classification of Domain Analysis Methods

Accordingly, there are following types of DA methods:

a) DA methods for software product reuse
b) DA methods for software process reuse
c) DA methods for software technology reuse
d) DA methods for software experience reuse

2.3.5 Purpose of all Domain Analysis Methods is:

a) Improve reusable elements
b) Populate libraries of reusable elements
c) Integrate DA into the software process
d) Decrease adaptation costs
e) Construct reusable elements

2.3.6 Existing Domain Analysis Methods are:

a) Feature Oriented Domain Analysis Method(FODA)

b) Joint Object Oriented Domain Analysis (JODA)
c) Organizational Domain Modelling (ODM)
d) Domain Specific Software Architecture(DSSA)
e) Domain Analysis and Design Process(DADP)

2.4 Domain Analysis Methods

2.4.1 Feature Oriented Domain Analysis Method (FODA)

The feature-oriented domain analysis method (FODA) was developed at the Software Engineering Institute. It is a domain analysis method which mainly focuses on the "features" of the domain systems. Application in a domain gives many capabilities. These capabilities are modeled in FODA as features. In order to model these features, FODA defines a method for domain analysis which is based on three activities:

a. Context Analysis: The context analysis activity defines the scope of the domain with the objective of scoping the domain under investigation [20].It identifies the relationship between the application in the domain and the elements that are external to the domain. Context analysis is used to define the inputs and outputs to and from the domain. The results from context analysis are the context diagram and structure diagrams. Context diagrams communicate the domain to the environment, while structure diagrams communicate the domain to other domains, which could be part of the original domain.

b. Domain Modeling: The domain modeling process classifies commonalties and variabilities that characterize the applications within the domain, by modeling the functions, data and relationship between applications in the domain. It describes the problem space in the domain that is addressed by software. Domain modeling is composed of following steps:

Information analysis: An informational model describes the applications in terms of entities and their relationship.

Feature analysis: A feature model describes what the applications do in terms of operations.

Operational analysis: It describes the control and data flow in the application domain and the relations between the objects in the informational model and the features of the feature model.

c. Architecture Modeling: Architecture modeling defines a framework for constructing applications in the domain. It identifies concurrent processes and common assets. It also allocates features, function and data objects to the processes and assets.

2.4.2 Joint Object Oriented Domain Analysis (JODA)

The Joint Object Oriented Domain Analysis method advocates the idea that software objects are more understandable and customizable than traditional functions and subroutines [1]. The Joint Integrated Avionics Working Group (JIAWG) reuse subcommittee developed JODA. JODA consists of three phases:

a. Preparing the Domain: It is concerned with collecting information about the domain under consideration either by interviewing domain experts or by reengineering existing system. It also includes investigating the stability and maturity of technologies in the domain and their anticipated future [1].

b. Domain Scoping: It includes definition of the domain services, dependencies, and the development of whole part, subject, and inheritance diagrams [1].

c. Modeling the Domain: The last phase in JODA is modeling the domain by defining the object life histories and state event response, investigating operation scenarios, packaging and grouping reusable objects.

2.4.3 Domain Specific Software Architecture (DSSA)

The DSSA domain models are developed by DARPA for command and control applications [21]. It is more concerned in identifying the models to be created rather than the analysis process. A domain-specific software architecture (DSSA) is the architecture for a specific domain. However, it should be common enough to support a number of applications of the domain .There are five stages in domain specific software architecture.

a. Domain Model: Defines what can be accomplished-emphasize is on user needs. The DSSA Domain Model corresponds to the concept model (i.e. information model in FODA) rather than a full domain model.

b. Reference Requirements: The DSSA Reference Requirements are corresponding to the feature model in FODA. The DSSA Reference Requirements include both functional and non-functional requirements. It emphasizes on problem space.

c. Reference Architecture: DSSA Reference Architecture is architecture for a family of systems including mainly of an architecture model, configuration decision tree, design record (i.e. description of the components), and constraints and rationale. It emphasizes on solution space.

d. Function and Domain Models: Function and dynamic model illustrate the functional process in a method similar to the structured analysis diagram. It includes data flow diagram and control flow diagram. These models offer a hierarchical decomposition of the domain functionality.

e. Object Models: Object model are created using the object modeling technique. These models consist of class diagrams containing class attributes, class methods and generalization and association relationship.

2.4.4 Domain Analysis and Design Process (DADP)

Defense Information System Agency (DISA) developed the Domain Analysis and Design Process. The DADP method takes a problem or solution space approach. The analysis aspect of DADP is concerned with identifying and defining problems within a group of related system in the domain. The design aspect of DADP is concerned with the development of domain-specific solutions in terms of architecture and reusable assets [5]. DADP consist of four phases:

a. Identifying the Domain: The outcome of this step is a set of business models, definition of system capabilities, domain models, and definitions of external interfaces, knowledge reuse opportunities, groups of systems sharing same capabilities, and some descriptions and documentation of current systems and anticipated systems.

b. Scoping the Domain: The domain is scoped with more than three systems in mind. In this phase, the domain analyst also identifies opportunities for reuse among systems in the same domain and reuse across other domains. The domain knowledge and team experiences are also documented.

c. Analyzing the Domain: This step involves analyzing the problem space information. Commonalities and common objects adaptation requirements are identified. Domain models are constructed and verified.

d. Designing the Domain: At this step, the solution space is addressed by providing solution in terms of common designs and implementations of domain objects [22].

2.5 Feature Model

Feature models are modeling details, used in the context of domain analysis, which considers features as the source for analyzing and describing commonalities and variabilities of systems within a domain [23, 24, 25]. Feature modeling is defined as a mechanism to manage variability in a system family. A feature is an attribute or quality of a system which is of interest to a stakeholder and each feature represents a common or variable aspect of a product. A feature model represents the typical features of a family of products in the domain and relationships between them. i.e., tree-like structures consist of nodes that represent features of modeled SPL and their interrelationships. It describes the decomposition of features into sub-features in a hierarchical manner. For each sub-feature below a certain feature it can be specified if it is mandatory, alternative or optional. There are four types of features in feature modeling: mandatory, optional, alternative and or [26].

A mandatory feature have to be integrated in every member of a product line family, if its parent feature is included; optional features characterize choices to choose from, depending on requirements for a particular product. In case of alternative feature, exactly one feature from a set of alternative(XOR) features must be included ,if a parent of the set is included, an (or) feature describes a group of features from which at least one of the features must be included in a product line family. Commonalities between the software product lines are modeled as common features which are named as mandatory features. Variabilities are modeled as variable features which are classified as “alternative features”, “or features” or “optional features”.

At last, feature models also include feature relationships that cannot be captured with a tree structure. Such relationships are called integrity constraints [27]. These constraints

are includes and excludes constraints. An includes constraint specify a relationship between two features that ensures that one feature is chosen when another one is. The excludes integrity constraint specifies a relationship between two features that ensures that one feature is not chosen when another one is.

2.6 Language Extended Lexicon

The Language Extended Lexicon (LEL) was originally created for requirements modeling and is an appropriate model so as to capture the language of a domain in order to identify and define the symbols of that situation, to accomplish an improved understanding of it. The language extended lexicon is an illustration of the symbols in the language. The principle of the language extended lexicon (LEL) is to confine the vocabulary used in the universe of discourse (UofD) [28]: "Universe of discourse is the general perspective where the software should be developed and operated. The universe of discourse includes all the sources of information and all recognized people connected to the software". The LEL aims at registering significant vocabulary in the universe of discourse (UofD). Natural language is the only notation that is commonly readable and understandable by the stakeholders, thus encouraging them to participate dynamically in first steps of software development. Moreover, it eases and encourages stakeholders' contribution. They present an attractive means of communication between the clients/users LEL registers the vocabulary of a given Universe of Discourse. The LEL is anchored to an easy idea: the focus of the LEL is on the language, rather than the details of the problem. As such, lexicon acquisition focuses on the language and not on the problem. Its focus is on the application domain language, rather than the details of the problem. Each entry in the LEL has a name and is composed of notions and behavioral responses. The notions should aim to capture the meaning of the symbol and its relationships with other entries. The behavioral responses state the results, which come from the use of this symbol in UofD, or any effect caused by another symbol to this one.

When describing symbols in the LEL, two principles must be followed: The circularity principle prescribes the maximization of the practice of LEL symbols when describing LEL entries, whereas the minimal vocabulary principle prescribes the minimization of the practice of symbols exterior to LEL when describing these symbols. LEL symbols may

be classified according to its common use in the UofD. One option is to classify each entry in the LEL as a subject (active entity, usually an organization or person that represents significant behavior), an object (passive entity), a verb phrase or a state [29].

Object

Notion: It defines the object and identifies additional objects with which the object has a relationship.

Behavioral response: It describes the actions that might be applied to the object.

Subject

Notion: It defines who the subject is.

Behavioral response: It registers actions executed by the subject.

State

Notion: It describes what it means and the actions which might triggered the state.

Behavioral response: It describes other situations and actions related to it.

Verb phrase

Notion: It describes who executes the action, procedures involved in the action and when it happens.

Behavioral response: It describes the constraints on the occurrence of the action, identifies the action triggered in the environment and new situations that occur as the consequence.

The above definitions suggest what to include in the notion and behavioral response of a symbol according to what the symbol is define.

Chapter 3

Problem Statements

3.1 Problem Statement 1

Some programming tasks cannot be neatly encapsulated in objects or the behavior that doesn't fit into a single module but must be scattered throughout the code called “Code Tangling”. The problem is to handle code tangling.

3.2 Problem Statement 2

There are a number of domain analysis methods. All of these support different quality attributes .All the methods support reusability. The advantage of JODA method is that software objects are more understandable and customizable than traditional functions and subroutines [1] while DADP method corrects much of the vagueness in the process of STARS, particularly in domain model construction and definition [1]. There is a need to check which quality attributes are supported by JODA and DADP.

3.3 Problem Statement 3

Feature models are modelling details, used in the context of domain analysis, which considers features as the source for analyzing and describing commonalities and variabilities of systems within a domain [23, 24, 25]. A feature model is not simply comprehensible to stakeholders. This is mainly inconvenient for interaction with stakeholders during the first stages of software development. For this explanation, natural language is still broadly used to model requirements information. The language extended lexicon (LEL) was originally created for requirements modeling and is an appropriate model to capture the language of a domain in order to identify and define the symbols of that situation, to accomplish an improved understanding of it. To obtain design level of a system there is need to identify steps to transform LEL symbols to UML diagram. There is need to identify reuse in the transformation process from natural language LEL to UML diagrams in feature models.

Chapter 4

Proposed Solutions

4.1 Solution 1: Role of Aspect Oriented Programming in Software Product Line

4.1.1 Why Aspect Oriented Programming (AOP)

Software product lines signify a successful approach to software reuse. Instead of motivated for general component reusability, every product lines targets a specific domain. Reusability is achieved by separating common features from variable features and projecting this along the design and implementation lines. Preferably, an exact application is developed by configuring the product line, by selecting variable features to be included. Furthermore, the new artifacts may be incorporated into the product line, too. The requirements of a software system includes functional requirements (e.g. data query etc) as well as non-functional requirements (e.g. performance etc). Non-functional requirements are likely to cut across functional requirements. Some concerns cannot be tidily separated and hence, they are scattered across numerous modules in the software system. Such concerns are referred as crosscutting concerns because they are realized by fragments of code that allow identical behavior across several modules. Maintaining a crosscutting concern is a way of modifying each fragment of the scattered code realizing that concern. The application developer frequently programs this part in the interiority of components, which leads to the development efficiency reduction and is simple to cause code-tangling and code-scattering [30]. The fact that multiple concerns from one dimension are realized by one concern of another dimension is called tangling while scattering means that a concern of one dimension is realized by multiple concerns of another dimension [33]. This increasing the coding time, error proneness, and the maintenance cost. Ideally, the modular structure of the core and variable features of a SPL must not be submitted in the presence of change requests. The inefficacy of the variability mechanisms to hold changes might guide to several undesirable consequences associated to the product line stability, including invasive wide changes, significant ripple

effects. On the other hand, in despite of introducing crosscutting to implement the separation of components and aspects, the interaction between them is puzzled. It brings difficulties to the software reuse. Development of product lines is a complex process. As in any software development, this is in part due to crosscutting concerns. This is mainly obvious in the product line configuration in which features that signify crosscutting concerns cannot be easily included nor taken out once they have been included. AOP is aimed at sustaining the encapsulation of crosscutting features into new modular units - the aspects - all the way through new composition mechanisms. The goal is to make the variation of crosscutting features more modular and evolvable.

Aspect-oriented techniques redound to improving the design level of software, the implementation of separation of concerns and the reusability of components. Separation of concerns is at the heart of software development. It will help to enhance understanding of the software system and strengthen the adaptability, maintainability and reusability of the final software [31].The primary idea of AOP is to encapsulate the crosscutting behavior into modular units known as aspects. These units are composed of advices that realize the crosscutting behavior, and point-cut descriptors, which assign the points in the program where the advices are included. Aspect-oriented techniques allow the handling of crosscutting, improve the software reuse and remove the code-tangling and code-scattering. The significant features provided by aspect-oriented languages were meant to enable developers to encapsulate tangled code in a very flexible way, hence improving maintainability of the system by allowing developers to modify single units instead of scattered code fragments.

4.1.2 Commonalities between SPL and AOP

The commonalities between SPL and AOP are as follows:

a. Software variability impacts a software system in a related method as crosscutting concerns do. In this way, we have a connection between aspects (crosscutting concerns) and variability. Both aspects and variability are orthogonal concepts which are independent of the core system (commonality /core concerns) and can freely be joint with it.

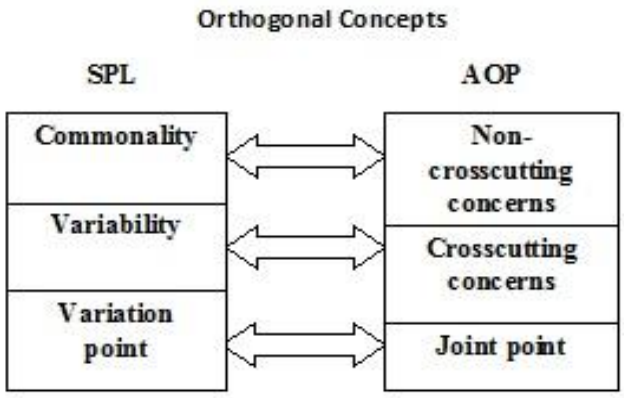

Fig 4.1: Commonalities between SPL and AOP

b. Moreover, software variability may impact software commonalities as well as other software variability's. In the same way, crosscutting concerns (aspects) may impact non-crosscutting ones

as well as other crosscutting concerns. As a result, aspect-oriented techniques can also be used for modelling the variable and the common concerns in a software product line individually.

c. Software commonalities execute like non-crosscutting concerns. In this way, we have a connection between commonality and core concerns.

d. Software Variation points in software are the locations in features (common features) where other features can be added (variable features) to construct the various products. In the same way, joint points in AOP are the places where crosscutting concerns can be woven in. In this way, we have a connection between variation point and joint point.

Fig 4.1 represents the commonalities between SPL and AOP.

4.2 Solution 2: Quality Attributes in Domain Analysis

4.2.1 ATM Case Study

ATM machine is a computer. It has a small display and a control panel like a keyboard for input. It runs a program that is usually written by the bank. The program by banks usually follow a pattern. Most banks start by asking for the language. Then it asks you to insert your card. This card has a magnetic strip on the back that it reads information about your bank account .It uses this information to look up your information and decide what to do next. Usually this is followed by entering your password by you. After that, it's up to the program to decide what to do next.

4.2.2 Joint Object Oriented Domain Analysis (JODA) for ATM

a. Preparing the Domain

The actors of the ATM case study identified are user, ATM machine, bank. The functions (use-cases) of the ATM system are: login; get balance information; withdraw cash; transfer funds; deposit cash. The following constraints are identified while investigating the stability and maturity of technologies in the domain of ATM are ATM PIN encryption should be triple DBS and DSS compliant, PIN blocks should never be stored in ATM log files, personal account number (PAN) should be truncated and protected in ATM logs, secure remote access controls must be established for all ATMs and antivirus and malware detection system should be installed, updated and tested frequently.

b. Domain Scoping Application Engineering

Features(Services) identified for ATM system are deposit cash or cheques into your accounts, transfer funds between your own accounts, check your balance, change your PIN, view transactional history, order statements/cheque books,send in other instructions to the bank via the mail deposit system, make payments to other same bank customers. ATM card also be used against your foreign currency accounts. With this card, you can withdraw cash from your accounts at 500,000 ATMs worldwide.

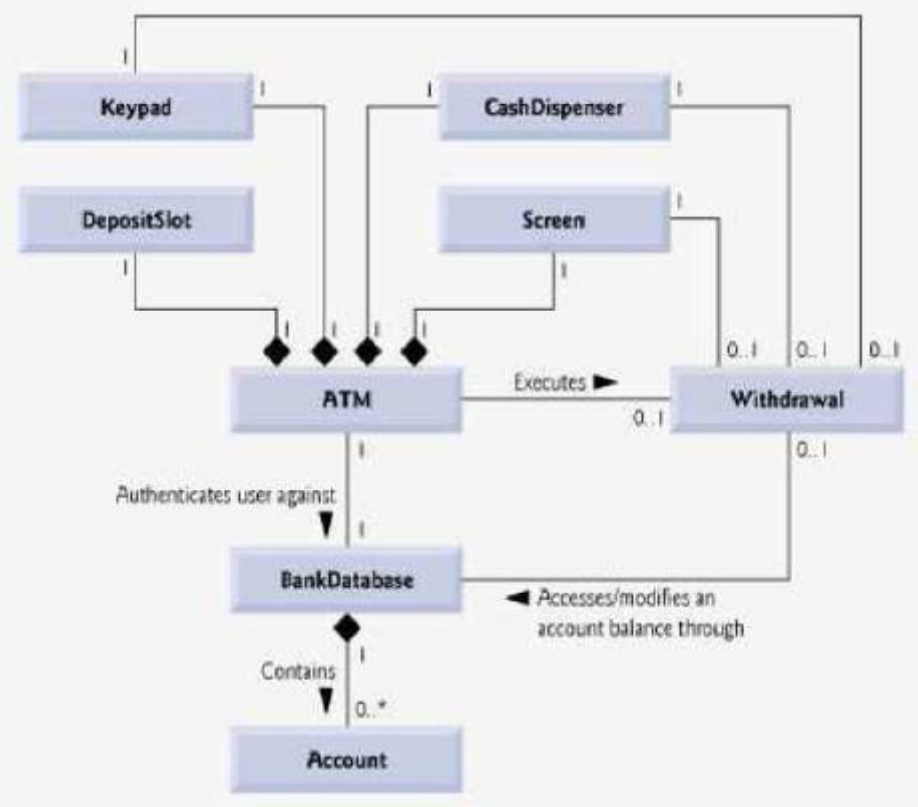

Fig 4.2: Class diagram of ATM [22]

To identify the dependencies, inheritance and whole part relationships, the class diagram is drawn as shown in Fig 4.2.The above diagram supports cohesion and coupling.

c. Modelling the Domain

The object diagram also supports cohesion and coupling. The objects identified are shown in object diagram in Fig 4.3 below. The operations scenarios are shown as state chart diagram in Fig 4.4.

A customer can perform many transactions while at an ATM. A possible number of transactions a customer might perform at a single visit to an ATM are: Deposit money into a checking account, transfer funds from a checking to a savings account, withdraw money from checking and print balance.

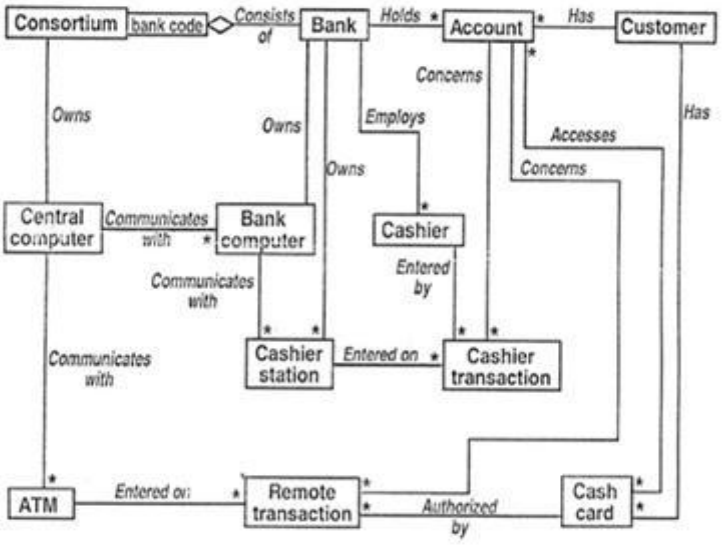

Fig 4.3: Object diagram of ATM [34]

At least four occasions require access to the customer's balance. So the point for code reusability has been identified during domain analysis. You can write a function that contains the logic and structures to handle the access to the customer's balance and then reuse that function when needed. Code reusability saves programming time immediately and for changes, if any in future.

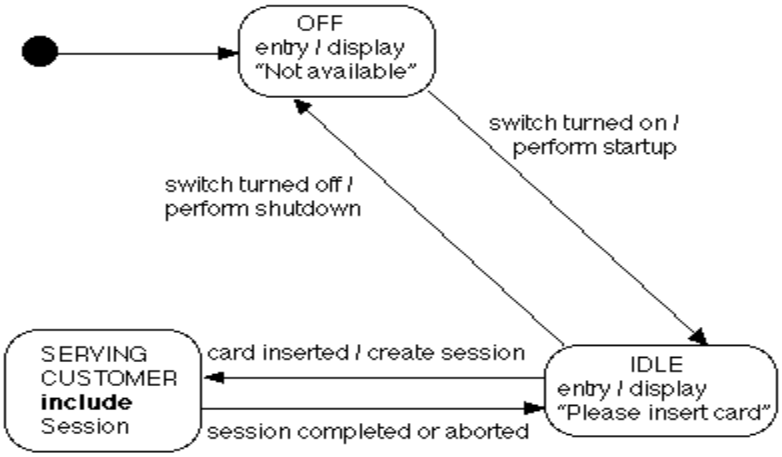

Fig 4.4: State Chart diagram of ATM [34]

4.2.3 Domain Analysis and Design Process for ATM

a. Identifying the Domain

The actors of the ATM system identified are User, ATM machine and bank. The functions (use-cases) of the ATM system are login, get balance information; withdraw cash; transfer funds and deposit cash. In Fig. 4.5 , the use case diagram of ATM is shown.

System capabilities of ATM system identified are : Deposit cash or cheques into your accounts, transfer funds between your own accounts, check your balance, change your PIN, view transactional history, order statements/cheque books, send in other instructions to the bank via the mail deposit system and make payments to other same bank customers.

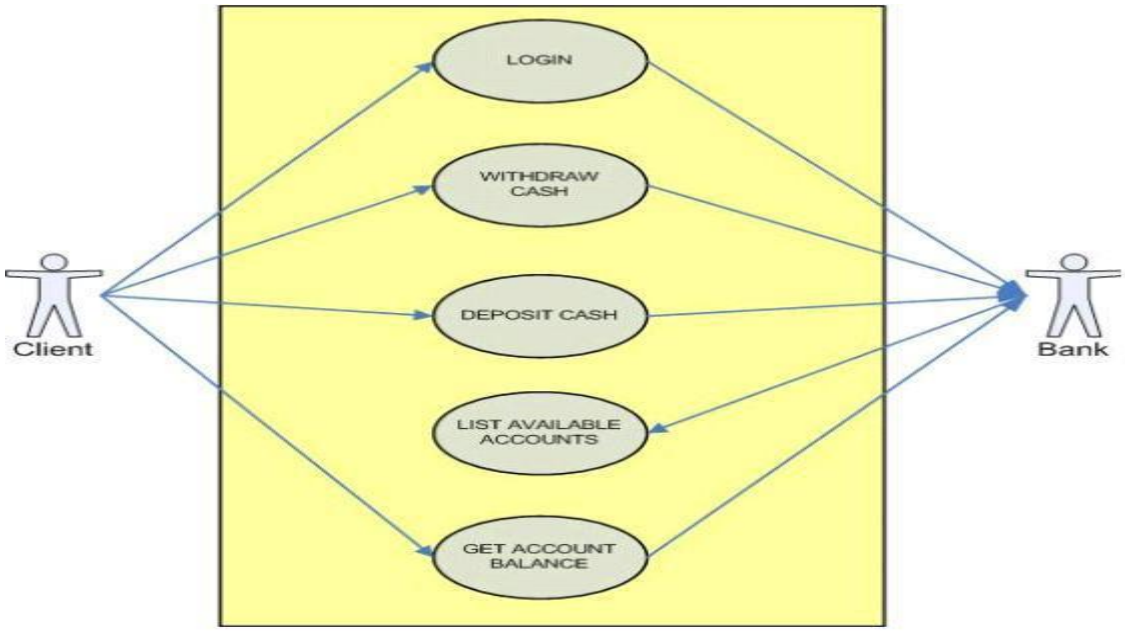

Fig 4.5: Use Case diagram of ATM

ATM card also be used against your foreign currency accounts. With this card, you can withdraw cash from your accounts at 500,000 ATMs worldwide.

External interface of ATM identified are

User Interfaces: The interface of the ATM must fulfill ergonomic requirements.

Hardware Interfaces: The ATM network has to provide hardware interfaces to various printers, various ATM machines and several types of networks.

Software Interfaces: The ATM network has to provide software interfaces to the software used by different banks and different network software.

Communication Interfaces: There is no restriction of the ATM network to a specific network protocol as long as the performance requirements are satisfied.

b. Scoping the Domain

In this phase, the domain analyst also identifies opportunities for reuse among systems in the same domain and reuse across other domains. The domain knowledge and team experiences are also documented.

c. Analyzing the Domain

The problems identified for ATM system are [8]: Transactional secrecy and integrity, customer security, bank on our security, device operation integrity, increased availability, accelerated speed to market and customer identity integrity.

There have also been a number of incidents of fraud by man-in-the-middle attacks, where criminals have attached fake keypads or card readers to existing machines. These have then been used to record customers' PINs and bank card information in order to gain unauthorized access to their accounts.

There are many possible errors in the system. These errors may be mechanical (such as card transport mechanisms, keypads, hard disk failures, envelope deposit mechanisms), software (such as OS, device driver; application), communications, or purely due the operator errors.

The Commonalities identified are:

User interface are: A screen to displays messages, a keypad for numeric input, a cash dispenser, a deposit slot.

ATM session includes: Display a welcome message and prompt the user to enter an account number, the user enters a five-digit account number, using the keypad, the screen prompts the user to enter the PIN, the user enters a five-digit PIN, using the keypad, if the user enters a valid account number and the correct PIN for that account, the screen

displays the main menu and if the user enters an invalid account number or an incorrect PIN, the screen displays an appropriate message, then the ATM returns to Step 1 to restart the authentication process.

ATM Main menu includes: View my balance, withdraw cash and deposit funds.

d. Designing the Domain

At this step, the solution space is addressed by providing solution in terms of common designs and implementations of domain objects [22].

Transactional secrecy and integrity: The security of ATM transactions relies mostly on the integrity of the secure processor which may use encryption of personal information, using digital encryption scheme (DES**)** or remote key loading techniques. Message authentication code (MAC) or partial MAC may also be used to ensure messages have not been tampered with while in transit between the ATM and the financial network**.**

Customer identity integrity: ATM manufacturers have put in place counter measures to protect the equipment they manufacture from threats. Alternate methods to verify cardholder identities such as finger and palm vein patterns, iris, and facial recognition technologies have been deployed in some countries.

Device operation integrity: Openings on the customer-side of ATMs are often covered by mechanical shutters to prevent tampering with the mechanisms when they are not in use. Alarm sensors are placed inside the ATM and in ATM servicing areas to alert their operators when doors have been opened by unauthorized personnel.

Customer security: Security cameras can be placed at the cabin where ATM machine is placed or security guards can be deputed at ATM machine cabins.

Reliability: When an ATM is placed in a public place, it typically has undergone extensive testing with both test money and the backend computer systems that allow it to perform transaction.

Card fraud: In an attempt to prevent criminals from shoulder surfing the customer's PINs, some banks draw privacy areas on the floor.

Increase Availability: Software diagnoses and fixes problems at the terminal level. Minimize downtime and lower costs.

Accelerate Speed to Market: A rich set of built-in business functions and an application based on the fundamentals of self-service best practices means integration in weeks not months and deployment of new features in hours, not days.

Bank on Our Security: Use advanced lockdown systems to protect customer data and built-in package signing to block rogue applications.

4.3 Solution 3: Transformation from LEL to UML Diagrams

4.3.1 The Transformation Process from LEL Symbols to UML Class Diagram

In this section we describe a transformation process to derive a UML class diagram from a natural language oriented requirements models (LEL). The process starts from well-known natural language requirements models, the Language Extended Lexicon (LEL) Model, which explains the vocabulary in a Universe of Discourse (UofD). The process consists in the application of a set of transformation rules to derive classes, attributes, methods and relationships taking into account the structure and semantic of LEL. The LEL symbols are used as source and defines the classes, attributes, methods and relationships in the derivation of the UML class diagram. The process consists of a set of steps that transform natural language oriented models (LEL) to a UML class diagram [35].

The Transformation Rules

Rule 1: **Transformation of subject to class**

a. Each LEL subject becomes a UML class.

b. For each entry in the notion of this LEL symbol that does not contain a LEL, the transformation identifies each noun and defines them as attributes.

One of the major problems of this transformation is that it misses noun groups. As the current implementation only detects separate nouns, every noun is a possible attribute, thus generating more and sometimes inappropriate attributes. Thorough noun groups detection may be included following various linguistic approaches.

Rule 2: **Transformation of object to class**

a. Each LEL object becomes a UML class.

b. For each entry in the notion of this LEL symbol that does not contain a LEL symbol, the transformation identifies each noun and defines them as attribute.

c. Methods to access and modify each attribute are defined by adding SET and GET prefixes for each attribute.

Rule 3: **Transformation of subject behavioral response to method**

a. Each entry in the behavioral response of a LEL subject that was modelled as a class by Rule 1 becomes a method of this class.

Rule 4: **Transformation of subject information to method parameter**

a. Each scenario comes from an entry in the behavioral response of a LEL subject that was modelled as a UML class.

b. The rule models actors and resources of each scenario as parameters of the method obtained by Rule 3 from the entry in the behavioral response that originated the scenario. The actor referring to the subject LEL symbol in consideration is excluded.

Rule 5: **Transformation of LEL relationships to class relationships**

This transformation applies to subject as well as object LEL symbols

a. The entries in the notion of each symbol in the LEL (called L1) modelled as a class is analyzed in order to detect other symbols in the LEL modelled as classes.

b. For every detected LEL symbol (L2), this rule defines a relationship between the corresponding classes, analyzing the verb involved to determine the type of relationship, taking into account the following issues**:**

Inheritance relationships: L1 and L2 have the same classification (object or subject). As well, L1 appears in one of the entries of the notion of L2. The concerned entries of L1 and L2 contain, in a complementary way, two kinds of verbs [36]: bottom-up verbs (is a, is a type of, is a class of) or top-down verbs (is, may be, may be classified as, classifies as). The rule does not consider if the methods of the subclasses are refinements of the corresponding superclass methods. The software engineer is the one who must take the appropriate decision depending on the semantic of every case.

Aggregation relationships: In the entries of the notion of the LEL symbol considered as container, verbs of the type "component_composition_verb" must show[36]: "to consist /

to contain / to include / to form, to compose, to divide" (these three last in passive voice).In the entries of the notion of the "component" symbol, verbs of the type content_composition_verb must appear [36]. Since it is not likely to automatically distinguish between an aggregation or a composition relationship, the transformation rule defines the relationship as an aggregation.

Association relationships: Any relationship between LEL symbols that does not represent a relationship of the previous types represents an association. The verb in the entry of the notion (classified as general verb in [36]) is taken as the name of the association.

It is important to mention that this strategy must be complemented with the participation of software engineers who will adjust the results obtained after the application of the transformation rules.

4.3.2 The Transformation Process from Class Diagram to Object Diagram

In this section we describe a transformation process to derive object diagram from class diagram. Object diagrams represent an instance of a class diagram. The elements in class diagram are in abstract form which represents the blue print and the elements in object diagram are in concrete form which represents the real world object. The UML class diagram is used as source and creates and name the instance of each class in the derivation of the object diagram. The process consists of a set of steps that apply transformation rules to UML class diagram to define object diagram.

Transformation rules

Rule 1: Create an instance of each class used in the class diagram and name the instance.

Rule 2: In the object diagram refer the object as instancename:classname

Rule 3: Assign values to each of the attribute of the object corresponding to the definition in the class.

Rule 4: Create an object corresponding to each class in the class diagram.

Rule 5: The relationship between the objects in the object diagram is same as the corresponding relationship in the classes in the class diagram.

The transformation process from LEL to UML diagrams is verified by using six case studies. These case studies are as follows: Admission process, online transaction system, ATM, online dairy farm, online book shop and online potato collection company.

4.3.3 Case Study of the Admission Process

Admission process describes software systems which enable organizations to do admissions over the Internet. They must be able to deal with a large number of students, and coordinate multiple resources to deliver the services to the students. This process enables admissions of students where organizations provide services to an individual. This is done through an admission process web site.

Feature Model of the Admission Process

The relationship between the student registration and entering data features is a mandatory feature; the relationship between course details and duration features is an optional feature. The set consisting of the manual and automatic features - fraud detection in an admission process application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the credit card, debit card, and money order features - to enable users to pay for selected courses, at least one payment method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.6 shows a feature diagram of an admission process that represents all possible services of admission process. An admission process typically consists of three parts, one dedicated to the interface that the student uses for registering and one dedicated to the interface that the student uses for browsing the course catalog and one concerned with administration/back office operations. The student registration enables the entering data and entering personal information. During the process of entering data, users specify their passwords. The password sub-tree on the left hand side specifies that each admission process application has their own user password specification and expiration policy. A user password specification policy includes the use of lower case (L. Case) and upper case (U. Case) characters as well as digits. A user password may never expire, or the expiry may be specified in days. During the process of entering personal information it includes fill student admission form. Further fill student admission form might mandate

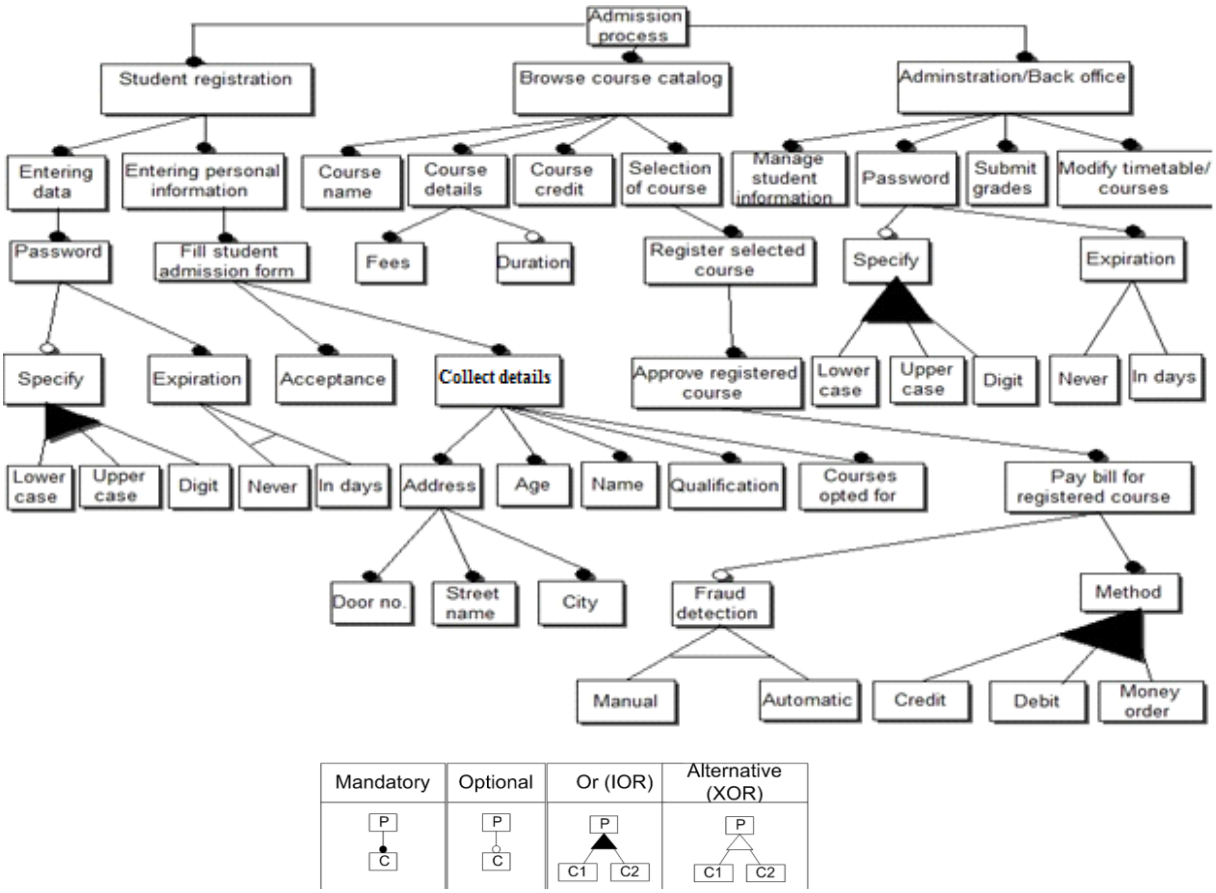

Fig 4.6: Feature model of admission process

the use of details collection and acceptance. During details collection it mandate the use of name, age, qualification, address and courses opted for. An address might mandate the use of door no., street name and city.

During the process of browse course catalog, it enables the course name, course details, course credit and selection of course. The course details sub-tree on the left hand side specifies that each course has their own fees and duration. The selection of course enables register selected course which further enables approve registered course. The approve registered course might mandate the use of pay bill for registered course. In addition, the pay bill for registered course allows for different payment methods and fraud detections. Administration/back office operations for this admission process are operations of manage student information of the admission process effectiveness, modification of timetable and courses (modify timetable/courses), submit grades and

password. The latter consists of managing information in the administration/back office such as product data (Content Management) and managing operational concerns of the admission process such as site search and domain name setup (Store Administration). Each administrator/back office of an admission process must have their own password as specified by the Password sub-tree on the right hand side. A password policy for administrating an admission process application includes the use of lower case (L. Case) and upper case (U. Case) characters as well as Digits and a password may Never expire, or the expiry may be specified. In the feature model in Fig 4.6, an includes constraints exist between the student registration and the browse course catalog features. An excludes integrity constraint exists between the Credit Card and the Manual features. Introducing semantics to the feature model of a system is suggested through natural language LEL specifications. These specifications explain constraints and preconditions, also allowing to specify relations for features.

The features customer registration, payment, fraud detection, manual, automatic, method, credit, debit and money order of case study online transaction system are reused in this case study. The relationship between student registration and pay bill for registered course in admission process is similar to the relationship between customer registration and payment in online transaction system.

Language Extended Lexicon Symbols for the Admission Process

A feature model is not simply comprehensible to stakeholders. This is mainly inconvenient when interaction with stakeholders during the first stages of software development. For this explanation, natural language is still broadly used to model requirements information. The entire elements of the model must pursue the same vocabulary as used in the lexicon. The words that are highlighted are signs of the LEL, used in accordance with the principle of circularity. LEL symbols, taken from the LEL of the admission process case study are given below, dotted line represents that many more can be included to it:

Admission process (Subject)

Table 4.1.1: Admission process LEL symbol

Notion	Behavioral Response
• It is a software system for the admission of students. • It is composed of **registration**, **browse course catalog** and **administration/back office**. • It has a starting date. • It has an application form. • It has an approximated period of duration in days. • It may have one or more employee.	• It may include special characters with them. • It may enable **student registration**. • It may enable **browse course catalog**. • It may enable **administration/back office**.

Student registration (Subject)

Table 4.1.2: Student registration LEL symbol

Notion	Behavioral Response
• It is the process for the **registration of student**. • It contains **entering data** and **entering personal information**. • It is a component of **admission process**. • It has one or more employee. • It has a form. • It has identification for students.	• It registers the student. • It may **enter personal information** of student. • It may **enter data**.

Browse course catalog (Subject)

Table 4.1.3: Browse course catalog LEL symbol

Notion	Behavioral Response
• It is the process of **browsing course catalog** by a student • It is a component of **admission process**. • It is composed of **course name, course details, course credit** and **selection of course**. • It has a list. • It can search according to author. • It can search according to version number.	• It may enable **course details**. • It may enable list of **course name**. • It may enable list of **course credits**.

Administration/back office (Subject)

Table 4.1.4: Administration/back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access **admission process**. • It is a component of the **admission process**. • It is composed **of manage student information, modify timetable/courses, submit grades** and **password**. • It may have one or more employee. • It has a unique identification.	• It **manages student information**. • It **modifies timetable/courses**. • It **submits grades** of student. • It manages **password**.

Entering data (Verb)

Table 4.1.5: Entering data LEL symbol

Notion	Behavioral Response
• It is the process of **entering data** of a student. • Performed by students. • It contains **password**.	• It accepts the **entered data**. • It manages the **password**

Password (Object)

Table 4.1.6: Password LEL symbol

Notion	Behavioral Response
• It is the process of providing security to the data entered by the student. • It is a component of **entering data**. • It contains **specify** and **expiration**. • It has a size. • It has unique identification. • It has an approximated period of duration in days.	• It provides authorization to student to access data related to **password.** • It determines **expiration**. • It maintains **specify**.

Specify (Verb)

Table 4.1.7: Specify LEL symbol

Notion	Behavioral Response
• It is the process of specifying **password**. • It is composed of **lower case, upper case** and **digit.** • Performed by students.	• It manages the type of **password**. • **Password** is selected.

Lower case (Object)

Table 4.1.8: Lower case LEL symbol

Notion	Behavioral Response
• It defines that the **password** is **specify** in **lower case.** • It has small alphabet. • It may have size.	• It may include special characters with them.

Upper case (Object)

Table 4.1.9: Upper case LEL symbol

Notion	Behavioral Response
• It has capital alphabet. • It defines that the **password** is **specify** in **upper case.** • It may have size.	• It may include special characters with them.

Digit (Object)

Table 4.1.10: Digit LEL symbol

Notion	Behavioral Response
• It has number. • It defines that the **password** is **specify** in digits.	• It may not have repetition of **digits**.

Expiration (Verb)

Table 4.1.11: Expiration LEL symbol

Notion	Behavioral Response
• It is the process of **expiration** of **password**. • It is composed of **never** and **in days**. • Performed by students.	• It may **never** expire. • **Password** may expire **in days**.

Never (Object)

Table 4.1.12: Never LEL symbol

Notion	Behavioral Response
• It defines that **password** may **never** expires.	• It may not have any fixed duration.

In days (Object)

Table 4.1.13: In days LEL symbol

Notion	Behavioral Response
• It defines that **password** may expires **in days**. • It has number of days. • It may be a week. • It has a starting date.	• It may provide some fix days.

Entering personal information (Verb)

Table 4.1.14: Entering personal information LEL symbol

Notion	Behavioral Response
• It is the process of **entering personal information** of students. • Performed by the student. • It is composed of **fill student admission form**.	• It may **fill student admission form**. • It accepts the **entered personal information** by student.

Fill student admission form (Verb)

Table 4.1.15: Fill student admission form LEL symbol

Notion	Behavioral Response
• It is the process of **filling student admission form**. • Performed by student. • It contains **details collection** and **acceptance.**	• It may accept student **admission form**. • It gives **details collection**.

Collect details (Subject)

Table 4.1.16: Collect details LEL symbol

Notion	Behavioral Response
• It is the process of collecting student details. • It contains **name, age, qualification, address** and **courses opted for**. • It is a component of fill student admission form.	• It provides the details of student.

Name (Subject)

Table 4.1.17: Name LEL symbol

Notion	Behavioral Response
• It is the **name** of student. • It is a component of **details collection**. • It has assigned a unique identification. • It has assigned a unique roll no.	• It provides the details of student **name**.

Age (Object)

Table 4.1.18: Age LEL symbol

Notion	Behavioral Response
• It defines the **age** of student. • It is a component of **details collection**. • It may be in years. • It may be calculated **in days**.	• It may have **age** relaxation for students.

Qualification (Object)

Table 4.1.19: Qualification LEL symbol

Notion	Behavioral Response
• It defines the **qualification** of student. • It is a component of **details collection**. • It has a degree. • It may be in years.	• It is the eligibility of a student to get admission.

Address (Object)

Table 4.1.20: Address LEL symbol

Notion	Behavioral Response
• It is where the student **address** is given. • It is a component of **details collection**. • It contains **door no. street name** and **city.** • It has house no. • It has a size. • It has pin code. • It may be in number and alphabet.	• It provides **door no**. to the **address** of student. • It provides **street name** to the **address** of student. • It provides **name** of the **city** to the student **address**. • Student fills all the fields of **address.**

Door no. (Object)

Table 4.1.21: Door no. LEL symbol

Notion	Behavioral Response
• It defines the **door no**. in the **address** of a student. • It is a component of **address**. • It may have number.	• It must be in numeral form. • Must have unique number.

Street name (Object)

Table 4.1.22: Street name LEL symbol

Notion	Behavioral Response
• It is the **name** of street in the **address** of a student. • It is a component of **address.** • It may be in alphabets. • It has a size.	• Must be in alphabets. • No special characters are used to display **street name.**

City (Object)

Table 4.1.23: City LEL symbol

Notion	Behavioral Response
• It is the name of **city** in the **address** of student. • It is a component of **address**. • It is associated with **address** of student. • It may be in alphabets. • It has a size.	• Must be in alphabets. • It provides **name** of **city** to which student belongs. • Must have unique pin code.

Courses opted for (Object)

Table 4.1.24: Courses opted for LEL symbol

Notion	Behavioral Response
• It is where the student selects the **courses opted for.** • It is a component of **details collection**. • It has a unique identification number. • It has an author. • It has a title.	• Student can opt one or more courses.

Acceptance (Verb)

Table 4.1.25: Acceptance LEL symbol

Notion	Behavioral Response
• It is the process of acceptance of student admission form. • It is a component of fill student admission form.	• It accepts the student admission form.

Course name (Object)

Table 4.1.26: Course name LEL symbol

Notion	Behavioral Response
• It describes the **name** of the courses. • It is a component of **browse course catalog**. • It has a list. • It may be in alphabets.	• Each **course name** must have unique id. • Student choose course according to **course name**. • It displays the list of courses.

Course details (Object)

Table 4.1.27: Course details LEL symbol

Notion	Behavioral Response
• It describes **course details**. • It is a component of **browse course catalog**. • It contains **fees** and **duration**. • It has a list. • It assigns a unique identification. • It assigns author. • It assigns a title. • It has an approximated period of duration in days.	• May provide **fees** of the course chosen by student. • May provide **duration** the of course

Fees (Object)

Table 4.1.28: Fees LEL symbol

Notion	Behavioral Response
• It is the process of displaying **fees** for courses. • It is a component of **course details**.	• **Fees** are according to the courses.

Duration (Object)

Table 4.1.29: Duration LEL symbol

Notion	Behavioral Response
• It describes the time period of a course. • It is a component of **course details**. • It has a starting date. • It has an approximate period of time in days.	• May be in week, month or year.

Course credit (Object)

Table 4.1.30: Course credit LEL symbol

Notion	Behavioral Response
• It is where the **course credits** are given. • It is a component of **browse course catalog**. • It may be in number.	• Must be in numeral. • Each course is assigned only unique credit.

Selection of course (Verb)

Table 4.1.31: Selection of course LEL symbol

Notion	Behavioral Response
• It is the process of **selection of course** by a student. • It contains **register selected courses**. • Performed by the student.	• It displays the list of courses selected by the student.

Register selected course (Verb)

Table 4.1.32: Register selected course LEL symbol

Notion	Behavioral Response
• It is the process of **registration of selected courses.** • Performed by the student. • It is composed of **approve registered course**.	• It assigns **registered selected courses** to the student.

Approve registered course (Verb)

Table 4.1.33: Approve registered course LEL symbol

Notion	Behavioral Response
• It is the process where **administration/back office approves registered course**. • Performed by **administration/back office**. • It contains **pay bill for registered courses**.	• The student can access to registered courses.

Pay bill for registered course (Verb)

Table 4.1.34: Pay bill for registered course LEL symbol

Notion	Behavioral Response
• It is the process of **paying bill for registered course**. • Performed by the student. • It contains **fraud detection** and **method.**	• The student has access to registered course. • May have **fraud detection**. • May have **methods** to **pay bill for registered course.**

Fraud detection (Subject)

Table 4.1.35: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting fraud in case of **paying bill for registered courses**. • It is composed of **automatic** and **manua**l. • It has transaction identification. • It is a component of **pay bill for registered course.**	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.1.36: Manual LEL symbol

Notion	Behavioral Response
• It is a manual process of **fraud detection**. • It has identification.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.1.37: Automatic LEL symbol

Notion	Behavioral Response
• It is an **automatic** process of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Method (Subject)

Table 4.1.38: Method LEL symbol

Notion	Behavioral Response
• It is the process by which a student can **pay bill for registered course.** • It contains **credit, debit** and **money order**. • It is a component of **pay bill for registered course**. • It has a secret code. • It has a machine.	• It **pays bill for registered course**.

Credit (Object)

Table 4.1.39: Credit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student can **pay bill for registered course.** • It has a card. • It has a bank account. • It has a account holder. • It has card no.	• Student has to enter pin number.

Debit (Object)

Table 4.1.40: Debit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student **can pay bill for registered course.** • It has a card. • It has a secret code. • It has a bank account. • It has a account holder.	• Student has to enter pin number.

Money order (Object)

Table 4.1.41: Money order LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student can **pay bill for registered course.** • It has a account. • It has a customer. • It has a receipt slip.	• Student has to enter money order number.

Manage student information (Verb)

Table 4.1.42: Manage student information LEL symbol

Notion	Behavioral Response
• It is the process of **managing student information.** • Performed by the **administration/back office**.	• Update student information. • Edit student information. • Available student information to **administration/back office**.

Modify timetable/courses (Verb)

Table 4.1.43: Modify timetable/courses LEL symbol

Notion	Behavioral Response
• It is the process of **modifying timetable/courses** of students. • Performed by the **administration/back office.**	• Edit timetable/courses of students. • May update timetable/courses of students. • May available timetable/courses of students.

Submit grades (Verb)

Table 4.1.44: Submit grades LEL symbol

Notion	Behavioral Response
• It is the process of submission of grades of students. • Performed by the **administration/back office**.	• May update grades. • May edit grades of students. • Display grades of students. • Students gets report card.

The Transformation Process Using a Case Study of Admission Process

The transformation process takes as the source model a LEL model from a case study of the admission system, and follows the steps described below to derive a UML class diagram from LEL symbols of the admission system:

The Transformation Rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol admission process that is Table 4.1.1, the class shown in Fig 4.7 is defined.

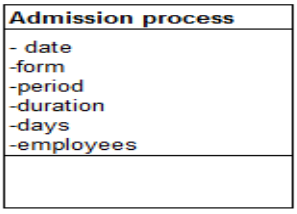

Fig 4.7: Admission process class

As Rule1 indicates, second entry in the notion is discarded because it contains another LEL symbol; from the other five entries, the rule identifies nouns and defines them as attributes.

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol Password that is Table 4.1.6, gives as result the class and attributes shown in Fig 4.8. The following

attributes have been obtained: identification, size, period of duration in days (Rule 2 takes each of them separately). In the last case, the problem is that the dictionary does not recognize noun groups, as we have mentioned before. Moreover, the attribute days obtained by applying Rule 2 would not be an attribute following the manual approach because human judgment would have realized they are the way in which periods are considered.

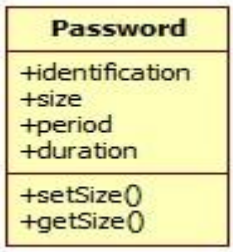

Fig 4.8: Password class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the LEL symbol admission process that is Table 4.1.1, the methods described in Fig 4.9 are obtained.

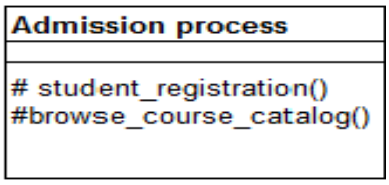

Fig 4.9: Methods of admission process

Rule 4: **Transformation subject information to method parameter**

For example, for each method previously defined by Rule 3 (Fig 4.9), parameters are identified considering the scenarios involved. As parameters come from resources and actors, they are modeled as classes when the corresponding resource and actor is a subject or object LEL symbol (Rule 1 and Rule 2).When the resource or the actor does not belong to the LEL, two things may occur. It may be a word that does not need a LEL entry because it belongs to the minimum vocabulary or it may represent a set. In the

former case, it is modeled with a primitive class or type, and in the later one no new classes are needed because the parameter is a set of a class which is already defined.

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with method as the superclass and credit, debit and money order as subclasses.

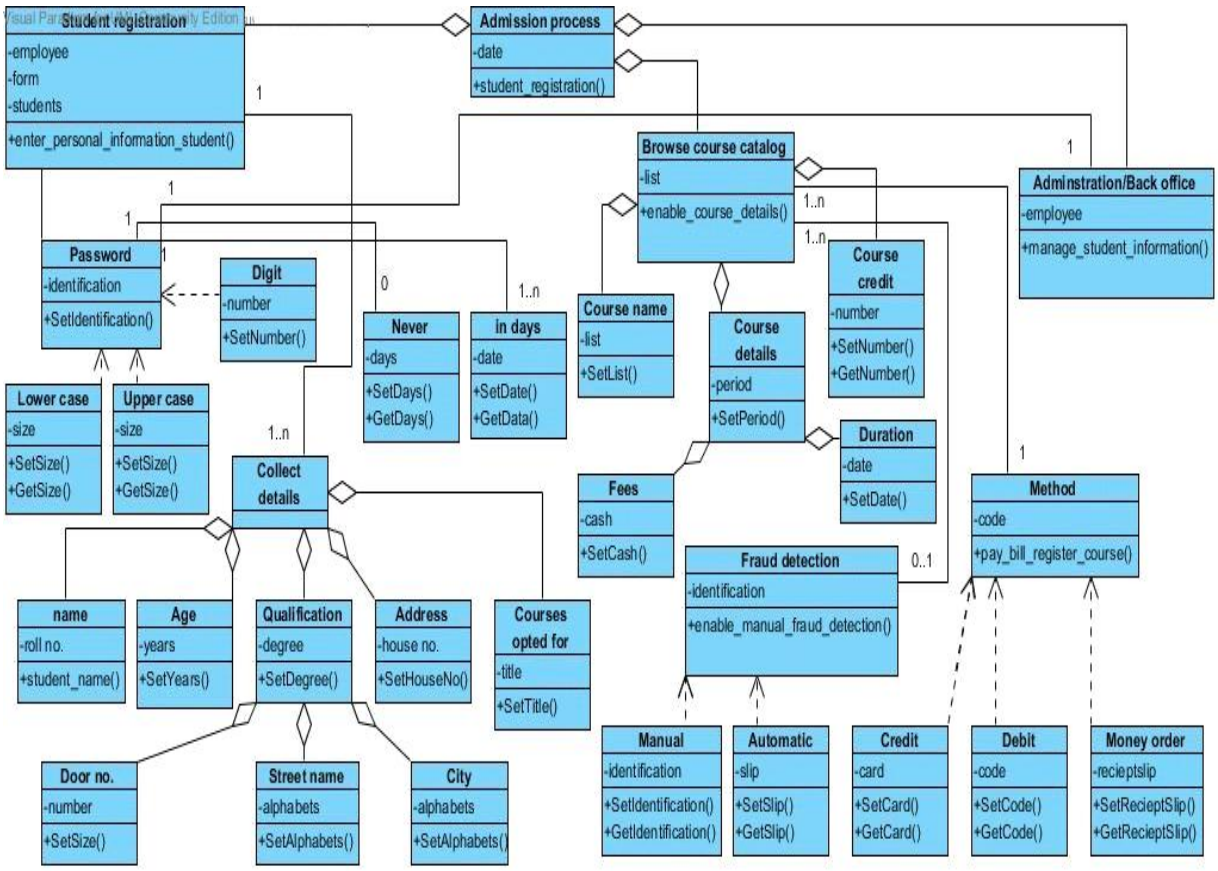

Fig 4.10: UML Class diagram of admission process

Method (Subject)

Notion

- It is the **method** by which a student can **pay bill for registered course.**

……

Credit (Object)

Notion

- It is the **method** through which a student can **pay bill for registered course.**

Debit (Object)

Notion

- It is the **method** through which a student can **pay bill for registered course.**

……

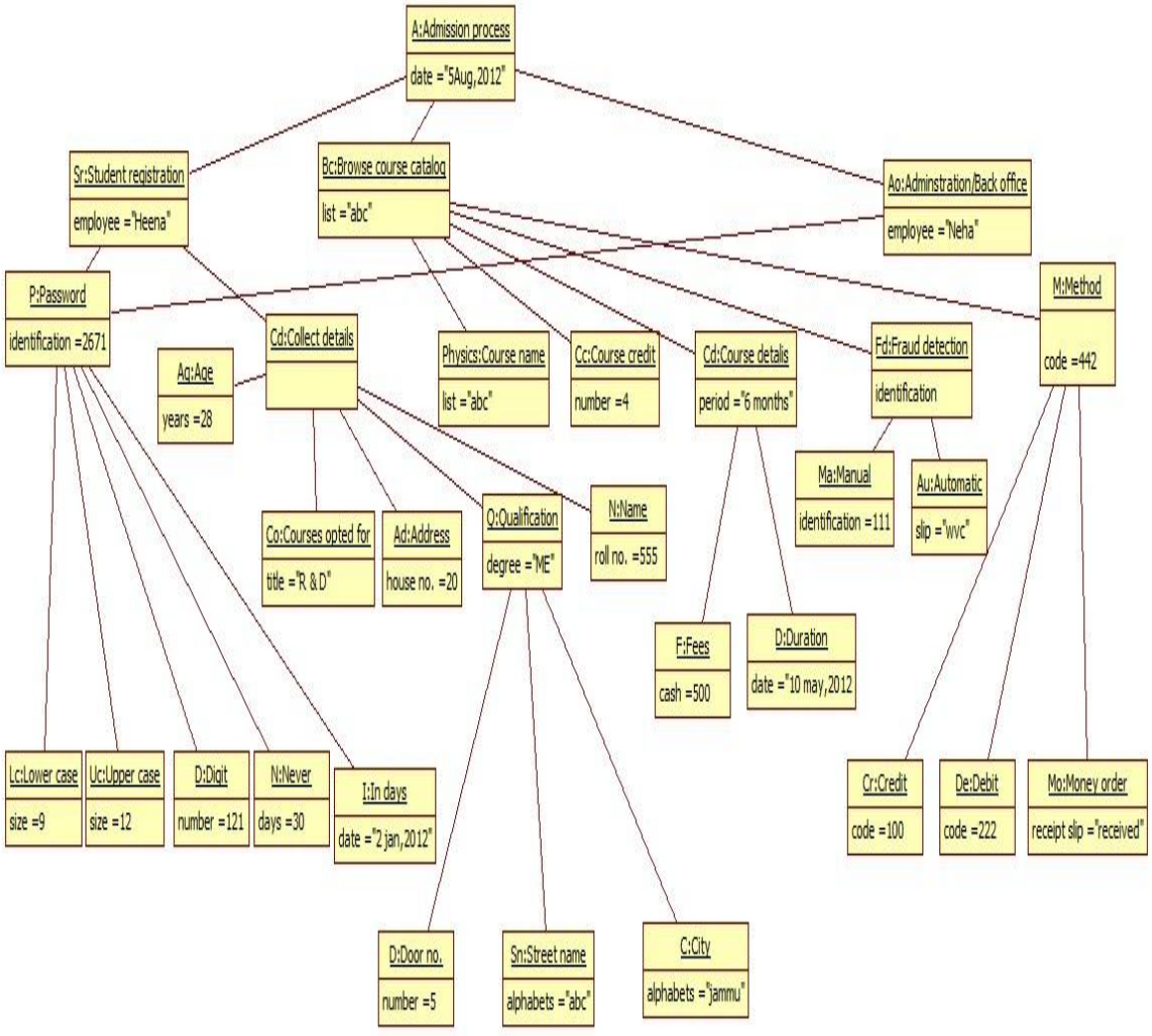

Fig 4.11: Object diagram of Admission process

Money order (Object)

Notion

- It is the **method** through which a student can **pay bill for registered course.**

……

Fig 4.10 shows the UML class diagram which was defined considering the structure and the construction process of LEL model.

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of admission process shown in Fig 4.10, the object diagram shown in Fig 4.11 is obtained.

4.3.4 Case Study of the Online Transaction System

Online transaction system describes software systems which enable customers to do transactions over the Internet. They must be able to deal with a large number of customers, and coordinate multiple resources to deliver the services to the customers. This process enables transactions by the customers where online transaction system provides services to an individual. This is done through an online transaction process web site.

Feature Model of the Online Transaction System

The relationship between the store front and home page features is a mandatory feature; the relationship between payment and fraud detection features is an optional feature. The set consisting of the Manual and Automatic features - fraud detection in the online transaction system application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the Credit Card, Debit Card, and Money Order features - to enable users to pay for selected item, at least one payment method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.12 shows a feature diagram of the online transaction system that represents all possible services of online transaction system. The online transaction system consists of two parts, one is store front that is dedicated to the interface that the customer uses to access the services of online transaction system and other is back office that is concerned with back office operations. During the process of store front, it enables the home page and customer registration. The customer registration might mandate the use of payment. In addition, the payment allows for different payment Methods and Fraud Detections. In the feature model in Fig 4.12, an includes constraints exist between the store front and the back office features. An excludes integrity constraint exists between the Credit Card and the Manual features.

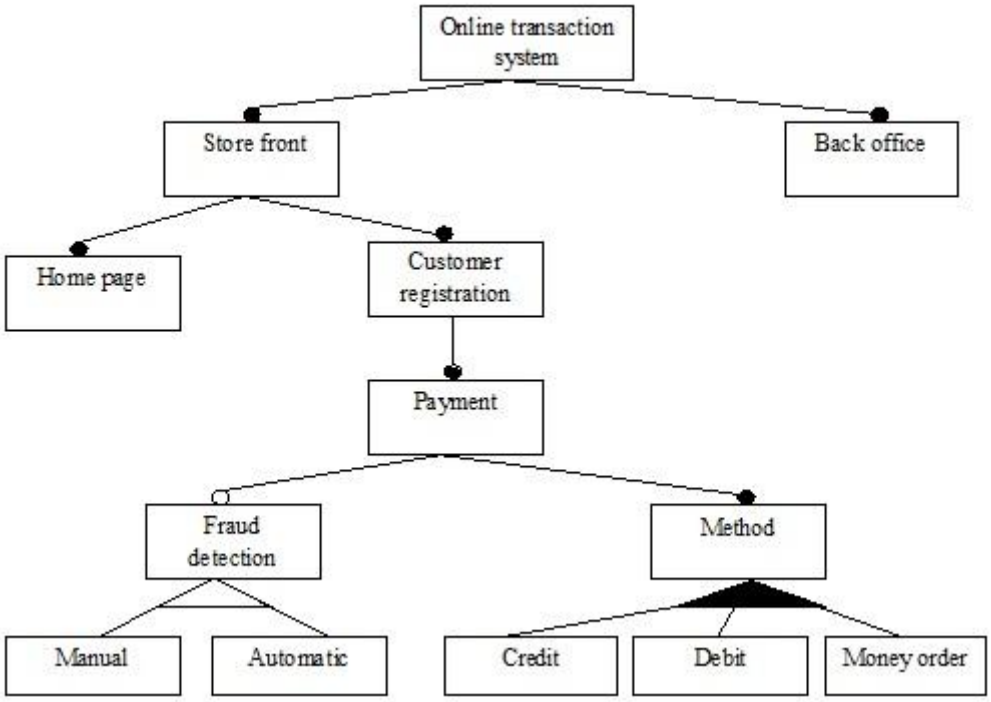

Fig 4.12: Feature model of online transaction system

Language Extended Lexicon Symbols for the Online Transaction System

LEL symbols of the online transaction process case study are given:

Online transaction system (Subject)

Table 4.2.1: Online transaction system LEL symbol

Notion	Behavioral Response
• It is a software system which is devoted to the online transaction by **customers** through Internet. • It is composed by a **store front** and a **back office**. • It may have account.	• It coordinates the **store front** and the **back office**.

Store front (Subject)

Table 4.2.2: Store front LEL symbol

Notion	Behavioral Response
• It is a component of the **online transaction system**. • It is the interface the customer uses to access the **services** of **online transaction system**. • It contains a **home page** and **customer registration**. • It has a unique identification.	• It may enable **customer registration**.

Back office (Subject)

Table 4.2.3: Back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access the **online transaction system**. • It is a component of the **online transaction system**. • It may have one or more employee. • It has a unique identification.	• It **manages online transaction system**.

Home page (Object)

Table 4.2.4: Home page LEL symbol

Notion	Behavioral Response
• It is the first page when **customer** accesses the site. • It has a welcome message. • It is a component of the **store front**.	• It may navigate to next page.

Customer registration (Subject)

Table 4.2.5: Customer registration LEL symbol

Notion	Behavioral Response
• It is the process for the **registration of customer**. • It is a component of **store front**. • It contains **payment.** • It has one or more employee. • It has a form. • It has identification. • It is the requesting of **customer** information. • It maintains **customer** information.	• Determine the future actions of the **customer**. • It registers the **customer**.

Payment (Verb)

Table 4.2.6: Payment course LEL symbol

Notion	Behavioral Response
• It is the process of paying bill to purchase the item through **online transaction**. • Performed by the **customer**. • It contains **fraud detection** and **method.**	• The **customer** has access to the purchased item. • May have **fraud detection**. • May have **methods** to **pay bill for the purchased book.**

Fraud detection (Subject)

Table 4.2.7: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting fraud in case of paying bill for the purchased item through **online transaction**. • It is composed of **automatic** and **manua**l. • It has transaction identification. • It keeps track of the **customer registration.**	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.2.8: Manual LEL symbol

Notion	Behavioral Response
• It is a **manual** process of **fraud detection**. • It has identification.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.2.9: Automatic LEL symbol

Notion	Behavioral Response
• It is an **automatic** process of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Method (Subject)

Table 4.2.10: Method LEL symbol

Notion	Behavioral Response
• It is the process by which a customer can pay bill to purchase the item through **online transaction.** • It contains **credit, debit** and **money order**. • It has a secret code. • It keeps track of modes of **payment** for **customer registration**. • It has a machine.	• It makes **payment** for the purchased item.

Credit (Object)

Table 4.2.11: Credit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a customer can pay bill to purchase the item**.** • It has a card. • It has a bank account. • It has an account holder. • It has card no.	• **Customer** has to enter pin number.

Debit (Object)

Table 4.2.12: Debit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a customer can pay bill to purchase the item**.** • It has a card. • It has a secret code. • It has a bank account. • It has an account holder.	• **Customer** has to enter pin number.

Money order (Object)

Table 4.2.13: Money order LEL symbol

Notion	Behavioral Response
• It is the **method** through which a customer can pay bill to purchase the item**.** • It has a account. • It has a customer. • It has a receipt slip.	• **Customer** has to enter money order number.

The Transformation Process Using a Case Study of the Online Transaction System

The transformation process takes as the source model a LEL model from a case study of the online transaction system, and follows the steps described below to derive a UML class diagram from LEL symbols of the online transaction system:

The transformation rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol store front that is Table 4.2.2, the class shown in Fig 4.13 is defined.

Fig 4.13: Store front class

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol Home page that is Table 4.2.4, gives as result the class and attributes shown in Fig 4.14. We would have obtained the following attributes: message.

Fig 4.14: Home page class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the LEL symbol store front shown in Table 4.2.2, the method described in Fig 4.15 is obtained.

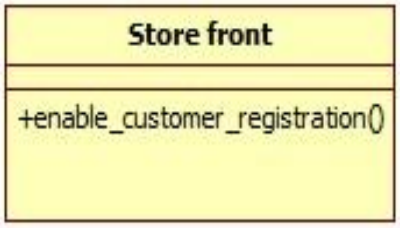

Fig 4.15: Method of store front

Rule 4: **Transformation subject information to method parameter**

Same as Rule 4 in case study of admission process.

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with method as the superclass and credit, debit and money order as subclasses.

Method (Subject)

Notion

- It is the process by which a customer can pay bill to purchase the item through **online transaction.**

  ……

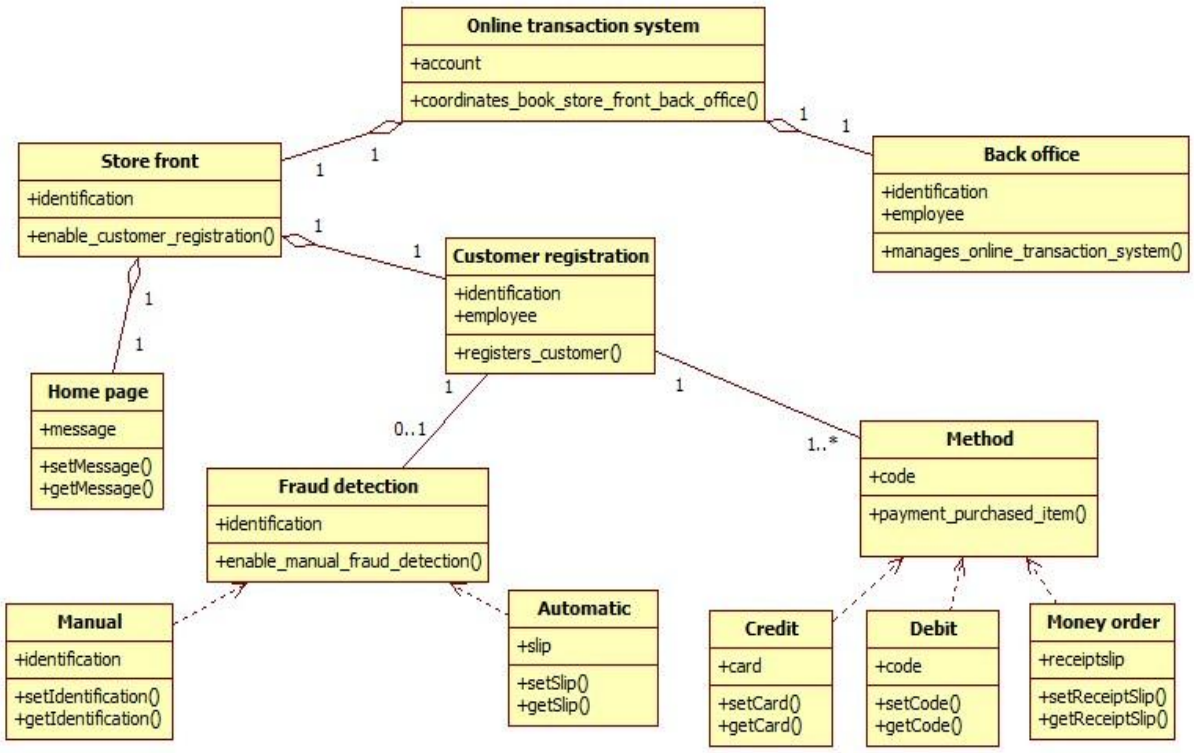

Fig 4.16: Class diagram of online transaction system

Credit (Object)

Notion

- It is the **method** through which a customer can pay bill to purchase the item**.**

……

Debit (Object)

Notion

- It is the **method** through which a customer can pay bill to purchase the item**.**

……

Money order (Object)

Notion

- It is the **method** through which a customer can pay bill to purchase the item**.**

……

Fig 4.16 shows the UML class diagram of admission process which was defined considering the structure and the construction process of LEL model of admission process.

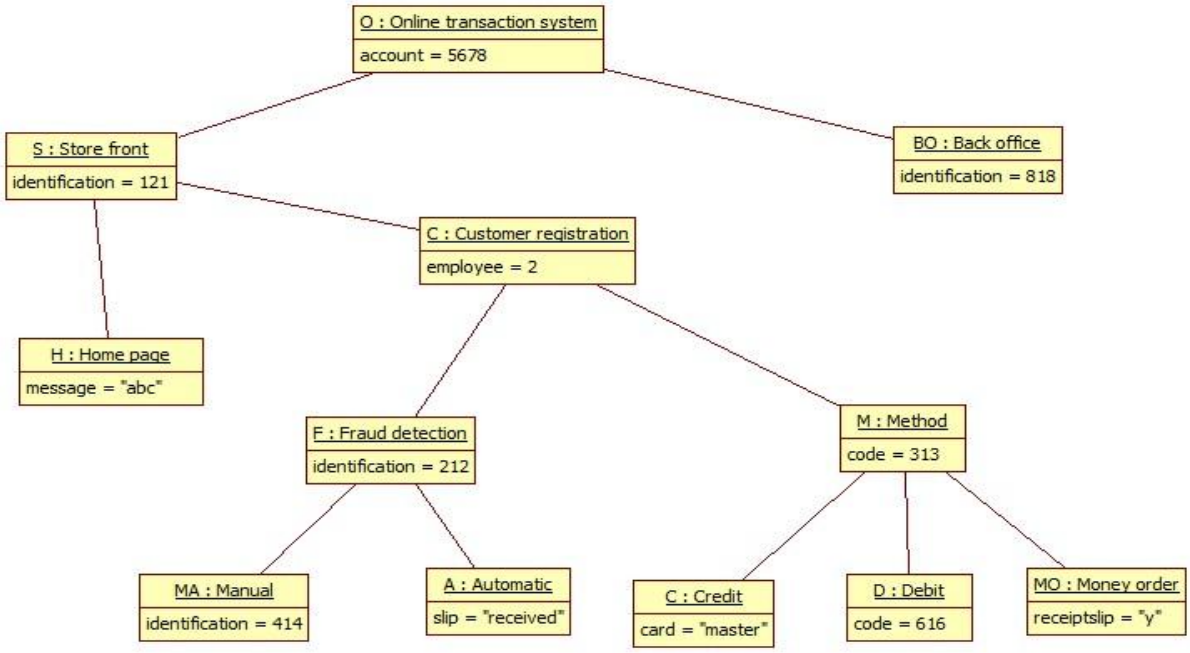

Fig 4.17: Object diagram of online transaction system

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of online transaction system shown in Fig 4.16, the object diagram shown in Fig 4.17 is obtained.

4.3.5 Case Study of the ATM

The ATM is a software system which allows customers to enter their own transaction using cash cards as identification. It has a card reader, transaction, a cash dispenser, a receipt printer and fraud detection. It runs a program that is usually written by the bank that usually follows a pattern. It starts by asking for the language. First step is to insert your card which has a magnetic strip on the back that it reads information about your bank followed by entering your password. After that, it's up to the program to decide what to do next. It must be able to deal with a large number of customers and coordinate multiple resources to deliver the services to the customers.

Feature Model of the ATM

The relationship between the store front and card reader features is a mandatory feature; the relationship between store front and fraud detection features is an optional feature.

The set consisting of the Manual and Automatic features - fraud detection in the ATM application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the deposit, withdrawal and balance inquiry features - to enable users to enter their transaction , at least one transaction method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.18 shows a feature diagram of the ATM that represents all possible services of the ATM. The ATM consists of two parts, one is store front that is dedicated to the interface that the customer uses to access the services of the ATM and other is back office that is concerned with back office operations. During the process of store front, it enables the card reader, transaction, a cash dispenser, a receipt printer and fraud detection. In addition, the

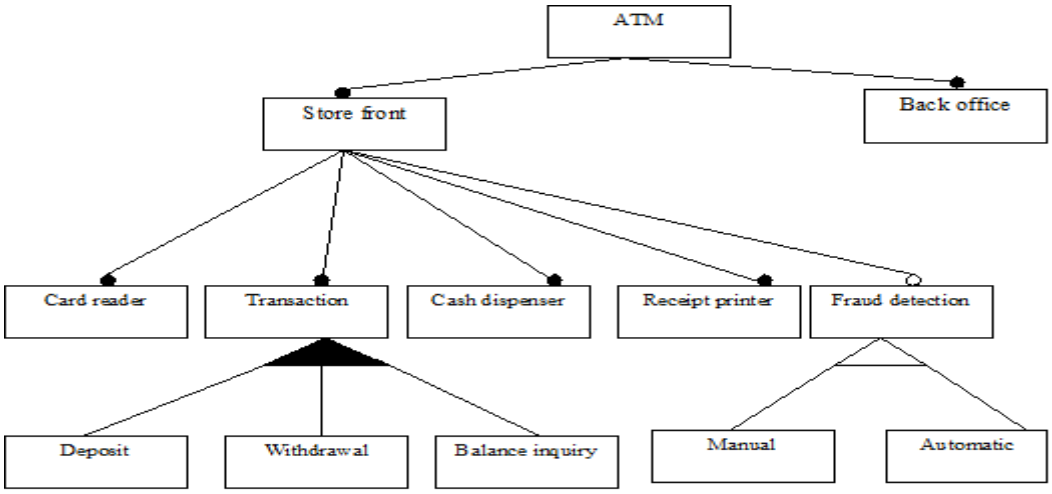

Fig 4.18: Feature model of the ATM

transaction allows for different transaction methods. In the feature model in Fig 4.18, an includes constraints exist between the store front and the back office features. An excludes integrity constraint exists between the withdrawal and the manual features.

The features store front, back office, fraud detection, manual and automatic of case study online transaction system are reused in this case study.

Language Extended Lexicon Symbols for the ATM

LEL symbols of the ATM case study are given below:

ATM (Subject)

Table 4.3.1: ATM LEL symbol

Notion	Behavioral Response
• It is a station which allows customers to enter their own **transaction** using **cash cards** as identification. • It is composed by a **store front** and a **back office**. • It may have bank name. • It has an atmid. • It may have state name.	• It may enable **transaction**. • It may enable **fraud detection**.

Store front (Subject)

Table 4.3.2: Store front LEL symbol

Notion	Behavioral Response
• It is the interface the customer uses to access the **ATM**. • It is a component of the **ATM**. • It contains a **card reader, transaction,** a **cash dispenser,** a **receipt printer** and **fraud detection**. • It has a unique identification.	• It may enable **transaction**.

Back office (Subject)

Table 4.3.3: Back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access the **ATM**. • It is a component of the **ATM**. • It may have one or more employee. • It has a unique identification.	• It manages the **ATM**.

Transaction (Subject)

Table 4.3.4: Transaction LEL symbol

Notion	Behavioral Response
• A banking operation involving the bank is defined as a **transaction**. A **transaction** might involve **deposit cash**, **withdrawal** and **enquiry**. • It is composed of **deposit, withdrawal and balance inquiry**. • It has unique pin. • It may have balance. • It is a component of the **store front.**	• It may **deposit cash**. • It may **withdrawal cash**. • It may enable balance **enquiry**.

Card reader (Object)

Table 4.3.5: Card reader LEL symbol

Notion	Behavioral Response
• It is the process for swiping and reading the **ATM card**. • It is a component of the **store front.** • It may have atm.	• Customer can swipe the **card**. • It may eject card.

Cash dispenser (Object)

Table 4.3.6: Cash dispenser LEL symbol

Notion	Behavioral Response
• It is the process of **dispensing cash**. • It is a component of the **store front.** • It may have initialcash and totalcash.	• It provides **cash**.

Receipt printer (Object)

Table 4.3.7: Receipt printer LEL symbol

Notion	Behavioral Response
• It is the process for **printing** customer **receipt**. • It is a component of the **store front.** • It may have balance**.**	• It generates **receipt**.

Fraud detection (Subject)

Table 4.3.8: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting access of an account by an unauthorized user. • It is composed of **automatic** and **manual.** • It has transaction identification. • It keeps track of the **store front**.	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.3.9: Manual LEL symbol

Notion	Behavioral Response
• It is a manual process of **fraud detection**. • It has identification. • It is a component of **fraud detection**.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.3.10: Automatic LEL symbol

Notion	Behavioral Response
• It is an automatic process of **fraud detection**. • It is a component of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Deposit (Verb)

Table 4.3.11: Deposit LEL symbol

Notion	Behavioral Response
• It is the process to **deposit cash** in account of customer. • It is performed by customer.	• It updates the account of customer.

Withdrawal (Verb)

Table 4.3.12: Withdrawal LEL symbol

Notion	Behavioral Response
• It is the process to **withdrawal cash** from account. • Performed by customer.	• It may update the account of customer.

Enquiry (Verb)

Table 4.3.13: Enquiry LEL symbol

Notion	Behavioral Response
• It the process to do **enquiry** about balance details of account. • It is performed by customer.	• It provides balance details to the account holder.

The Transformation Process Using a Case Study of the ATM

The transformation process takes as the source model a LEL model from a case study of the ATM, and follows the steps described below to derive a UML class diagram from LEL symbols of the ATM:

The transformation rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol ATM shown in Table 4.3.1, the class shown in Fig 4.19 is defined.

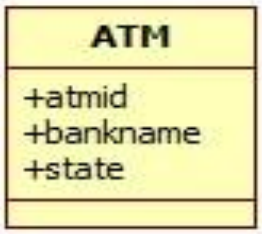

Fig 4.19: ATM class

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol receipt printer shown in Table 4.3.7, whose notion is described below, gives as result the class and attributes shown in Fig 4.20. We would have obtained the following attributes: balance.

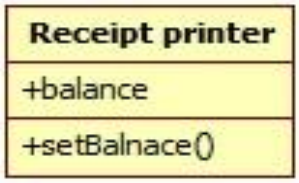

Fig 4.20: Receipt printer class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the LEL symbol ATM shown in Table 4.3.1, the method described in Fig 4.21is obtained.

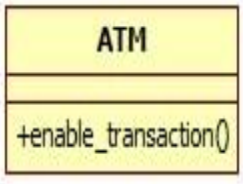

Fig 4.21: Method of ATM

Rule 4: **Transformation subject information to method parameter**

Same as Rule 4 in case study of admission process.

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with fraud detection as the superclass and manual and automatic as subclasses.

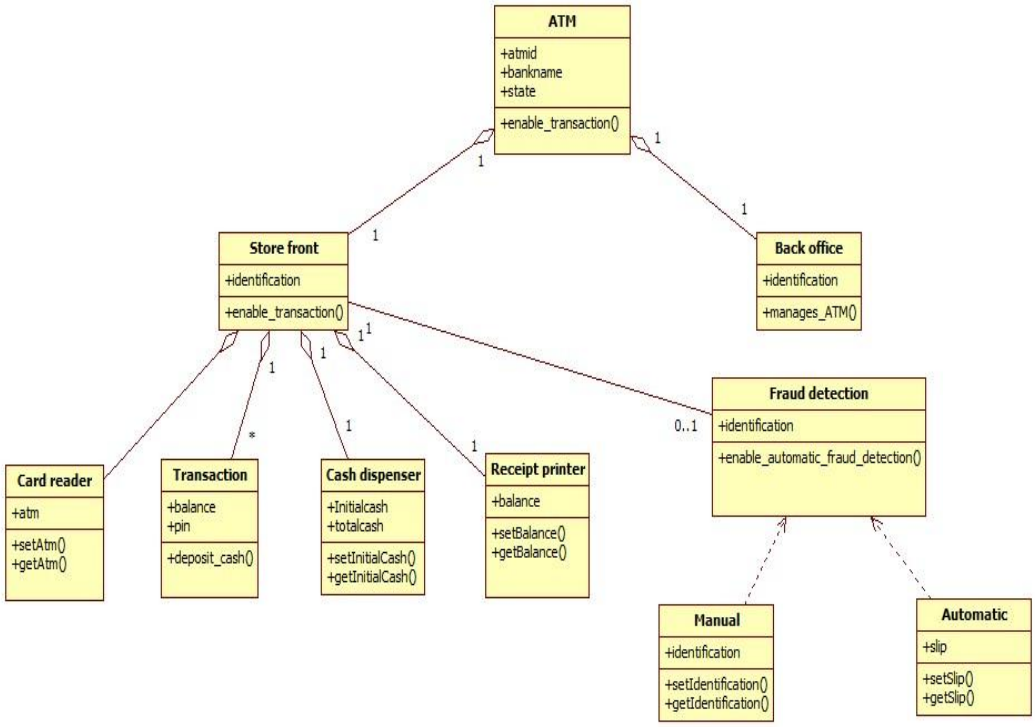

Fig 4.22: Class diagram of ATM

Fraud detection (Subject)

Notion

- It is the process of detecting access of an account by an unauthorized user.

……

Manual (Object)

Notion

- It is a manual process of **fraud detection**.

……

Automatic (Object)

Notion

- It is an automatic process of **fraud detection**.

……

Fig 4.22 shows the UML class diagram of the ATM which was defined considering the structure and the construction process of LEL model of the ATM.

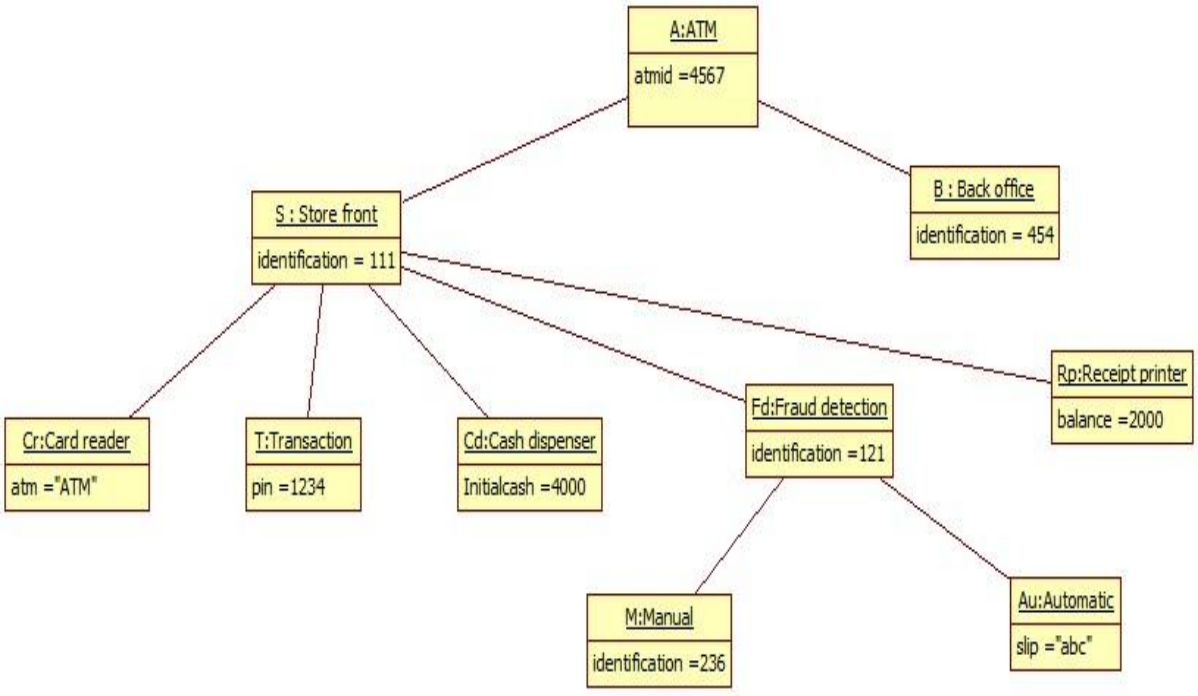

Fig 4.23: Object diagram of ATM

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of the ATM shown in Fig 4.22, the object diagram shown in Fig 4.23 is obtained.

4.3.6 Case Study of the Online Dairy Farm

Online dairy farm describes software systems which manages dairy farm over the internet. They must be able to deal with a large number of customers, and coordinate multiple resources to deliver the services to the customers. This is done through an online dairy farm web site.

Feature Model of the Online Dairy Farm

The relationship between the farm store front and home page features is a mandatory feature; the relationship between payment and fraud detection features is an optional feature. The set consisting of the Manual and Automatic features - fraud detection in the online dairy farm application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the Credit Card, Debit Card, and Money Order features - to enable dairy farmer to pay for the registration, at least one payment method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.24 shows a feature diagram of the online dairy farm that represents all possible services of online dairy farm. The online dairy farm consists of two parts, one is farm store front that is dedicated to the interface that the customer uses to access the services

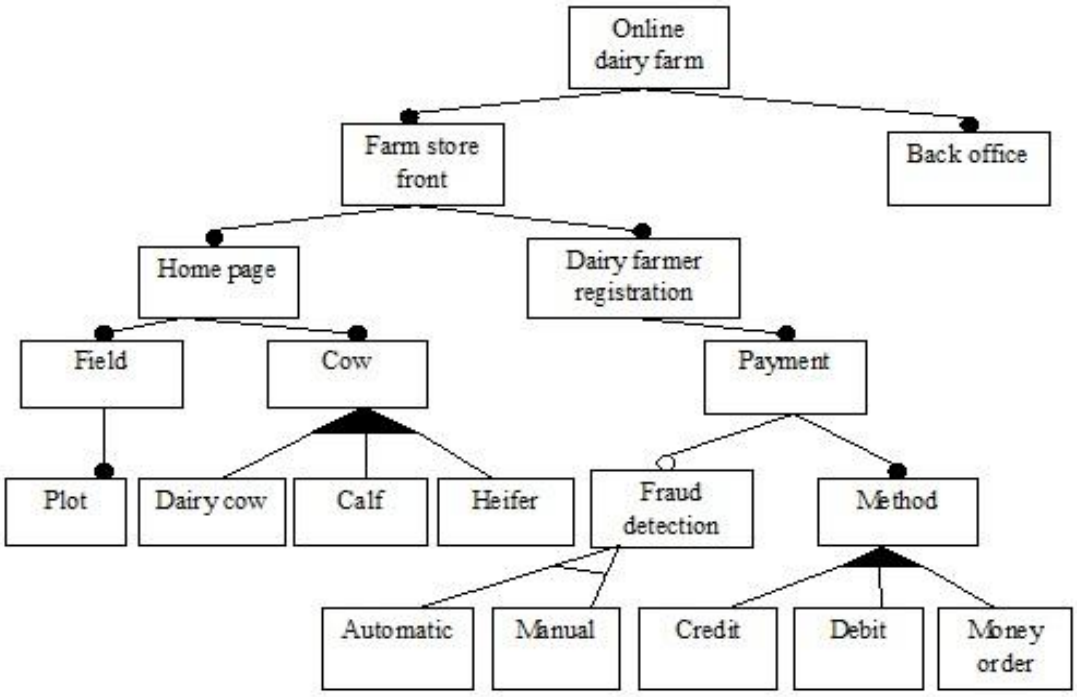

Fig 4.24: Feature model of the online dairy farm

and to register to the online dairy farm and other is back office that is concerned with back office operations. During the process of farm store front, it enables the home page and dairy farmer registration. The home page includes field and cow. The cow may be

dairy cow, calf or may be heifer. The field may include plot. The dairy farmer registration might mandate the use of payment. In addition, the payment allows for different payment methods and fraud detections. In the feature model in Fig 4.24, an includes constraints exist between the farm store front and the back office features. An excludes integrity constraint exists between the Credit Card and the Manual features.

The features store front, back office, home page, customer registration, payment, fraud detection, manual, automatic, method, credit, debit and money order of case study online transaction system are reused in this case study. The relationship between dairy farmer registration and farm store front in online dairy farm is similar to the relationship between customer registration and store front in online transaction system.

Language Extended Lexicon Symbols for the Online Dairy Farm

LEL symbols of the online dairy farm case study are given below:

Online dairy farm (Subject)

Table 4.4.1: Dairy farm LEL symbol

Notion	Behavioral Response
• It is managed by a **dairy farmer.** • It has identification. • It is composed of **farm store front** and **back office**. • It may have account.	• It coordinates the **farm store front** and the **back office**.

Farm store front (Subject)

Table 4.4.2: Farm store front LEL symbol

Notion	Behavioral Response
• It is the interface the customer uses to access the services of **online dairy farm**. • It is a component of the **online dairy farm**. • It contains a **home page** and **dairy farmer registration**. • It has a unique identification.	• It may enable **dairy farmer registration**.

Home page (Object)

Table 4.4.3: Home page LEL symbol

Notion	Behavioral Response
• It is the first page when **customer** accesses the site. • It has a welcome message. • It is a component of the **farm store front**.	• It may navigate to next page.

Dairy farmer registration (Subject)

Table 4.4.4: Dairy farmer registration LEL symbol

Notion	Behavioral Response
• It is the process for the **registration of dairy farmer**. • It is a component of **farm store front**. • It contains **payment.** • It has one or more employee. • It has a form. • It has identification.	• Determine the future actions of the **dairy farmer**. • It registers the **dairy farmer**.

Back office (Subject)

Table 4.4.5: Back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access the **online dairy farm.** • It is a component of the **online dairy farm**. • It may have one or more employee. • It has a unique identification.	• It manages **online dairy farm**.

Payment (Verb)

Table 4.4.6: Payment course LEL symbol

Notion	Behavioral Response
• It is the process of paying bill for the **registration**. • Performed by the **dairy farmer**. • It contains **fraud detection** and **method.**	• The **dairy farmer** has access to the **online dairy farm**. • May have **fraud detection**. • May have **methods** to **pay bill for the registration.**

Fraud detection (Subject)

Table 4.4.7: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting fraud in case of paying bill for the **registration**. • It is composed of **automatic** and **manua**l. • It has transaction identification. • It keeps track of the **dairy farmer registration.**	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.4.8: Manual LEL symbol

Notion	Behavioral Response
• It is a **manual** process of **fraud detection**. • It has identification.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.4.9: Automatic LEL symbol

Notion	Behavioral Response
• It is an **automatic** process of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Method (Subject)

Table 4.4.10: Method LEL symbol

Notion	Behavioral Response
• It is the process by which a **dairy farmer** can pay bill for **registration.** • It contains **credit, debit** and **money order**. • It has a secret code. • It keeps track of modes of **payment** for **dairy farmer registration**. • It has a machine.	• It makes **payment** for the **dairy farmer registration**.

Credit (Object)

Table 4.4.11: Credit LEL symbol

Notion	Behavioral Response
• It is the **method** through which **dairy farmer** can pay bill for the **registration.** • It has a card. • It has a bank account. • It has a account holder. • It has card no.	• **Dairy farmer** have to enter pin number.

Debit (Object)

Table 4.4.12: Debit LEL symbol

Notion	Behavioral Response
• It is the **method** through which **dairy farmer** can pay bill for the **registration.** • It has a card. • It has a secret code. • It has a bank account. • It has a account holder.	• **Dairy farmer** have to enter pin number.

Money order (Object)

Table 4.4.13: Money order LEL symbol

Notion	Behavioral Response
• It is the **method** through which **dairy farmer** can pay bill for the **registration.** • It has a account. • It has a customer. • It has a receipt slip.	• **Dairy farmer** have to enter money order number.

Field (Object)

Table 4.4.14: Field LEL symbol

Notion	Behavioral Response
• Land where cows eat pasture. • It has identification. • It has a precise location in the **dairy farm**. • It has a size. • It is divided into a set of **plot**.	• It assigns **plot**.

Cow (Subject)

Table 4.4.15: Cow LEL symbol

Notion	Behavioral Response
• It may be a **calf**, a **heifer** or a **dairy cow.** • It has age.	……

Dairy cow (Subject)

Table 4.4.16: Dairy cow LEL symbol

Notion	Behavioral Response
• It is a female **cow** which has leads at least one **calf.** • It has age.	……

Calf (Subject)

Table 4.4.17: Calf LEL symbol

Notion	Behavioral Response
• It is a **cow** of less than 12 months age**.** • It has age.	……

Heifer (Subject)

Table 4.4.18: Heifer LEL symbol

Notion	Behavioral Response
• It is a female **cow** of 12 months age on or more which has not had a **calf.** • It has age.	……

Plot (Object)

Table 4.4.19: Plot LEL symbol

Notion	Behavioral Response
• It is a part of a **field**. • It has identification**.** • It has a size.	……

The Transformation Process Using a Case Study of the Online Dairy Farm

The transformation process takes as the source model a LEL model from a case study of the online dairy farm, and follows the steps described below to derive a UML class diagram from LEL symbols of the online dairy farm:

The transformation rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol farm store front shown in

Table 4.4.2, the class shown in Fig 4.25 is defined.

Fig 4.25: Farm store front class

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol Plot shown in Table 4.4.19, gives as result the class and attributes shown in Fig 4.26. We would have obtained the following attributes: identification and size.

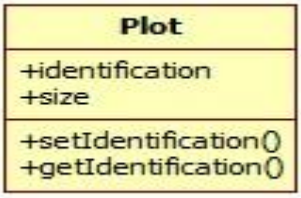

Fig 4.26: Plot class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the subject LEL symbol farm store as shown in Table 4.4.2, the method described in Fig 4.27 is obtained.

Fig 4.27: Method of farm store front class

Rule 4: **Transformation subject information to method parameter**

Same as Rule 4 in case study of admission process.

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with cow as the superclass and dairy cow, calf and heifer as subclasses.

Analyzing the LEL symbol field shown in Table 3.84, a notion with a LEL symbol modeled as a class (Plot, Fig 4.26) containing the "component-composition" verb is "divided into" is found. Besides, a "content-component" verb is found in the notion of the LEL symbol plot ("it is a part of …" Table 4.4.19). Therefore, the transformation Rule 5 defines an aggregation relationship between field and plot classes.

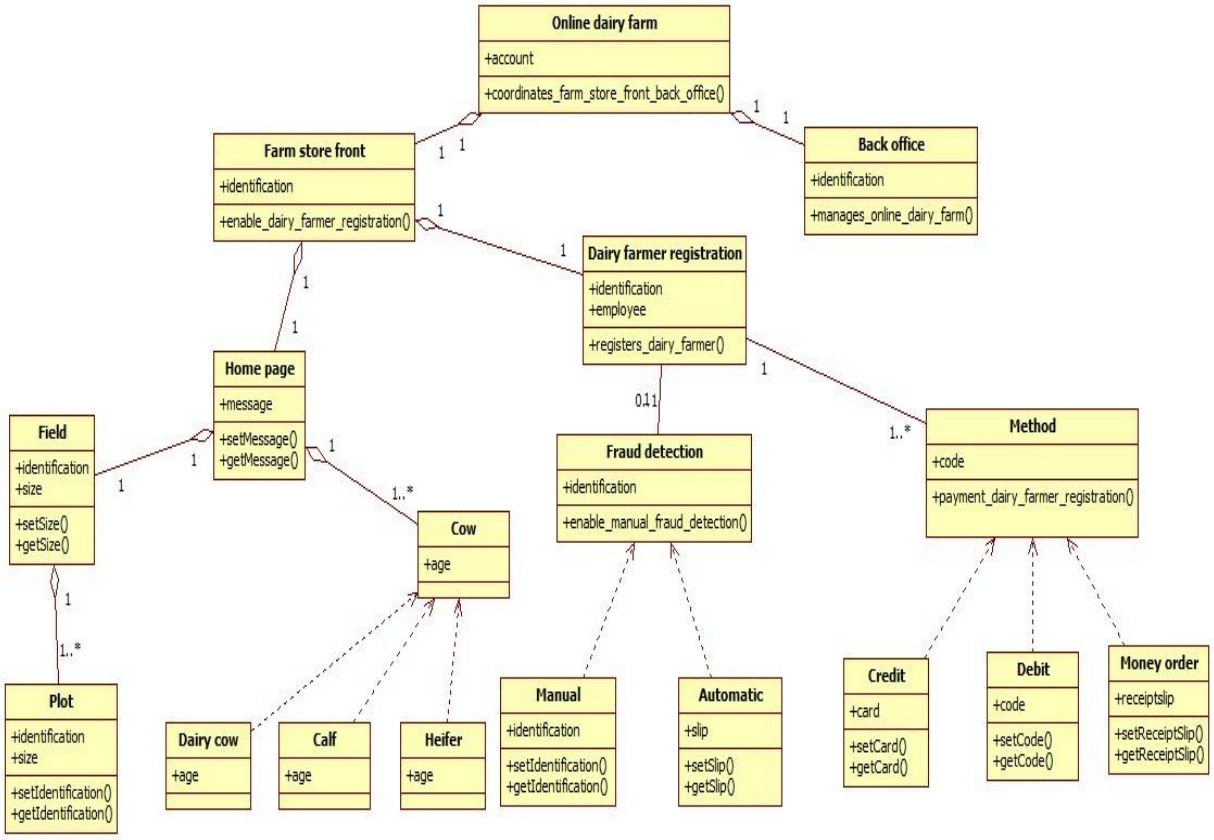

Fig 4.28: Class diagram of online dairy farm

Cow (Subject)

Notion

- It may be a **calf**, a **heifer** or a **dairy cow.**

……..

Dairy cow (Subject)

Notion

- It is a female **cow** which has lead at least one **calf.**

……..

Calf (Subject)

Notion

- It is a **cow** of less than 12 months age**.**

……..

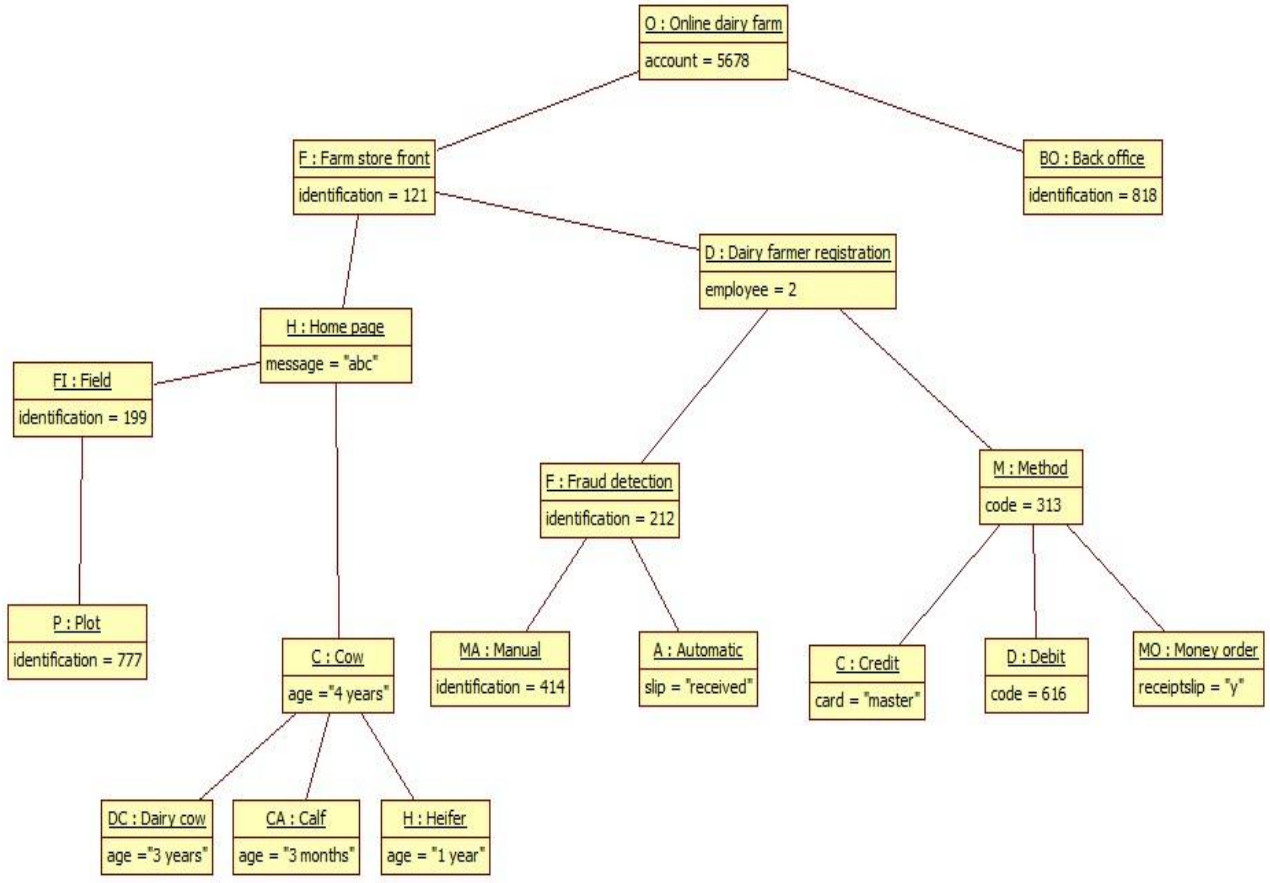

Fig 4.29: Object diagram of online dairy farm

Heifer (Subject)

Notion

- It is a female **cow** of 12 months age on or more which has not had a **calf.**

……..

As shown in LEL symbol field in Table 4.84.

Fig 4.28 shows the UML class diagram of the online dairy farm which was defined considering the structure and the construction process of LEL model of the online dairy farm.

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of the online dairy farm shown in Fig 4.28, the object diagram shown in Fig 4.29 is obtained.

4.3.7 Case Study of the Online Book Shop

Online book shop describes software systems which enable customers to buy books over the Internet. They must be able to deal with a large number of customers, and coordinate multiple resources to deliver the services to the customers. This process enables customers to buy books through transactions over the internet where online book shop provides services to an individual. This is done through an online book shop web site.

Feature Model of the Online Book Shop

The relationship between the book store front and home page features is a mandatory feature; the relationship between payment and fraud detection features is an optional feature. The set consisting of the Manual and Automatic features - fraud detection in the online book shop application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the Credit Card, Debit Card, and Money Order features - to enable users to pay for selected book, at least one payment method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.30 shows a feature diagram of the online book shop that represents all possible services of online book shop. The online book shop consists of two parts, one is store front that is dedicated to the interface that the customer uses to purchase the book from the online book shop and other is back office that is concerned with back office operations. During the process of book store front, it enables the home page and customer registration. The home page includes books. The books may be new books, or may be used books. The customer registration might mandate the use of payment. In addition, the payment allows for different payment methods and fraud detections. In the feature model in Fig 4.30, an

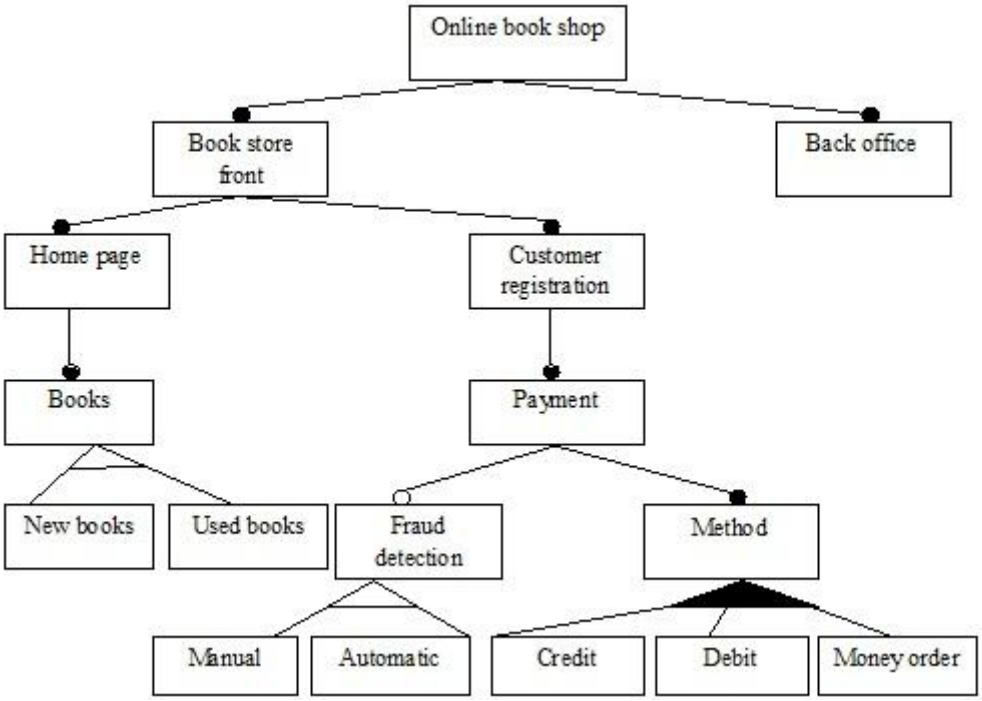

Fig 4.30: Feature model of the online book shop

includes constraints exist between the book store front and the back office features. An excludes integrity constraint exists between the Credit Card and the Manual features.

The features store front, back office, customer registration, payment, fraud detection, manual, automatic, method, credit, debit and money order of case study online transaction system are reused in this case study. The relationship between book store front in online book shop is similar to the relationship between store front in online transaction system.

Language Extended Lexicon Symbols for the Online Book Shop

LEL symbols of the online book shop process case study are given below:

Online book shop (Subject)

Table 4.5.1: Online book shop LEL symbol

Notion	Behavioral Response
• It is a software system which is devoted to selling **books** to **customers** through Internet. • It is composed by a **book store front** and a **back office**. • It may have account.	• It coordinates the **book store front** and the **back office**.

Book store front (Subject)

Table 4.5.2: Book store front LEL symbol

Notion	Behavioral Response
• It is a component of the **online book shop**. • It is the interface the customer uses to purchase the **book** from the **online book shop**. • It contains a **home page** and **customer registration**. • It has a unique identification.	• It may enable **customer registration**.

Home page (Object)

Table 4.5.3: Home page LEL symbol

Notion	Behavioral Response
• It is the first page when **customer** accesses the site. • It has a welcome message. • It is a component of the **book store front**.	• It may navigate to next page.

Customer registration (Subject)

Table 4.5.4: Customer registration LEL symbol

Notion	Behavioral Response
• It is the process for the **registration of customer**. • It is a component of **book store front**. • It contains **payment.** • It has one or more employee. • It has a form. • It has identification.	• Determine the future actions of the **customer**. • It registers the **customer**.

Back office (Subject)

Table 4.5.5: Back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access the **online book shop**. • It is a component of the **online book shop**. • It may have one or more employee. • It has a unique identification.	• It manages **online book shop**.

Payment (Verb)

Table 4.5.6: Payment course LEL symbol

Notion	Behavioral Response
• It is the process of paying bill in order to purchase the **book**. • Performed by the **customer**. • It contains **fraud detection** and **method.**	• The **customer** has access to the purchased book. • May have **fraud detection**. • May have **methods** to **pay bill for the purchased book.**

Fraud detection (Subject)

Table 4.5.7: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting fraud in case of paying bill for the purchased **book**. • It is composed of **automatic** and **manua**l. • It has transaction identification. • It keeps track of the **customer registration.**	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.5.8: Manual LEL symbol

Notion	Behavioral Response
• It is a **manual** process of **fraud detection**. • It has identification.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.5.9: Automatic LEL symbol

Notion	Behavioral Response
• It is an **automatic** process of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Method (Subject)

Table 4.5.10: Method LEL symbol

Notion	Behavioral Response
• It is the process by which a customer can pay bill to purchase the **book.** • It contains **credit, debit** and **money order**. • It has a secret code. • It keeps track of modes of **payment** for **customer registration**. • It has a machine.	• It makes **payment** for the purchased **book**.

Credit (Object)

Table 4.5.11: Credit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student can pay bill to purchase the **book.** • It has a card. • It has a bank account. • It has a account holder. • It has card no.	• **Customer** has to enter pin number.

Debit (Object)

Table 4.5.12: Debit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student can pay bill to purchase the **book.** • It has a card. • It has a secret code. • It has a bank account. • It has a account holder.	• **Customer** has to enter pin number.

Money order (Object)

Table 4.5.13: Money order LEL symbol

Notion	Behavioral Response
• It is the **method** through which a student can pay bill to purchase the **book.** • It has a account. • It has a customer. • It has a receipt slip.	• **Customer** has to enter money order number.

The Transformation Process using a Case Study of the Online Book Shop

The transformation process takes as the source model a LEL model from a case study of the online book shop, and follows the steps described below to derive a UML class diagram from LEL symbols of the online book shop:

The transformation rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol book store front shown in Table 4.5.2, the class shown in Fig 4.31 is defined.

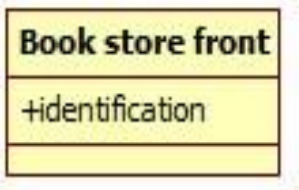

Fig 4.31: Book store front class

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol home page shown in Table 4.5.3, gives as result the class and attributes shown in Fig 4.32. We would have obtained the following attributes: message.

Fig 4.32: Home page class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the LEL symbol book store front shown in Table 4.5.2, the method described in Fig 4.33 is obtained.

Fig 4.33: Method of book store front

Rule 4: **Transformation subject information to method parameter**

Same as Rule 4 in case study of admission process.

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with method as the superclass and credit, debit and money order as subclasses.

Method (Subject)

Notion

- It is the process by which a customer can pay bill to purchase the **book.**

……

Credit (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

……..

Debit (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

……

Money order (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

……

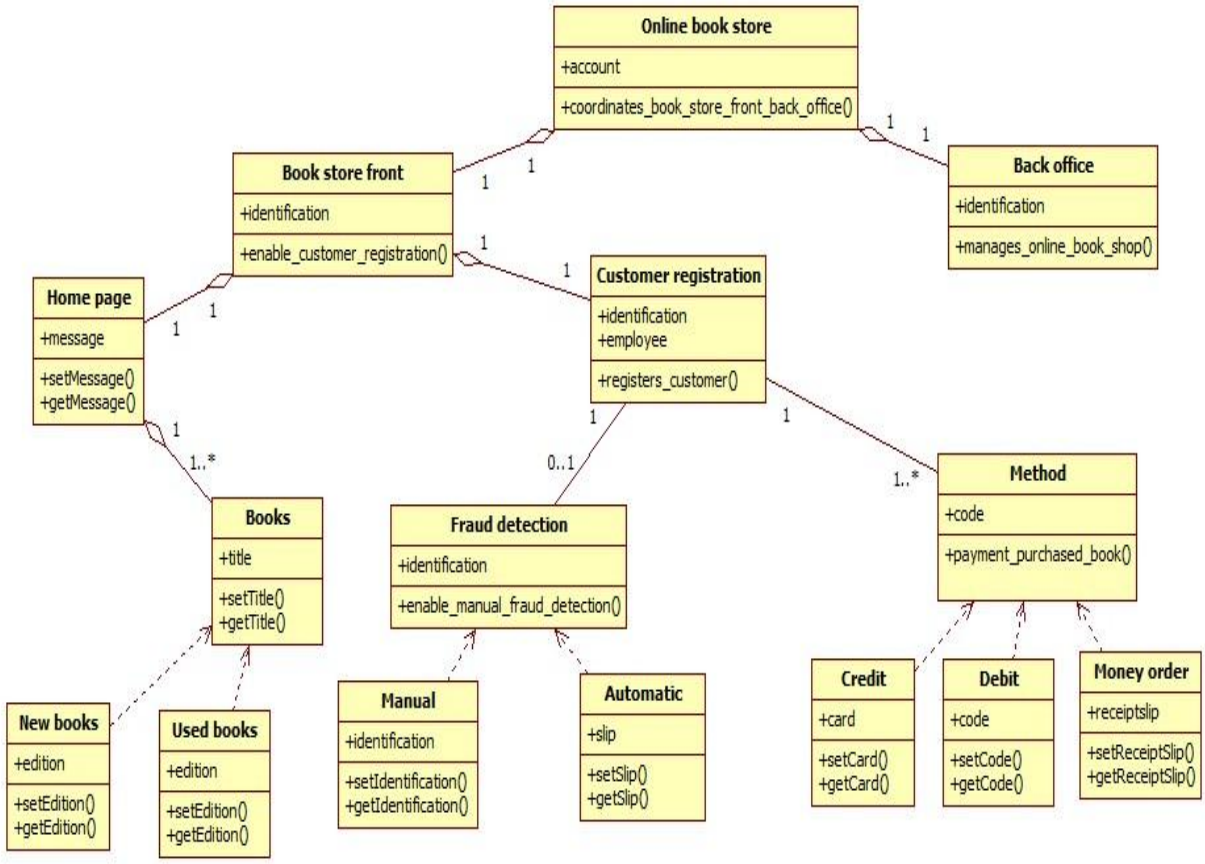

Fig 4.34: Class diagram of online book store

Fig 4.34 shows the UML class diagram of the online book store which was defined considering the structure and the construction process of LEL Model of the online book store.

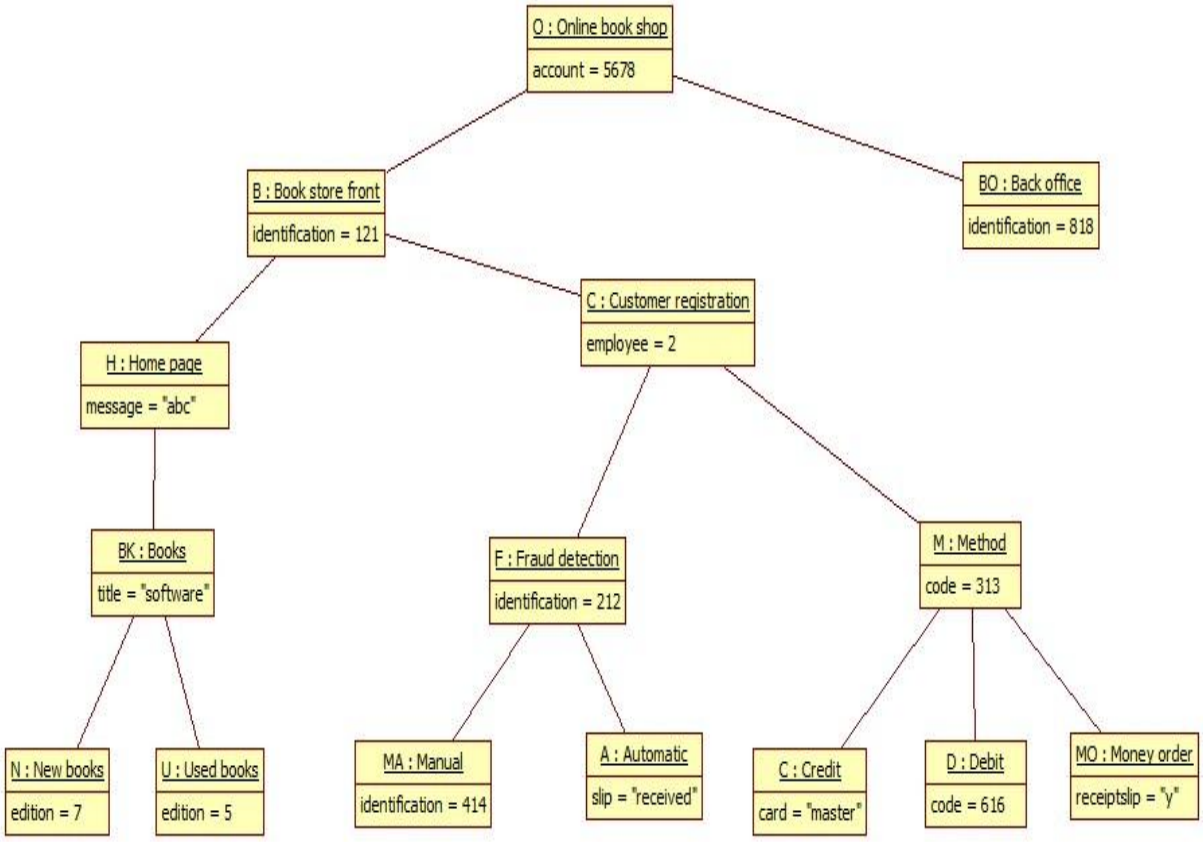

Fig 4.35: Object diagram of online book store

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of the online book store shown in Fig 4.34, the object diagram shown in Fig 4.35 is obtained.

4.3.8 Case Study of the Online Potato Collection Company

Online potato collection company describes software systems that buy potato to Producer over the Internet. They must be able to deal with a large number of customers, and coordinate multiple resources to deliver the services to the customers. This is done through an online potato collection company web site. Tables below show LEL symbols belonging to this case study

Feature Model of the Online Potato Collection Company

relationship between the potato store front and home page features is a mandatory feature; the relationship between payment and fraud detection features is an optional feature. The set consisting of the Manual and Automatic features - fraud detection in the online potato collection company application can be either manual or automatic, but not both as it is an alternative feature; the group consisting of the Credit Card, Debit Card, and Money Order features - to enable producer to pay for the registration, at least one payment method must be selected as it is an (or) feature. These features are organized into a feature model as Fig 4.36 shows a feature diagram of the online potato collection company that represents all possible services of online potato collection company. The online potato collection company consists of two parts, one is potato store front that is dedicated to the interface that the customer uses to access the services of online potato collection company and other is back office that is concerned with back office operations. During the process of potato store front, it enables the home page and producer registration. The home page enables producer, defect and potato purchase contract. The producer registration might mandate the use of payment. In addition, the payment allows for different

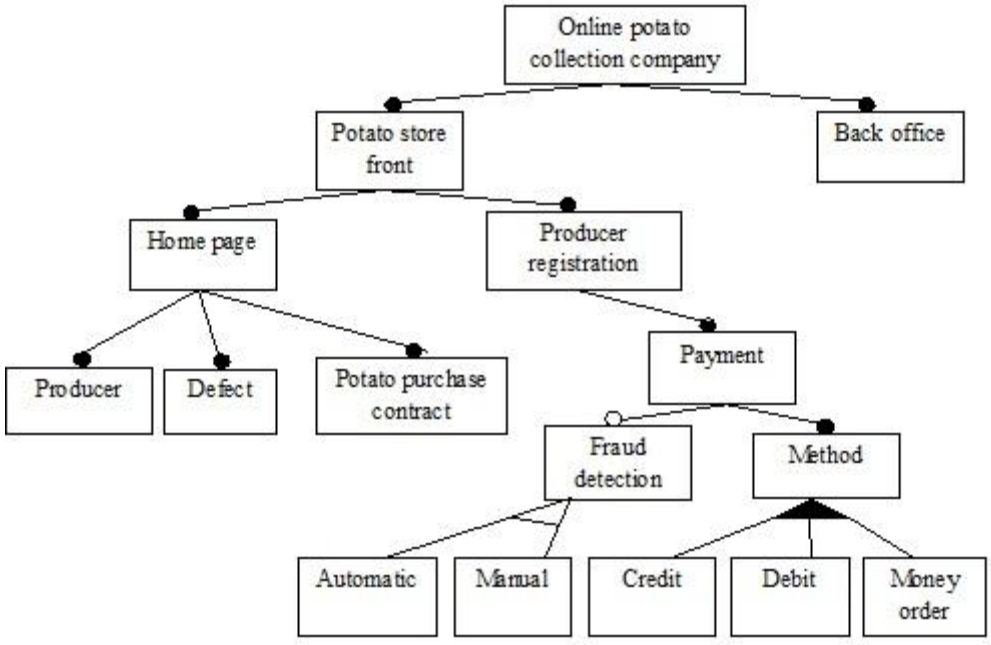

Fig 4.36: Feature model of online potato collection company.

The payment methods and fraud detections. In the feature model in Fig 4.36, an includes constraints exist between the potato store front and the back office features. An excludes integrity constraint exists between the Credit Card and the Manual features.

The features store front, back office, homepage, customer registration, payment, fraud detection, manual, automatic, method, credit, debit and money order of case study online transaction system are reused in this case study. The relationship between potato store front and producer registration in online potato collection company is similar to the relationship between store front and customer registration in online transaction system.

Language Extended Lexicon Symbols for the Online potato collection company

LEL symbols of the online potato collection company case study are given below:

Online potato collection company (Subject)

Table 4.6.1: Online potato collection company LEL symbol

Notion	Behavioral Response
• It is a software system of a company that buys **potato** to **Producer**. • It has identification. • It is composed by a **farm store front** and a **back office**. • It may have account.	• Sets the price of **potato** in agreement with **Producer.** • Delivers **production recommendations** to **Producer**. • Signs the **purchase and production of potatoes contract**. • **Receive potato** from **Producer**. • **Inspect potato**.

Producer (Subject)

Table 4.6.2: Producer LEL symbol

Notion	Behavioral Response
• Person in charge of planting, harvesting and delivering **potato** to **online potato collection company**. • It is one of the participants of the **contract**. • It is a component of **home page**. • He has a name.	• Signs the **purchase and production of potatoes contract**. • Produces **Potato.**

Defect (Object)

Table 4.6.3: Defect LEL symbol

Notion	Behavioral Response
• Problem found in a **potato** during the inspection. • It can be internal or external. • It is a component of **home page**. • It has number.	• It helps to define the limit of **tolerance** in **Grade of quality**.

Potato purchase contract (Object)

Table 4.6.4: Potato purchase contract LEL symbol

Notion	Behavioral Response
• Object of the **purchase and production of potatoes contract.** • It has identification. • It is a component of **home page**.	• It is produced by **Producer**.

Potato store front (Subject)

Table 4.6.5: Potato store front LEL symbol

Notion	Behavioral Response
• It is a component of the **online potato collection company**. • It is the interface the customer uses to access the services of **online potato collection company**. • It contains a **home page** and **producer registration**. • It has a unique identification.	• It may enable **producer registration**.

Home page (Object)

Table 4.6.6: Home page LEL symbol

Notion	Behavioral Response
• It is the first page when **customer** accesses the site. • It has a welcome message. • It is a component of the **potato store front**. • It is composed by a **producer, potato purchase contract** and **defect**.	• It may navigate to next page.

Producer registration (Subject)

Table 4.6.7: Producer registration LEL symbol

Notion	Behavioral Response
• It is the process for the **registration of producer**. • It is a component of **potato store front**. • It contains **payment.** • It has one or more employee. • It has a form. • It has identification.	• Determine the future actions of the **producer**. • It registers the **producer**.

Back office (Subject)

Table 4.6.8: Back office LEL symbol

Notion	Behavioral Response
• It is the interface that administrator uses to access the **online potato collection company**. • It is a component of the **online potato collection company**. • It may have one or more employee. • It has a unique identification.	• It manages **online potato collection company**.

Payment (Verb)

Table 4.6.9: Payment course LEL symbol

Notion	Behavioral Response
• It is the process of paying bill for the **registration**. • Performed by the **producer**. • It contains **fraud detection** and **method.**	• The **producer** has access to the **online potato collection company**. • May have **fraud detection**. • May have **methods** to **pay bill for the registration.**

Fraud detection (Subject)

Table 4.6.10: Fraud detection LEL symbol

Notion	Behavioral Response
• It is the process of detecting fraud in case of paying bill for the **registration.** • It is composed of **automatic** and **manua**l. • It has transaction identification. • It keeps track of the **producer registration.**	• It may enable **manual fraud detection**. • It may enable **automatic fraud detection**.

Manual (Object)

Table 4.6.11: Manual LEL symbol

Notion	Behavioral Response
• It is a **manual** process of **fraud detection**. • It has identification.	• It may detect **fraud** manually.

Automatic (Object)

Table 4.6.12: Automatic LEL symbol

Notion	Behavioral Response
• It is an **automatic** process of **fraud detection**. • It has a machine. • It generates a slip.	• It may detect **fraud** automatically.

Method (Subject)

Table 4.6.13: Method LEL symbol

Notion	Behavioral Response
• It is the process by which a customer can pay bill to purchase the **book.** • It contains **credit, debit** and **money order**. • It has a secret code. • It keeps track of modes of **payment** for **producer registration**. • It has a machine.	• It makes **payment** for the **producer registration**.

Credit (Object)

Table 4.6.14: Credit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a **producer** can pay bill for the **registration.** • It has a card. • It has a bank account. • It has an account holder. • It has card no.	• **Producer** has to enter pin number.

Debit (Object)

Table 4.6.15: Debit LEL symbol

Notion	Behavioral Response
• It is the **method** through which a **producer** can pay bill for the **registration.** • It has a card. • It has a bank account. • It has an account holder. • It has a secret code.	• **Producer** has to enter pin number.

Money order (Object)

Table 4.6.16: Money order LEL symbol

Notion	Behavioral Response
• It is the **method** through which a **producer** can pay bill for the **registration.** • It has an account. • It has a customer. • It has a receipt slip.	• **Producer** has to enter money order number.

The Transformation Process using a Case Study of the Potato Collection Company

The transformation process takes as the source model a LEL model from a case study of the online book shop, and follows the steps described below to derive a UML class diagram from LEL symbols of the online book shop:

The transformation rules

Rule 1: **Transformation subject to class**

By applying the transformation Rule 1 to the subject LEL symbol online potato collection company shown in Table 4.6.1, the class shown in Fig 4.37 is defined.

Fig 4.37: Online potato collection company class

Rule 2: **Transformation object to class**

The application of the transformation Rule 2 to the object LEL symbol potato purchase contract shown in Table 4.6.4, gives as result the class and attributes shown in Fig 4.38. We would have obtained the following attributes: identification.

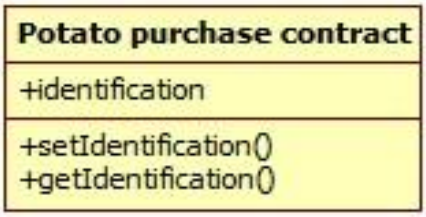

Fig 4.38: Potato purchase contract class

Rule 3: **Transformation subject behavioral response to method**

Applying the transformation Rule 3 to the LEL symbol online potato collection company shown in Table 4.6.1, the method described in Fig 4.39 is obtained.

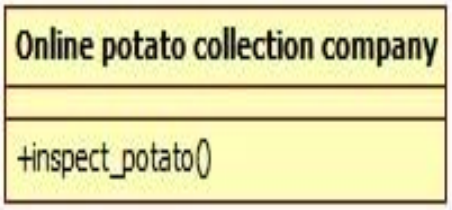

Fig 4.39: Method of online potato collection company

Rule 4: **Transformation subject information to method parameter**

Same as Rule 4 in case study of admission process.

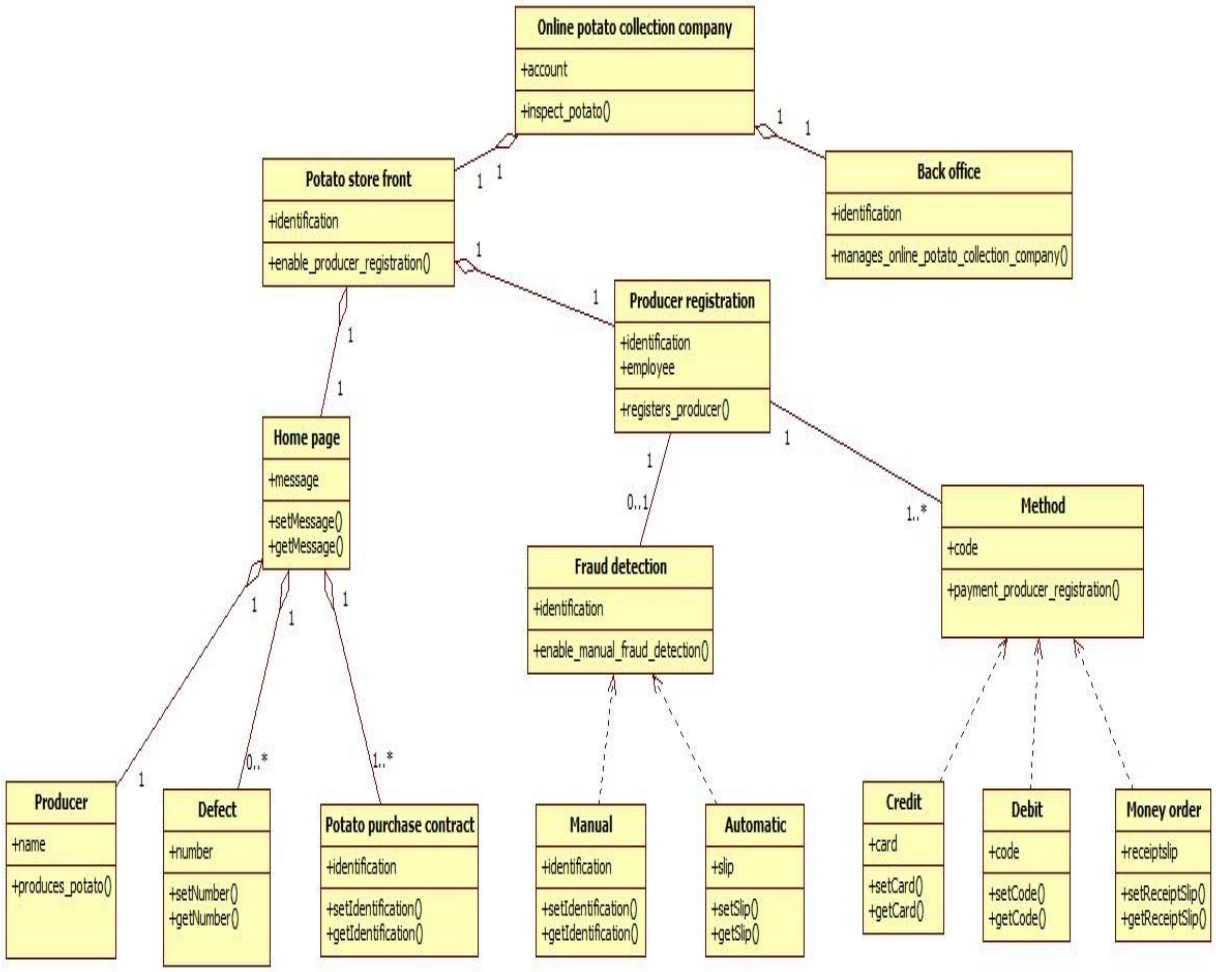

Fig 4.40: Class diagram of online potato collection company

Rule 5: **Transformation LEL relationships to class relationships**

By applying the transformation Rule 5 to the LEL symbols given below we obtain a hierarchy, with method as the superclass and credit, debit and money order as subclasses.

Method (Subject)

Notion

- It is the process by which a customer can pay bill to purchase the **book.**
- It contains **credit, debit** and **money order**.

……

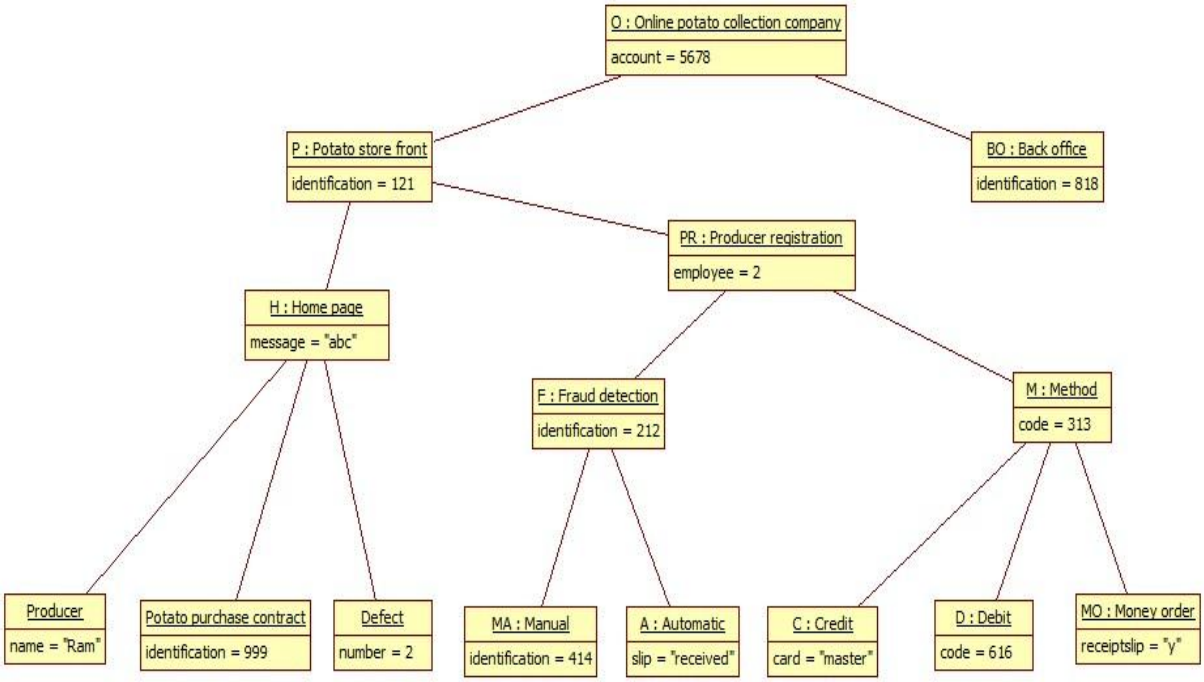

Fig 4.41: Object diagram of online potato collection company

Credit (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

……..

Debit (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

……

Money order (Object)

Notion

- It is the **method** through which a student can pay bill to purchase the **book.**

  ……

Fig 4.40 shows the UML class diagram of the online potato collection company.

Transformation from Class Diagram to Object Diagram

By applying the transformation rules to the class diagram of the online potato collection company shown in Fig 4.40, the object diagram shown in Fig 4.41 is obtained.

Chapter 5

Future Scope and Conclusion

5.1 Conclusion

Software product line engineering established to be one of the most successful paradigms for developing a variety of similar software applications and software-intensive systems at low costs, in short time, and with high quality, by exploiting commonalities and variability between products to attain high levels of reuse. The software reuse programs is applied in an effort to improve productivity and quality, programs are decomposed into appropriate cross-cutting building blocks, i.e., a building block affects many parts of the combined program.AOP enables modular implementation of crosscutting concerns. Using AOP, programmers get profit of modular reasoning for crosscutting concerns whereas without AOP they did not. Aspects oriented technology is useful for solving the problems of code scattering and tangling. The development of component reuse will prompt to the reuse of aspect, as they were orthogonal in the system.

Domain analysis can be seen as an extension of the conventional requirements analysis [4]. The primary advantage of domain analysis is that it provides flexibility. We have done domain analysis of ATM with JODA and DADP. Code reusability is common in both methods. JODA supports stability, coupling, cohesion, reusability and usefulness while DADP supports reusability, expandability and flexibility. DADP focuses on design also which can help identify problems and solutions also. The primary conclusion is that, when done right, domain analysis is a significant undertaking. Yet there is evidence that a significant benefit can be realized when reuse is systematically integrated into the development process. Even though the effort for domain analysis may be high, it is recommended that all steps must be taken in performing a domain analysis. Anything less will add risks to the potential payoff of the domain analysis and associated reusable assets.

Feature Model can be used as the source for software development in the context of Domain Analysis. It describes commonalities and variabilities of systems within a

domain. It would be essential to include linguistic approaches to achieve a better processing of the information. In spite of the wide use of many semi-formal notations, during the initial stages of software development, natural language is still chosen for describing the requirements of a software system because it contributes to elicit, model, and communicate requirements in an easy and friendly way [37, 38]. LEL provides a complete description of an application.LEL helps to set and merge the domain concepts allowing stakeholders to share a common vocabulary. It encourages and facilitates stakeholders contribution and on the other side, the accurate structure and well-defined construction process, it has made the manipulation of the information easier, and thus this information may be reinterpreted into more formal descriptions for example, a UML class diagram. The use of LEL and UML class diagram is motivated by the truth that stakeholders frequently speak of product characteristics in terms of "the product has and/or delivers" using them in order to communicate their problems, ideas and needs. Here transformation processes, using natural language oriented requirements models (LEL) to derive UML class diagram and object diagram from them are described. As the elements in class diagram are in abstract form which represents the blue print while the elements in object diagram are in concrete form which represents the real world object.

The features of the case study online transaction system are reused in other case studies. The store front and back office features of the online transaction system are reused in case studies of the admission process, ATM, online dairy farm, online book shop and online potato collection company. Similarly payment, method, credit, debit, money order and customer registration features of the online transaction system are reused in case studies of the admission process, online dairy farm, online book shop and online potato collection. The same way homepage feature of the online transaction system is reused in case studies of the online dairy farm, online book shop and online potato collection. It improves quality, increases productivity, reduces cost and time.

5.2 Future Scope

Even though the general notion of separation of concerns is an old idea, one can witness the nascence of a research area devoted to the investigation of new techniques to maintain advanced separation of concerns.

As future work, a new domain analysis method can be proposed for better domain analysis and more quality attributes may be added to each domain analysis method and steps can be improved to achieve this. It will increase the efficiency of domain analysis methods incorporating the quality in domain analysis.

As future work, we must improve some transformation rules in order to attain a better definition of the relationships between UML classes as well as between UML objects. Though, we want to improve this transformation process by defining another corresponding and independent model to capture and signify the relationships produced by the application of the transformation rules.

References

[1] A. Jatain and S. Goel , "Comparison of Domain Analysis Methods in Software Reuse", International Journal of Information Technology and Knowledge Management, Vol. 2, No. 2, pp. 347-352, 2009.

[2] R. Holibaugh, "Joint Integrated Avionics Working Group (JIAWG) Object-Oriented Domain Analysis (JODA)", Software Engineering Institute, Carnegie Mellon University, Special Report CMU/SEI-92-SR-003, 1992.

[3] S. Cohen and L. M. Northrop, "Object-Oriented Technology and Domain Analysis", In proceeding of ICSR 5th International Conference on Software Reuse, SEI, pp. 86-93, 1998.

[4] M. Simos, et. al., Organization Domain Modeling (ODM) Guidebook, Version 2.0. (STARS- VC-A025/001/00) Lockheed Martin Federal Systems, 1996.

[5] H. Mili, A. Mili, S. Yacoub and E. Addy, Reuse Based Software Engineering, John Wiley &Sons, 2002.

[6] M. Bowman, S.K. Debray and L.L. Peterson, "Reasoning about naming systems", ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 15, Issue 5, pp. 795 – 825, 1993.

[7] M. Mannion, J. Savolainen and T. Asikainen, "Viewpoint-oriented variability modeling", In proceedings of the IEEE International Computer Software and Applications Conference, 2009.

[8] P. Clements and L. Northrop, Software Product Lines: Practices and Pattern, Addison-Wesley Professional, August 2001.

[9] L. Blair and J. Pang, "Aspect-Oriented Solutions to Feature Interaction Concerns using AspectJ", In Proceedings of Seventh International Workshop on Feature

Interactions in Telecommunication Networks and Distributed Systems (FIW'03), 2003.

[10] K. Lee and K. C. Kang, "Feature Dependency Analysis for Product Line Component Design", Lecture Notes in Computer Science, Vol.3107, pp.69-85, 2004

[11] J. Coplien, D. Hoffmann, D. Weiss, "Commonality and Variability in Software Transactions Engineering", IEEE Software, Vol.15, No.6, pp. 37–45, November/ December 1998.

[12] M. Mannion, "Organizing for Software Product Line Engineering", In Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02) IEEE, 2002.

[13] R. Augustine, Seminar Report on Aspect Oriented Programming, Department Of Computer Science, Cochin University of Science and Technology, Kochi-682022,2007.

[14] M. Voelter, and I. Grover, "Product Line Implementation using Aspect-Oriented and Model-Driven Software Development", SPLC '07 Proceedings of the 11th International Software product Line Conference. IEEE Computer Society Washington, DC, pp. 233-242,2007.

[15] F. Losavio, A. Matteo and P. Morantes, "UML Extensions for Aspect Oriented Software Development", Journal of Object Technology, Vol.8, No.5, 2009, ETH Zurich, Chair of Software Engineering, Venezuela. Available from: http://www.jot.fm.

[16] E.S. de Almeida, A. Alvaro,D. Lucredio,V. C. Garcia,S. R. de Lemos Meira "A Survey on Software Reuse Processes", In Proceedings of the IEEE International conference on Information Reuse and Integration, pp.66-71, 2005.

[17] R. P. Díaz, "Domain Analysis: An Introuction", ACM SIGSOFT Software Engineering Notes, Volume 15, Issue 2, pp. 47 – 54, 1990.

[18] E. R. Comer, “Domain Analysis: A System Approach to Software Reuse”, In Proceedings of IEEE/AIAA/NASA 9th Conference on Digital Avionics Systems, pp: 224 – 229, 1990.

[19] X. Ferré, S.Vegas, “ An Evaluation of Domain Analysis Methods ”, In 4th CAiSE/IFIP8.1 International Workshop in Evaluation of Modeling Methods in Systems Analysis and Design,1999.

[20] R. W. Krut, Jr., “Integrating 001 Tool Support into the Feature-Oriented Domain Analysis Methodology”, Software Engineering Institute, Carnegie Mellon University, Technical Report: CMU/SEI-93-TR-011, 1993.

[21] A. Suganthy, T. Chithralekha, “Domain-Specific Architecture for Software Agents”, Journal of Object Technology, Volume 7, No. 6, pp. 77-100, 2008.

[22] A. Foka, Case Study: ATM machine I. CEID, University of Patras, Object Oriented Programming II (C++), 2010-2011.

[23] K. Czarnecki, S. Helsen and U. Eisenecker, “Staged configuration using feature models”, In Proceedings of the SPLC on Software Product Line, pp. 266–283, 2004.

[24] K. Kang, S. Kim, J. Lee and K. Kim, “FORM: a feature-oriented reuse method with domain-specific reference architectures”, Annals of Software Engineering, 1998.

[25] K. Lee, K. Kang and J. Lee, “Concepts and guidelines of feature modeling for product line software engineering”, In Proceedings of the ICSR-7 on Software Reuse, 2002.

[26] K. Czarnecki, and U. Eisenecker, “Generative programming: methods, tools and applications”, Addison-Wesley, Reading, MA, USA, June 2000.

[27] D. Benavides, S. Segura and A. Ruiz-Cortes, “Automated analysis of feature models 20 years later: A literature review”. Information Systems, Vol. 35, No.6, pp. 615 – 636, 2010.

[28] J.C.S.P. Leite, and A.P.M. Franco, "A Strategy for Conceptual Model Acquisition", In Proceedings of the First IEEE International Symposium on Requirements Engineering, pp. 243-246, 1993.

[29] M. C. Leonardi, M. V. Mauco, L. Felice, G. Montejano, D. Riesco and N. Debnath, "Integrating Natural Language Oriented Models with Feature Model", IEEE, pp. 185-190, 2009.

[30] C. Constantinos, "In: Revisiting Separation of Concerns with an Aspect-Oriented Framework", 14th European Conference on Object-Oriented Programming .Sophia Antipolis and Cannes, France, 12-16 June, 2000.

[31] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lapes, J.M. Longtier and J. Irmin, "Aspect-oriented Programming", In Proceeding of ECOOP'97, Springer Verlag, 1997.

[32] H. Siy, P. Aryal, V. Winter and M. Zand, "Aspectual Support for Specifying Requirements in Software Product Lines", Workshop on Early Aspects at ICSE, Minneapolis, USA, 2007.

[33] A. Nyssen, S. Tyszberowic and T. Weiler, "Are aspects useful for managing variability in software product lines? A case study", Aspects and Software Product Lines: An Early Aspects Workshop, at SPLC-Europe'05, 2005.

[34] www.wikipedia.com(http://en.wikipedia.org/wiki/Automated_teller_machine)

[35] N. Debnath, M. C. Leonardi, M. Ridao, M. V. Mauco, L. Felice ,G. Montejano and D. Riesco, "An ATL Transformation from Natural Language Requirements Models to Business Models of a MDA Project", IEEE, pp. 633-639, 2011.

[36] N. Juristo, A. Moreno and M.López, "How to Use Linguistic Instruments for Object- Oriented Analysis", IEEE Software, Vol. 17, No.3, pp. 80-89, 2000.

[37] B. Berry, A. Bucchiarone,S. Gnesi, G. Lami, G. and G. Trentani, “A new quality model for natural languages requirements specifications”, In Proceedings of the Twelfth International Workshop on Requirements Engineering: Foundation for Software Quality, 2006.

[38] B.R. Bryant, B. Lee, F. Cao, W. Zhao, C. Burt, R. Raje, A. Olson and M. Auguston, “From natural language requirements to executable models of software components”, In Proceedings of Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation, 2003.

List of Publications

[1] Megha Bhushan and Shivani Goel, "Quality Attributes in Domain Analysis", published in IEEE-ICAER, 11 held on October 15, 2011 at UIET, Panjab University, Chandigarh.

[2] Megha and Shivani Goel, "Role of Aspect Oriented Programming in Software Product Line", in proceedings International Conference on Advanced Computing Technologies (ICACT-2012), held on 8th -9th June, 2012 at Gurukul Vidyapeeth Institute of Engineering and Technology, Chandigarh (Accepted) .

[3] Megha and Shivani Goel, "Role of Aspect Oriented Programming in Software Product Line", International journal of Computers and Technology, ISSN No. 2277-3061 (Accepted).

[4] Megha and Shivani Goel, "Transformation from LEL to UML", International journal of computer application, ISSN: 0975 – 8887(Communicated).