Representing Nonfunctional Requirements using UML2

Submitted by

Irurn Ilyas
(128-FAS/MSSE/F06)

Supervised By:
Dr. Naveed Ikram

Co-Supervised By:
Atif Qureshi

Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University, Islamabad
2008

Representing Nonfunctional Requirements using UML2

By

Irum Ilyas
128-FAS/MSSE/F06

A Thesis

Submitted in partial fulfillment of the requirement for

the award of degree of

Master of Science (MS)

In Software Engineering

Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University, Islamabad

2008

Department of Computer Science
Faculty of Basic and Applied Sciences
International Islamic University
Islamabad.

Date: 11th March, 2009

Final Approval

It is certified that we have read the thesis submitted by Miss. Irum Ilyas and it is our judgment that this thesis is of sufficient standard to warrant its acceptance by IIUI for the degree of Master of Science in Software Engineering.

Committee:

External Examiner:

Dr. Iftikhar Niaz
Head of Department of Computer Science,
Ripha International University.

Internal Examiner:

Zohaih Zafar [illegible]
Lecturer.
Department of Computer science, FBAS, IIUI.

Supervisor:

Dr. Naveed Ikram
Associate Professor,
Department of Computer science, FBAS, IIUI.

Co-Supervisor:

Atif Qureshi
Lecturer,
Department of Computer science, FBAS, IIUI.

Abstract

The way requirements should drive the rest of the software development process has been a subject of many research projects in the past. Recent research shows that software development demands early incorporation of the nonfunctional requirements in order to produce quality software that can meet today's challenges. Customers are demanding quality which can only be achieved by considering nonfunctional requirements as early as possible. Since errors due to nonfunctional requirements are the most expensive and difficult to correct, improperly dealing with nonfunctional requirements can lead to more expensive software and a longer time to the market. This research work tries to fill the gap of identification of nonfunctional requirements and then how to deal nonfunctional requirements in software systems during design process. In this research I find how to tackles the problems of nonfunctional requirements, and how the models will satisfice these nonfunctional requirements. The models for the incorporation of nonfunctional requirements are including from Unified Modeling Language v2 artifacts. The Unified Modeling Language v2 artifacts are interaction overview diagram, composite structure diagram and component diagram. This integration can be used in the early stage of software development with ongoing projects or to enhance even legacy system with nonfunctional requirements. I validate my approach by performing a case study.

Acknowledgement

Primary and foremost, all praises for Almighty Allah, the kind and merciful. the creator of the universe, who provided me the right ability, strength and courage to complete the work presented here. I invoke peace for Hazrat Muhammad (peace be upon him), the last prophet of Allah who is eternally a torch of guidance for humanity as a whole.

Working on this study was the most valuable and worth learning experience of my life. It was the time to put the learning and knowledge of the past semesters to test. Along with the unique learning experience came the realization that there is so much to be learned from the practical field that cannot be gained from the books alone. After the completion of this dissertation, I think that, although, I have ventured every possible measure to make it a presentable piece of work, yet man is fallible and I am not an exception.

I would like to express my deep and sincere gratitude to my co-supervisor, Atif Qureshi and Supervisor, Dr. Naveed Ikram, Associate Professor in department of computer science and also to Department of Computer Science. Both supervisors have wide knowledge and logical way of thinking has been a great importance for me. Both have understanding, encouraging and personal guidance have provided a good basis for the present thesis.

Lastly, I am grateful to my family for the inspiration and moral support. I would not be here without their tremendously encouragement and understanding.

Irum flays

LIST OF ACRONYMS

Abbreviation	Title
RE	Requirement Engineering
SE	Software Engineering
NFRs	Non-Functional Requirements
FRs	Functional Requirements
UML	Unified Modeling Language
IOD	Interaction Overview Diagram
CSD	Composite Structure Diagram
CD	Component Diagram
SRS	Software Requirement Specification
POST	Point of Sale Terminal
OCL	Object Constraint Language
CBD	Component Based Development
L EL	Language Extended Lexicon
CrAS	Credit Authorization Service
ChAS	Check Authorization Service
SEI	Software Engineering Institute
XML	extensible Markup Language

TABLE OF CONTENTS

LIST OF FIGURES /GRAPHS

LIST OF TABLES

Chapter 1

Introduction

1. Introduction

This chapter describes the problem domain, main contributions and research problem, research method and outline of thesis.

1.1 Problem Domain

Requirement engineering (RE) is critical for the success of any major development project. The success of a software system [Cheng & Atlee, 2007] depends on how well it fits the needs of its users and its environment. Software requirements (SR) include these needs, and RE is the process through which the requirements are determined. Successful RE involves understanding the users need, customers, and other stakeholders; understanding the contexts in which the to be developed software will be used; modeling, analyzing, negotiating, and documenting the stakeholders requirements; validating that the documented requirements match the negotiated requirements; and managing requirements evolution [Cheng & Atlee, 2007] .

Software system, aside from implementing the entire functional requirements (FRs) must also deal with the nonfunctional requirements (NFRs) [Cysneiro's & Leite, 2004]. NFRs should be dealt with from the beginning and throughout the software development process [Cysneiro's & Leite, 2004]. NFRs have been frequently neglected or forgotten in software design [Cysneiro's & Leite, 2004]. NFRs were not considered when we model the FRs in use cases, sequence diagram, communication diagram and all other diagrams. NFRs are very important for the success of every project.

During the last decade NFRs got importance and from the literature it is realized that NFRs are very important for the success of every project [Cysneiro's & Leite, 2004], [Cysneiro's & Yu, 2003] & [Chung & Yu, 2000]. Besides the basic functionality, nonfunctional aspects are demanded by the market. Usually nonfunctional aspects are treated only at the design stage of a software system. And all these nonfunctional aspects must be treated as NFRs of the software [Cysneiro's & Leite, 1999]. NFRs should be dealt throughout the software development process [Cysneiro's & Leite, 2004].

Literature has been pointing out that NFRs are very difficult to achieve and at the same time are expensive to deal [Cysneiro's & Leite, 2004]. Ineffectively dealing with NFRs has led to a series of failures in software development, the case of London Ambulance System [Finkelstein & Dowell, 1996], where the deactivation of the system right after its deployment was strongly influenced by NFRs noncompliance [Cysneiro's & Leite, 2004] & [Finkelstein & Dowell, 1996]. Errors due to omission of NFRs or not properly dealing with them are among the most expensive type and most difficult to correct [Cysneiro's & Yu, 2003].

There have been reports showing that not properly dealing with NFRs have led to considerable delays in the project and consequently to significant increases in the final cost [Cysneiro's & Leite, 1999]. The development of a real time system by Paramax System Corp. experienced major delays in its deadlines and significant increasing costs which put the deployment in risk [Cysneiro's & Yu, 2003]. There were many reasons for that, but one of the most important reasons relies on the fact that performance (NFR) was neglected during the development of the software leading to several changes in both hardware and software architecture, as well as in both the design and code of the software [Cysneiro's & Leite, 2004].

1.2 Main Contributions

NFRs are gained very importance during last decade, yet it is continues to have promise in advancing the field. But not as much work had done in this area as other fields. The main inspiration for undertaking my research work comes from [Cysneiro's & Leite, 2004] that puts light on future focus of NFRs in software design. In this paper, they present a process to elicit NFRs, analyze their interdependencies, and trace them to functional conceptual models. This paper also suggests different directions for future research one of which is dealing with other Unified Modeling Language (UML) artifacts.

Mostly work has done in this field by the inspiration of Chung's work [Mylopoulos & Chung, 1992], [Chung & Nixon, 1995] & [Chung & Yu, 2000]. [Mylopoulos & Chung, 1992] proposes a comprehensive framework for representing and using NFRs during the development process. The framework consists of five basic components which provide for the representation of NFRs in terms of interrelated goals. Such goals can be refined through refinement methods and can be evaluated in order to determine the degree to which a set of NFRs is supported by a particular design.

Chung's paper [Chung & Nixon, 1995] are dealing with NFRs and have conducted three empirical studies of small portions of software systems, in order to give an initial evaluation of a framework [Mylopoulos & Chung, 1992] for dealing with NFRs in the software development process. The studies dealt with several NFRs, primarily accuracy, security and performance.

Leite, 2001]. NFRs have also been integrated with functional models [L.Cysneiros & Leite, 2004] [L.Cysneiros & Leite, 2001].

My contribution will contribute to fill the gap of identification of NFRs and then how to deal NFRs in software systems during design process. In this research I find how to tackles the problems of NFRs, and how the models will satisfice these NFRs.

Software systems are becoming large and complex day by day. This complexity includes not only static structure of classes but their relationship with each other their functionality, behavior, state etc., it is difficult to grasp this information as a whole for any system which leads to misunderstanding of that system. Modeling of software systems help to minimize this complexity by abstracting out vital information from that system. Software systems can be modeled from different point of view-s.

According to the 4+1 architectural views model was proposed by Kruchten [P. Kruchten,1995], the views are structural view, behavioral vicw, implementation view, environmental view and one is use case view. As I found in literature [L.Cysneiros & Leite, 20011, [L.Cysneiros & Leite, 2004] the work on NFRs modeling of use case view and some diagrams of structural view and behavioral view has already been done. There is need to do some work on behavioral view and structural view and implementation view. As there is no work done on implementation view so I am continuing my research on implementation view, CSD and CD are considered in my research from implementation view. CSD is also included in structural view. As these diagrams have a lack of information about non functional aspects of the software system, especially NFRs related to different components and their implementation. In literature I find that there is need to improve the understanding of NFRs at component and implementation level and also to complete the understanding of NFRs in

these diagrams to make these diagrams complete and also to improve the quality of software system at run time in implementation view.

In behavioral view, there is work done on sequence diagram and collaboration diagram. But some diagrams are left for the integration of NFRs. The focus of my research work on behavioral view is on IOD, this diagram is a combination of interaction diagram including sequence diagram, communication diagram and timing diagram. I want to find that, is there any impact in diagram when we combine interaction diagrams in an IOD, any addition of instances, classes, association or any NFR which may change the IOD. The literature of software engineering does not provide any mechanism to incorporate NFRs in these types of models.

My research focus is to find out a mechanism to incorporate NFRs in these views, so that it will help us to understand that particular view (Implementation, Behavioral) of system.

The research questions of this thesis are:

1. How to deal and tackle the problems of NFRs in software during design process?
2. How NFRs can be incorporated in UML 2 models?

1.5 Outline of thesis

The rest of the thesis is organized as follows:

Chapter **2** presents a survey on the related research work. It covers two research fields namely:

i) NFRs modeling, its importance, its approaches and how we deal with NFRs; and

ii) UML2 Diagrams which are used for modeling named as IOD, CSD and CD.

Chapter **3** proposes a strategy how we integrate NFRs in the UML2 diagrams. The strategy deals with UML2 diagrams to represent the NFRs.

Chapter 4 is the case study which shows the implementation of the strategy. The performance and security of a POST are presented as a proof of concept.

Chapter 5 presents conclusion and discusses the future research direction.

Chapter 2

Literature Review

2. Literature Review

Requirements are the essential part for the development of any system. Requirements are FRs and NFRs. FRs captures the proposed behavior of the system, in terms of the services or tasks the system is required to perform. NFRs along with FRs play a significant role in software development. This chapter covers the concept of NFRs. The section deals with the basic concept of NFRs and importance to consideration of NFRs and approaches for modeling of NFRs, activities of NFRs, NFR framework and application of NFR framework in UML artifacts.

2.1 NFRs

2.1.1 What are NFRs?

IEEE defines NFRs as:

"NFRs in sofhvare system engineering are a software requirement that describes not what the software will do, but how the software will do it, for example, software performance requirements, software external interface requirements, design constraints, and software quality attributes. NFRs are difficult to test; therefore, they are usually evaluated subjectively" [Subrina, 2006].

NFRs define global constraints on a software system or subsystem, on a functional requirement, on the development process or on the deployment process. They are global in the sense that they arise from all parts of the system and from their interactions [L.Cysneiros & Yu, 20031.

A requirement that specifies system properties, such as environmental and implementation constraints, performance, platform dependencies, maintainability, extensibility, and reliability. **A** requirement that specifies physical constraints on a functional requirements [I. Jacobson, Booch & Rumbaugh, 1999].

Requirements which are not specifically concerned with the functionality of a system. They place restrictions on the product being developed and the development process, and they specify external constraints that the product must meet [G. Kotonya & Sommerville, 1998].

"NFRs are global requirements on its development or operational cost, performance, reliability, maintainability, portability, robustness, and the like" [Mylopoulos & Chung, 1992].

NFRs are also known as Quality Requirements [Mylopoulos & Chung , 1992] & [Boehm & Barry , 1996] and distinct to FRs, NFRs state constraints to the system as well as particular notion of qualities a system might have, for example, security, reliability, performance, safety, usability, accuracy. So we can say that FRs state "what" the system must do, whereas NFRs constrain "how" the system must accomplish the "what". As a consequence, NFRs are always linked to FRs [ISSCO, 1995] [Kimer & Davis, 1996].

NFRs are requirements that impose restrictions on the product being developed [L. Xu & Ziv, 2005].

Software systems implementing all the desired functionality must also deal with the non functional requirements of the system. NFRs are constraints or conditions on the functionality of a system. NFR is focused on how the software must perform something instead of on what the software must do. NFRs are the requirements such as reliability,

security, accuracy, safety, look and feel requirements, performance, as well as organizational, political and cultural requirements.

2.1.2 Why NFRs are important?

There has been a lot of work showing that complex systems must deal with non-functional aspects [Mylopoulos & Chung, 1992] & [Chung & Nixon, 1995]. These nonfunctional aspects should be dealt within the process of NFRs definition [Cysneiro's & Yu, 2003].

Errors due to the ignorance of NFRs or not properly dealing with them are among the most expensive type and most difficult to correct [Mylopoulos & Chung, 1992], [Cysneiro's & Leite, 1999] & [Ebert, 1997].

Besides that, the market is increasing its demands for software that implemented all the desired functionality but also copes with nonfunctional aspects such as: reliability, security, accuracy, safety, performance as well as others [Cysneiro's & Leite, 2004]. These nonfunctional aspects must be treated as NFRs of the software. They still, should be dealt with from the beginning of software development process [Chung & Nixon, 1995] & [Subrina & Tahvildari,], throughout the whole life cycle.

Although NFRs have been presented in many software development methods, but they are presented in later stages of requirement, not deal with the first class of requirement like in requirement elicitation phase. NFRs should be considered in requirement elicitation phase. Then model properly into the design phase with FRs in the UML artifacts. Otherwise. they are not proceeding on the next phases of the software. Modeling NFRs allows them to be organized for better visualization and understanding. It will help software engineers to analyze NFRs [Cysneiro's & Yu, 2003]. When NFRs are not properly model in the design

phase which may lack the deficiencies of the software. Researchers consider the design of software as the basic foundation of building a high quality product. Some examples of systems are given as a proof, which shows that the deficiency of the system due to ignorance of NFRs in design phase and not properly model.

A more serious problem related to NFRs can be seen in the London Ambulance Service Report [Finkelstein & Dowell, 1996] & [Cysneiro's & Yu, 2003]. The London Ambulance System was deactivated just after its deployment because, among other reasons, many NFRs were neglected during the system development such as: reliability (vehicles location), cost (emphasis on the best price), usability (poor control of information on the screen), and performance (the system did what was supposed to do but he performance was unacceptable).

The development of a real time system by Paramax System Corp. experienced major delays in its deadlines and significant increasing costs which put the deployment in risk. There were many reasons for that, but one of the most important reasons relies on the fact that performance was neglected during the development of the software leading to several changes in both hardware and software architecture, as well as in either the design or code of the software [Cysneiro's & Yu, 2003] & [Lindstrom, 1993].

2.1.3 Approaches for dealing NFRs

Most of the early work on NFRs focused on measuring how much a software system is in accordance with the set of NFRs that it should satisfy, using some form of quantitative analysis [Fenton & Pfleeger , 1997] & [Keller , 1990] offering predefined metrics to assess the degree to which a given software object meets a particular NFRs.

Previous research may be either characterized as process oriented or product oriented. Process oriented technique to integrate NFRs into the design proves while product oriented approaches focus on evaluating the end product to determine whether it satisfies the NFRs [Hill & Wang, 2004].

Product-oriented approaches are those concerned with measuring how much software complies with NFRs. They do not help to prevent problems but are helpful to evaluate the degree of compliance with non-functional needs [Cysneiro's & Yu, 2003].

Process-oriented approaches focus on the software development process. It aims to help software engineers searching for alternatives to sufficiently meet NFRs while developing the software [Cysneiro's & Yu, 2003].

Recently, a number of works proposed to use approaches which explicitly deal with NFRs before metrics are applicable [L.Cysneiros & Leite, 2001], [Chung & Yu, 2000]. These works propose the use of techniques to justify design decisions on the inclusion or exclusion of requirements which will impact on the software design. Unlike the metrics approaches, these latter approaches are concerned about making NFRs a relevant and important part of the software development process [Cysneiro's & Yu, 2003].

2.1.5 Related Work

The idea of integrating NFRs with FRs at design level is not a new one. A survey of the approaches found that most of the research works used the UML model to represent a system as shown in table2.2 and these works propose some extensions in order to add NFRs during the design phase with the model representation of the FRs. The different frameworks proposed by different researchers are included in related work and how these frameworks are applied to UML artifacts.

2.1.5.1 NFR Framework

Lawrence Chung et al. [L. Chung & J. Mylopoulos 1992] presents a comprehensive framework for representing and using NFRs during the development process. The framework consists of five basic components which provide for the representation of NFRs in terms of interrelated goals. Such goals can be refined through refinement methods and can be evaluated in order to determine the degree to which a set of NFRs is supported by a particular design. The framework consists of five major components: a set of goals for representing NFRs, design decisions, and arguments in support of or against other goals: a set of link types for relating goals or goal relationships (hereafter links) to other goals; a set of generic methods for refining goals into other goals; a collection of correlation rules for inferring potential interactions among goals; and finally, a labeling procedure which determines the degree to which any given NFRs is being addressed by a set of design decision. During the design process, goals are organized into a goal graph structure, very much in the spirit of AND/OR trees used in problem solving.

Luiz Marcio Cysneiro's et al [L.Cysneiros & Leite, 2004] present a process to elicit NFRs, analyze their interdependencies, and trace them to functional conceptual models. They focus their attention on conceptual models expressed using UML. Extensions to UML are proposed to allow NFRs to be expressed. They showed how to integrate NFRs into the class, sequence, and collaboration diagrams. They also showed how use cases and scenarios can be adapted to deal with NFRs. They use the Language Extended Lexicon (LEL) driven approach to describe the application domain in LEL to provide context for both FRs and NFRs. This policy assures that a common and controlled vocabulary is used in both functional and nonfunctional representations. Later they analyze those domains separately and build the functional view of the system using UML diagrams. Then they build the non-functional view of the system using NFR framework. They extend the NFR framework to adopt their notations. Finally, they integrate the NFRs with the functional representation of the system by proposing some extensions to UML models.

Moreira et al [Ana. Moreira & Brito, 2002] propose a model for integrating crosscutting quality attributes with FRs by the UML use case diagram and sequence diagram. They adopt NFR Framework's goal analysis Framework (without visual notations and diagrams) to analyze NFRs textually for cross-cutting relevance to one or more use cases. The template they proposed to specify a quality attributes was also influenced by the approaches of Chung et al [J. Mylopoulos & Chung , 1992] .They propose a template for quality attributes with specific fields, including description, focus, source, decomposition, priority, obligation, and influence. Then, they integrate those quality attributes with FRs using standard UML diagrammatic representations (e.g. use case diagram, sequence diagrams) extended with some special notations.

Subrina **Anjum et al** [Subrina & Tahvildari, 2005] proposes a framework to incorporate NFRs, as reusable components, with standard UML notations. This framework can also be integrating those reusable NFRs with the extracted UML representations of legacy systems during the reverse engineering process. This framework uses the standard XMI representation of UML models without proposing any extension to it. This framework starts with such extracted UML model of a legacy system. It consists of three phases 1) Identification of FRs and NFRs, 2) Specification of FRs and NFRs and 3) Integration of NFRs.

Dimitrov **et al** [E. Dimitrov & Schmietendorf, 2002] describe three approaches for UML-based performance engineering. The three approaches are: 1) Direct representation of performance aspects using UML, 2) Expanding UML to deal with performance aspects and 3) Combining UML with formal description techniques.

In the first approach, they propose some methods for specifying performance aspects with some of the UML models with standard UML notations. For the use case model, they define a load and time-weighted use case diagram for specifying performance aspects with the standard UML notations. For the interaction diagram, they propose to add some additional time information by labeling the messages with relative constraints such as assigning time attributes to the messages and to the method executions. According to this approach the labeling of messages with time will be interpreted as latency and labeling of methods with time will be interpreted as time for method execution. For state transition diagram. they proposed to assign a thinking time along with the probability assigned to each transition. In the second approach, they expand the UML for supporting Real-time Object-Oriented Modeling methods for supporting performance aspect. They used UML tailoring mechanism

which are based on stereotypes, tagged values, and constraints to convert the Real-time Object-Oriented Modeling constructs to UML. In the third approach they combine UML with Message Sequence Charts and software development lifecycle type formal description techniques to support performance of a software system.

The frameworks on NFRs are defined above as different researchers proposed different frameworks. As mentioned above research works, some focused on particular NFR and some are system dependent. Cysneiro used the NFR framework which described by Chung. They proposed the strategies for the integration of NFRs on UML artifacts. So Cysneiro works inspired from Chung worked. The work of Cysneiros is major work on integration of NFRs on UML artifacts. Chung works on NFR framework, this NFR framework is most comprehensive for the representing and using nonfunctional requirements during the development process. Dimitrov explained his work on specific NFR, and apply on a specific system; they didn't propose any framework for the integration of NFRs. Moreira proposed a template to specify a quality attributes was also influenced by the approaches of Chung. Moreira works on NFR related to Cysneiros as Moreira described template to record quality attributes and Cysneiros used LEL to record NFRs. Cysneiros used the NFR graphs for the operationalization of NFR which is also based on Chung's worked, whereas Moreira didn't used any graphs for the operationalization of NFRs. Subrina described a framework for integrating NFRs with the UML design of a software system which can be applied during the re-engineering process of a legacy system. Although the framework can also be used during forward engineering but when if the developers follow the standard XMI during their model design. My concern is related to forward engineering so Subrina works also specific in case

of forward engineering, if the developers can't follow the XMI, so they can't apply this framework for the integration of NFRs.

My contribution will contribute to fill the gap of identification of NFRs and then how to deal NFRs in software systems during design process. I am proposing a strategy that tackles the problems of NFRs, and proposing a systematic process to assure that the models will satisfice these NFRs. This strategy is based on the use of domain glossary to build functional and nonfunctional perspectives. Using this glossary, I shall show how to record NFRs against FRs. Then how to integrate NFRs in UML 2 models, and present a systematic way to integrate NFRs into the functional model of UML 2.

2.1.5.2 Application of NFR Framework on UML Artifacts

The application of NFR framework on different UML artifacts like use case diagram, class diagram, sequence diagram, collaboration diagram and activity diagram. Different researcher applied different ways for the integration of NFRs in UML artifacts.

2.1.5.2.1 Application of NFR Framework on Use case

Supakkul & **Chung** et **al** [S. Supakkul & Chung, 2004] proposed a use case and goal-driven approach to integrate FRs and NFRs. They used the UML use case model to capture functionality of the system and they also used the NFR Framework [L. Chung & Yu, 2000] to represent NFRs. They proposed to associate the NFRs with four use case model elements: actor, use case, actor-use case association and the system boundary. Figure 2.1 shows these NFR association points. They name these associations \Actor Association Point", \Use Case

Association Point", \Actor-Use Case Association (AU-A) Point", and \System Boundary Association Point" respectively. Having such an extension to the UML use case model. NFRs can be integrated at the design level with FRs and can provide better understanding of the requirements model.

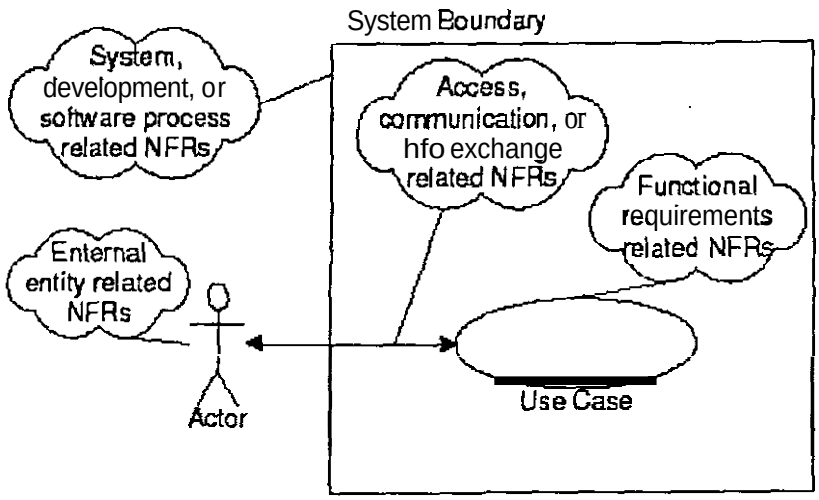


Figure 2.1: NFR Association points [S. Supakkul & Chung, 2004]

Cysneiros et al [L.Cysneiros & Leite, 2004] used the strategy for the integration of NFRs in use case diagrams. They first identified LEL symbol appears in the use case diagram. They also identified if any LEL symbol appears in any of the use cases or actors of the diagram. For each LEL symbol they find, they search the set of NFR graphs to identify those where this symbol appears. They may eventually, find one or more NFR graphs that contain the symbol being searched for. They take every graph where the symbol appears and check if the use case diagram realizes the dynamic operationalizations in the graph. LEL intends to capture every meaningful term used in the Universe of Discourse. If a use case diagram does not have at least one LEL symbol, either there are symbols in LEL that may have aliases not

yet specified or a symbol is missing. Most of all, all the actors in a use case diagram must be LEL symbols. If a symbol is missing, LEL must be updated and all the processes that are carried out within the nonfunctional perspective must take place again, e.g., search for NFRs that may apply to this symbol, create NFR graphs. Every use case or actor included, due to NFR satisfaction, must be followed by an expression using the pattern: {NFR_Type [NFR_topic]}. The use of this expression aims at adding traceability between the functional and nonfunctional perspectives.

Moreira et al [A.Moreira & Brito, 2002] proposed a strategy to first identify requirements then specify it and then integrate it. According to his approach which is applied on use case, first identify requirements then from those requirements identify actors and use cases and identify quality attributes from those requirements. After identifying requirements and actors and use case then specify all these in use case diagram and then specify quality attributes in a template. All quality attributes are specified or recorded in a template with this information: "name, including description, focus, source, decomposition, priority, *obligation*, influence, where, requirements and contribution". Then finally integrate these quality requirements in a use case diagram. They augmented the use case diagram to include a new stereotyped use case for each quality attribute and make the crosscut initial use cases include the new ones. Then they showed these stereotypes in the use case diagram with the include relationship.

Dimitrov et al [E. Dimitrov & Schmietendorf, 2002] used a process based on ISO 14756 for a formalized description of the workload within the framework of use case diagrams. This requires many steps: identifying activity types and recording the number and types of users, deriving task types by assigning activity types to task mode and defining service level requirements, defining chain types as fixed sequence of task types, defining the percentage

occurrence frequency q and the user preparation time for concrete chain types while the program runs and laying down the reference yalue of mean execution time t_{Ref} (response times) for concrete task types. They define a load and time-weighted use case diagram for specifying performance aspects with the standard UML notations.

The different researchers had done a different work on integration of NFRs in use case diagram. Moreira and Cysneiros worked on use case diagram both works inspiration come from Chung work [L. Chung & Yu, 2000]. For the integration of NFRs both researchers used the Chung work on NFR. Chung [L. Chung & Yu, 2000] described the most comprehensive framework on NFR for representing and dealing with NFR. Cysneiros used the NFR graphs for the integration of NFRs in use case diagram from Chung NFR framework [L. Chung & Yu, 2000]. Moreira used the approaches of Chung for the integration of NFRs in use case diagram and integration is also familiar with the Cysneiros worked on use case diagram. Dimitrov worked on NFR for just a specific NFR which is performance and didn't give any work related to any other NFR.

2.1.5.2.2 Application of NFR Framework on Class Diagram

Cysneiros et al [L.Cysneiros & Leite, 2001], [L.Cysneiros & Leite, 2004] described integration of the nonfunctional perspective into the class model is based on the use of LEL. It means that every class belonging to the class diagram has to be named using a LEL symbol. The use of LEL as an anchor to construct both perspectives is facilitating their integration. It can also be used for validating both models since, if for some reason one cannot find a LEL symbol for naming a class, it means that either any LEL symbol has an alias that was not yet considered, or the symbol is missing in LEL definition and, therefore,

should be added to it. If that is the case, one may go over the nonfunctional perspective again, i.e., evaluate the symbol for possible NFRs, represented in the graphs. Using this anchor, the integration process is centered on searching for a symbol that appears in both models, and evaluating the impacts of adding the NFRs operationalizations to the class diagram. They start the process by picking out a class from the class diagram. There is no order for choosing one class or another. They search all the NFR graphs looking for any occurrence of this symbol. For each graph where the name of the class are searching for appears, have to identify the dynamic and static operationalizations from this graph.

For dynamic operationalizations found, check if the operations that belong to this class already fulfill the needs expressed in the graph's operationalizations. On the other hand, for static operationalizations, check if the class attributes already fulfill the needs expressed in the graph's operationalizations. If they do not, then they are adding operations and attributes to the class. Note that, adding new operations may sometimes call for the inclusion of new attributes in order to implement the desired operation or vice-versa.

2.1.5.2.3 Application of NFR Framework on Sequence Diagrams and Collaboration Diagrams

Cysneiros et al [L.Cysneiros & Leite, 2004] described a strategy for integrating NFRs into the sequence diagram by examined every class of the class diagram. For every operation included because of NFR satisfaction, they searched the sequence and collaboration diagrams where this class appears. For each diagram they find, they must check if the new operations added due to NFR satisfaction will imply any change in this sequence diagram. It may be necessary to add classes, messages, or both to the diagram. If there is any pre or post condition attached to an operation, they may also need to specify it attached as a note to a message. And must be able to represent that new message together with pre and post conditions in the sequence diagrams were added due to NFR satisficing. This is done by using a note linked to the message where the condition will apply. This note will contain the expression that portrays the pre or post condition. Any message included in these diagrams due to NFR satisfaction will have the same traceability expression they used with attributes and operations. The integration on communication diagrams were done in a similar way.

Moreira et al [A.Moreira & Brito, 2002] described the integration of NFRs in UML sequence diagram. When they found the result from the use case diagram after integration of NFRs then they further precede. They used stereotypes in the use case diagram for integration. Then they used those stereotypes in the sequence diagram. They showed through the arrows and with the grey rectangles in the lifeline of sequence diagram. They identified the points wherein the constraint applies with the units of time.

Dimitrov et al [E. Dimitrov & Schmietendorf, 2002] used sequence diagram as special interaction diagrams offer sufficient potential to obtain and present performance information.

They represent time relations by introducing an explicit time axis (time progress from top to bottom). The vertical layout of messages in the diagram helps define the messages chronological sequence. For the interaction diagram, they proposed to add some additional time information by labeling the messages with relative constraints such as assigning time attributes to the messages and to the method executions. According to this approach the labeling of messages with time will be interpreted as latency and labeling of methods with time will be interpreted as time for method execution.

As I conclude all the work done by different researchers like Moreira worked on sequence diagram was done on the basis of use case diagram means result of integration of use case diagram is used in sequence diagram. Those use cases which are added after integration are included in sequence diagram through mentioned boxes in lifelines and units of time are also showed in sequence diagram. Cysneiros worked on sequence diagram is done through after the integration of NFRs in Class diagram. They first check that any NFRs inclusion changes the impact of class diagram then these changes also manipulate in sequence diagram like addition of any new class, any operation or any attribute. Dirnitrov just added some additional time information by labeling the messages with relative constraint.

2.1.5.2.4 Application of NFR Framework on Activity Diagram

Atif et al [M. Usman, Atif, Rizwan & Shahzad] described the integration of NFR into activity diagram. This integration requires the identification of each activity from use cases and domain glossary. The activities identified in use cases must exist in domain glossary. If there is any activity in use cases but not present in domain glossary then domain glossary must be updated. The activities are then represented in activity diagrams. They assumed that activity diagrams are drawn for important activities of Universe of Discourse. Usually activity diagrams are drawn from use cases. They then explored domain glossary to identify NFRs for each activity. From NFR graph, they used satisficing sub goals at leaf nodes to operationalize the NFR in activity diagram. The operationalization of NFR requires looking for relevant actions in activity diagram and then mapping satisficing sub goals of NFR graph on these actions. The operationalization of NFR through activity diagrams gives result in new actions or as constraints on existing actions. The mapping either is an action or constraint. New action is proceeded or follows existing action and is being referred to as ***pre*** and ***post*** actions. The constraints are appended in existing actions and are referred to as in actions. The ***pre*** actions mean that these must be performed before the execution of actual action; ***post*** *actions* are performed after the execution of actual action and in actions are treated as a condition or constraint on actual action.

2.1.6 Comparison among Different Approaches

Concept/Research Work	Approach	**Paper**
NFR Framework	Based on NFRs	[L. Chung & Yu ,2000]
NFR in Conceptual model through NFR graphs and LEL Symbols.	Based on NFRs	[L.Cysneiros & Leite, 2001] & [L.Cysneiros & Leite , 2004]
Model to identify and specify quality attributes at an early stage	Based on NFRs	[Ana Moreira & Brito, 2002]
A framework for Performance engineering	Based on NFRs(UML based approach for performance modeling)	[E. Dimitrov & Schrnietendorf ,2002]
A framework for integrating NFRs with FRs in the use case model.	Based on NFRs (A use case and Goal driven approach)	[S. Supakkul & Chung, 2004]
Homogeneous UML use-case model	Based on NFRs	[Brian, 2004]
Extending UML with UML profile	Based on NFRs	[S. Supakkul & Chung, 2005]
NFRs in Software Architecture	Based on NFRs	[L .Xu & Ziv ,20051
UML profile for modeling design decision and for modeling NFRs in a ggeneric way.	Based on NFRs	[L. Zhu & I. Gorton, 2007]

Table 2.1: Comparison among different approaches

Table 2.1 shows the research work which is based on NFRs and shows the different approaches to handle the NFR of the system. The research works are presented in chronological order. The most comprehensive work on NFRs is based on NFR framework which was proposed by Chung [L. Chung & Yu, 2000]. The Chung NFR framework is the most comprehensive for the representation and dealing for NFRs. The template Moreira proposed to specify a quality attributes was influenced by the approaches of Chung et al.

Then Cysneiros also used Chung et al NFR framework [L. Chung & Yu, 2000] and apply it into use case view, behavioral view and structural view as mentioned in table 2.2. Different researchers proposed different frameworks mentioned in table 2.1 and some of the researcher applied frameworks on UML artifacts shown in table 2.2. Table2.2 shows the frameworks which are applied on UML artifacts. Table 2.2 shows these works according to five views. Some views are covered but some views are not covered yet like implementation view and inclusion of some new diagrams in behavioral view and structural view in version UML2.

se Case ew	Behavioral view						Structural view		Implement ation view		Environm ent view	Papers
se case agram	SD	CoD	IOD	State	Activity	Timing	Class	CSD	CD	CSD	Deployme nt diagram	
✓	✓											[A. Moreira Brito, 2002]
✓	✓			✓	✓						✓	[E. Dimitrov & Schmieter dorf 2002]
✓												[S. Supakkul & Chung 2004]
✓	✓	✓					✓					[L. Cysneiros & Leite 2004]

Table 2.2: Views are used to specify NFRs

According to the 4+1 architectural views model was proposed by Krutchen, the views are structural view, behavioral view, implementation view, environmental view and one is use case view. Software systems can be modeled from these views. As shown in table 2.2, implementation view is not covered in any researcher in any paper. The CSD and CD are included in implementation view. As these diagrams have a lack of information about non functional aspects of the software system, especially NFRs related to different components and their implementation. In literature I find that there is need to improve the understanding of NFRs at component and implementation level and also to complete the understanding of NFRs in these diagrams to make these diagrams complete and also to improve the quality of software system at run time in implementation view.

As shown in table 2.2, the behavioral view covers work on sequence diagram and collaboration diagram. But some diagrams are left for the integration of NFRs. IOD is diagram is the combination of all interaction diagrams like sequence diagram and communication diagram and timing diagram. I want to find that, is there any impact in diagram when we combine interaction diagrams in an IOD, any addition of instances, classes, association or any NFR which may change the IOD. The literature of software engineering does not provide any mechanism to incorporate NFRs in these types of models. My research focus is to find out a mechanism to incorporate NFRs in these views, so that it will help us to understand that particular view (Implementation, Behavioral) of system.

Chapter 3

Stratew for Integration of NFRs

3. Strategy for the Integration of NFRs

Developing quality software always requires the proper elicitation of NFRs with the FRs. If the NFRs are not reflected properly in the design phase, the quality of the software will suffer. Researchers consider the design of software as the basic foundation of building a high quality product.

Software models like requirements, conceptual, and design models are realizations of FRs. Few attempts are made to model NFRs in UML artifacts like [A. Moreira & Brito, 2002], [E. Dimitrov & Schmietendorf, 2002], [S. Supakkul & Chung, 2004] & [L. Cysneiros & Leite , 2004]. Research in RE has shown recently that besides modeling FRs, there is need to incorporate NFRs in functional models as well [L.Cysneiros & Leite, 2001] & [Dardenne & Van , 1993]. "Modeling NFRs allows them to be organized for better visualization and understanding" [Cysneiro's & Yu, 2003]. It will help software engineers to analyze NFRs. This integration will improve the understanding and quality of functional models.

Chung worked on NFRs [J. Mylopoulos & Chung, 1992] & [L. Chung & Yu, 2000], the most comprehensive framework for the integration of NFRs and dealing for NFRs [J. Mylopoulos & Chung, 1992]. Cysneiros used the Chung [L. Chung & Yu, 2000] worked on NFR and applied it into UML1 diagrams, the worked on the integration of NFR in UML artifacts were done in a very comprehensive way so I am further extending on UML2 diagrams with using NFR framework and Graphs. As mentioned in table 2.1 different research works, some focused on particular NFR and some are system dependent. Cysneiros used NFR framework and applied it into use case view, behavioral view and structural view. But some views are not covered yet like implementation view and inclusion of some new diagrams in behavioral

view and structural view in UML2. My work is focus on implementation view, behavioral view and structural view. I am proposing the strategy for the integration of NFRs in IOD, CSD and CD which are not dependent for a particular system and not for a particular NFR. My proposing strategy deal NFRs with respect to the system. The different steps for the integration of NFRs are as follows:

1. Eliciting NFRs
2. Building Domain Glossary
3. Representation of NFRs
    a. NFR Framework
    b. Refining NFRs using NFR graph
4. Creating NFR Graphs
5. Integrating NFRs in Use case diagram
6. Integrating NFRs in Class Diagram
7. Integrating NFRs in Sequence and Communication Diagram
8. Integrating NFRs in IOD
9. Integrating **NFRs** in CSD
10. Integrating NFRs in CD.

The step 1 and 2 are our contribution whereas step 3 and 4 are included from Chung [L. Chung & Yu, 2000] & [J. Mylopoulos & Chung, 1992] NFR framework. The steps 5 to 7 are included from Cysneiros [L. Cysneiros & Leite, 2004] integration strategies for NFRs. The steps 8 to 10 are ow contribution for the integration of NFRs in UML2.

3.1 Eliciting NFRs

The elicitation of NFRs, Cysneiros [L. Cysneiros & Leite, 2004] elicited NFRs through LEL symbols. LEL registered the vocabulary of universe of Discourse. They build LEL for the functional and nonfunctional perspective. First build LEL for the functional perspective then nonfunctional perspective is added. I assume that requirements have already been elicited and specified in requirements document like Software Requirement Specification (SRS), use cases etc. These requirements documents will be used to search NFRs. There is no hard and fast rule to identify NFRs. I have used a simple principle to identify NFRs from requirement documents. Typically verbs are treated as functional and adverbs are treated as NFRs. For example an FR might be stated as "customer want to complete sales processing". On the other hand NFR might be phrased as "Customer want to complete sales processing very quickly". Some other NFRs for POST are: The system shall be highly available since the effectiveness of sales depends on its availability. The system shall be portable to a range of different platforms to support aproduct line of POST systems. The system shall be usable by clerks with a minimum of training and with a high degree of efficiency.

3.2 Building Domain Glossary

The concept of domain glossary is to record tasks and their corresponding NFRs. Tasks are added in glossary by analyzing the requirements documents. Tasks are functional in nature e.g. items, payment etc. NFRs are constraint on these tasks e.g. Payment authorization service that they will make or guarantee the payment to the seller, and the management of

items. Each task is represented in domain glossary by its name, description and it's associated NFRs.

Domain glossary is based on a vocabulary system composed of tasks where each task is expressed through its description and associated NFRs. The description explains the meaning of the task and its relation with other task. The associated NFRs specify the non functional aspects of the mentioned task.

3.3 Representation of NFRs

I am using Chung [L. Chung & Yu, 2000] NFR framework to represent NFRs. The NFR Framework for representing and using nonfunctional requirements during the development process.

3.3.1 The NFR Framework

Chung [J. Mylopoulos & Chung, 1992] proposed a comprehensive framework for the representation and dealing NFRs during the development process. Chung framework based on five basic components which provide for the representation of NFRs in terms of interrelated goals. Such goals can be refined through refinement methods and can be evaluated in order to determine the degree to which a set of NFRs is supported by a particular design.

The NFR framework view NFRs as goals that might conflict among each other and must be represented as soft goals to be satisficed. Each goal decomposed into sub-goals represented by a graph structure inspired by the And / Or trees used in problem solving. The

decomposition is done using contribution links. Contribution links can be categorized as an or contribution or an ***and*** contribution. Contribution links allow one to decompose NFRs to the point that one can say that the operationalizations to this NFR have been reached (i.e., the goals are no longer "soft"). Operationalizations can be viewed as FRs which has arisen from the need to meet NFRs. These operationalizations may add some new methods, attributes, entities and constraints in functional models. The NFR framework also used correlation links to show contributions (positive and negative) from one NFR to another.

3.3.2 Refining NFRs using NFR graph

The NFR framework [J. Mylopoulos & Chung, 1992] was extended to represent the operationalizations in two different ways [Cysneiro's & Yu, 2003]. They are called dynamic and static operationalizations. Dynamic operationalizations are those that call for some action to be carried out. Static operationalizations express the need for some data to be used in design of the software to store information which is necessary for satisficing the NFR [Cysneiro's & Yu, 2003].

3.3.3 Creating NFR Graphs

To build the NFR model, must go through every task looking for descriptions that express the need for an NFR. For each NFR found, one must create an NFR graph where this NFR will be the root of the graph. This graph must be further decomposed into sub goals and then express all the operationalizations that are necessary to satisficed this NFR. This can be

accomplished either using the knowledge base on NFRs or investigating what descriptions and associated NFRs are added to domain glossary to satisficed NFRs.

If during decomposition any new term is identified, it is added in domain glossary. NFRs are treated as goals, which can be decomposed in satisficing sub goals. The terms used for these goals are task in the domain glossary. Sub goals can further be decomposed in other satisficing goals. **A** goal will only be satisficed when all of its sub goals are satisficed. Satisficing NFRs may result in additional functionality. The leaf nodes of NFR graph will be used to operationalize the NFR. This Operationalization may add some new attributes, methods, entities and constraints in functional model.

3.4 Integrating NFRs

Cysneiros [L. Cysneiro's & Leite, 2004] strategies for the integration of NFRs in use case, sequence diagram, collaboration diagram and class diagram. I am using these strategies. I am explaining these strategies because I am applying these strategies on a case study. Cysneiros explained these strategies with respect to LEL which is build for the nonfunctional perspective. I am building domain glossary for the nonfunctional perspective so I am applying these strategies according to domain glossary instead of LEL symbols. I want to cover all views in one particular system so I am using these strategies and applying it on a case study POST. Then also my contribution is on strategy for integrating NFRs in IOD, CSD and CD. The consideration of NFRs in all views in one particular system is considering in this research work.

3.4.1 Integrating NFRs in Use Cases

Cysneiros [L. Cysneiro's & Leite, 2004] explained the strategy for the integration of NFRs in use case in this way.

The integration of NFR in use cases requires the identification of each use case from the domain glossary. They identified if any domain glossary task appears in the diagram. They also identified if any domain glossary **task** appears in **any** of the use cases or actors of this diagram. For each domain glossary **task** they find, they search the set of NFR graphs to identify those where this task appears. For example, the use case of process sale has security NFR.

From NFR graph, they used satisficing sub goals at leaf nodes to operationalize the NFR in use case. The operationalizations of NFR requires looking for relevant use case in use case diagram and then mapping satisficing sub goals of NFR graph on these actions. The operationalizations of NFR through use case diagram will result in new actions or as constraints on existing use case. The mapping will either be an action or constraint. Figure 3.3 [L. Cysneiro's & Leite, 2004] shows the integration process for use case or scenario.

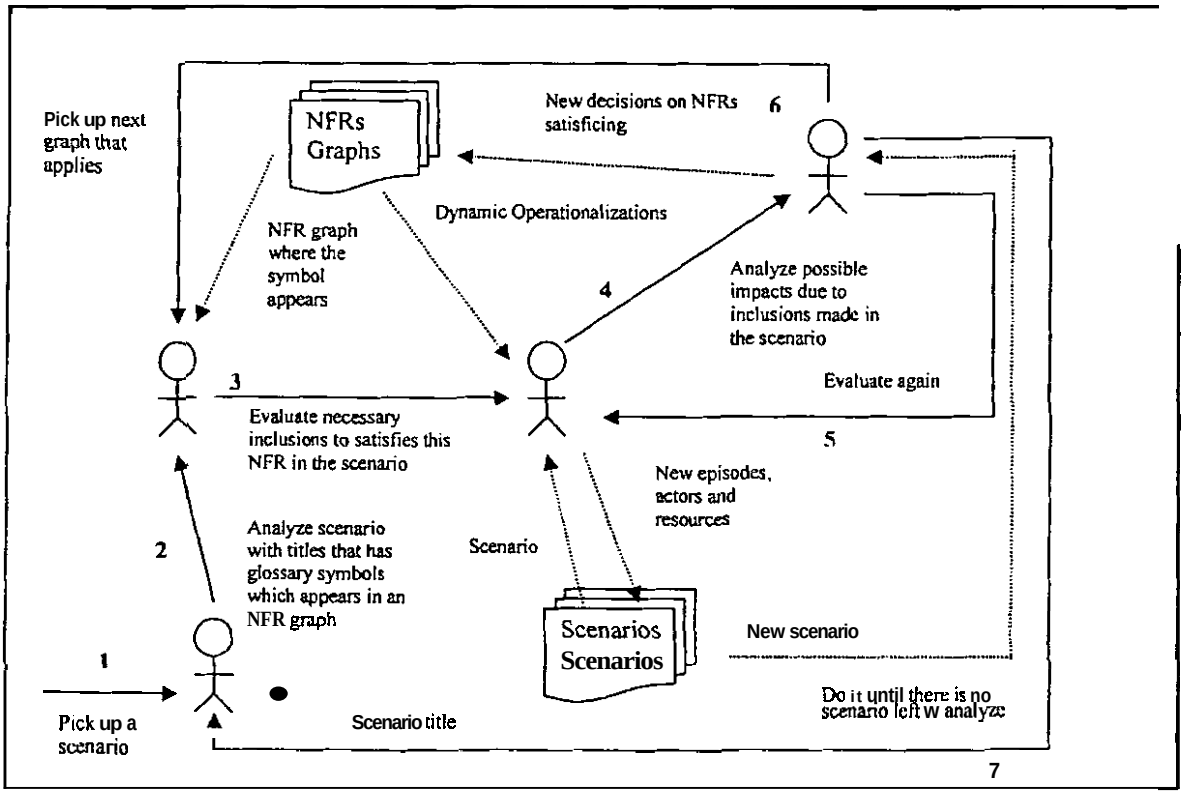


Figure **3.1:** The use case or scenario integration process [L. Cysneiro's & Leite, 2004]

3.4.2 Integrating NFRs in Class Diagram

Cysneiros [L. Cysneiro's & Leite, 2004] explained the strategy for the integration of NFRs in class diagram in this way.

Integration of the non functional perspective into the class model will be based on the use of domain glossary. Here, it means that every class belonging to the class diagram has to be named using a name domain glossary. The use of domain glossary as an anchor to construct both perspectives to facilitates their integration. It can also be used for validating both models since, if for some reason one cannot find in domain glossary for naming a class, it means that either that was not yet considered, or is missing in domain glossary and therefore should be added to it. The integration process is starting on searching for a task that appears in both models, and evaluating the impacts of adding the NFRs operationalizations to the class diagram. Figure 3.4 [L. Cysneiro's & Leite, 2004] shows the integration method for the class diagram. They start the integration by picking out class or another. They search all the NFRs graphs looking for any occurrence of this task. For each graph where the name of the class they are searching for appears, they have to identify the dynamic and static operationalizations from this graph. For dynamic operationalizations found, they have to check if the operations that belong to this class already fulfill the needs expressed in the graph's operationalizations. On the other hand, for static operationalizations they have to check if the class attributes already fulfill the needs expressed in the graphs operationalizations. If they do not, then they have to add operations and attributes to the class.

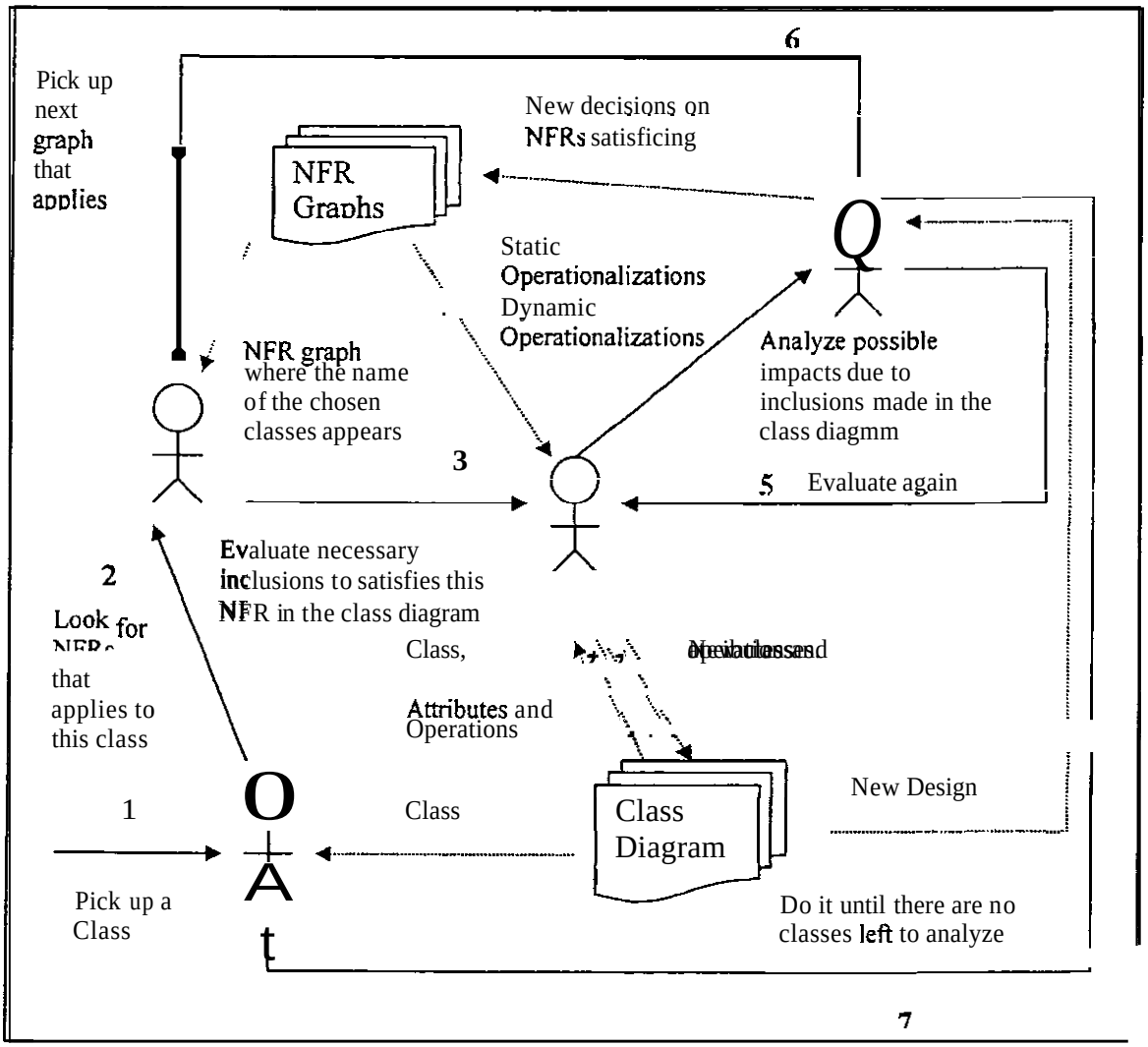


Figure 3.2: The class diagram integration process [L. Cysneiro's & Leite, 2004]

tr

3.4.3 Integrating NFRs in Sequence and Communication Diagrams

Cysneiros [L. Cysneiro's & Leite, 2004] explained the strategy for the integration of NFRs in sequence and communication diagrams in this way.

Integrating NFRs into the sequence and collaboration diagrams is done by examining every class of the class diagram. For every operation included because of NFR satisfaction, they may search the sequence and collaboration diagrams where this class appears. For each diagram they found, they must check if the new operations added due to NFR satisfaction will imply any change in this sequence or collaboration diagram.

It may be necessary to add classes, messages, or both to the diagram. If there is any pre or post condition attached to an operation, they may also need to specify it attached as a note to a message.

They must be able to represent that new messages together with pre and post condition in the sequence and communication diagrams were added due to NFR satisficing. This is done by using a note linked to the message where the conditions will apply. This note will contain the expression that portrays the pre or post condition, any message included in these diagrams due to NFR satisfaction will have the same traceability expression they used with attributes and operations. Figure 3.5 [L. Cysneiro's & Leite, 2004] shows the integration method for the sequence and communication diagram.

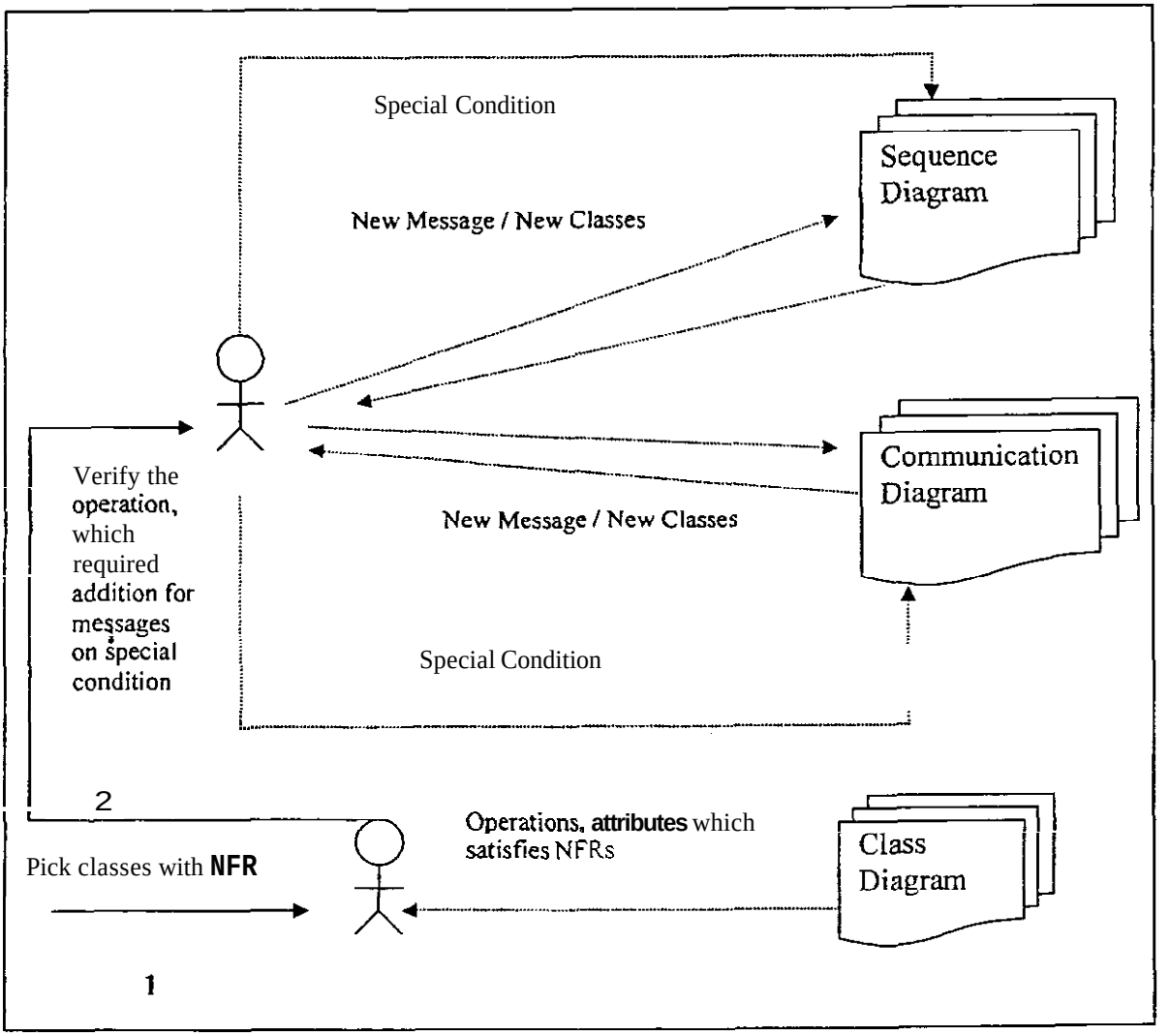


Figure 3.3: The sequence and communication integration process

[L. Cysneiro's & Leite, 2004]

3.4.4 Integrating NFRs into IOD

The IOD focuses on the overview of the flow of control of the interactions. An IOD is a form of activity diagram in which nodes represents interaction diagrams. Interaction diagrams can include sequences, communication, interaction overview and timing diagrams and IOD is a combination of sequence, communication and timing diagram.

The purpose of integration of NFRs in IOD is, sequence diagram are produced against every individual event in a use case and a use case have more than one sequence diagrams. Although NFRs are modeled on use cases [L.Cysneiros & Leite, 2004] as well but it is difficult to map those NFRs to each individual sequence diagram against that use case. There is a need to have a single model realizing an individual use case in terms of sequence diagram. IOD represents an individual use case in terms of sequence diagram. The integration of NFRs fulfills those needs. Figure 3.6 shows the integration process for the IOD.

The integration process for the IOD is as follows:

1. Pick up an interaction,
2. Analyzes it and look for NFRs that applies to this interaction,
3. Evaluate necessary addition to satisficed this NFR in IOD,
4. Analyze all possible impacts due to addition made in IOD and then,
5. Evaluate this process again for the verification for any addition,
6. Analyze all the interaction in this ways when there is no interaction left to analyze.

In this way some new interactions or new messages/classes or note with pre or post condition are added in IOD. When I apply NFRs on IOD, the addition of new interactions or new messages/classes or note show changes in the model. These addition or changes makes

the model perfect and complete and enhance its understanding and also improves the quality of the system.

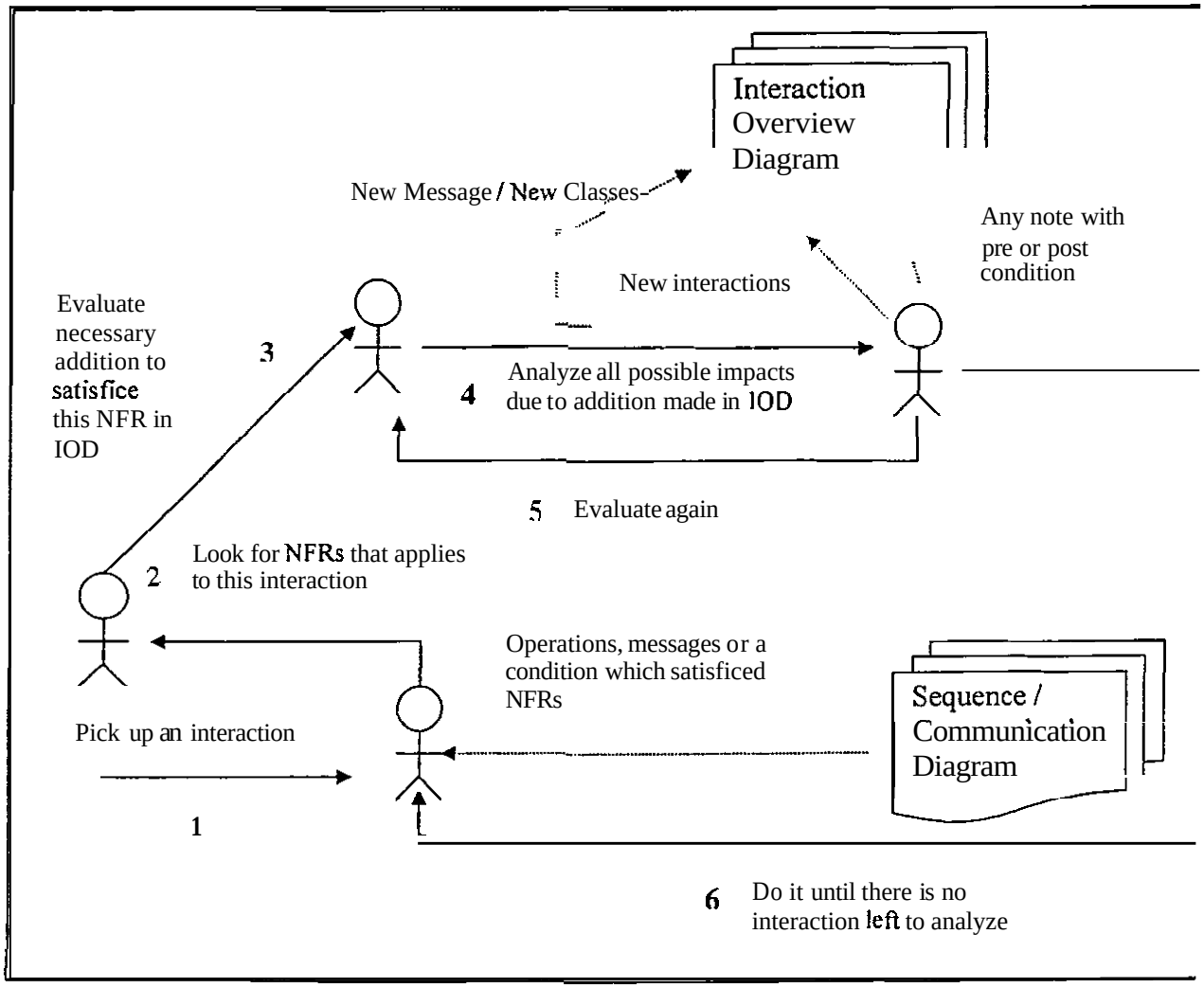


Figure **3.4:** The IOD integration process

3.4.5 Integrating NFRs into CSD

The CSD introduced in UML2 which has a lack of information about non functional aspects of that system, especially NFRs related to different composite classes and their implementation. The CSD deals with the internal structure of a classifier, including its interaction points to other parts of the system. It shows the configuration and relationship of parts that together, perform the behavior of the containing classifier.

The integration of NFRs in CSD covers the implementation and structural view of software system. Figure 3.7 shows the integration process for CSD.

The integration process for the CSD is as follows:

1. Pick up a composite class,
2. Analyzes it and look for NFRs that applies to this composite class,
3. Evaluate necessary addition to satisficed this NFR in CSD,
4. Analyze all possible impacts due to addition made in CSD,
5. Evaluate this process again for the verification of any addition,
6. Analyze all the composite classes in this ways when there is no composite class left to analyze.

In this way, addition of some new composite class or new parts in composite class or any association between parts and composite classes are added in CSD. When I apply NFRs on CSD, the addition of new composite class or new part or any new association show changes in the model. These changes make the model complete, clear the understanding of system at implementation and structural level and improves the quality of the system.

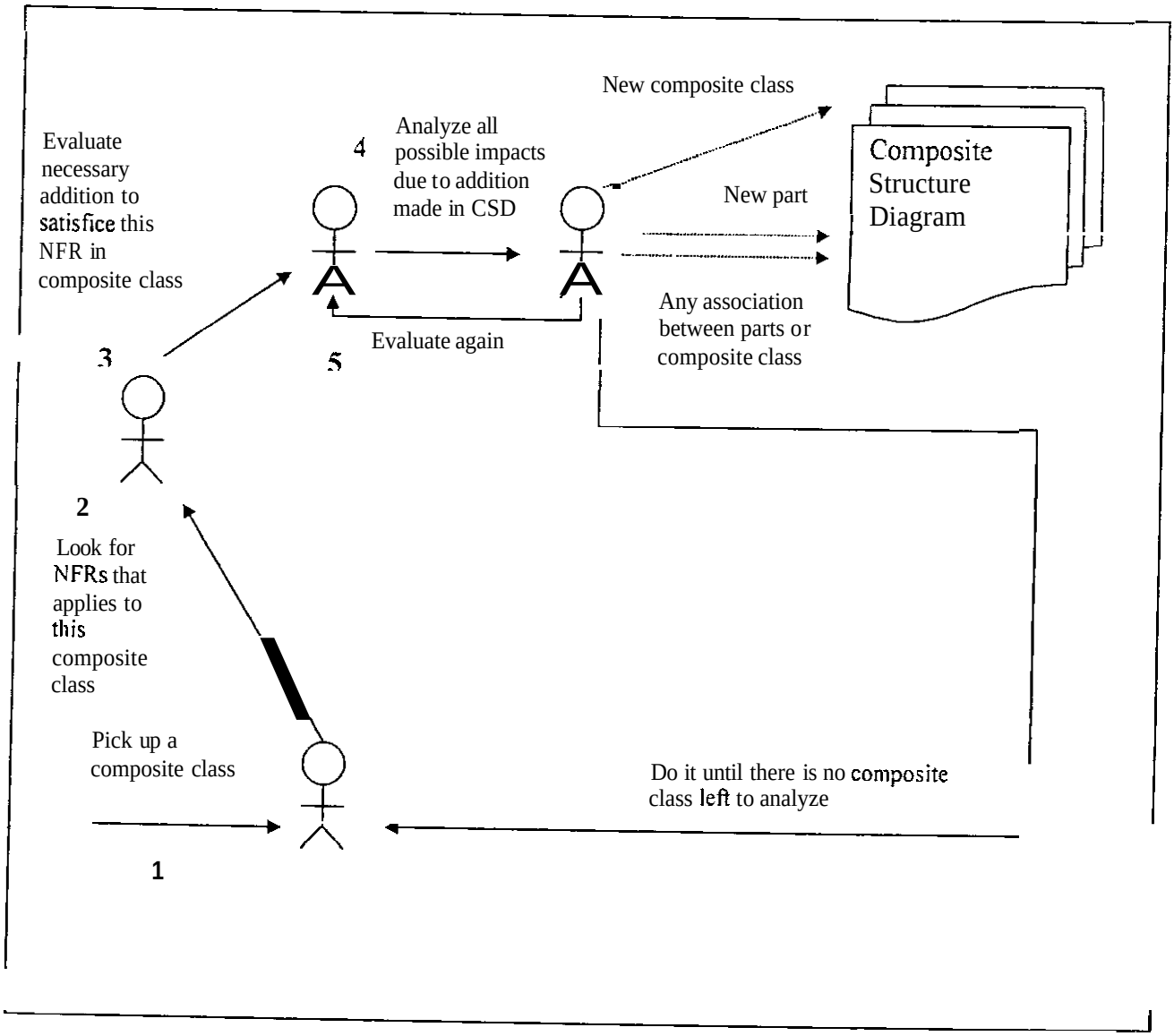


Figure 3.5: The CSD integration process

3.4.6 Integrating NFRs into CD

The CD main purpose is to show the structural relationships between the components of a system. In UML1.1, a component represented implementation items, such as files and executables. Unfortunately, this conflicted with the more common use of the term component, "which refers to things such as COM components. Over time and across successive releases of UML, the original UML meaning of components was mostly lost. UML2 officially changes the essential meaning of the component concept; in UML2, components are considered autonomous, encapsulated units within a system or subsystem that provide one or more interfaces. Although the UML2 specification does not strictly state it, components are larger design units that represent things that will typically be implemented using replaceable modules. But unlike UML1, components are now strictly logical, design time constructs. The idea is that you can easily reuse and or substitute a different component implementation in your designs because a component encapsulates behavior and implements specified interfaces.

The integration of NFRs in CD covers the implementation view of software system. The CD has lack of information about non functional aspects of that system, especially NFRs related to different components and their implementation. Figure 3.8 shows the integration process for component diagram.

The integration process for the CD is as follows:

1. Pick up a component,
2. Analyzes it and look for NFRs that applies to this component,
3. Evaluate necessary addition to satisficed this NFR in CD,
4. Analyze all possible impacts due to addition made in CD,

5. Evaluate this process again for the verification of any addition,

6. Analyze all the components in this ways when there is no component left to analyze.

Some NFRs are not operationalized at component level then I attach these NFRs in a form of a special note with condition, mention the condition for which it will be use. But we have to consider the association between components. Sometimes some components required or provide interface then we have to inspect these interfaces and model the related NFRs on these components by attaching some notes to that association or relationship which show the creation of adding those notes. In this way, addition of some notes between components is added in CD. When I apply NFRs on CD, the addition of new notes shows changes in the model. These changes make the model complete, enhances the understanding of system at component level and improves the quality of the system at implementation and run time.

Chapter 4

Case Study(POST)

4. POST (Case Study)

4.1 Introduction

This case study is about a POST system. A POST is a computerized system used to record sales and handle payments; it is typically used in a retail store. It includes hardware components such us a computer and bar code scanner, and software to run the system. It interfaces to various service applications, such as third-party credit and check authorization systems, tax calculator and inventory control. The main objectives of the system are to deal with sale, returned items and handle payments of those returned items. These systems must he relatively fault-tolerant; that is, even if remote services are temporarily unavailable (such as the inventory system), they must still be capable of capturing sales and handling at least cash payment.

POST **Functions**

I am discussing two main use cases of POST because these use cases are the main use cases for any sale system. First sale is required to run for any sale system and then returned of those sale items.

1. Sales

(a) Sale items

(b) Return purchased items

4.1.1 Use Case descriptions

4.1.1.1Use case: Sale items

Actors: Customer, Cashier, Accounts Receivable/To Pay System

Description: A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects payment. On completion, the Customer leaves with the items.

Steps o r transactions

1. This Use Case begins when a Customer arrives at a POST checkout with items to purchase.

2. The Cashier records the identifier of each item.

3. The System determines the item price and description and adds information to the current sales transaction.

4. On completion of item entry, the Cashier indicates to the POST that item entry is complete.

5. The System calculates and presents the sale total.

6. The Cashier tells the customer the total.

7. The Customer chooses payment type:

(a) If cash payment, see section *Pay* by ***Cash***

(b) If credit payment, see section *Pay* by **Credit**

(c) If check payment, see section *Pay* by Check

8. The System logs the completed sale.

9. The System prints a receipt.

10. The Cashier gives the receipt to the Customer.

11. The Customer leaves with the items purchased.

Alternative

• Line 2. If there is more than one of the same item, the Cashier can enter the quantity as well. The subtotal of these items is shown.

• Line 2. Invalid identifier entered. Indicate error.

• Line 7. Customer didn't have enough money. Cancel sales transaction.

Section: *Pay by Cash*

Steps or transactions

1. The Customer gives cash payment - the "cash tendered" – possibly greater than the sale total.

2. The Cashier records the cash tendered.

3. The System shows the balance due back to Customer.

4. The Cashier deposits the cash received and extracts the balance owning. The Cashier gives the balance owning to the Customer.

Alternative

• Line 4. Insufficient cash in drawer to pay balances. Ask for cash from Supervisor cashier, or ask Customer for a payment closer to sale total.

Section: ***Pay*** by Credit

Steps or transactions

1. The Customer communicates their credit information for the credit payment.

2. The System generates a credit payment request and sends it to an external Credit Authorization Service (CrAS).

3. CrAS authorizes the payment.

4. The System receives a credit approval reply from the CrAS.

5. The System posts (records) the credit payment and approval reply information to the Accounts Receivable/To Pay System. (The CrAS owes money to the Store, hence AIR must track it).

6. The System displays authorization success message.

Alternative

• Line **3.** Credit request denied by CrAS. Suggest a different payment method.

Section: ***Pay*** by ***Check***

Steps o r transactions

1. The Customer writes a check and identifies itself.

2. The Cashier records identification information and request check payment authorization (ChAS).

3. The System generates a check payment request and sends it to an external ChAS.

4. The ChAS authorizes the payment.

5. The System receives a check approval reply from the CrAS.

6. The System displays authorization success message.

Alternative

• Line 4. Check request denied by ChAS. Suggest different payment method.

4.1.1.2Use case: Return purchased items

Actors: Customer*, Cashier

Cross Reference: F1.b

Description: A Customer arrives at a checkout with purchased items to return. The Cashier records the returned items. On completion, the Cashier refunds the Customer the total money.

Steps o r transactions

1. This Use Case begins when a Customer arrives at a POST checkout with items to return.

2. The Cashier records the identifier of each item.

3. The System determines the item price and description and adds information to the current return transaction.

4. On completion of item entry, the Cashier indicates to the POST that item entry is complete.

5. The System calculates and presents the refund total.

6. The Cashier gives the customer the money back.

7. The System logs the completed return.

Alternative

• Line 2. If there is more than one of the same item, the Cashier can enter the quantity as well. The subtotal of these items is shown.

• Line 2. Invalid identifier entered. Indicate error.

• Line *6*. Insufficient cash in drawer to pay total. Ask for cash from Supervisor Cashier.

4.2 Integrating NFRs in POST

POST case study is used for validation of NFRs integration process. I am integrating NFRs on use case diagram, class diagram, sequence and communication diagram, IOD, CSD and CD. For the nonfunctional perspective as I mentioned in chapter **3,** I am building a domain glossary to record the NFRS against each functional requirement so domain glossary for POST is shown in table 4.1.

Task	Description	NFRs
Process Sale	A Customer arrives at a checkout with items to purchase. The Cashier records the purchase items and collects payment.	Security of payments either in credit or through checks. Performance involves in payment authorization. Maintain the records of sale items.
Handle Returns	A Customer arrives at a checkout with purchased items to return. The Cashier records the returned items.	Accuracy and Consistency of the item which is to return.
Process Rental	A cashier gives the item on rent.	Security of Customer which takes item on rent.
Manage Users	System administrator has to manage the details of users of the system.	
Manage Security	System administrator has to manage the security of all tasks	

Register	Register class deals the new sale; enter items, payments and end of sale.	Security involves for making payments. Maintain record of the items. Maintain and Update the records of the return sale.
Payment	Payment class deals with the payments.	Security involves for handling payments.
Sale	Sale class deal with sale and make the quantity of sale, get total and make payment.	Security involves for making payments. Maintain the records of sale
Product Catalog	Product Catalog gives the description of the product.	Catalog provides useable description of the product.

Table 4.1: Domain glossary

4.2.1 Integration of NFRs in POST Use Case

Figure 4.1 shows the use case diagram of POST without integration of NFRs.

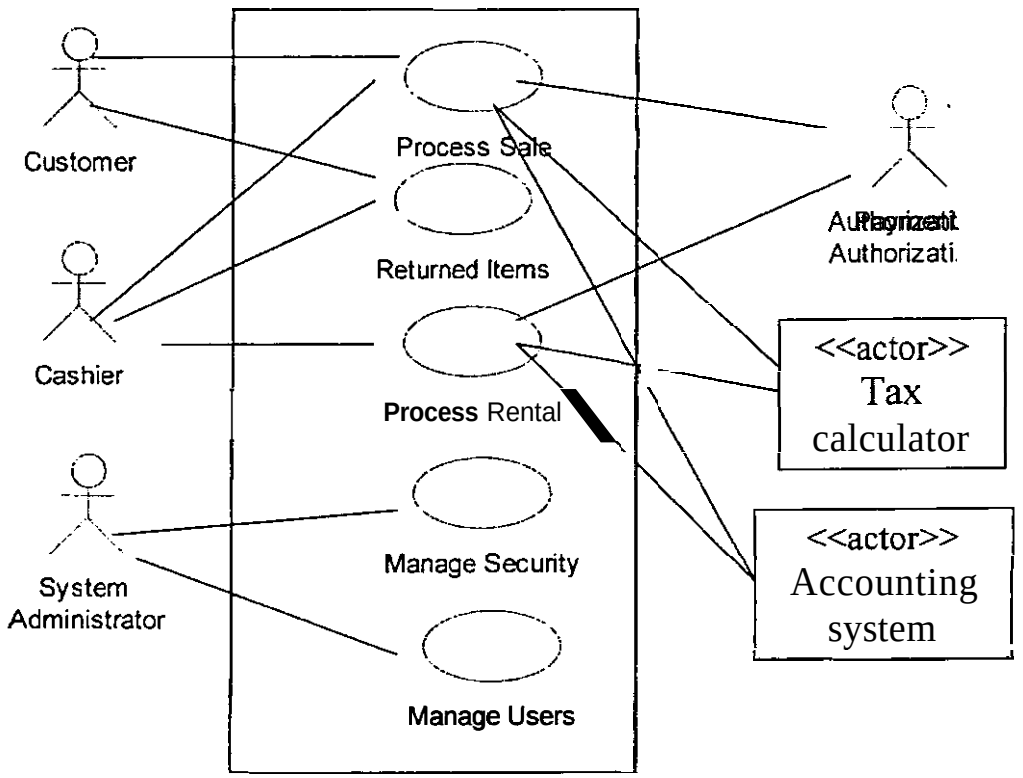


Figure 4.1 : Use case diagram for POST before integration

In the figure **4.2** shows the decomposition of NFR Security. In figure 4.2 Security [Payment] refers to NFR security regarding the process sale task. I found that in order to achieve security of payment either in check or credit. These are represented as sub goals as Security [Payment.Check] and Security [Payment.Credit]. In figure **4.2** these sub goals are further decomposed in two sub goals of Security [Payment.Check] and three sub goals of Security [Payment.Credit]. These sub goals are represented in figure **4.2** as Security [Payment.Check.RecordIdenti.Info], Security [Payment.Check.Authorize] and Security

[Payment.Credit.Communicate Credit Info], Security [Payment.Credit.Approve Credit Payment], Security [Paymnet.Credit.Request Credit Payment].

The Security [Payment.Check.Authorize.GenerateCheckPayment] and Security [Payment.Check.Authorize.ApproveCheck Payment].

By further refining these goals we come to know that goal

- Security [Payment.Check.Authorize.GenerateCheck Payment] is decomposed in satisficing sub goals Security [Payment.Check.Authorize.GenerateCheck Payment.SecureCommunication] and Security [Payment.Check.Authorize.ApproveCheckPayment.AuthenticateAccess].they are represented as leaf nodes in figure 4.2. Through analysis we found that Security of Payment can be achieved by restricting the access to authenticated operators and secondly by providing secure communication between system and bank.
- Security [Payment.Credit.CommunicateCreditInfo] is decomposed into Security [Payment.Credit.CommunicateCreditInfo.SecureCommunication] and Security [Payment.Credit.RequestCreditPayment] is decomposed into Security [Payment. Credit.RequestCreditPayment.AuthenticateAccess] and . Security [Payment.Credit.ApproveCreditPayment] is decomposed into Security [Payment.Credit.Approve Credit Payment.Authenticate Access].
- Security [Messaging Services] must be secure, but they are not operationalized.
- Security [Database Services] must be secure, but they are not operationalized.

They are represented as leaf nodes in figure 4.2. Through analysis I found that security of payment can be achieved by restricting the access to authenticated operators and secondly by providing secure communication between system and bank.

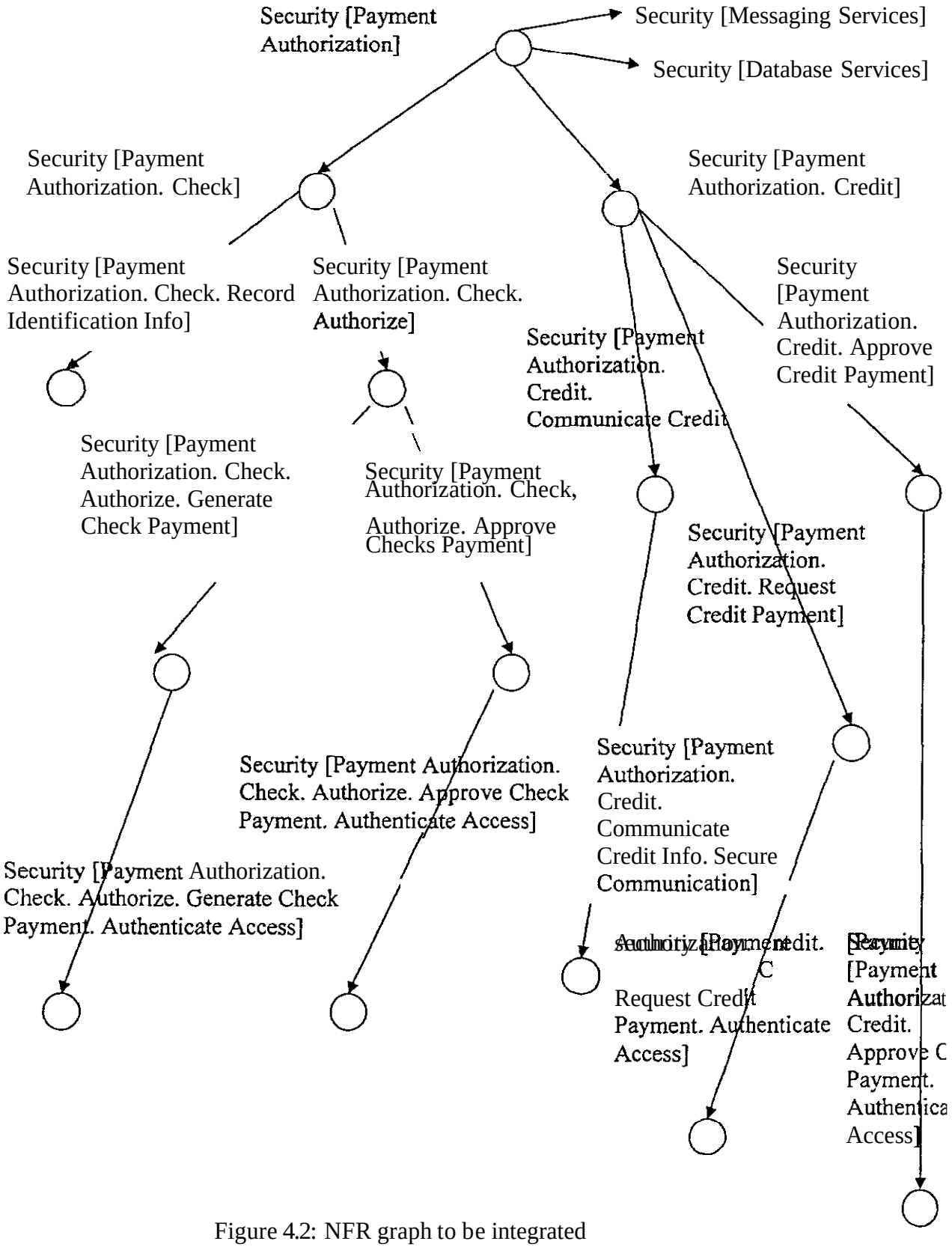


Figure 4.2: NFR graph to be integrated

Figure 4.3 shows the NFR use case diagram from NFR graph figure 4.2, the satisficing sub goals are integrated in use case diagram to operationalize the NFR.

To achieve the sub goal of security, we need to restrict unauthenticated access to system and to maintain access log of operators. Secure communication with bank is also required to protect the Customer's bank account details. So the authenticated access in case of credit and check payment is added in use case diagram and it should verify it through credit or check authorization system. Maintain item and manage returned item are added as a use case which conforms the availability of sale items and then maintain the records of the sale items. The items which are returned to the system they must be managed properly or record properly. This operation will ensure that the information entered/ changed is updated correctly at all places and there are no in consistencies.

I believe that these new actions will enhance the understanding and quality of use case diagram. It will help designers and developers to conform to the FRs and NFRs of users. Implementation of these actions is left on designers and developers. This process has to be carried out for all the use case diagrams that compose the software specification.

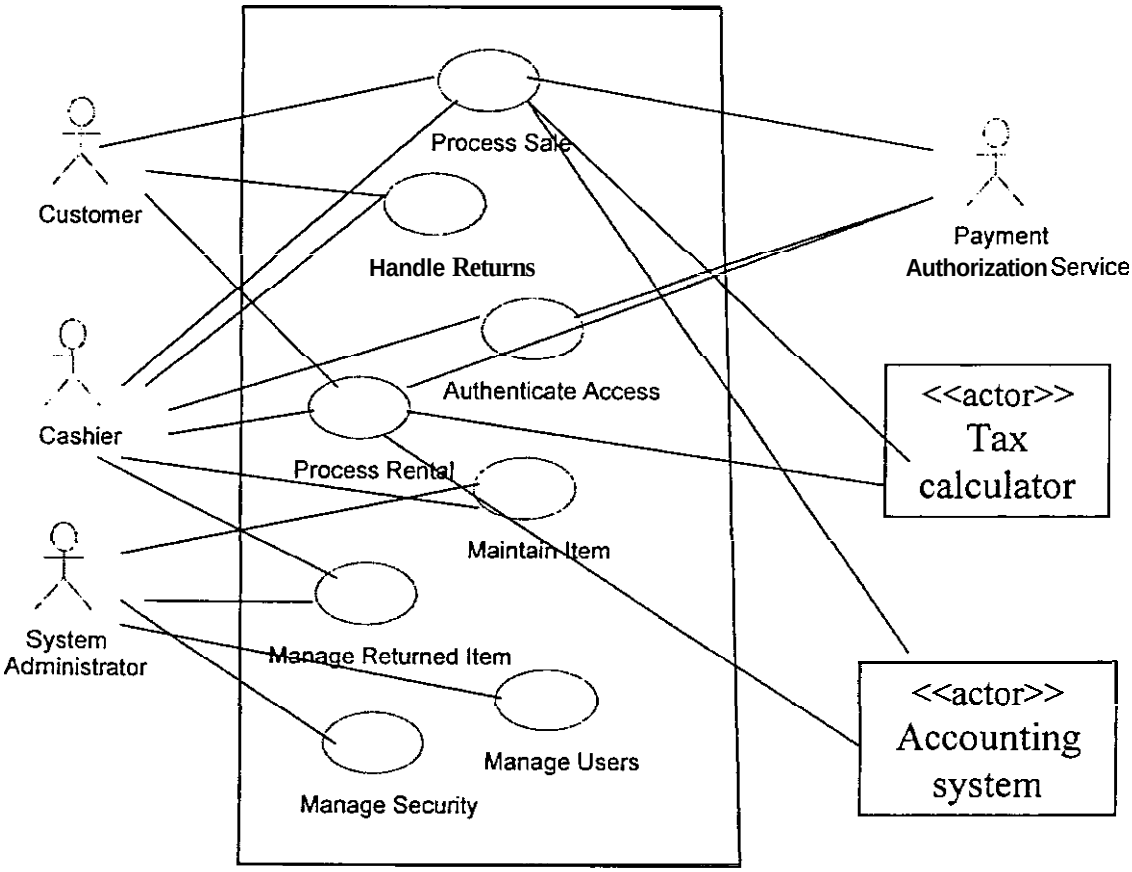


Figure **4.3:** Use case diagram for POST after integration

4.2.2 Integration of NFRs in POST Class Diagram

Integration of NFRs into class diagrams for a POST according to the integration strategy in figure 3.4. Let us take for example a class sale items from POST. I had to search all NFR graphs in the non functional perspective looking for the symbol sale item. One of the NFR graphs found is shown in figure 4.6, where I can see that the operationalizations state that the software must maintain the available records of the sale items. I have to check if any operation in the class sale already performs these actions. If they do not, I should add operations to handle these actions.

Notice that, if the class that I am analyzing is part of a generalization, association, I have to check if any super class or subclass of this class does not have operations, or attributes that satisfy the needed operationalizations.

According to Cysneiro's, integration of NFRs into class diagrams calls for some extensions to be made on how to use UML notation for these diagrams. Cysneiro's explained his paper four heuristics on how to proceed.

1. Classes created to satisfice an NFR may have the name of the class followed by traceability link that points out to the NFR whose operationalizations demanded the class be created. This link will follow the syntax: {NFR [LEL Symbol]}. Since NFRs are often more difficult to be on designer's minds than FRs, having this traceability link avoids classes to be withdrawn from the class diagram during a reviewing process, because one could not find any reason why this class must exist.

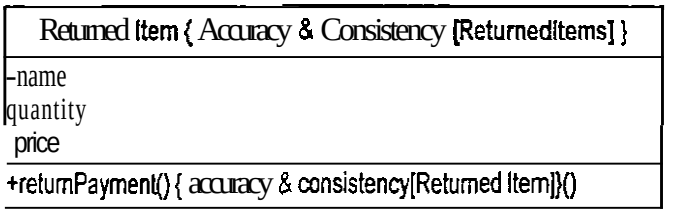


Figure **4.4:** Class Returned Item after integration

Figure **4.4,** shows an example of such a class. This class is created when I am analyzing the class Sale in the class diagram for POST. I searched the set of NFR graphs in the non functional perspective looking for the class Sale for items. The returned item should be maintained so this NFR is added due to maintain the record of return items.

It is important to make it clear that the creation of a new class to satisfice an NFR will always be a design decision. The software engineer could have chosen, in this case, to add the same attributes and operations present in the'class shown in figure **4.4,** to another existing class such as sale class.

2. Adjacent to each operation that has been included to satisfice an NFR; add a link to the nonfunctional perspective. As in heuristic one, this is to enforce traceability between models, so the designer can easily check nonfunctional aspects whenever the changes anything in this class. The link will follow the same pattern as in heuristic one.

Let us take for example the class sale mentioned before. Suppose I add an operation named authenticateAccess() in order to perform one of the operationalizations, I should represent it as follows: authenticateAccess() {Security [Payment]}. These operationalizationsas showed in figure 4.5.

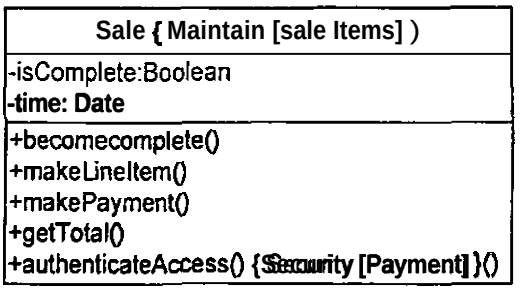


Figure 4.5: Class Sale after integration

3. If an NFR calls for pre or post conditions to apply for an operation, they may add these pre or post conditions to the respective operations.

This heuristic is used for dealing with operational restrictions should be inputted as pre or post conditions to an operation and whenever possible should be stated using Object Constraint Language (OCL). These pre and post conditions can also be stated in a note linked to the class.

4. Adjacent to each attribute that has been added to satisfice an NFR, may use the same expression they used in the operations to establish a link to the nonfunctional perspective.

Figure 4.9 is class diagram after integration, shows an example with the results of applying these heuristics. This figure shows the class diagram of POST after integration process. During the integration process, I analyzed each class I had in the class diagram. I picked out the class from class diagram before integration and analyze it; I searched the nonfunctional perspective looking for any occurrence of this symbol.

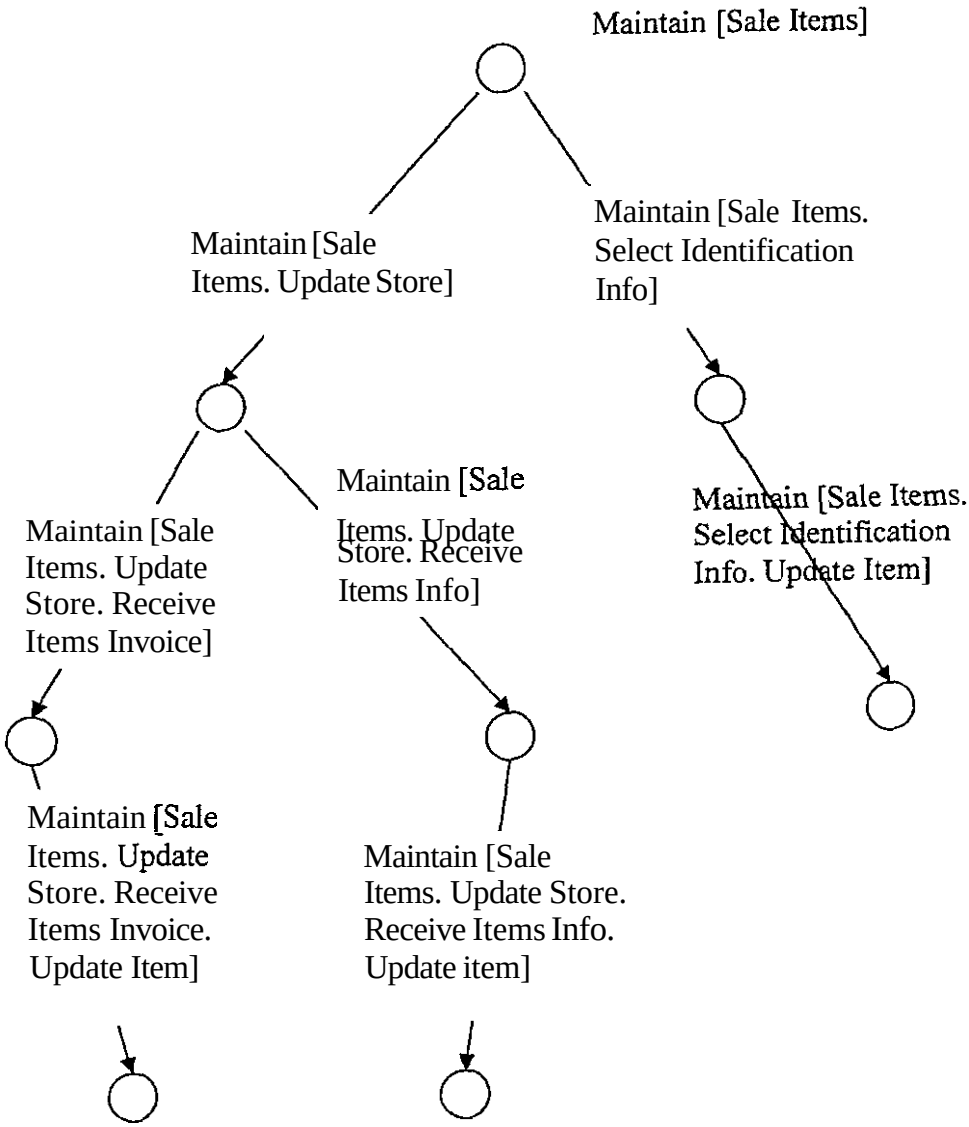


Figure 4.6: NFR graph for sale items

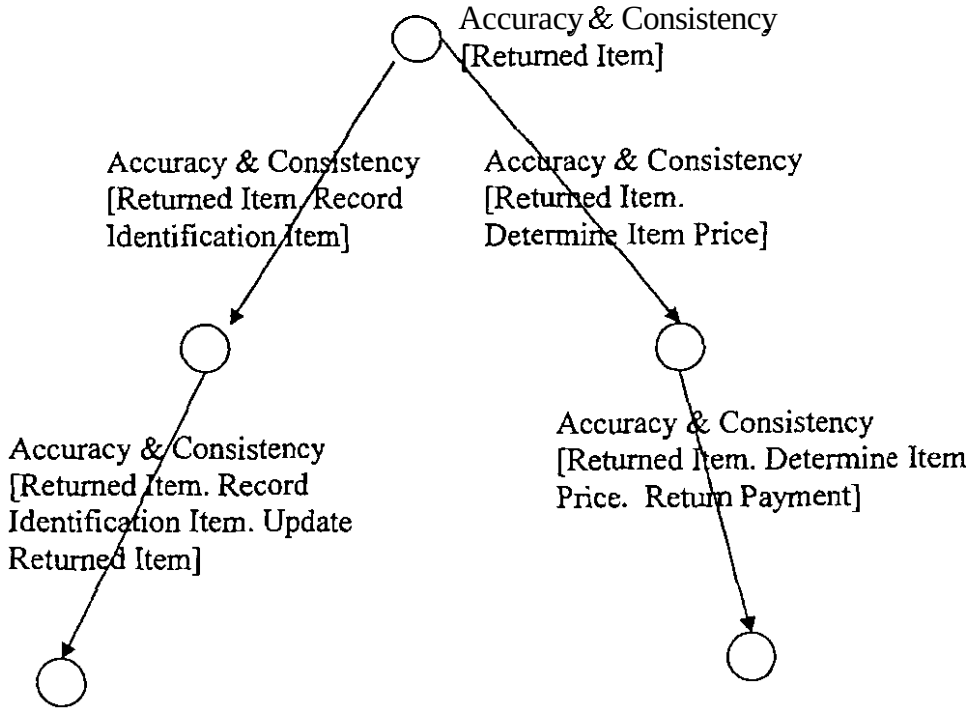


Figure 4.7: NFR graph for returned item

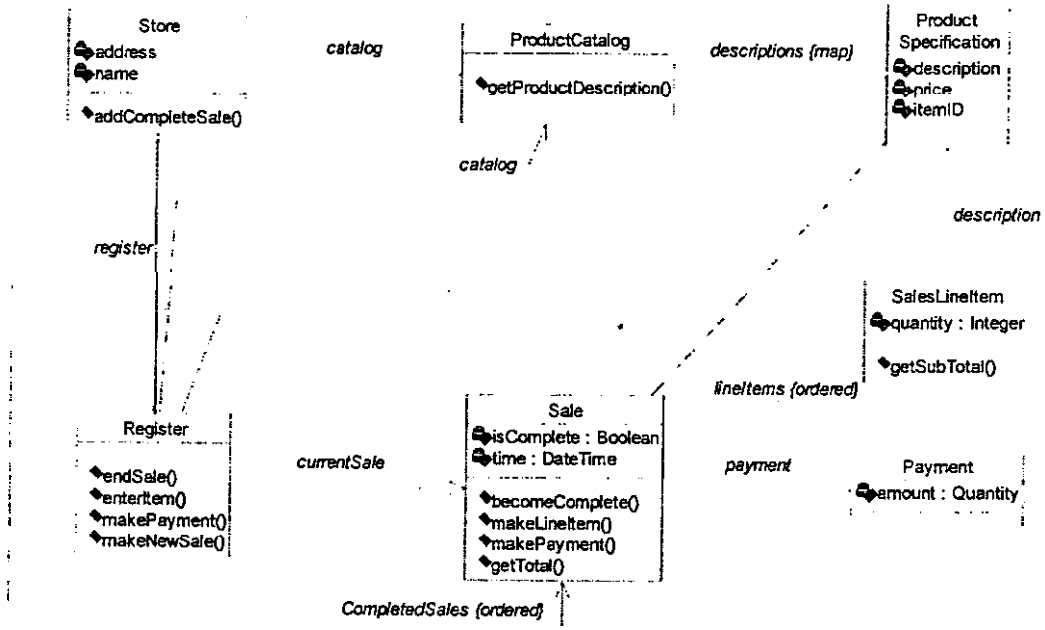


Figure 4.8: Class diagram before integration

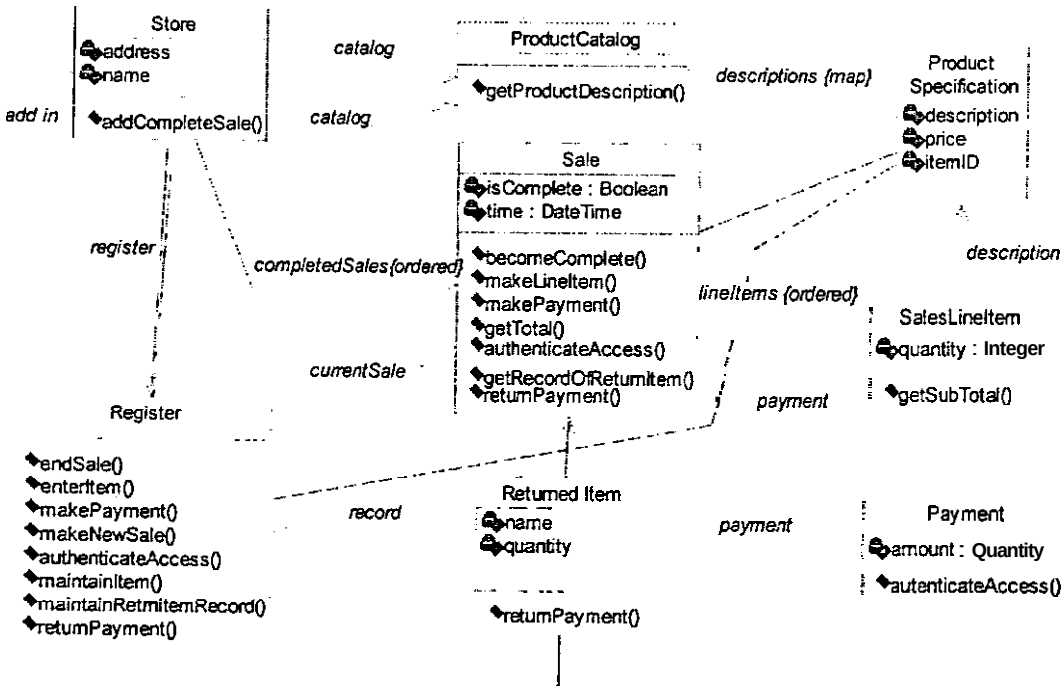


Figure 4.9: Class diagram after integration

4.2.3 Integration of NFRs in POST Sequence Diagram and Communication Diagram

Integrating NFRs into the sequence and communication diagrams is done by examining every class of the class diagram. I am following the strategy for integration of NFRs in sequence and communication diagrams as mentioned in figure 3.5.

Let us take for example the class Payment shown in figure 4.10. Applying the strategy, I searched the sequence diagram or communication diagram seeking for instances of the above class. Figure 4.11a shows the one we found.

Examining the existing diagram, and the operations included in the class due to NFR satisficing, along with the special conditions represented in the note, I concluded that some changes had to be made in this diagram. Figure 4.11b shows the resulting diagram. I can see in this figure that the message tagged with the number 3 to authorize the payment, I added the note to satisfice a security NFR and performance NFR to authenticate access and authentication< 1 minute. This is necessary so the software could check, the payment authentication of the customer, if the customer account work and authenticated, and then perform the operation. For the Payment authentication I attached another note, to message number **3,** which means that the authentication process should be less than one minute.

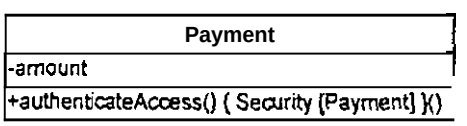


Figure 4.10: Class Payment after integration

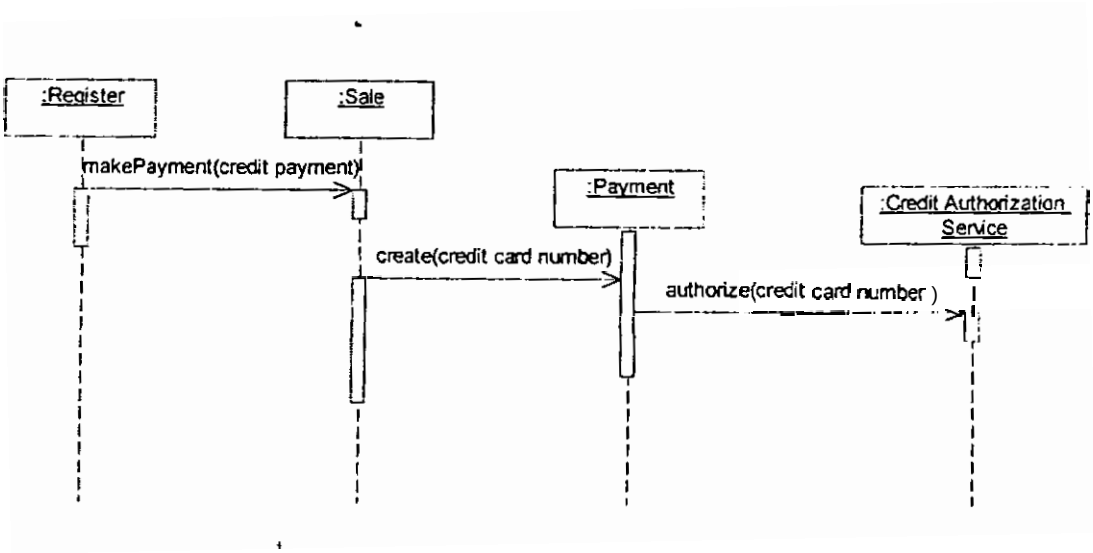


Figure 4.11a: Sequence diagram for makePayment (credit payment) before integration

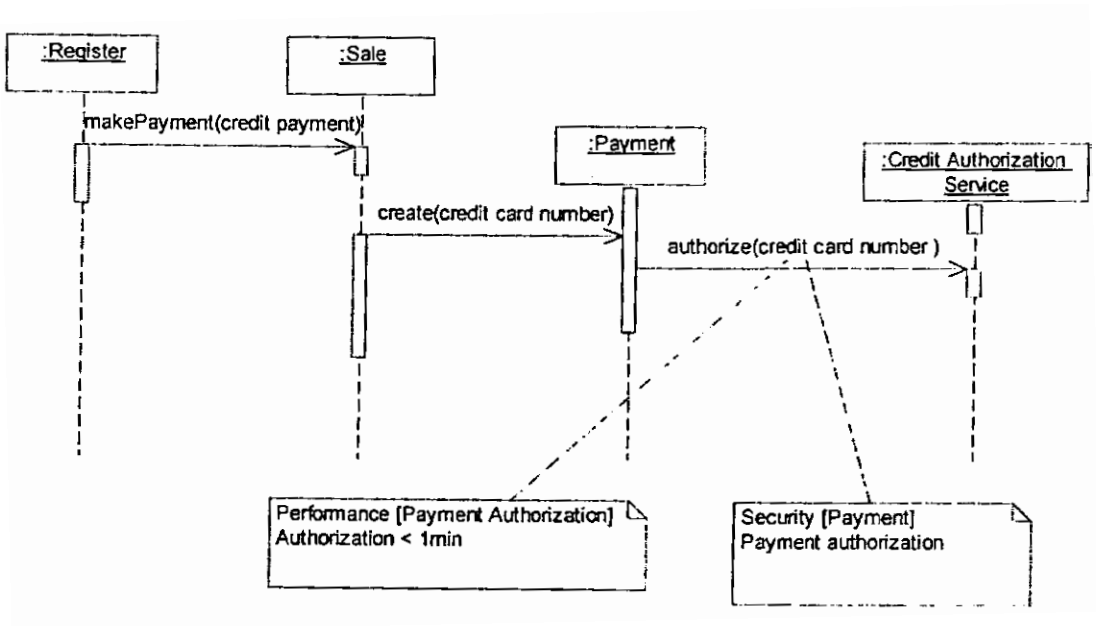


Figure 4.11b: Sequence diagram for makePayment (credit payment) after integation

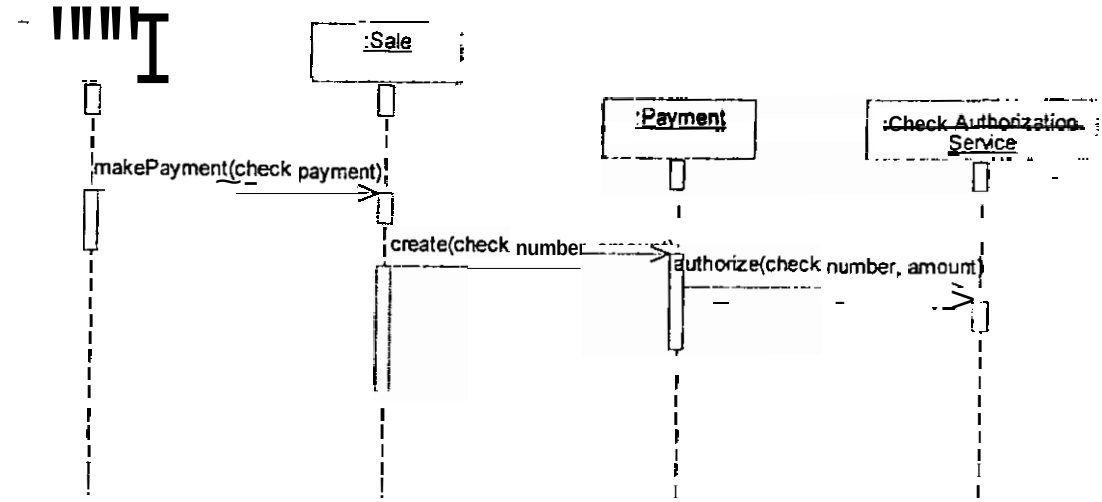


Figure 4.12a: Sequence diagram for makePayment (check payment) before integration

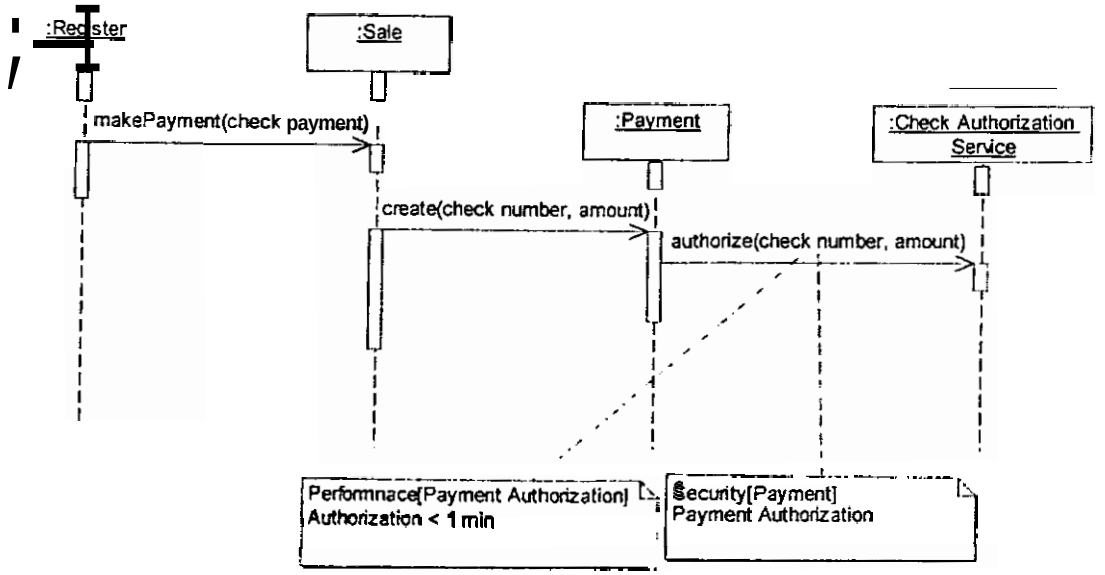


Figure 4.12b: Sequence diagram for makePayment (check payment) after integration

Let us take another example the class sale shown in figure 4.5. Applying the strategy, I searched the communication diagram seeking for instances of the above class. Figure 4.13a shows the one we found. Examining the existing diagram, and the operations

included in the class due to NFR satisficing, along with the special conditions represented in the note, I concluded the some changes had to be made in this diagram. Figure 4.13b shows the resulting diagram. I can see in this figure that the messages tagged with the number 8 is added to satisfice the maintainability NFR to record the sale item. This is necessary so the software could check the sale items should be recorded in the register and update all the record according to the transactions. Then the number 9 message added due to the returned item that should be recorded in the register. And must check the accuracy and consistency of the returned items.

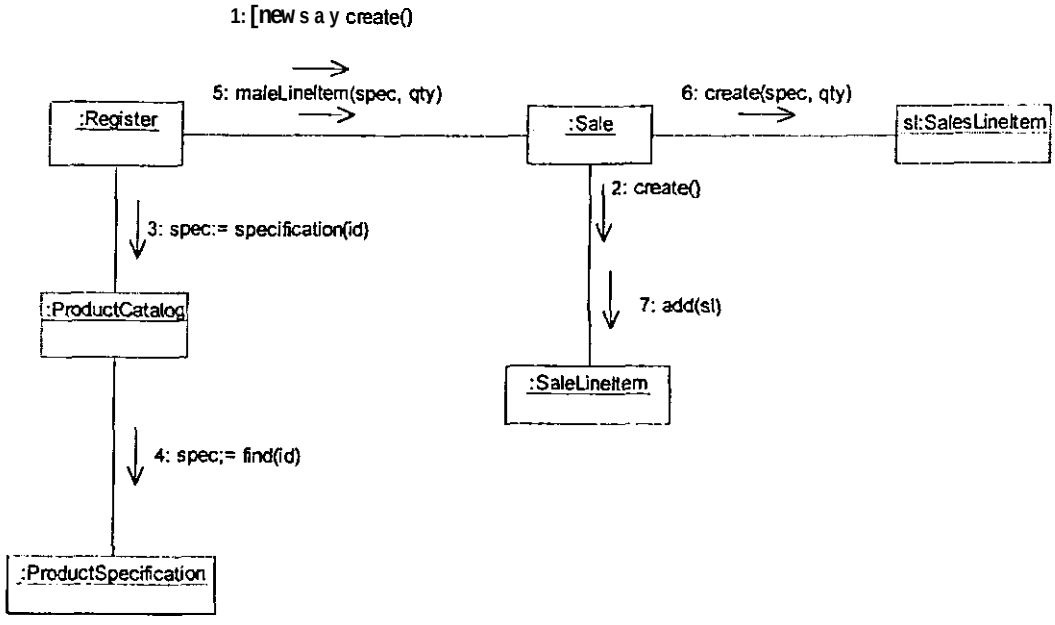


Figure 4.13a: Communication diagram for enterItem before integration

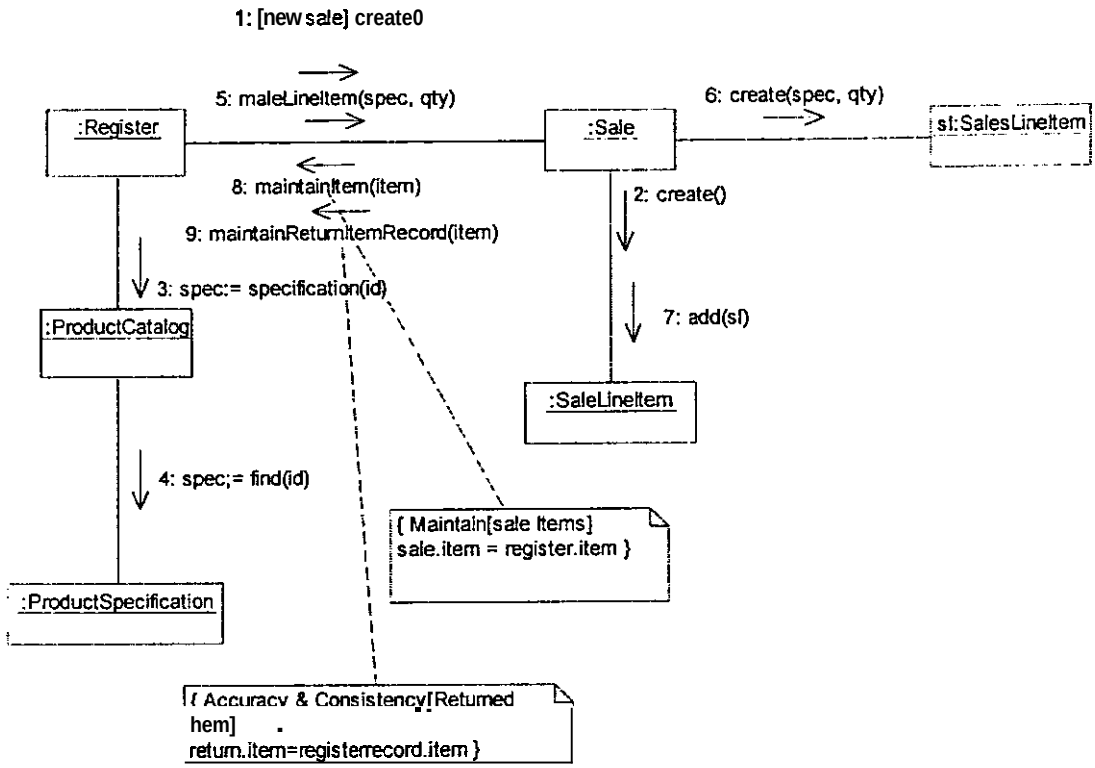


Figure 4.13b: Communication diagram for enterItem after integration

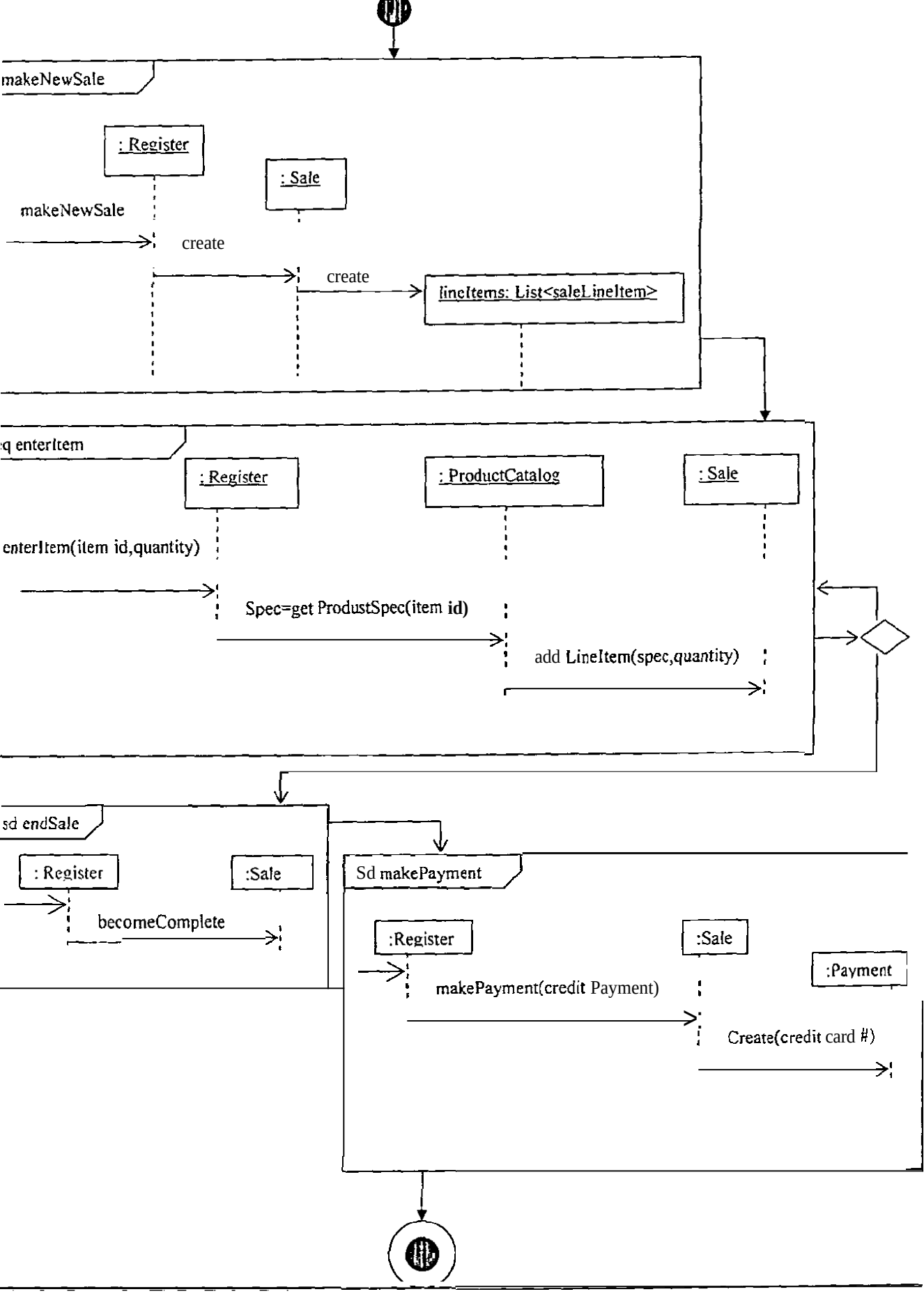
4.2.4 Integration of NFRs in POST IOD

Integrating NFRs into the IOD is done by examining the messages of sequence and communication diagram. We follow the strategy for integrating NFRs in IOD as mentioned in figure 3.6.

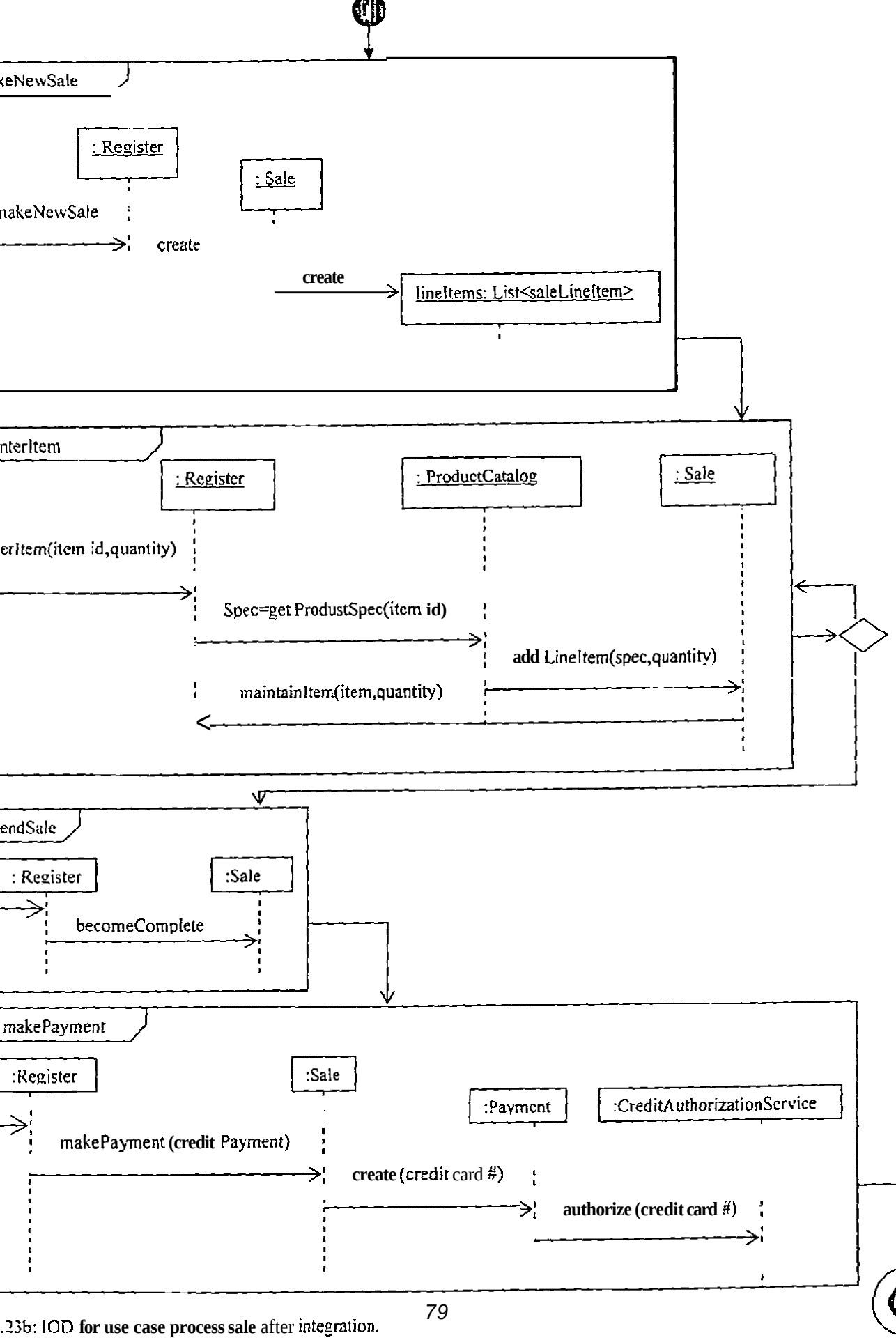
Let us take for example the use case returned items or handle returns. Figure 4.24a shows the IOD for use case returned items before integration. During the integration of NFRs in class diagram, *a* class and some methods are added in POST case study due to different NFRs. These changes have also impact in IOD. Like in class diagram, class returned items are added during the NFRs integration. The retumed item is added in a result of when the returned items record are maintained and updated. During this procedure, we must check the accuracy and consistency of the returned item and maintain the record then return payment to customer.

Figure 4.24b shows the IOD for use case retumed item after integration. Before integration the IOD for returned items called three reference diagrams. Because when we analyze NFRs, the addition of new reference diagram is added in the diagram. When customer retumed the item or items, cashier must check the record of item and then the accuracy and consistency of items and then return payment. Cashier checks all the record of returned item from the saleLineItem. So saleLineItem is called in this use case for the inspection of record of returned items. Now, saleLineItem is added in IOD for returned item use case. Cashier checks the record of returned item from the message getRecordOfReturnedItem(), the invoice generated last time when he bought all the items so get the record of returned item and then return the payment of returned item to customer. And also maintain and update the record of returned item.

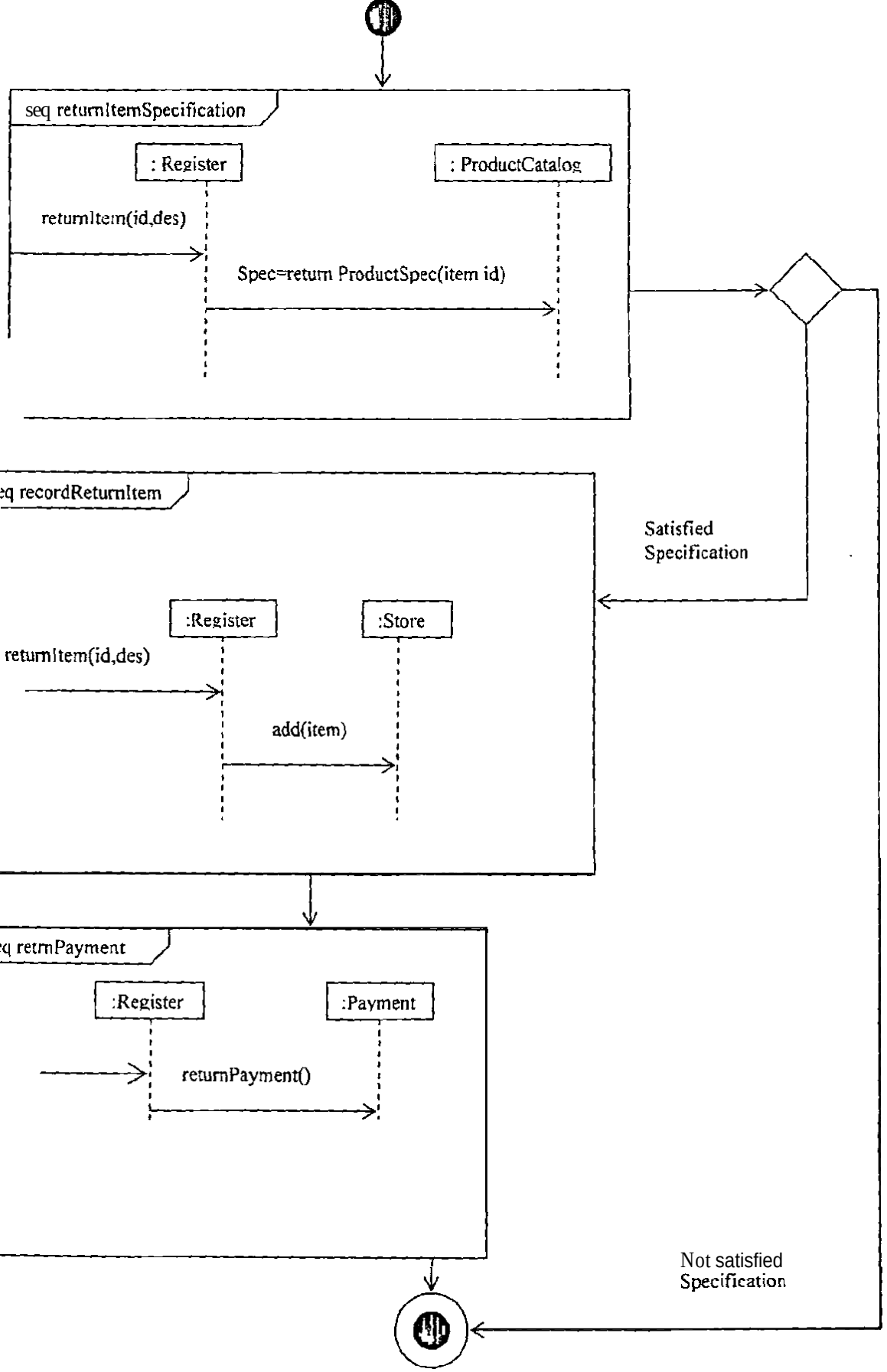
This process makes the easy way for designers and especially for developers to cover the NFRs in a single diagram against a use case. This process is useful because developer see all possible NFRS and interactions against a single use case.



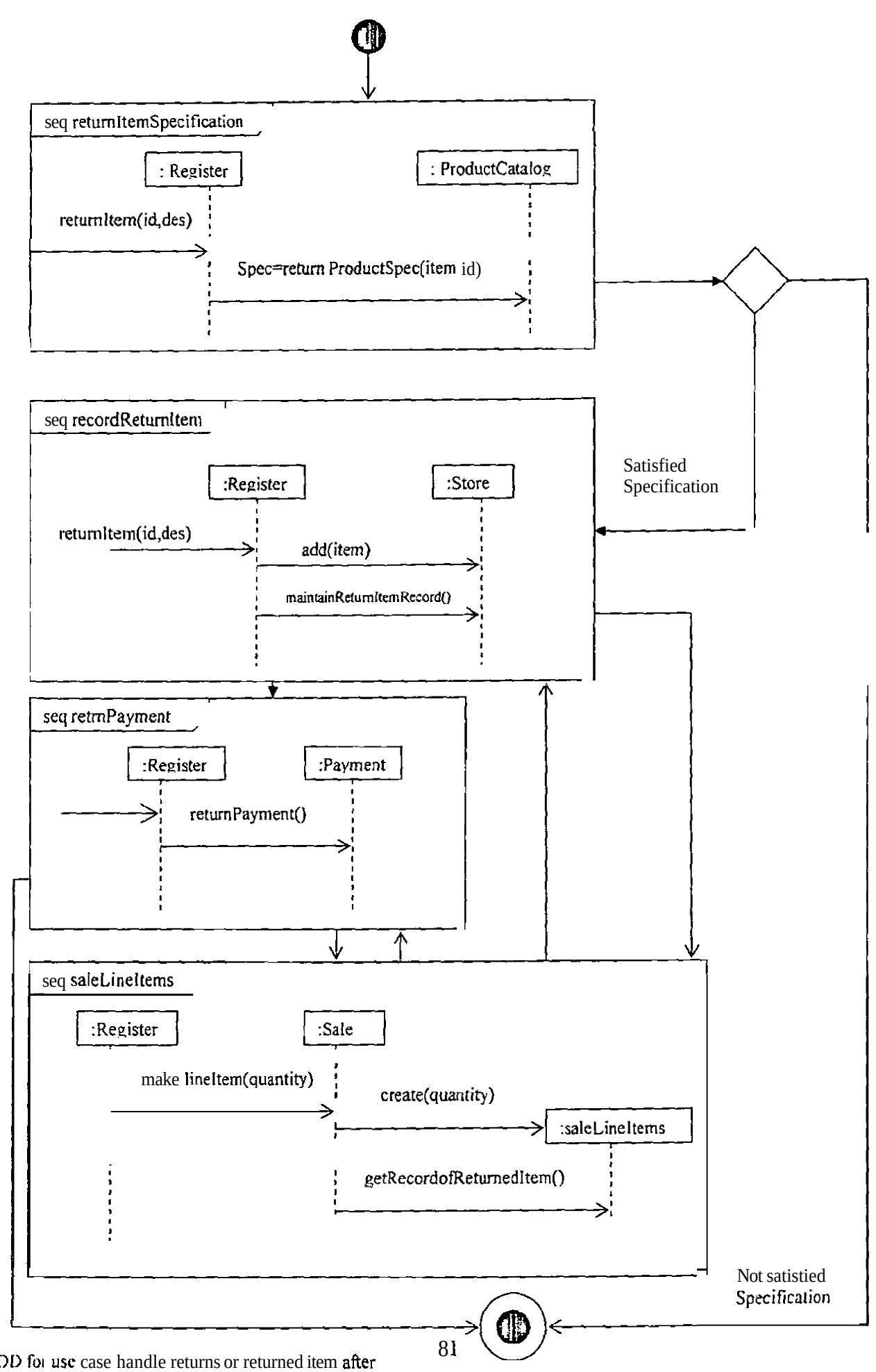
3a: IOD for use case process sale before integration.



.23b: IOD **for use case process sale** after integration.



4a: IOD for use case handle returns or returned item before integration.



81

OD for use case handle returns or returned item after

4.2.5 Integration of NFRs in POST CD

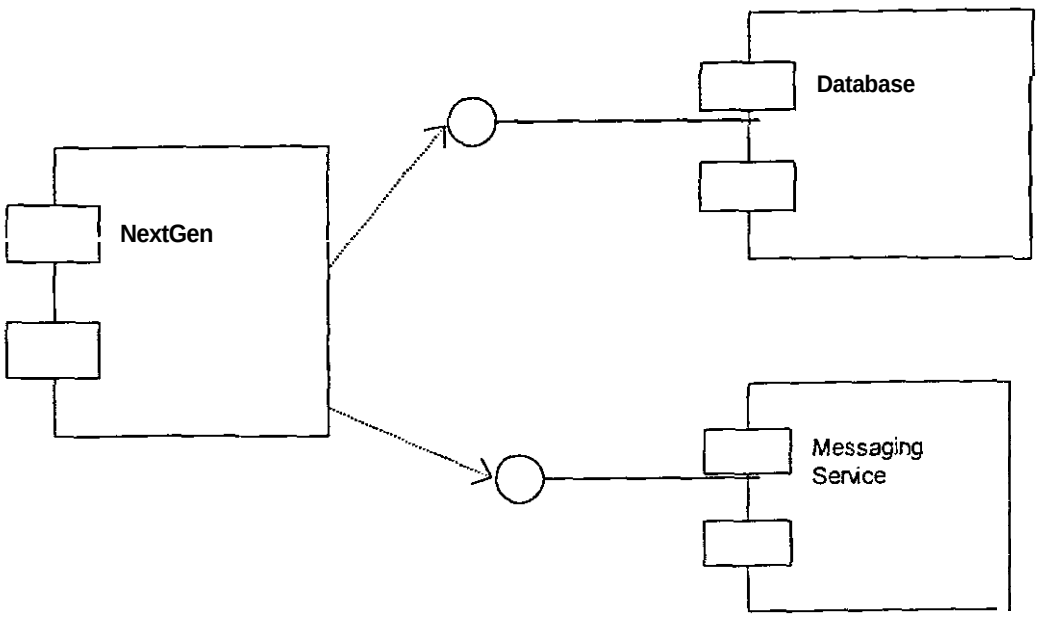


Figure 4.25a: CD for POST before integration

"**A** component represents a modular, deployable and replaceable part of a system that encapsulates implementation and exposes a set of interfaces [OMG 01]."

I am following the strategy for integrating NFRs in CD as mentioned in figure 3.8.Figure 4.25a represents the CD for the system POST.

Let us take for example the component "NextGen" and another component is "Messaging Service". When the component of NextGen is requiring an interface from messaging service, then the messaging service must be secure between the NextGen and messaging service component. Security [Messaging Service] for the payment authorization or for the payment must be secure and must be secure communication are formed between these two components. When the component of NextGen is requiring an interface from database. then

check all possible NFRs related to these interactions. The one is that the security [database] must be maintained. Figure 4.25b represents the CD after integration.

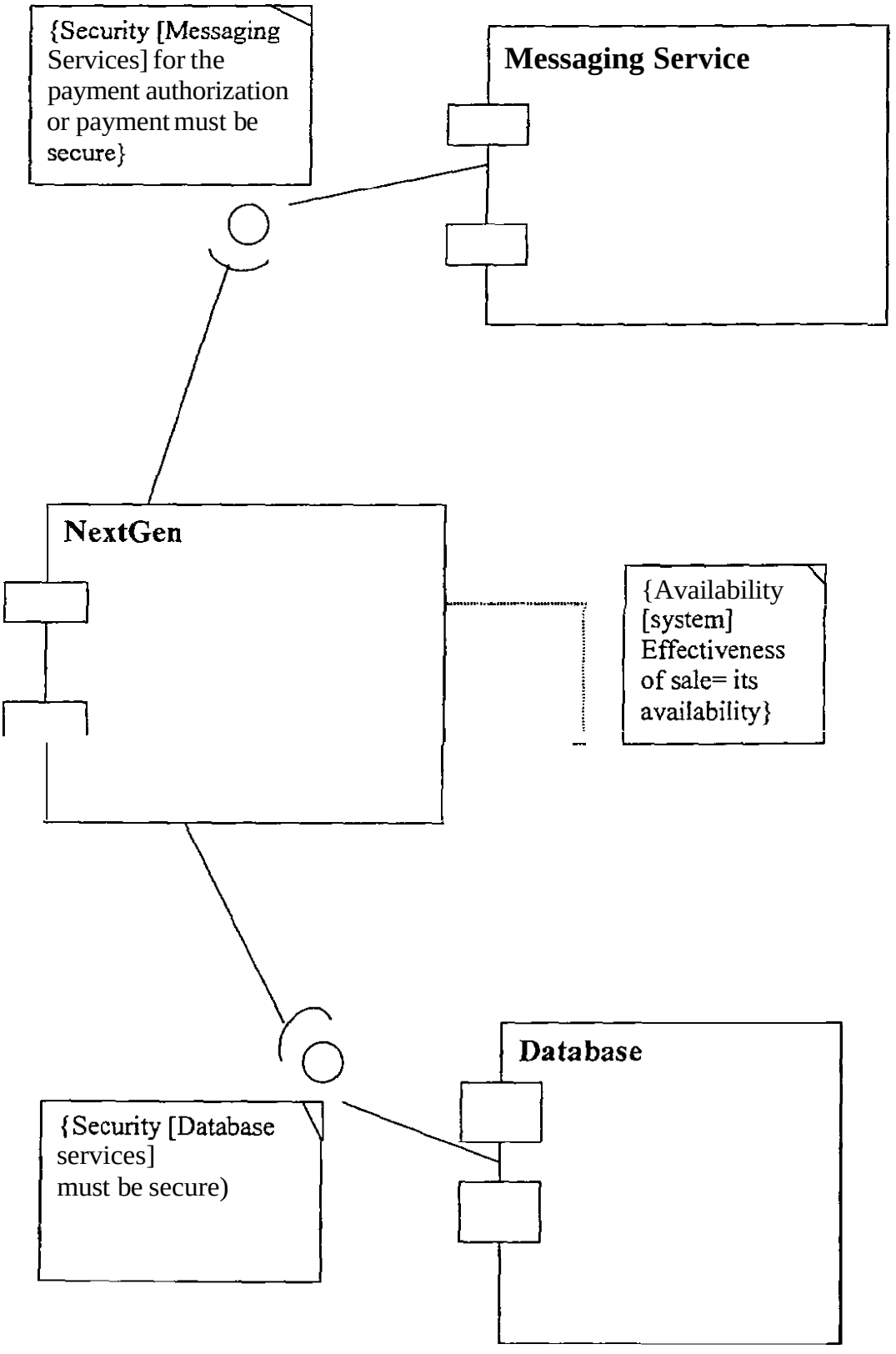


Figure 4.25b: CD for POST after integration process

Another way to integrate NFRs in CD is by using "UML profile" [L. Zhu & I. Gorton, 20071. If we use this UML profile for NFR then first we have to update class diagram of system. We have to show all NFRs related to the system in the class diagram. Then we can associate those NFRs to component in CD. According to this profile, first we consider the six element framework from the Software Engineering Institute (SEI) [Len, Clements & Kazman, 2003]. These six elements are stimulus, source of stimulus, environment, artifact, response and response measures. The six elements are specified for the NFRs. Figure 4.26 shows the UML profile for NFR. When we consider this profile we have to consider the related NFRs to the system and then specify six elements and then add in the class diagram and then incorporate in CD.

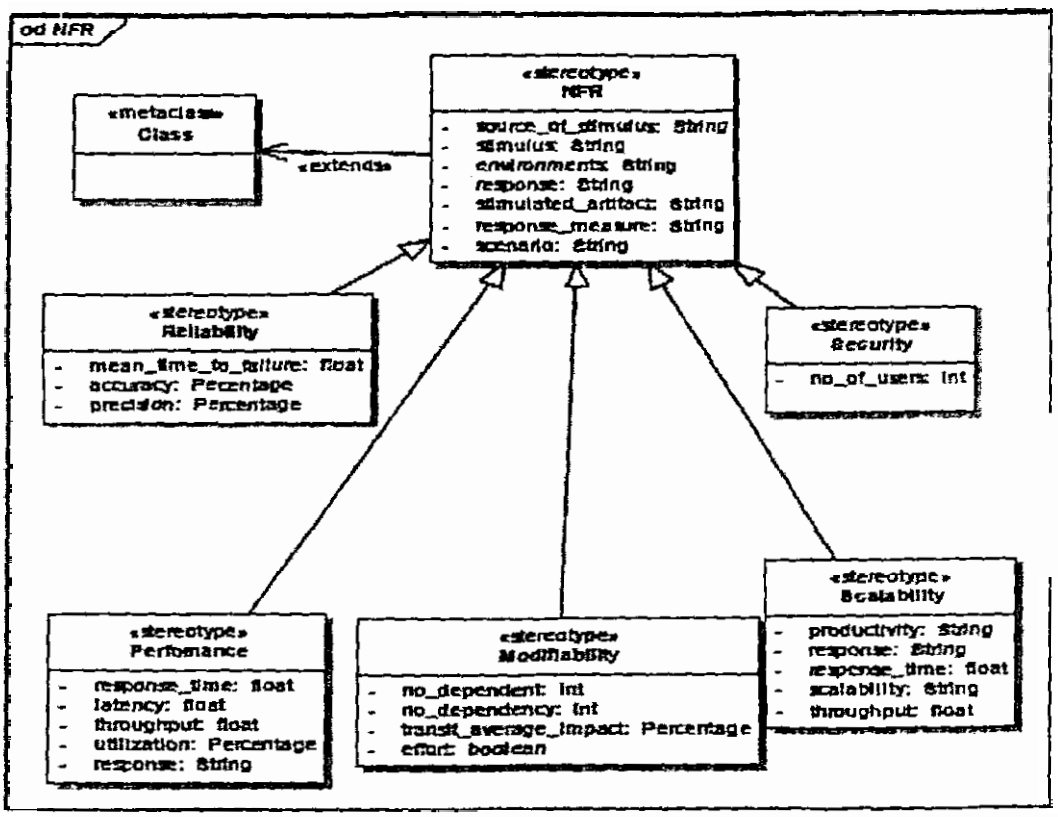


Figure 4.26: UML profile for NFR.

4.2.6 Integration of NFRs in POST CSD

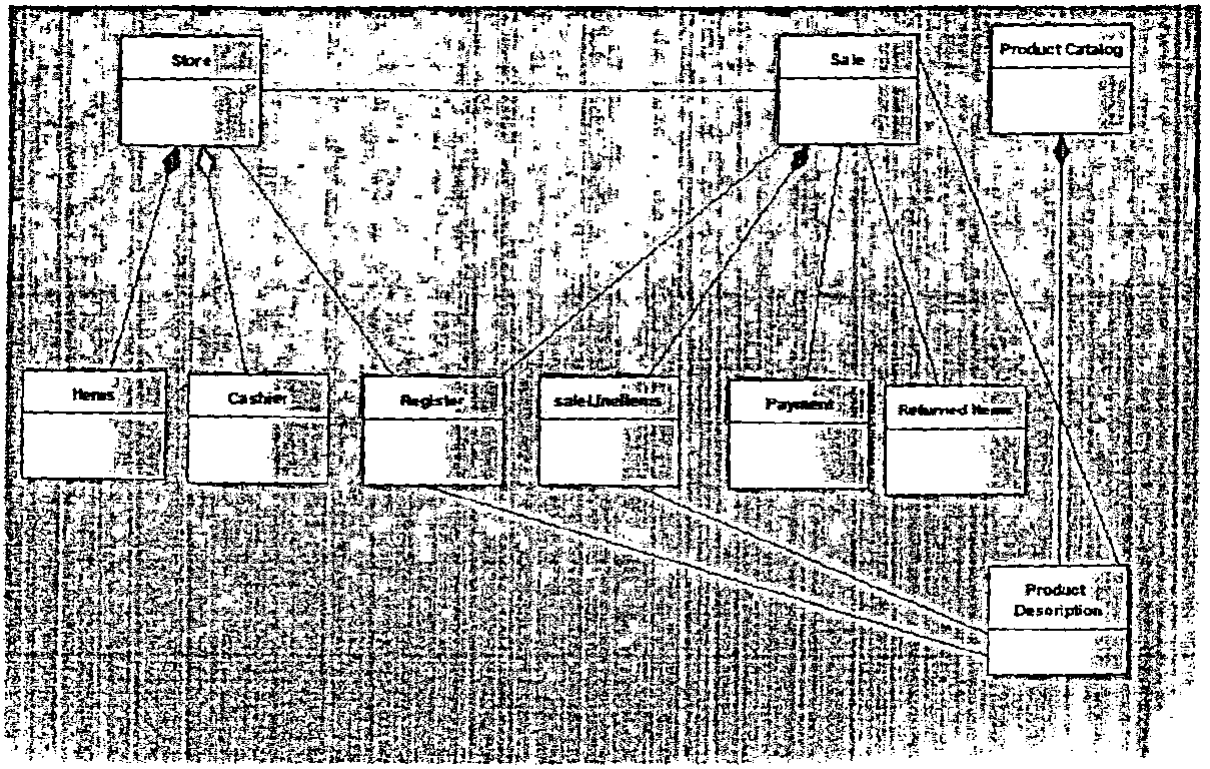


Figure 4.27: Composition and association of classes

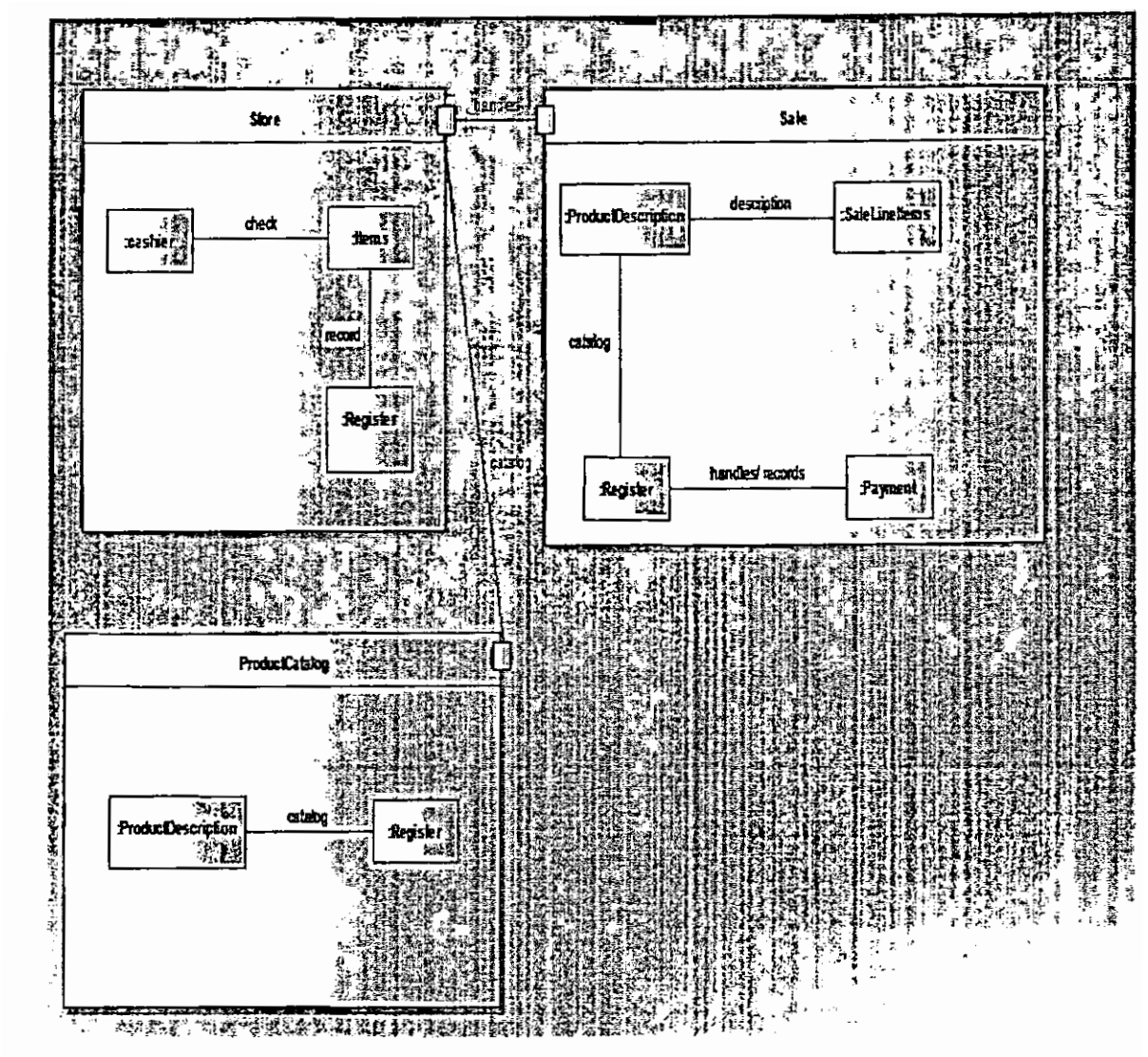


Figure 4.28a: CSD before integration

CSD shows the internal structure of a class and the collaborations that this structure makes possible [OMG Superstructure, 2007]. Composite class contains different parts and these parts have strong relationships with their composite class. The composite classes are interacting through ports with other composite class.

The integration of NFRs into the CSD is done by examining the class diagram. I am following the strategy for integrating NFRs in CSD as mentioned in figure 3.7.

When I integrate NFRs in CSD, classes are involved in CSD but when these classes involved in composite class and have relationship with other composite class, so we have to consider the nonfunctional aspect of these composite class or relationships.

Let us take for example the composite class "Sale" which consists of number of different parts like ProductDescription, saleLineItems, Register and Payment. When we integrate class diagram, we found new class named as Returned Item, so this class has also a relationship with the composite class sale. According to the figure 4.27, the sale has a returned item class. So this class shows as a part of composite class sale, shown in figure 4.28b.

Without this integration or addition in the diagram is incomplete, so the understanding of system is partial. The integration of CSD makes it complete and easily understandable.

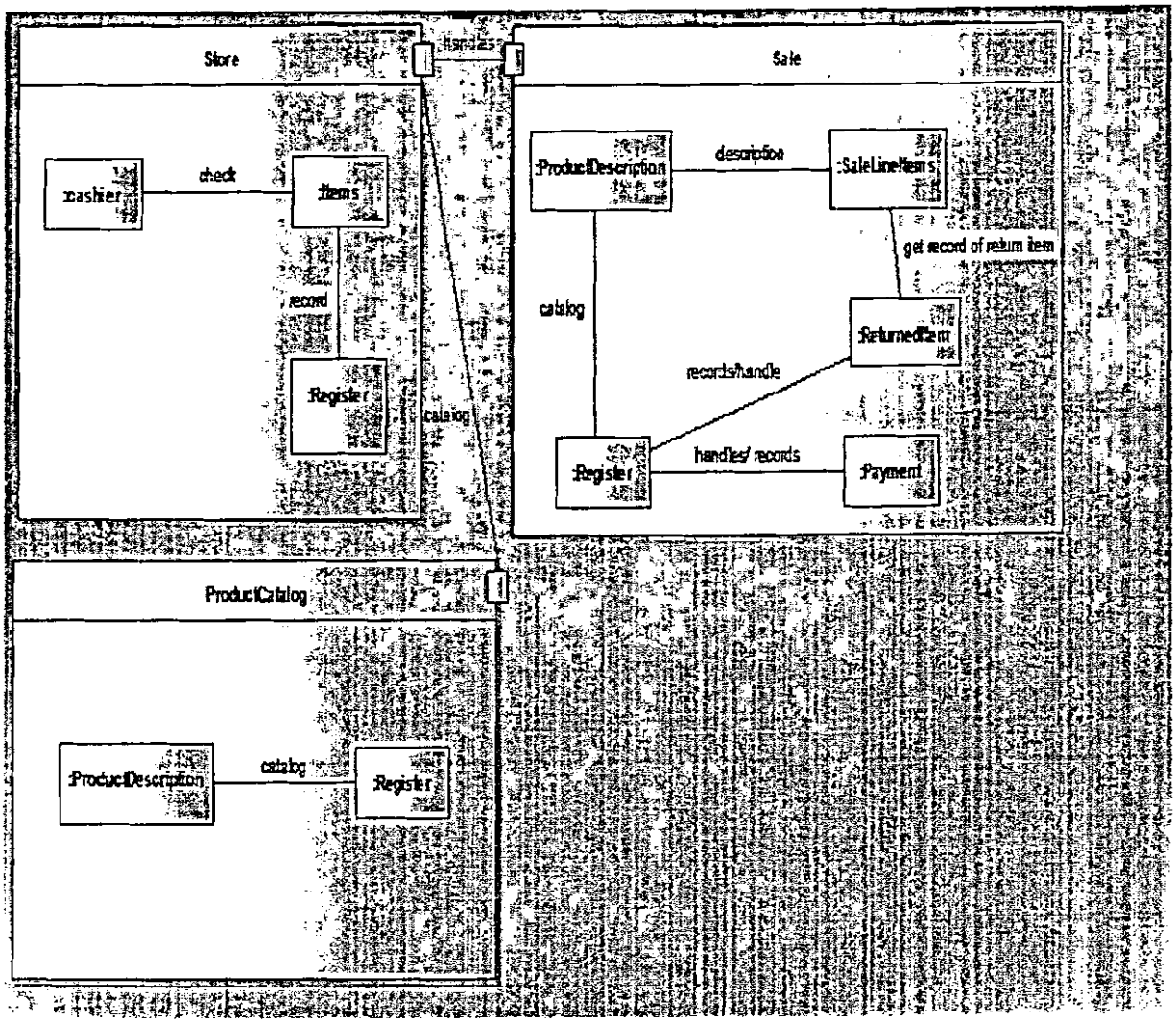


Figure 4.28b: CSD after integration

.s mentioned above in CD, another way to integrate NFRs in CSD is by using "UML rofile" [L. Zhu & I. Gorton, 2007]. If we use this UML profile for NFR then first we have to ıpdate class diagram of system. We have to show all NFRs related to the system in the class diagram. Then we can associate those NFRs to composite class in CSD. According to this profile, first we consider the six element framework fiom the SEI [Len, Clements & Kazman, 2003]. These six elements are stimulus, source of stimulus, environment, artifact, response and response measures. The six elements are specified for the NFRs. As figure 4.26 show the UML profile for NFR. When we consider this profile we have to consider the related NFRs to the system and then specify six elements and then add in the class diagram and then incorporate in CSD.

Chapter 5

Conclusion

5. Discussion

This chapter presents a discussion on the contributions of this thesis and limitations of the research, followed by an outline of the future work.

5.1 Conclusion and Limitations

Despite the fact that NFRs are very difficult to attain and at the same time are expensive to deal with, a few research work [L.Cysneiros & Leite, 2001], [L.Cysneiros & Leite, 2004] have focused on them as first class requirements in a development process. Developers have mainly focused on the FRs of the system during the design phase. However, the key point behind the success of a software system lies in specifying the NFRs along with FRs during the design phase. A survey [D. Grimshaw, Godfrey, 2001] from a small sample of organizations, of the state of the practice in terms of NFRs as shown that:

1. nonfunctional are often overlooked,
2. questioning users is insufficient,
3. methods do not help the elicitation of NFRs, and
4. there is a lack of consensus about the meaning and utility of NFRs.

Researchers are trying to find ways to identify the NFRs easily during the design of a software system. Software systems are becoming large and complex day by day. This complexity includes not only static structure of classes but their relationship with each other their functionality, behavior, state etc., it is difficult to gasp this information as a whole for any system which leads to misunderstanding of that system. Modeling of software systems help to minimize this complexity by abstracting out vital information from that system. The

roblem of effectively designing and analyzing software system to meet its NFRs is critical
the system success.

nly recently, research results are showing ways of dealing with NFRs [L.Cysneiros & Leite.
001], [L. Chung & Yu, 2000] at the software definition level. Eliciting and specifying NFRs
difficult [L.Cysneiros & Leite, 2004]. Few attempts have been made to elicit NFRs
.Cysneiros & Leite, 2001]. NFRs have also been integrated with functional models
.Cysneiros & Leite, 2004] [L.Cysneiros & Leite, 2001].

y contribution also contributes to fill this regard in software development during design
ocess. The problem with the identification of NFRs and then how to deal NFRs in software
stems during design process. I proposed a strategy that tackles the problems of NFRs, and
oposes a systematic process to assure that the models will satisfice these NFRs during
sign process. The objective is to provide a consistent way of dealing with NFRs at all
ages of development and in different views. The main contribution is to show that a
mbination of widely used functional or conceptual models with an NFR framework is
fective in improving the overall quality of the requirement process. This strategy is based
the use of domain glossary to build functional and nonfunctional perspectives. Using this
ossary, firstly I showed how to record NFRs against FRs. Secondly, I also showed how to
egrate NFRs in UML 2 functional models, and presented a systematic way to integrate
FRs into the functional model of UML 2.

veloping quality software always requires the proper elicitation of NFRs with the FRs.
e elicitation process of NFRs is based on the use of a domain glossary. Domain glossary is
sed on a vocabulary system composed of tasks where each task is expressed through its
scription and associated NFRs. The description explains the meaning of the task and its

relation with other task. The associated NFRs specify the non functional aspects of the mentioned task.

Secondly, I showed the NFRs integration strategies in a systematic way. These strategies are building for NFRs to integrate in UML 2 functional models. The integration process also deals with the representation of multiple NFRs for the same class, and also to address dynamic model instead of addressing only static models.

I improved our strategy by applying the case study POST. The result found in the case study, suggest that the use of this strategy can lead to a final functional or conceptual model with better quality, as well as to a more productive software development process and also the exclusion of misunderstandings of the system. Through this modeling of NFRs, we analyze the system in an organized way and we achieve the better visualization and understanding of the system. Although my strategy may be used for almost any type of NFR, I understand that its results will be more effective when addressing NFRs that effectively demand actions to be performed by the system, and therefore, affects the software design. The lack of automation between the task use in glossary and the construction of the NFR graphs also poses some concerns about the time spent in this job, as well as in the accuracy of the process.

NFRs such as Maintenance and Portability are not easily operationalized in a specific point of the artifact, but rather will be more related on how the design is organized. My strategy help to elicit such NFRs, but since they are not operationalizable, they are dealt in abstract level in my strategy.

On the other hand, NFRs such as Security, Performance, Accuracy, Consistency. and other, frequently demand the design to be carefully studied in order to satisfice these NFRs. Hence,

it will be more likely that these NFRs will be the type of NFRs that my strategy will help the most.

This strategy is useful for the integration of all NFRs related to software system while some profiles are also available for the integration of NFRs but these profiles are dealt to specific NFRs not for all NFRs related to system.

I believe that there is no research and its results are an ultimate solution to any research problem under investigation. I assume that the case which I selected and presented is good representative and successful for all kinds of software. Yet, there is a limitation: NFRs such as Portability and Maintenance are not easily operationalized in a specific point of the artifact; they are dealt in proposed strategy but in an abstract level.

Based on the literature survey, proposed strategy and the result of case study, I can claim that I found satisfactory answers to the research question outlined in the chapter 1. It is also possible to argue that the proposed solution positively affects the software systems and provides a better way and quality to represent the software system through the consideration of NFRs.

5.2 Future Research Directions

My current research focuses on incorporating or integrating NFRs into the design of a system. The artifacts of UML 2 include IOD, CSD and CD is covered in our research work. The future research directions are as follows:

1. The automation of integration strategy will certainly improve its productivity.
2. The extensions of this strategy to other UML artifacts which are not yet covered like package diagram and object diagram.
3. Apply this strategy on different case studies and controlled experiments for improvements,
4. All NFRs are not discemable at run time [Standish Group, 1994], my strategy deals with discernable NFRs. Non discemable NFRs are difficult to operationalize. So work need to he carried out to model non discemable NFRs like modifiability. reusability etc.

References

[A. Finkelstein & Dowell, 1996] A. Finkelstein & J. Dowell. (1996). *A Comedy of Errors: The London Ambulance Service Case Study.* Proceedings of 8th International Workshop on Software Specification and Design, IEEE Computer Society Press.

[A.Moreira & Brito, 2002] **A.** Moreira, I. Brito, & J. Arajo. (2002) *Crosscutting quality attributes for requirements engineering.* In The fourteenth International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy.

[A. Moreira & Brito, 2002] **A.** Moreira, I. Brito, J. Arajo. (2002). A *Requirements Model for Quality Attributes.* In the 1'' International Conference on Aspect-Oriented Software Development, Dresden, Germany.

[Betty & Atlee, 2007] **Betty** H.C. **Cheng** & **Joanne** M. Atlee. (2007). *Research Directions in Requirements Engineering.* IEEE Future for Software Engineering, Washington, USA.

[Brian, 2004] Brian Berenbach. (2004). *Towards a unified model for requirements engineering.* In Proceedings of the 4th International Workshop on Adoption-Centric Software Engineering (ACSE), Edinburgh, Scotland.

[Boehm & Barry, 1996] Boehm, Barry and In. (1996). *Identifying Quality-Requirement Conflicts.* In the Proceedings of the 2nd International Conference on requirement Engineering, Washington, USA.

[Boehm , 1978] Boehm, B. ''Characteristics of Software Quality'' North Holland Press, 1978.

[Dardenne & Van , 1993] Dardenne **A, Van** Lamsweerde A, Fickas S. (1993). *Goal Directed Requirements Acquisition.* Science Computer Programming.

[Donald, 2004] Donald Bell. (2004) UML *basics: The Component diagram.* IBM Global Services.

[D.J. Grimshaw, Godfrey & Draper, 2001] D.J. Grimshaw, W. Godfrey, and G.W. Draper. (2001). *Non-Functional Requirements Analysis: Deficiencies in Structured Methods.* Information & Software Technology, Elsevier.

[E. Dimitrov & Schrnietendorf , 2002] E. Dimitrov and A. Schrnietendorf. (2002). *UML-based Performance Engineering Possibilities and Techniques.* IEEE Software, Ottawa, Canada.

[E. Insfran & Pastor ,2002] E. Insfran, O. Pastor and R. Wieringa. (2002). *Requirements Engineering-Based Conceptual Modeling.* Springer-Verlag London Limited.

[E. Insfran & Pastor , 1999] E. Insfran, Wieringa R, Pastor O. (1999). *Using* TRADE *to Improve an Object-Oriented Method.* Technical report, Computer Science Department, University of Twente, Enschede, The Netherlands.

[Ebert ,1997] Ebert, C. (1997). *Dealing with Nonfunctional in Large Software System's.* Annals of Software Engineering, USA.

[Fenton & Pfleeger , 1997] Fenton, N.E. and Pfleeger, S.L. (1997). *Software Metrics: A Rigorous and Practical Approach.* In the *2''* International Thomson Computer Press, Elsevier.

[G. Booch & Stevens, 2003] Grady Booch, Perdite Stevens, John Whittle. (2003). *The Unified Modeling Language" Modeling Languages and Applications.* 6th Intemational conference San Francisco, CA, USA.

[G. Kotonya & Sommerville, 1998] G. Kotonya, I. Somrnerville. (1998). *Requirements Engineering: Processes and Techniques.* John Wiley & Sons.

[G. Salazar, P. Botella, 2000] G. Salazar Zarate, P. Botella. (2000). *Use of UML modeling nonfunctional aspects.* In Proceedings of 13th International Conference on Software and System Engineering and their Applications, London, UK.

[H. Wada & J. Suzuki, 20061 Hiroshi Wada, Junichi Suzuki, Katsuya Oba. (2006). *A Model-Driven Development Framework for Non-functional Aspects in Service Oriented Architecture.* IEEE Intemational Conference on Service Computing, Pisa.

[H. Espinoza, H. Dubois, 2006] Huascar Espinoza, Hubert Dubois, Sebastien Gerard, Dorina C Perrius & Murray Woodside. (2006). *Annotating UML Models with Nonfunctional Properties for Quantitative Analysis.* Springer Berlin, Heidelberg.

[ISSCO , 1995] Evaluation of Natural Language Processing Systems. (1995). http://www.issco.unige.ch1'ewg95.

Kirner & Davis , 996]	Kirner T.G., Davis A .M. (1996). *Nonfunctional Requirements of Real-Time Systems.* Advances in Computers, Academic Press, New York.
Keller . 1990]	Keller, S.E. et al. (1990). *Specifying Software Quality Requirements with Metrics.* In Tutorial System and Software Requirements Engineering IEEE Computer Society Press.
. Chung & J. Mylopoulos & , 992]	L. Chung, J. Mylopoulos, E. Yu, and B. Nixon. (1992). *Representing and Using Nan-Functional Requirements: A Process-Oriented Approach.* IEEE Transactions Software Engineering.
.Cysneiros & Leite 2004]	Luiz Marcio Cysneiros, Julio Cesar Sampaio do Prado Leite. (2004). *Nonfunctional Requirements: From Elicitation to Conceptual Models. IEEE* Transactions on Software Engineering.
. Chung & Nixon , 995]	L. Chung and B. Nixon. (1995). *Dealing with Nonfunctional Requirements: Three Experimental Studies of a Process-Oriented Approach.* Proceedings of 17th International Conference on Software Engineering, IEEE.
.Cysneiros & Yu , 003]	L.M. Cysneiros and E. Yu. (2003). *Non-Functional Requirements Elicitation.* Perspective in Software Requirements, Kluwer Academics.
.Cysneiros & eite, 2001]	L.M. Cysneiros, J.C.S.P. Leite, and J.S.M. Neto. (2001). *A Framework for Integrating Nonfunctional Requirements into Conceptual Models.* Requirements Engineering Journal, IEEE.
.Cysneiros & eite, 2001]	L.M. Cysneiros and J.C.S.P. Leite. (2001). *Using UML to Reflect Nonfunctional Requirements.* Proceedings of 11th CASCON Conference.
. Chung & Yu , 00]	L. Chung, B. Nixon, E. Yu, and J. Mylopoulos. (2000). *Non-Functional Requirements in Software Engineering.* Kluwer Academics.
.Cysneiros & ite, 1999]	L.M. Cysneiros and J.C.S.P. Leite. (1999). *Integrating Non-Functional Requirements into Data Modeling* Proceedings of 4'' International Symposium Requirements Engineering, Limerick. Ireland.

[Larman , 1998] Larman C. (1998). *Applying UML and Patterns. An introduction to Object Oriented Analysis and Design and Iterative Development.* Prentice-Hall.

[L. Xu & Ziv ,20051 Lihua Xu, Hadar Ziv, Debra Richardson, Zhixiong Liu. (2005). *Towards Modeling Non-Functional Requirements in Software Architecture.* Annual Aspect Oriented Software Development Conference, Limerick, Ireland.

[Lindstrom , 1993] Lindstrom, D.R. (1993). *Five* Ways *to Destroy a Development Project.* IEEE Software.

[L. Zhu & I. Gorton, 2007] Liming Zhu, Ian Gorton. (2007). *UML Profiles for Design Decisions and Non-Functional Requirements.* Second workshop on sharing and reusing architectural knowledge. Architecture, rationale and design intent (Shark-ADI'07) IEEE. Minneapolis, MN.

[Len, Clements & Kazman, 2003] Len Bass, Paul Clements and Rick Kazman. (2003). *Software Architecture in Practice. 2''* edition; Addison- Wesley.

[M. Usman, Atif, Rizwan & Shahzad] Muhammad Usman, Atif Qureshi, Rizwan bin Faiz, Shahzad Rafiq. Modeling non-functional requirements in activity diagrams, MRSP research group, Faculty of engineering and sciences, Muhammad Ali Jinnah university, Islamabad, Pakistan.

[OMG Superstructure 2007] *OMG Unified Modeling Language: Superstructure.* Version 2.1.1 (with change bars) formal/ 2007-02-03.

[OMG 01] P.Botella & X. Burgues, 2001] OMG (2001). OMG *UML specification.* www.omg.org. Pere Botella, Xavier Burgués, Xavier Franch, Mario Huerta, Guadalupe Salazar. (2001). *Modeling Non-functional Requirements.* Proceedings of Journals of Integrating the Requirements Application JIRA, Sevilha.

[R. Hill & Wang , 2004] Raquel Hill, Jun Wang, Klara Nahrstedt. (2004). *Quantifying non-functional requirements: A process oriented approach.* Proceedings of the 12th IEEE International Requirements Engineering Conference.

[S. Supakkul & Chung, 2004] S. Supakkul and L. Chung. (2004). *Integrating frs and nfrs: A Use Case and Goal Driven Approach.* In 2nd International

Conference on Software Engineering Research, Management and Applications, Los Angeles, CA.

S. Supakkul & hung, 2005] Sam Supakkul and Lawrence Chung. (2005). A UML ***Profile for Goal-Oriented and Use Care-Driven Representation of NFRs and FRs.*** ***In*** Proceedings of the 3rd International Conference on Software Engineering Research, Management and Applications, Mt. Pleasant, MI.

Subrina & ahvildari ,2005] Subrina Anjum Tonu, Ladan Tahvildari. (2005). ***Towards a Framework to Incorporate NFRs into UML Models.*** Proceeding of IEEE WCRE Workshop on Reverse Engineering to Requirements (RETE), Pittsburgh, Pennsylvania, USA.

Subrina, 2006] Subrina Anjum Tonu. (2006). ***Incorporating NFRs with UML Models.*** A thesis, university of waterloo, Ontario, Canada.

Standish Group, 994] The Standish Group. (1994). ***Causes of Failed Software Projects.***

Yu & Eric,1997] Yu, Eric. (1997). ***Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering.*** Proceedings of the 3rd International Symposium on Requirements Engineering, New York, USA.

APPENDIX A

UML 2.0 Diagrams

UML 2 diagrams which are used for the integration of NFRs. The IOD, CSD and CD are involved in integration from UML 2.

1. IOD

The IOD focuses on the overview of the flow of control of the interactions. It is a variant of the activity diagram where the nodes are the interactions or interaction occurrences. The IOD describes the interactions where messages and lifelines are hidden. You can like up the "real" diagram and achieve high degree navigability between diagrams inside the IOD [OMG Superstructure, 2007].

An IOD is a form of activity diagram in which nodes represents interaction diagrams. Interaction diagrams can include sequences, communication, interaction overview and timing diagrams. Most of the notation for IOD is the same for activity diagram. For example, initial, final, decision, merge, fork and join nodes are all the same. However. IOD introduce two new elements: interaction occurrences and interaction elements [OMG Superstructure, 2007].

a. InteractionUse

An InteractionUse refers to an interaction. The InteractionUse is shorthand for copying the contents of the referred interaction where the InteractionUse is. To be accurate the copying must take into account substituting parameters with arguments and connect the formal gates

with the actual ones. It is common to want to share portions of an interaction between several other interactions. An InteractionUse allows multiple interactions to reference an interaction that represents a common portion of their specification [OMG Superstructure, 2007].

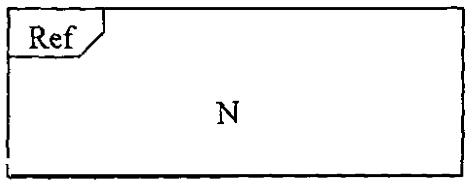


Figure 1: InteractionUse

b. Interaction Element

Interaction elements are similar to interaction occurrences, in that display a representation of existing interaction diagrams within a rectangular frame. They differ in that they display the context of the references diagram inline.

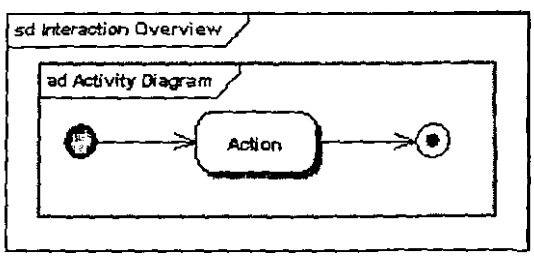


Figure 2: Interaction element [OMG Superstructure, 2007]

All the same controls from activity diagram (fork, join, merge, etc.) can be used on IOD to put the control logic around the lower level diagrams.

2. CSD

A UML2 CSD shows the internal structure of a class and the collaborations that this structure makes possible. This can include internal parts, ports through which the parts interact with

each other or through which instances of the class interact with the parts and with the outside world, and connectors between parts or ports. A composite structure is a set of interconnected elements that collaborate at runtime to achieve some purpose. Each element has some defined role in the collaboration [OMG Superstructure, 2007].

CSD **in *the*** UMLZ specification [OMG Superstructure, 2007]

The key composite structure entities identified in the UML2 specification are internal structure, parts, ports, collaborations, structured classes, and actions.

Internal Structure

The internal structure sub package provides mechanisms for specifying structures of interconnected elements that are created within an instance of a containing classifier. A structure of this type represents a decomposition of that classifier and is referred to as its "internal structure."

a. Part

A part is an element that represents a set of one or more instances which are owned by a containing classifier instance. So for example, if a diagram instance owned a set of graphical elements, then the graphical elements could be represented as parts; if it were useful to do so, to model some kind of relationship between them. Note that a part can be removed from its parent before the parent is deleted, so that the part isn't deleted at the same time.

A part is shown as an unadorned rectangle contained within the body of a class or component element.

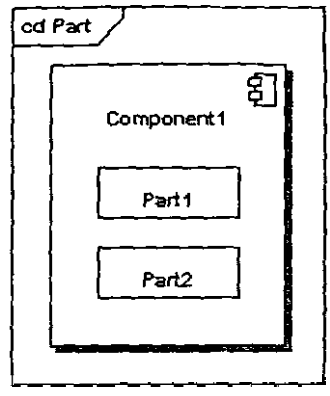


Figure **3:** Part [OMG Superstructure,2007]

b. Port

The Ports sub package provides mechanisms for isolating a classifier from its environment. This is achieved by providing a point for conducting interactions between the internals of the classifier and its environment. This interaction point is referred to as a "port." Multiple ports can be defined for a classifier, enabling different interactions to be distinguished based on the port through which they occur. By decoupling the internals of the classifier from its environment, ports allow a classifier to be defined independently of its environment, making that classifier reusable in any environment that conforms to the interaction constraints imposed by its ports: **A** port is shown as a named rectangle on the boundary edge of its owning classifier.

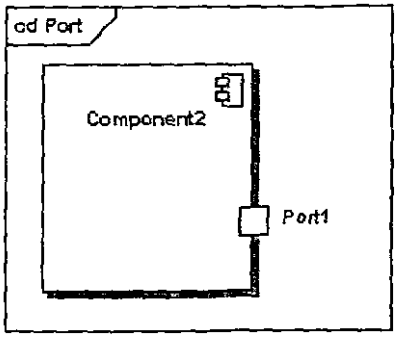


Figure 4: Port [OMG Superstructure, 2007]

c. Collaborations

Objects in a system typically cooperate with each other to produce the behavior of a system. The behavior is the functionality that the system is required to implement. A behavior of collaboration will eventually be exhibited by a set of cooperating instances (specified by classifiers) that communicate with each other by sending signals or invoking operations. However, to understand the mechanisms used in a design, it may be important to describe only those aspects of these classifiers and their interactions that are involved in accomplishing a task or a related set of tasks, projected from these classifiers. Collaborations allow us to describe only the relevant aspects of the cooperation of a set of instances by identifying the specific roles that the instances will play. Interfaces allow the externally observable properties of an instance to be specified without determining the classifier that will eventually be used to specify this instance. Consequentially, the roles in collaboration will often be typed by interfaces and will then prescribe properties that the participating instances must exhibit, but will not determine what class will specify the participating instances.

A collaboration element is shown as an ellipse.

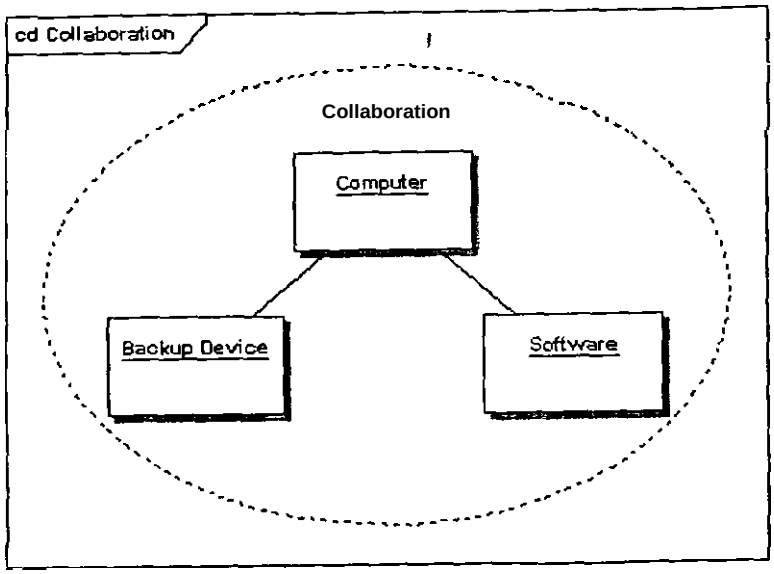


Figure 5: Collaboration [OMG Superstructure, 2007]

d. Structured Classes

The structured classes sub package supports the representation of classes that may have ports as well as internal structure.

e. Actions

The actions sub package adds actions that are specific to the features introduced by composite structures (e.g., the sending of messages via ports).

3. CD

"A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type. whose

conformance is defined by these provided and required interfaces" [OMG Superstructure, 2007].

In component-based development (CBD), CD offer architects a natural format to begin modeling a solution. CD allows an architect to verify that a system's required functionality is being implemented by components, thus ensuring that the eventual system will be acceptable. In addition, CD is useful communication tools for various groups. The diagrams can be presented to key project stakeholders and implementation staff. While CD are generally geared towards a system's implementation staff, CD can generally put stakeholders at ease because the diagram presents an early understanding of the overall system that is being built. Developers find the CD useful because it provides them with a high-level, architectural view of the system that they will be building, which helps developers begin formalizing a roadrnap for the implementation, and make decisions about task assignments and/or needed skill enhancements. System administrators find component diagrams useful because they get an early view of the logical software components that will be running on their systems.

The Basics of CD

Drawing a component in UML2 is now very similar to drawing a class on a class diagram. In fact, in UML2 a component is merely a specialized version of the class concept. This means that the notation rules that apply to the class classifier also apply to the component classifier. In UML2, a component is drawn as a rectangle with optional compartments stacked vertically. **A** high-level, abstracted view of a component in UML2 can be modeled as just a rectangle with the component's name and the component stereotype text and/or icon. The component stereotype's text is «component»" and the component stereotype icon is a

rectangle with two smaller rectangles protruding on its left side (the UML1.4 notation element for a component). Figure 2 shows three different ways a component can be drawn using the UML2 specification.

Figure 6: The different ways to draw a component's name compartment [OMG Superstructure, 2007]

When drawing a component on a diagram, it is important that you always include the component stereotype text (the word "component" inside double angle brackets, as shown in Figure 2) and/or icon. Reason is: in UML, a rectangle without any stereotype classifier is interpreted as a class element. The component stereotype and/or icon distinguish this rectangle as a component element.

APPENDIX B

Integration of NFRs in Sequence and Communication Diagrams

The integration of NFRS in sequence and communication diagrams for hvo major use cases are given below:

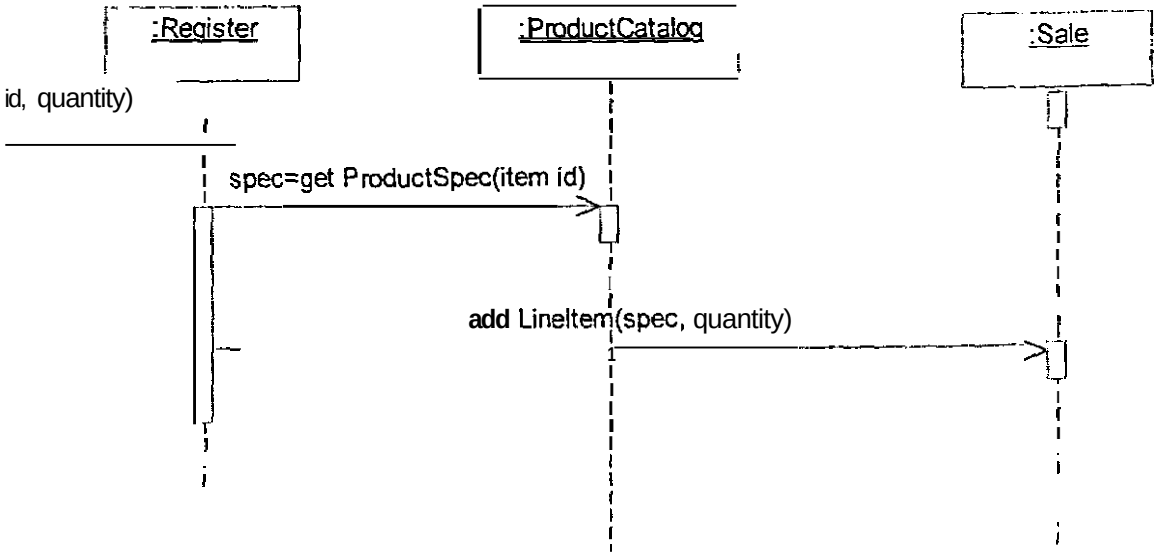


Figure 4.14a: Sequence diagram for enterItem before integration

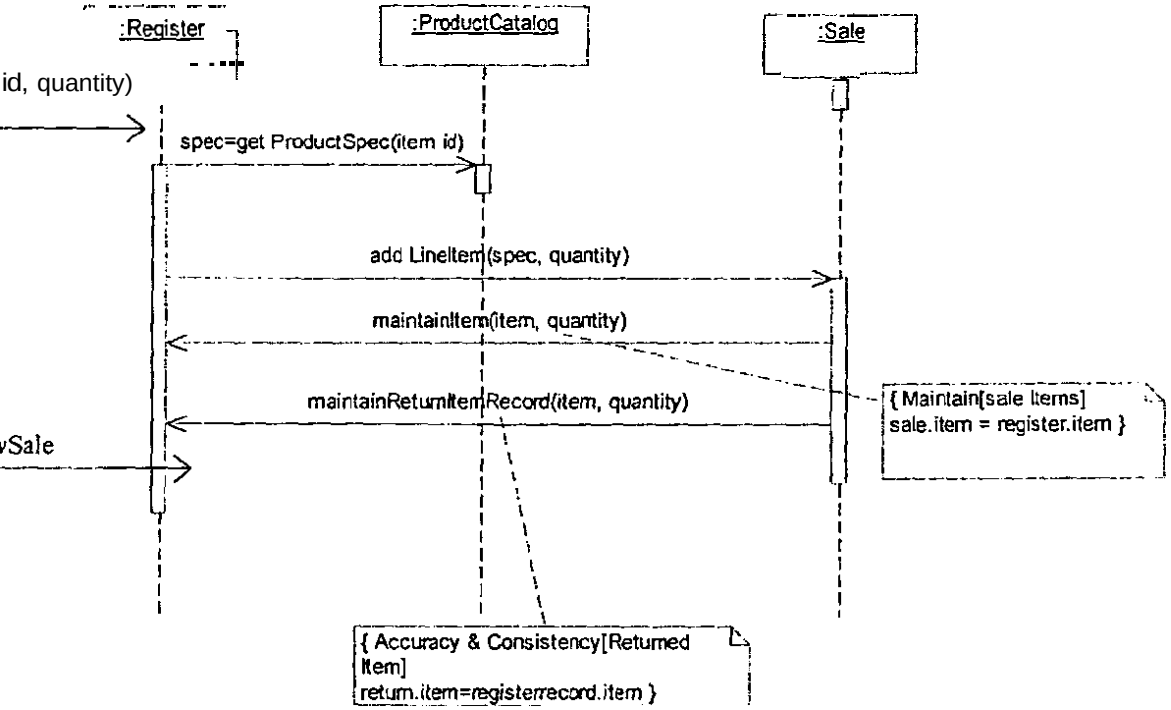


Figure 4.14b: Sequence diagram for enterItem after integration

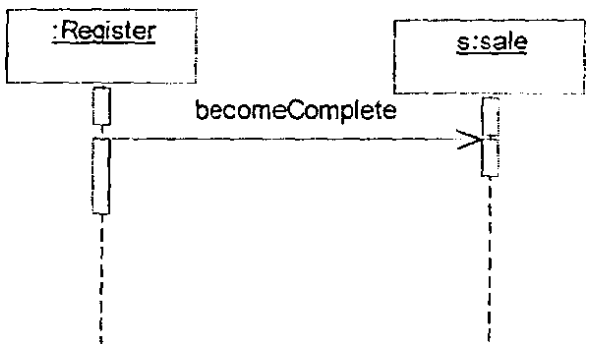


Figure 4.15a: Sequence diagram for endSale before integration

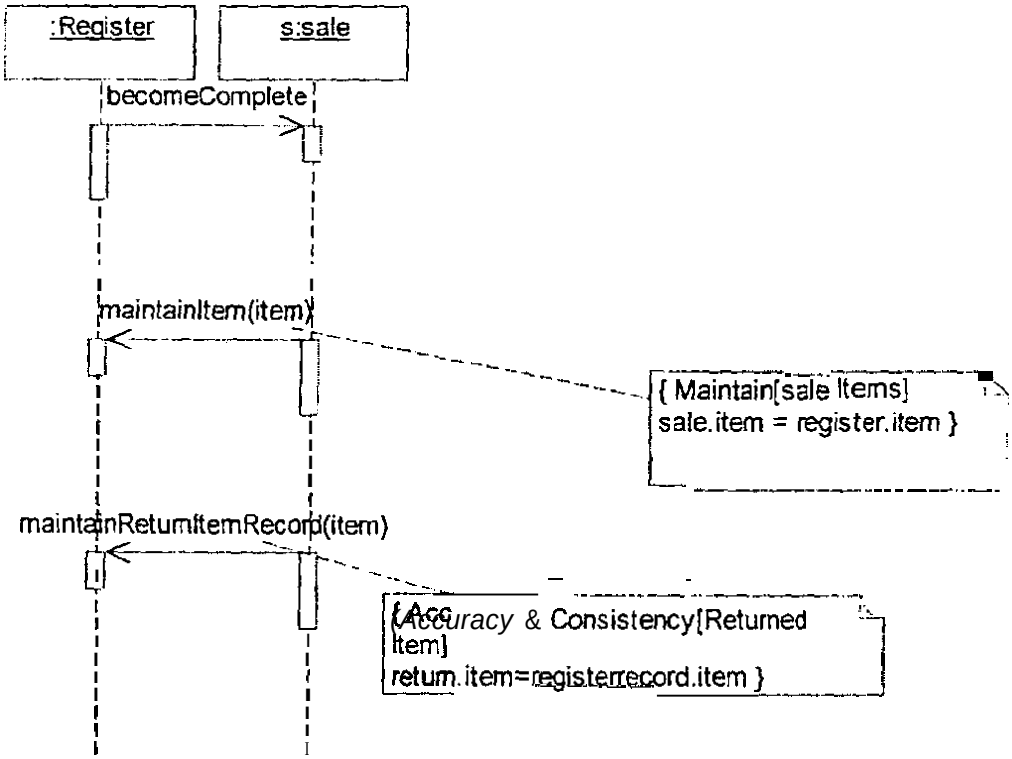


Figure 4.15b: Sequence diagram for endSale after integration

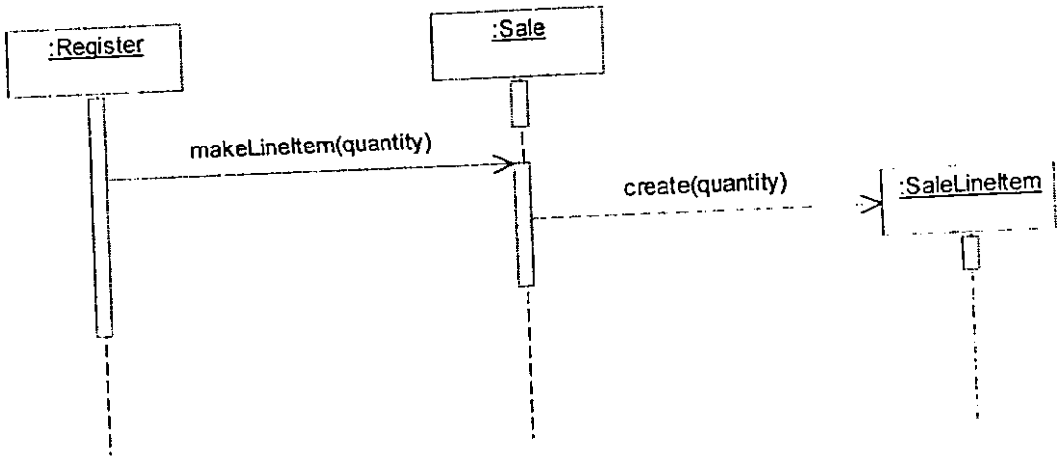


Figure 4.16a: Sequence diagram for saleLineItem before integration

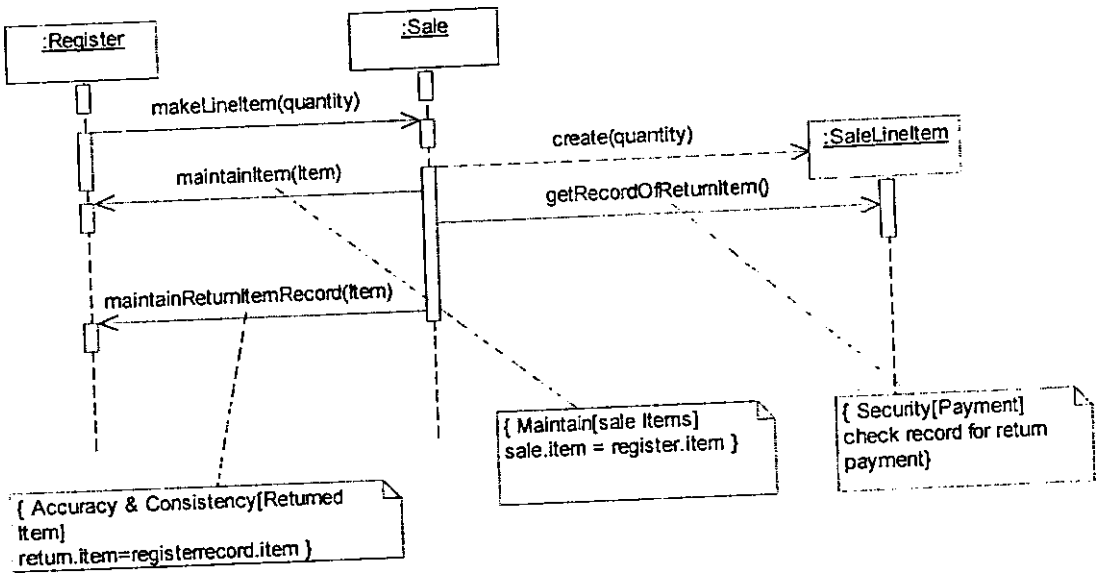


Figure 4.16b: Sequence diagram for saleLineItem after integration

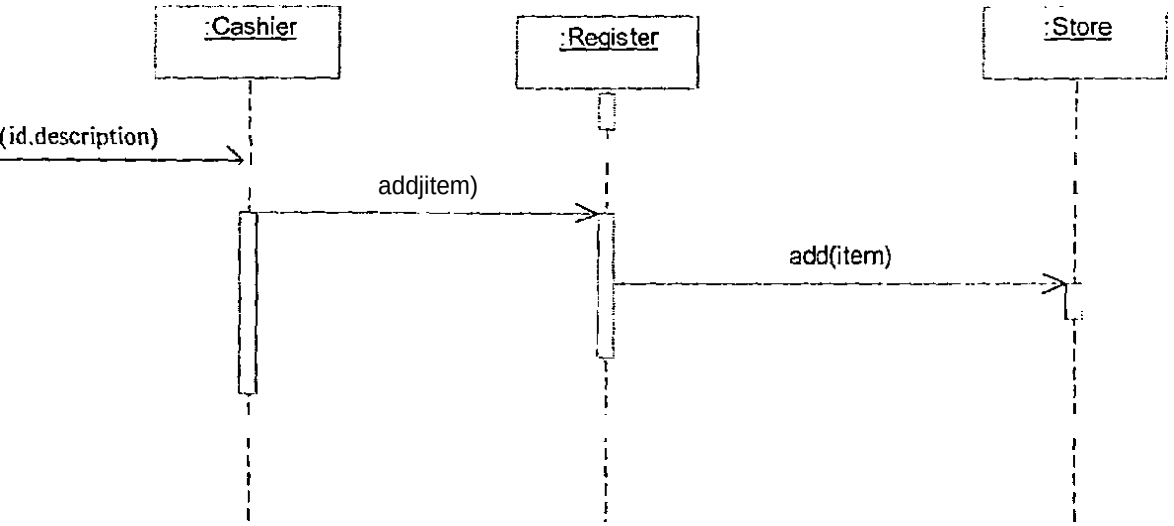


Figure 4.17a: Sequence diagram for recordReturnItem before integration

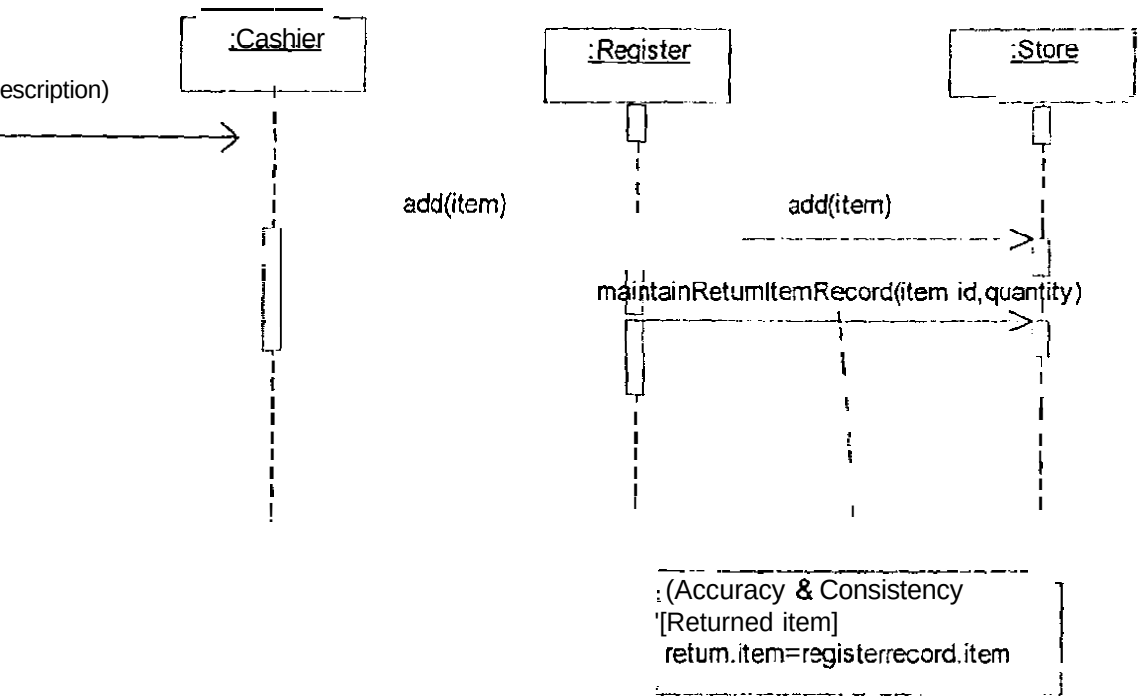


Figure 4.17b: Sequence diagram for recordReturnItem after integration

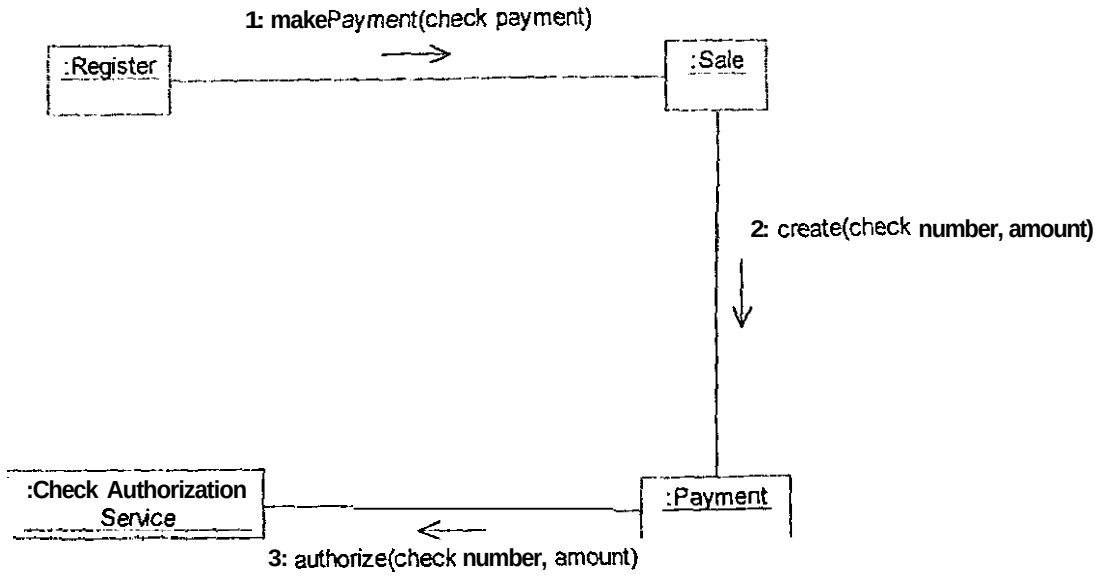


Figure 4.18a: Communication diagram for makePayment (check payment) before integration

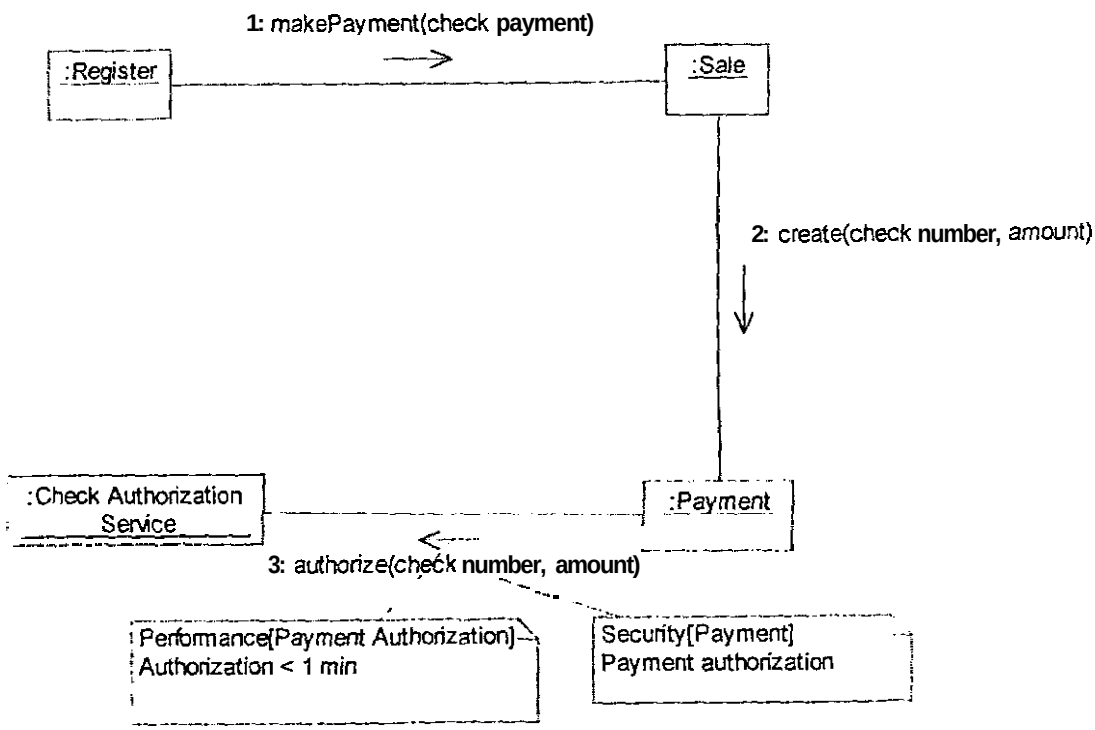


Figure 4.18b: Communication diagram for makePayment (check payment) after integration

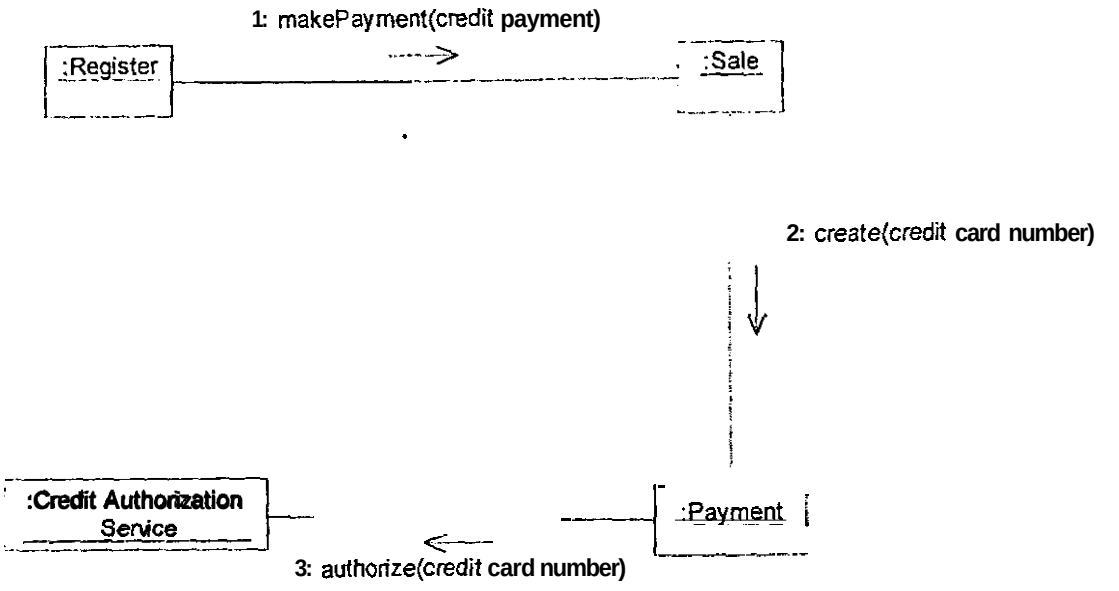


Figure 4.19a: Communication diagram for makePayment (credit payment) before integration

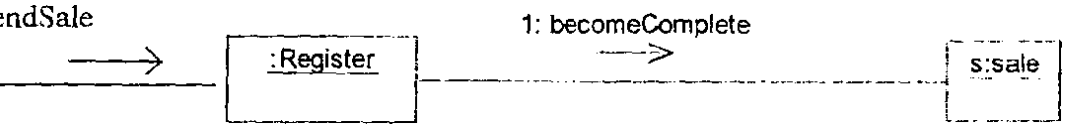


Figure 4.20a: Communication diagram for endSale before integration

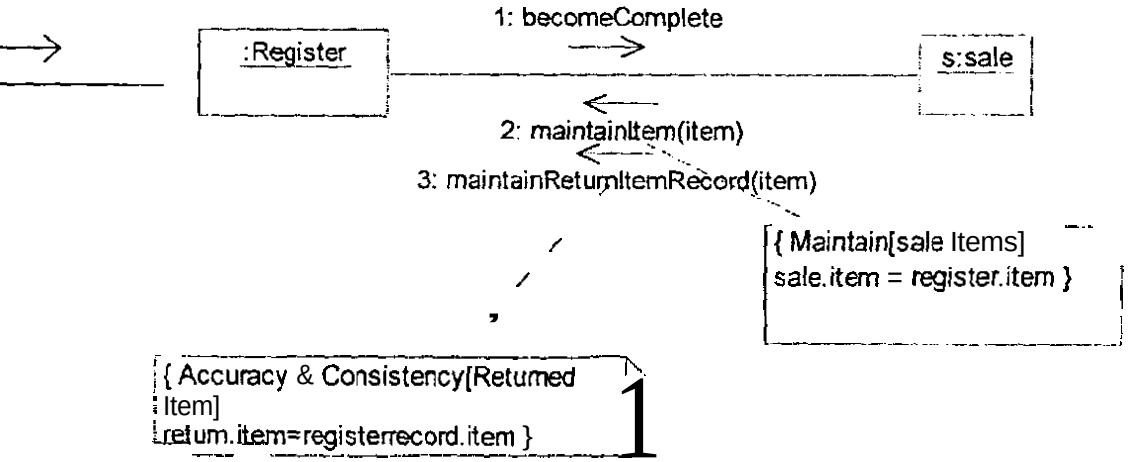


Figure 4.20b: Communication diagram for endSale after integration

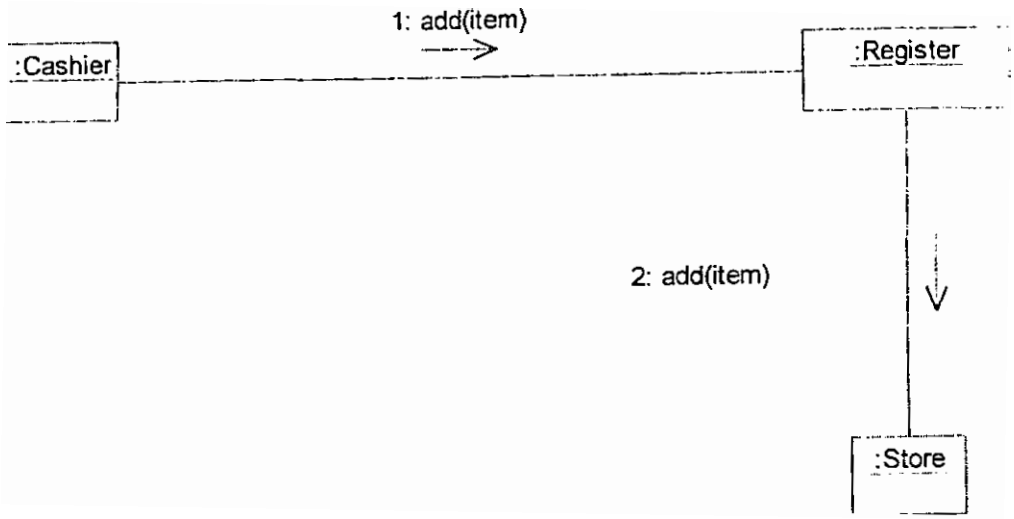


Figure 4.21a: Communication diagram for recordReturnItem before integration

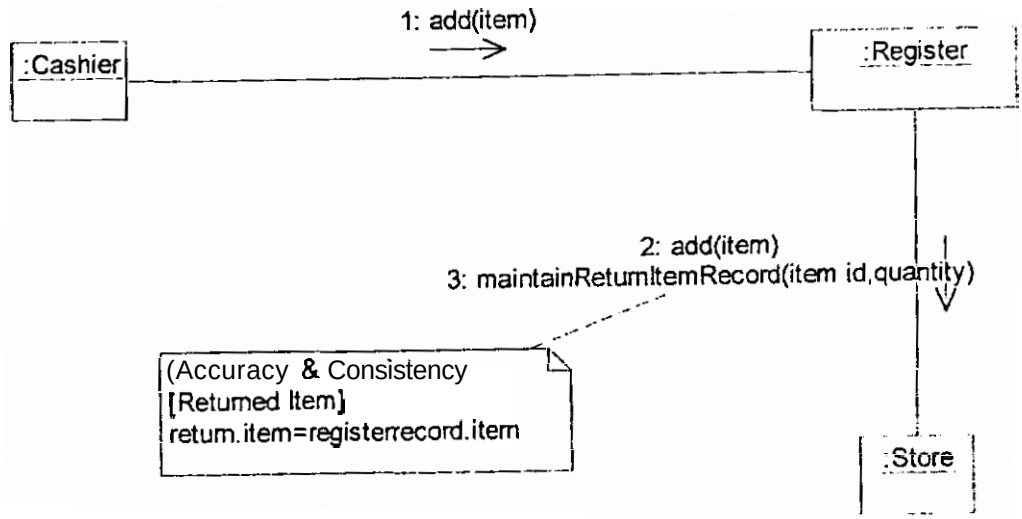


Figure 4.21b: Communication diagram for recordReturnItem after integration

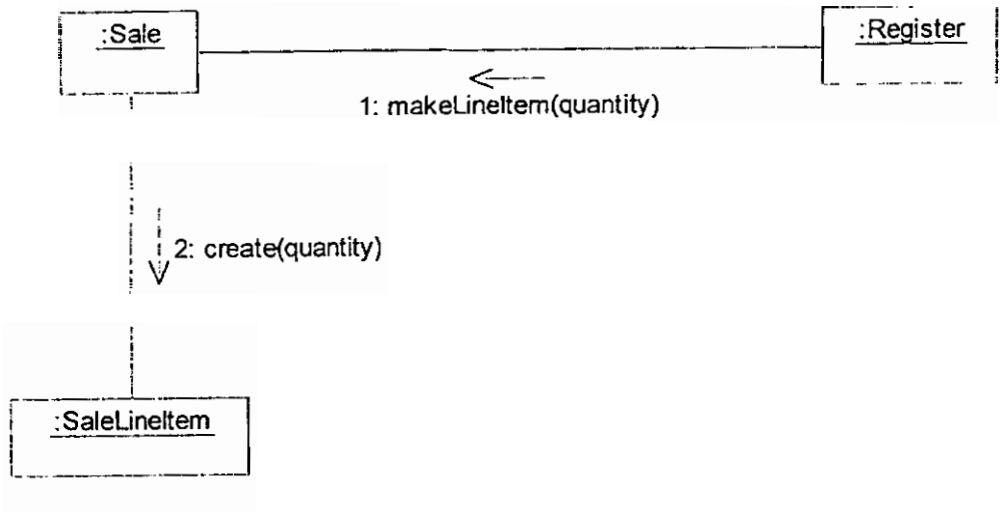


Figure 4.22a: Communication diagram for SaleLineItems before integration

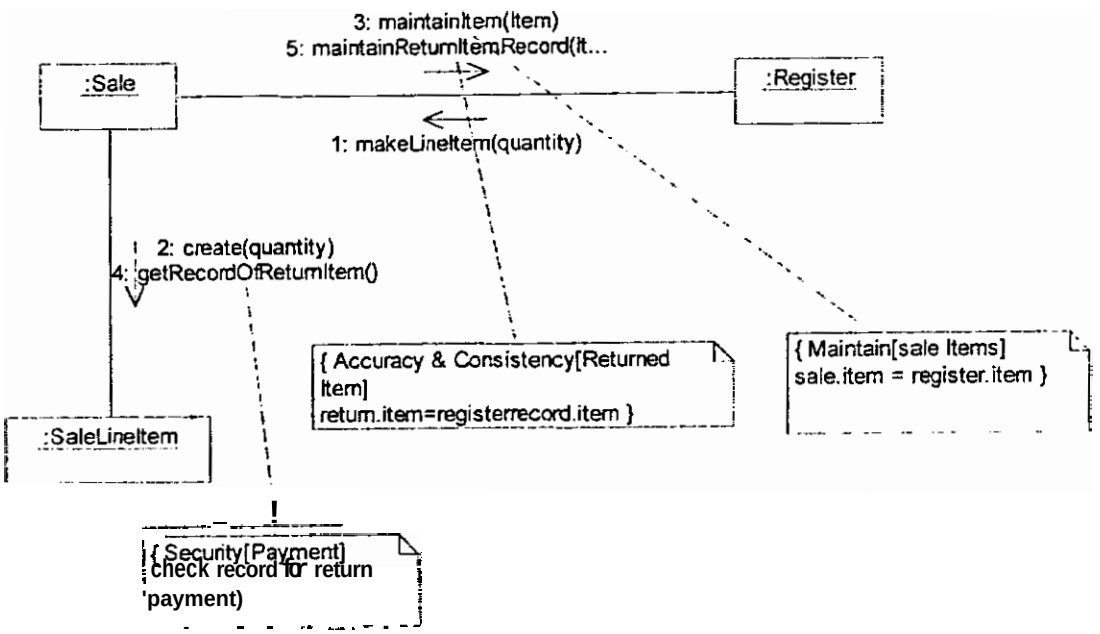


Figure 4.22b: Communication diagram for saleLineItems after integration.