A SERVICE ORIENTED ARCHITECTURE - PERFORMANCE SUPPORT SYSTEMS

A SERVICE ORIENTED ARCHITECTURE

FOR

PERFORMANCE SUPPORT SYSTEMS

By

SYED ASGHAR ALI BOKHARI

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfillment of the Requirements

for the Degree

Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2007) McMaster University

(Computing and Software) Hamilton, Ontario

TITLE: A Service Oriented Architecture for Performance Support Systems

AUTHOR: Syed Asghar Ali Bokhari

M.S.E.E. (University of Oklahoma)

MSc. (McMaster University)

SUPERVISOR: Dr. W. F. S. Poehlman

NUMBER OF PAGES: xx, 192

To My Dear Parents

Abstract

This thesis documents research encompassing the design of dynamic electronic performance support systems. Essentially, an *Electronic Performance Support System* (EPSS) is complex distributed software that provides on-the-job support in order to facilitate task performance within some particular target application domain. In view of the rapid pace of change in current business and industrial environments, the conventional practice of issuing a new release of Electronic Performance Support System (EPSS) every few years to incorporate changes, is no longer practical. An EPSS is required to adapt to the changes as soon as possible and without the need for major code modification. This is accomplished by creating a design in which task specific knowledge is not hard coded in the software but is extracted on the fly. The design also enables a loose coupling among different modules of the system so that functionalities may be added, removed, modified or extended with minimum disruption.

In this thesis we show how to combine service-oriented architecture with the concepts of software agents to achieve a software architecture that provides the required agility. Traditionally Unified Modeling Language (UML), which lacks formal semantics, has been the tool of choice for design and analysis of such systems and that means formal analysis techniques cannot be used for verification of UML models, whereas Software Engineering practices require analysis and verification at an early stage in the software development process. In this thesis we present an algorithm to transform UML state chart models to Object Coloured Petri (OCP) nets that have a strong mathematical foundation and can be implemented by standard tools such

as Design/CPN for simulation and dynamic analysis in order to verify behavioural properties of the model. We show how to apply this technique to verify some of the desirable behavioural properties of the proposed EPSS architecture. To demonstrate the feasibility of our approach we have successfully implemented a prototype of an EPSS based on the proposed design.

The main contributions of this research are: 1. Proposed an anthropomorphic architecture for a dynamic PSS. 2. Combined the concepts of services oriented architecture and software agents to achieve dynamic updating of task specific knowledge and minimal coupling between different modules of complex software to allow painless evolution. 3. Brought formal methods to the design phase in the development of agent based software systems by proposing an algorithm to transform UML state diagrams to OCP nets for dynamic analysis. 4. Modelled the dynamic creation and deletion of objects/agents using OCP net concepts and Design/CPN. 5. Proposed an architecture that can be used for creating families of agile PSS.

Acknowledgments

First of all I would like to thank my servisor Dr. W. F. S. Poehlman, who initially encouraged me to persue doctoral studies and over the past several years, has been a great source of motivation and inspiration. Without his continuous support this work would not have been possible. I am very fortunate for having him as my supervisor for this work.

I am also fortunate for having an exceptional supervisory committee and wish to thank Dr. Norman Archer, Dr. Emil Sekerinski, and Dr. Ridha Khedri for their support and assistance in completing this research. All of them have provided invaluable feedback and have played a key role in shaping this document. I would like to extend special thanks to Dr. Ridha Khedri who freely spent time to discuss and clarify ideas whenever the need arose.

Many thanks are also due to my wife Andleeb and my children for their understanding and acceptance of my continuous absence from home even on weekends for several years, to enable me to carry out my research. I could not have continued with this work without their cooperation and support.

I also thank Dr. Ryszard Janicki for his support during this research. His valuable suggestions have greatly contributed to improve the quality of this work. Thanks are also due to John Nakamura for his ongoing assistance, particularly for his help on Latex. Finally I thank everyone at the Department of Computing and Software, McMaster University, for providing an excellent environment for research.

Contents

List of Tables

List of Figures

List of Acronyms

ACL: Agent Communication Language

ADAPTS: Adaptive Diagnostics and Personalized Techincal Support

AI: Artificial Intelligence

AOSE: Agent Oriented Software Engineering

AUML: Agent Unified Modeling Language

BDI: Beliefs, Desires, Intentions

BKD: Blended Knowledge Delivery

CAL: Client Agent Local

CAR: Client Agent Remote

CBT: Computer Based Training

CCPN: Conditional Coloured Petri Nets

CO-OPN: Concurrent Object Oriented Petri Nets

CPN: Coloured Petri Nets

CRM: Customer Relations Management

CSV: Comma Separated Value

DSS: Decision Support System

EHA: Extended Hierarchical Automaton

EIS: Executive Information System

EOS: Elementry Object System

EPSS: Electronic Performance Support System

ERP: Enterprise Resource Planning

ES: Expert System

FIPA: Foundation of Intelligent and Physical Agents

GSPN: Generalized Stochastic Petri Net

GUI: Graphical User Interface

HOL: Higher Order Logic

IIOP: Internet Inter-Orb Protocol

JADE: Java Agent Development Environment

JITIR: Just In Time Information Retrieval

KIF: Knowledge Interchange Format

KM: Knowledge Management

KQML: Knowledge Query and Manipulation Language

MAS: Multi Agent System

OCPN: Object Coloured Petri Nets

OMG: Object Management Group

OPN: Object-oriented Petri Nets

OOPN: Object Oriented Petri Nets

OOPr/T: Object Oriented Predicate Transition

PCD: Performance-centered Design

PN: Petri Net

PROMELA: Process Meta Language

PSS: Performance Support System

PVS: Prototype Verification System

RMI: Remote Method Invocation

RS: Request Supervisor

SCC: Strongly Connected Component

SOA: Service Oriented Architecture

SPIN: Simple Promela Interpreter

SPN: Stochastic Petri Net

SRA: Service Registry Agent

TPN: Timed Petri Net

UA: User Agent

UML: Unified Modeling Language

VCM: Virtual Class Manager

Chapter 1

Introduction

1.1 Motivation

The Information Organization of the 21st century is very different from the Industrial Organization of the past. "Supporting workers in modern businesses has become increasingly complex in recent years" [2]. Today business processes require an employee to interact with data typically scattered "between four and twelve systems" [3] in order to perform his/her task. These systems are normally divided by functional boundaries and employ different user interfaces and different methods of representing data. Sometimes even similar terms have different meanings on different systems. The main hurdle to performance is that "in order to do the work the performer has to develop a mental model of the entire corporate system architecture, where the data reside, what screen they reside on, how to navigate to that screen, and how to integrate information that is in multiple places" [3]. The time to develop new products and deliver them to customers is very short. Customer satisfaction often requires designing products and services to meet individual customer needs. New customer-focused production paradigms that stress quality, flexibility, and new technologies are just some of the factors that have increased competency requirements of shop floor work [4]. Even the production of traditional products like cars results

in much higher pressures on the proficiency of workers due to requirements of new procedures and skills demanded by the introduction of new technology in production processes. *Electronic Performance Support System* (EPSS) is software that extracts task specific knowledge from different systems possibly distributed over a number of functional areas and provides the right information to the user at the right time in a much shorter time compared to that taken by conventional means to extract it. A thorough survey of the current use of this term is presented in Chapter 2.

The information needs of employees at different levels of an organizational hierarchy are not the same even when they are dealing with the same project. A line manager needs to monitor the day-to-day running of the business to achieve the goals set for the current financial period. His/her needs for information may be different from those of a sales/marketing executive working in the same unit. In a manufacturing environment, the information needs (relating to the same project) of the production manager, the design engineer, the production supervisor and production workers are different. If an EPSS is to meet its promise of enhancing the performance of its users then each individual should obtain the information he/she needs according to the nature of his/her duties, and this information must be as current as possible reflecting most, if not all, recent changes. During several years that the author worked as a corporate manager he frequently faced the frustration of knowing that data were available within the firm but we could not access such data in a desired integrated form, because doing so required changes in the software systems that could not be easily accomplished. Researchers and experts have strongly expressed the need for flexible, dynamic and adaptive EPSS's [5]. Can a software system be designed such that it meets the information needs of employees at different levels of an organization; where the output is not in a fixed format but can be changed according to the needs of users; where the information is not hard coded and is always current?

1.2 Challenges

1.2.1 Research Problems

Change is the only constant factor in today's business and industrial environment. Organizations are faced with the challenge of rapid change. Data that are a year out of date may not be valid in today's changed environment. Business and industry leaders previously planned five years ahead but this period has been shrinking and in some cases a year is too long a time horizon. Performance Support Systems must present a systems' view and coordinate the activities of an organization by acting as the nervous system of the organization, and must react to any positive or negative change immediately. As described by Bill Gates [6], only those corporations that have more rapid access to information will outperform the competition. The conventional method for dealing with various changes is to issue a new release of EPSS, updated to incorporate these changes. This method of discrete releases, however, cannot keep up with the rate of change in modern businesses. Often the new version is outdated before it is released. It is widely recognized that adaptability of an EPSS is severely limited due to hard-wired content and support. There is a need for "something dynamic that evolves continuously to support performance, without the need for significant developer intervention" [3].

In this thesis we investigate problems such as:

- Can we build an EPSS without hard-wired content such that the desired information is retrieved on the fly from original sources of information so that the user always receives the latest information?
- Can we modify parts of the sytem without shutting down the complete system and without the need to re-test the entire system?
- What is an appropriate architecure for such a system during the design phase?

- Can we verify some of the desired properties such as liveness, boundedness, deadlock-freeness, fairness etc for such a system?
- What is the technical feasibility of building such a system?

1.2.2 Research Objectives

- Examine the feasibility of partioning the application into discrete units of functionality and implementing each unit as a software agent.
- Use Service Oriented Architecture to achieve a loose coupling between different agents implementing various services, resulting in an agile EPSS.
- Create a conceptual model and various UML models such as a use-case model, interaction models, a logical model and state diagrams.
- Examine the possibility of transforming UML state diagrams to some type of formal model for simulation of the model and for verification of desired properties.
- Build a prototype to verify the feasibility of creating such a system.

1.3 Contributions

1.3.1 Conceptual Model

We have created an anthropomorphic design detailed in Chapter 4, in which different components of the system are given human like attributes such as *Director, Request Supervisor*, and *Agent* depending upon the responsibilities they are assigned to carry out. This human-centric approach creates a better understanding of the system and may reduce distance betwee man and machine, between design and implementation and between client and developer.

1.3.2 Adaptability

An important contribution of this research is the design of EPSS in which information is not hard-coded but is extracted from different sources distributed over an organization, on the fly. This ensures that the user always has access to the most recent information to help improve his/her performance.

1.3.3 Flexibility

Another important contribution is that domain specific functionalities may be added, removed, modified or extended with minimum disruption as the core application is partitioned into small discrete units of functionality based on related business rules or function points called "services". These services are loosely coupled via a service-oriented architecture. An autonomous software agent implements each service. The requirement of an additional service means implementing another autonomous agent that does not necessitate any change in the main system. If a service needs modification due to changed business rules or user requirement, only that service is disrupted and rest of the system remains in tact.

1.3.4 Accessibility

A user is able to access the system remotely via a desktop or a lap top computer from anywhere on the corporate intranet.

1.3.5 A Sound Architecture

In order to build a system that has the characteristics mentioned above it is necessary to have a strong architectural foundation – an architecture that has most of the desirable properties such as fairness, liveness, deadlockfreeness, boundedness and home state. Traditionally UML (Unified Modeling Language) or AUML (Agent UML) models have been used for the design and analysis of multi-agent systems [7]. However

UML is based on semi-formal semantics and formal methods for analysis cannot be directly applied to these models. Although formal methods have been frequently used in requirements specification and analysis phase their use to create formal models at the design stage is limited [8]. On the other hand current software engineering practices require dynamic analysis at an early stage in the design of a software system so that its behavioural properties may be verified. This led us to focus on the possibility of using Coloured Petri Nets (CPNs) for modeling the proposed EPSS design, in order to carry out dynamic analysis. A strong mathematical foundation, along with a graphical representation, makes Petri nets ideally suitable for dynamic analysis. One of the most significant contributions of this research is an algorithm that transforms UML state chart models for object-oriented systems to Object Coloured Petri Nets (OCPNs). This algorithm was extended to agent-oriented systems in order to verify the desirable properties of the proposed EPSS design.

1.3.6 Implementation of a Prototype

In order to validate the concept a prototype was successfully implemented using JADE (Java Agent Development Environment) as middleware. Tests were carried out to examine the performance of the system under different loads and by changing the content of the data sources, with excellent results.

1.3.7 Publications

This research has resulted in one journal paper, five conference papers and three technical reports as detailed below:

Journal Paper

"A Distributed Agile Electronic Performance Support System with Software Agents and Service Oriented Architecture from Design to Implementation", accepted for

publication in the International Transactions on System Science and Applications 2007.

Conference Papers

- "In Persuit of an Agile Performance Support System Design with SOA and Softare Agents", in the Proceedings of the IADIS International Conference e-Society 2006, Dublin, Ireland, July 13-16, 2006.
- "Translation of UML Models to Object Coloured Petri Nets with a view to Analysis", in the Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06), San Fransisco, CA, USA, July 5-7, 2006.
- "Validation of Information Systems using Petri Nets", in the Proceedings of the 8th International Conference on Enterprise Information Systems (ICEIS'2006), Paphos, Cyprus, May 23-27, 2006.
- "Performance Support System: a Knowledge Management Tool", in the Proceedings of the 26th McMaster World Congress, Hamilton, Ontario, Canada, January 19-21, 2005.
- "Design of an Agile Performance Suport System", in the Proceedings of the 42nd Annual ISPI International Performance Improvement Conference and Exposition, Tempa, Florida, April 20-23, 2004.

Technical Reports

- "Design/CPN Analysis Reports for EPSS", Tech. Rep. CAS-06-07-SP, McMaster University, Computing and Software Department, 2006.
- "Formalization of UML Statecharts: Approaches for Handling Composite States", Tech. Rep. CAS-05-07-SP, McMaster University, Computing and Software De-

partment, 2005.

- "What is a Performance Support System?", Tech. Rep. CAS-03-15-SP, McMaster University, Computing and Software Department, 2003.

1.4 Organization

This thesis is organised as follows: Chapter 2 presents an introduction to performance support systems. Here we also examine different interpretations of the term by various researchers and give our point of view. We then briefly discuss related work on performance support systems carried out by researchers in the recent past. Chapter 3 introduces the main features of service-oriented architectures and various concepts related to software agents. It also distinguishes the strong notion of agency from the weak notion of agency used in this research. We briefly examine the use of formal methods in Agent Oriented Software Engineering (AOSE) and present a rationale for using higher-level Petri Nets as a formal dynamic analysis technique. A brief introduction to UML, with explanation of terms later used in this thesis, is presented next, followed by an introduction to Object Coloured Petri Nets that we have used for modelling of EPSS.

Chapter 4 presents design considerations and gives details of a conceptual design. We then focus on creation of a UML model for the prototype. Chapter 5 examines recent research on transforming UML models to formal models amenable to dynamic analysis, and presents an algorithm to transform UML state diagrams to Object Coloured Petri Nets. It also gives examples to highlight how the algorithm can be extended from object-oriented systems to agent-oriented systems. Chapter 6 provides the simulation details of the proposed OCPN based model, using the Design/CPN tool. It also gives results of dynamic analysis carried out by building occurrence graphs. Chapter 7 contains implementation details of a prototype, and we discuss our choice of JADE as the middleware for its implementation. We ensure

that the agents developed by us are FIPA (Foundation of Intelligent and Physical Agents) compliant by using FIPA agent communication language (ACL) and FIPA agent interaction protocols with an application specific ontology called EPSS-ontology developed in this research. This chapter also discusses the relevance of agent communication languages, interaction protocols, and ontologies. Chapter 8 describes details of a methodology that can be followed to design a dynamic performance support system. We then present conclusions of this research and guidelines for future work.

Chapter 2

Performance Support Systems

2.1 Introduction

In this chapter, we briefly trace the history of Performance Support Systems and examine how different research groups have interpreted or used the term according to their research interests and backgrounds. We then present our understanding of the term and highlight the fact that the experts in the field strongly believe that a performance support system must be dynamic to support the ever-changing environments of today's business and industry, which is the focus of our current research. Finally we examine the essential characteristics of a dynamic performance support system in the light of requirements that they place on the software architecture.

A Performance Support System (PSS) or an Electronic Performance Support System (EPSS) as it is more commonly called today, is a software system that enhances the performance of its users. *"Understanding the EPSS vision remains far from common"* [5] even a decade after Gloria Gery [9] published her groundbreaking book in which she coined the new term. Towards the end of 1980's she worked with a group from AT&T to determine what tools could be added to workstations to give users precise information just in time to support their performance at work. The term "electronic performance support systems" emerged from this project to describe

integrated suites of these tools.[10] Gery [9] defines EPSS as:

"an integrated electronic environment that is available to and easily accessible by each employee and is structured to provide immediate, individualized on-line access to the full range of information, software, guidance, advice and assistance, data, images, tools, and assessment and monitoring systems to permit job performance with minimal support and intervention by others."

Gery's definition covers a wide area. Some examples of commercial EPSS packages listed by [11] are: Auxilium Performance Builder - A web-based application that is designed to increase the performance of departmental or functional groups that perform activities focused on achieving shared goals. Epiplex - a software to help large companies improve the performance of their enterprise application-enabled business processes. Step 7 Lite - is intended for use by automation specialists to engineer and program solutions to automation problems.

People have different perceptions about Performance Support Systems or even what they should be called. This largely stems from the manner by which terms such as performance, and performance improvement or performance support are interpreted as discussed in the following sections.

2.2 What is Performance and Performance Improvement?

Dictionary meanings of performance are:

n; any recognized accomplishment

n; the act of performing; of doing something successfully; using knowledge as distinguished from merely possessing it

The following definitions of Performance Improvement were taken from the John Hopkins University's Web site:[12]

1. Performance Improvement is a process for achieving desired institutional and individual results.

2. Performance Improvement (PI) is a process for enhancing employee and organizational performance that employs an explicit set of methods and strategies. Results are achieved through a systematic process that considers the institutional context, describes desired performance, identifies gaps between desired and actual performance, identifies root causes, selects, designs and implements interventions to fix the root causes, and measures changes in performance. PI is a continuously evolving process that uses the results of monitoring and feedback to determine whether progress has been made and to plan and implement additional appropriate changes.

It is obvious that Performance can be interpreted in many different ways. It can be:

1. A measurable output of an employee's work in terms of number of units produced per unit of time

2. Time taken to complete a job.

3. Accuracy with which a job is performed.

4. Efficiency in handling an assignment.

5. Making the appropriate decision under given circumstances.

2.3 Interpretations of the Term EPSS

The term Performance Support System has been used for a wide variety of different systems, for example:

On-line help:

- With ease of software use (Macros etc),

- With how to use software,
- With finding information.

Manipulation of information in order to learn:

The focus in these applications is on the reduction of training time, reduction of costs associated with training and improvement of training quality. They include:

- Video, audio, image and text representation of information;
- Use of hypermedia and multimedia for presentation of information. For example, to explain assemblies and maintenance of complex mechanical systems [13];
- On-line technical documentation; for example, on line manuals to support manufacturing processes;
- Some enterprise software is highly customized (e.g. Enterprise Resource Planning (ERP) Systems and Customer Relations Management (CRM) systems). Therefore the generic software from a vendor of such systems may not be useful and the use of performance support systems has been suggested in such cases [14];
- On-line reference material. For example, a performance support system that provides tutorial and reference for bodily injury claim investigators [15];
- Learning systems. For example, flashcard software for product knowledge memorization [16].

Manipulation of information for performance support:

- Systems that address configuration problems for companies that sell multiple products/services [17];
- Systems for fault diagnosis and support on the shop floor [4];

- Systems for Management Decision Support [18];

The merging of traditional Expert Systems (ES) with EPSS was a *natural* fit. Carr [19] brought knowledge into the equation for EPSS. These systems offer a fertile field for knowledge-based technology. According to him:

"A PSS is a computer-based system that uses knowledge-based systems, hypertext, on-line reference, extensive data bases, and allied technologies to provide support to performers on the job, where they need it, when they need it, in the form most useful to them."

He argued that because of the emphasis that Performance Support Systems put on performance improvement, they eliminate the feeling of threat and strangeness associated with expert systems, as improving on the job performance is important to everybody. PSS's are more tightly wired to the needs of the performance situation and they perform in a much broader variety of roles. PSS's, therefore, he predicted will be highly successful and widely accepted. There has been much progress in both the research and the applications of EPSS during the past few years. Gery also shares these ideas as she says, *"When designers have the point of view of the performer situated in a real work context, success is inevitable"* [10]. AI (Artificial Intelligence) is an essential characteristic of an EPSS according to Carr, but not according to Gery. Flexibility is another important feature of Performance Support Systems. Carr suggests that a PSS can assume one of the following four basic roles:

- It can act as a librarian. In this role, it helps the performer find, organize, and interpret the information required to carry out a task.

- It can function as an advisor. It embodies and shares some specialized expertise that the performer needs to carry out the task.

- It can be an instructor. In this role, it trains the performer in some aspect of the work to be done. Just as the advisor role is closest to that of an expert system, the instructor role is an outgrowth of Computer Based Training (CBT).

- It can serve as a doer. In this role, it does the work with or without assistance from the human performer.

  Advocates of EPSS can generally be divided into three groups:

  a) those arguing in favour of training and learning as a means for performance improvement;

  b) those who believe the EPSS is to support performance when required to do so (just-in-time approach); and

  c) those who want to see everything combined in one platform.

There has been an interesting debate on whether this performance improvement can be achieved by providing just enough information, just in time to handle a situation or by additional training about a new situation or both. Rosenberg [20] is of the view that *"performance rather than learning is the direct goal."* Raybould [21] suggests that: *"The purpose of the EPSS is to provide just-in-time performance support at the moment of need. The support could be in the form of procedural assistance or granular training."* Kevin Cloe says, *"The focus is not knowing, it is performing"* [22].

This thinking led to the building of systems that enable employees to complete an assignment without prior training in which *"Task-specific templates and job aids assist users in performing to an organization's 'best practices' standards. Wizards and cue cards walk users through an approved procedure"* [23] This also included user assistance for a particular software product. Performance in this case is the ability to use the product efficiently and with minimal training.

Equally interesting is the discussion about what can be called an EPSS and what should not be categorized as such. According to Debrough [24] an EPSS must provide specific information related to doing a domain specific job when it is needed, on demand by the user. Otherwise it will be no different from traditional training that provides information but not when it is needed and that is not specific and may include irrelevant data. Also the trainee cannot choose what he/she needs. Tables 2.1

and 2.2, reproduced from [25], highlight the difference between performance support systems, help systems and Computer based training (CBT).

Table 2.1: EPSS vs Help Systems

EPSS	Help Systems
Support a broad range of job tasks.	Support software-related tasks.
Provide "what if" type of advice.	Provide passive information only.
Can support complex, interrelated tasks.	Provide descriptions of procedures.
Accept and process user input.	Present information as a result of menu options.

Table 2.2: EPSS vs CBT

EPSS	CBT
Hypertext environment with multiple access.	Infobase structured within predetermined presentation sequences.
Contains a range of support mechanisms to assist user in performing a task.	Outlines procedures to be followed by the user. Does not actually help with execution.
Accepts and manipulates user input.	Checks user input against model answer or calculated answer.
Available on demand and in context.	Forms an event in a larger teaching process. Available when its turn comes up.
Emphasis on user construction of individualized learning sequence, based on need.	Structured according to design of the developer.
Granular	Modular

Raybould [26] felt that the previous definitions of EPSS were too restrictive and resulted in *"misconceptions of EPSS as intelligent job aids or as the cue card, coach or wizard structures."* He advocated a *systems-thinking* approach and redefined EPSS as follows:

"An EPSS is the electronic infrastructure that captures, stores and distributes individual and corporate knowledge assets throughout an organization, to enable an individual to achieve a required level of performance in the fastest possible time and with the minimum of support from other people."

Raybould advocated that an EPSS is distinct from traditional systems development as well as from expert systems development in that the former focuses on data, not on knowledge and the latter focuses on knowledge but not on enabling performance. He brought individual and corporate knowledge assets in the realm of EPSS. His methodology suggests extracting, organizing, and storing knowledge and making it available to employees at their time of need.

Today the corporate memory is subject to so many and so frequent changes that extraction of knowledge is required on an on-going basis. Recently a number of other researchers have also advocated the merging of EPSS and KM fields [3, 27, 28], although there is a natural tilt towards training/learning or performance, depending upon the background of the researchers.

2.4 Our Understanding of the Term

We agree with Jim Elsenheimer [28] that the information overload that corporations in the 21st century are facing, has made it paramount to make all the information available in corporate memory meaningful in terms of the performance we want to achieve. If EPSS provides this information, it should distill information into usable chunks. He advocates a training approach and recommends extraction of tacit as well as explicit knowledge, and making it available to users. His tapping of tacit knowledge and the approach for doing so, sound similar to that advocated by proponents of expert systems and in our opinion suffers from the same difficulties and disadvantages.

Based on an Artificial Intelligence (AI) and Expert Systems (ES) background, we developed a performance support system for the secondary water system of a nuclear power station [29, 30, 31] during the early 1990's. Our understanding of the term matches the explanation of Bill Miller [32]. He believed that an EPSS should support the performance of an employee on the job by enabling him/her to perform a task in less time, with fewer errors, with better results and with less training or

external support. We are of the view that: A performance support system is a suite of software tools that an employee can use to perform his/her job effectively, efficiently, and accurately within a time that enhances the employee's performance. It caters to the needs of an individual user by presenting the perspective of information that is just sufficient and just in time for the performance of that particular user, on demand by the user.

A PSS extracts relevant knowledge from information that may be distributed over a number of functional areas and provides it to the user in a much shorter time, compared to that taken by conventional means to extract it. This is analogous to reviewing each piece of information separately and connecting the relevant information in a human brain. The focus in this research is on user performance rather than learning or training. The user must have prior training to handle the kind of job that he/she is trying to do and should be able to do it without a PSS. In such a case, that will require much time and cognitive effort, and the results may not be satisfactory. The PSS is there to support performance, and not to provide performance. In this context, the PSS can be thought of as suite of productivity tools. On request from a user, it can also find information that the user may provide to others as part of his/her job (e.g., providing order status to a client). Also it should be user tunable, so that each user may arrange the toolbox in a manner that best meets his/her performance needs. Therefore the PSS must present a perspective that is relevant to the user's job.

The Blended Knowledge Delivery model (BKD) of Andy Zolper [27] provides a view that helps clarify the overlapping domains of KM (Knowledge Management), EPSS, HPS (Human Performance Support) and IK (Internalized Knowledge). In his effort to segregate KM and EPSS, however, he limited EPSS to a system whose content is organized in advance. Although traditionally this has been the case, we strongly believe that in order to meet the ever changing environment of business and industry in the 21st century, KM and EPSS must constructively cooperate if

the goal is to provide a solution in which the vast innate abilities of employees are complemented by our efforts, not complicated or confused. We, therefore, would modify his definition of EPSS to:

The practice of searching and extracting task specific knowledge from disparate knowledge sources distributed over the organization, and delivering it to employees when they need it, where they need it, and in the form that will best support their performance.

With this definition EPSS can include information that supports the performance of a worker, a supervisor, a business executive, or a manager and can overlap with Executive Information System (EIS) and Decision Support System (DSS) technologies. Such a system will *"extend cognitive ability by abstracting the procedure or task from irrelevant details"*, and will *"extend memory by relieving people of the necessity of remembering details, or even of the necessity of learning."* [33]

2.5 Dynamic / Adaptive Systems

Supporting workers in modern businesses has become increasingly complex in recent years [2]. Researchers and experts have strongly expressed the need for dynamic or adaptive EPSS's [5]. The conventional method for dealing with various changes in business processes is to issue a new/updated release of EPSS incorporating the changes. This method of discrete releases, however, cannot keep up with the rate of change in modern businesses. Often the new version is outdated before it is released. It is widely recognized that adaptability of EPSS is severely limited due to hard-wired content and support. There is a need for *"something dynamic that evolves continuously to support performance, without the need for significant developer intervention."* [3]

Another problem highlighted by Gery [3] is the fact that today's business processes require an employee to interact with data scattered "between four and

twelve systems" in order to perform his/her task. These systems are normally divided by functional boundaries and use different user interfaces and different methods of representing data. Sometimes similar terms have different meanings on these systems. The main hurdle to performance is that

"in order to do the work the performer has to develop a mental model of the entire corporate system architecture, where the data resides, what screen it resides on, how to navigate to that screen, and how to integrate information that is in multiple places."[3]

An EPSS can greatly support the performance of a worker by transparently presenting a view of the relevant latest data that he/she needs. We used an anthropomorphic model and the concept of agents in OPUS [29] and we strongly agree with Gery that the new agent technology is a fundamental aspect of dynamic PSS.

2.6 Essential Characteristics of EPSS

Debrough [34] examined different characteristics of EPSS and Gery [10] listed 19 characteristics of a performance-centered design (PCD). We examine the main features of EPSS in the light of requirements that they place on software architecture.

Graphical User Interface (GUI): The user interface should follow the principles of PCD elaborated by Gery [10]. Recognizing that the performance needs of different users are not the same, customizability is the key design feature for the GUI so that the system presents the information according to the mental model of the user.

Flexibility: The system should have minimum coupling between the modules that extract chunks of relevant knowledge and the sources of knowledge to facilitate modification to a part of the system without the need to re-test the entire system.

Adaptability: It should be possible to easily adapt the system to meet particular information needs of a user. It should also be possible to update content without code level changes.

Response: In order that the user obtains the support he/she needs just in time, the system should respond within a time frame that enhances performance. This means mechanisms should be in place to avoid deadlock due to different processes competing for the same resource.

Transparency: The user should not be required to know where the information is located and whether it is retrieved sequentially by a single process or concurrently by a number of processes. He/She also does not need to know if others are using the system.

Availability: As the user needs the system just in time to perform a task it should be available when required. It should be fault tolerant so that in case of breakage of communication, messages are not lost. It should also provide alternatives to guard against the possibility of non-availability of a machine in the system.

Solicited information: The system should provide only relevant information, discrete and to the point - just enough information to carry out the task.

System Performance: Performance is determined by a number of factors, for example: Response time – how quickly the system responds to user interaction. Throughput – number of jobs handled per unit of time, and Network Capacity used.

Incremental Growth: It should be possible to add new functionality easily.

Security: As the system provides access to corporate information, safeguards should be built in to prevent unauthorized uses.

2.7 Recent Research

Sen [35] studied the distance learning paradigms involving both synchronous (such as online distance lectures over the Web) and asynchronous (such as independent learning from web materials) techniques and noted the need for distance learners to access information on distributed machines in different formats. He also noted that the presentation of the same information to different users may be different, depending upon the level of understanding of the individuals using the information. Sen implemented a Virtual Class Manager (VCM) that provides an asynchronous, distributed, open information access and management environment to particular users, including supervisors and learners. In order to make the system flexible and adaptable, he proposed a multi-tier architecture in which the front end is the client and the back end is the database. Note that his focus was on learning whereas our focus is on performance.

Brusilovsky et al. [36] explored some possible scenarios for using adaptive hypermedia for adaptive performance support systems, in the context of the Adaptive Diagnostics and Personalized Technical Support (ADAPTS) project that provides an intelligent, adaptive electronic performance support system for maintaining complex equipment. ADAPTS adjusts the diagnostic strategy to the identity of the technician and what is being done, dynamically, adapting the sequence of setups, tests, and repair/replace procedures based on the technician's responses. The technician receives dynamically selected technical support information appropriate for the contexts of the setup, test, and repair/replace procedure being performed. The focus of this approach is on performance improvement similar to ours. However the system adapts to the information currently coded into the software but does not dynamically handle any changes to the information content. For example, any change in diagnostic procedure or use of a new kind of sub-system will result in issuing a new release of the software. They do not discuss possibilities of formal verification of the properties of their EPSS.

Banerji [37] has identified the need for just in time support to improve the perfor-

mance of knowledge workers and has proposed a four-layer architecture for electronic performance support systems. The system he proposed is limited to providing on-the-job support to facilitate task performance within some particular target application domain. It does not dynamically handle change in information content and is not adaptable. It is integrated software for use on a stand-alone system.

Rhodes [38] proposes a class of just in time information retrieval agents named JITIRs that can, based on the local context of a person, extract and present useful information. JITIRs continuously watch a user's environment, and present information that may be useful without any explicit action on the part of the user. The information a JITIR provides can come from any number of pre-indexed databases of documents (e.g. email archives, notes files, or documents from commercial databases such as the INSPEC collection of technical paper abstracts). However, JITIRs are software agents only, in the sense that they do not represent distributed agent architectures or agent oriented programming models.

2.8 Summary

This research explores the design of a dynamic EPSS. In this chapter we described what an EPSS is and discussed different interpretations of the term by various researchers. We also gave our interpretation of the term and listed some of the essential characteristics that an EPSS must have. At the end we briefly discussed some of the recent work on EPSSs by other researchers.

In order to fulfil the need for a dynamic and adaptive EPSS as pointed out in section 2.5 of this chapter, we propose that the concepts of software agents and service oriented architecture be combined for application towards the design of that EPSS. The next chapter presents a brief introduction to some of these concepts and examines different modeling possibilities for analysis of an agent based software system.

Chapter 3

Service-Oriented Architecture and Software Agents

3.1 Service Oriented Architecture

Service-oriented architecture (SOA) is a sofware architectural concept where applications are partitioned into discrete units of functionality called "services". Each service implements a small set of related business rules or function points. These services are made available to consumers/client applications. Whenever a business rule must be modified to support changing business requirements, only the service which implements that business rule needs modification, while the remainder of the application remains intact. The most importannt aspect of SOA is that it provides a loose coupling between different services that are composed into an application by the application developers or system integraters, without needing to know their implementation details.

As shown in figure 3.1 the basic building blocks of SOA are:

Service Providers: Processes that can provide specific services.

Service Clients: Processes that need a particular service.

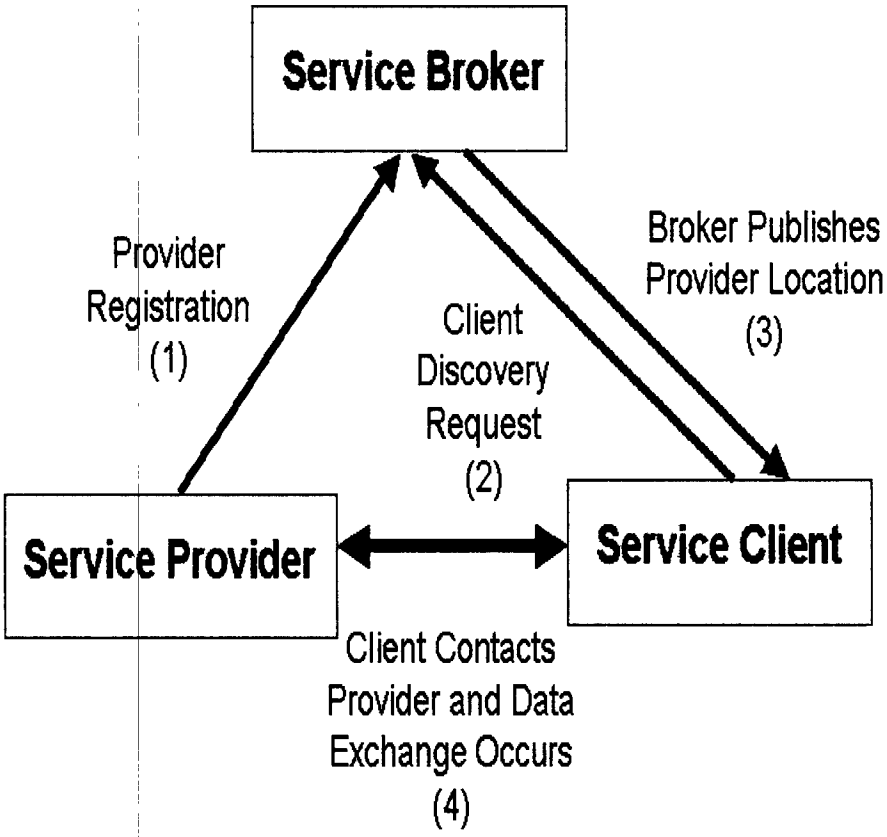

Figure 3.1: Basic Building Blocks of Service-Oriented Architecture.

Service Brokers: Processes responsible for registering and categorizing published services and providing search facilities to clients for locating the desired services.

SOA, like any other architecture, is not tied to a specific technology. A wide range of technologies such as CORBA, RPC, DECOM, RMI or Web Services can be used for its implementation. According to [39] a service is a behavior that is provided by a component for use by any other component based only on the interface contract. A service has a network-addressable interface. A service stresses interoperability and may be dynamically discovered and used. The services have the ability

to be invoked over a network and this can be accomplished in many different ways. The technologies used to invoke the interface of the services stress interoperability. Location transparency is also stressed so that services may be discovered and used dynamically.

Although SOA does not need Web Services for its implementation, their use is so common that often SOA is assumed to be equivalent to Web Services [40]. WebServices combine four technologies for implementing SOA i. e. HTTP/HTPPS as the primary network protocol, SOAP/XML for the payload format, UDDI(Universal Description and Discovery Interface) for service registry, and WSDL(Web Services Description Language) to describe the service interface.

Web Services allow packaging of applications using hetrogenous services owned by different organizations and implemented on different platforms in different languages. Such applications can only work if they are properly coordinated such that the sender and receiver of a message know, and agree in advance, the format and structure of the (SOAP) messages that are exchanged, and the sequence and conditions in which the messages are exchanged [41].

The first objective is realized by WSDL and its extensions, however it does not define the sequence and conditions in which messages are exchanged. This is accomplished by WS-CDL(Web Services Choreography Language) that produces a shared common or "global" definition of the sequence and conditions in which messages are exchanged. This describes the observable complementry behaviour of all the participants involved. Each participant can then use the definition to build and test solutions that conform to the global definition. Choreography is typically associated with the public message exchanges that occur between multiple web services, rather than a specific business process that is executed by a single party [42].

Generally, corporate entities are not willing to delegate control of their business processes to their integration partners. Orchestration, also called executable process in BPEL(Business Process Execution Language), deals with the description of the

interactions in which a given service can engage with other services, as well as the internal steps between these interactions. Orchestration always represents control from the perspective of one party. This differs from choreography, which is more collaborative and allows each involved party to describe its part in the interaction [42]. Much research is currently underway to explore different aspects of choreography, archestration and other related issues [43, 44, 45].

It is possible represent the global flow of messages specified by interaction protocols using WS-CDL by global calculus and the end point behaviour by a typed π calculus. Both calculi are based on a common notion of structured communication, called a session. A session binds a series of communications between two parties into one, distinguishing them from communications belonging to other sessions. Using such formalization, a UML interaction diagram representing interaction between Web Services can be formally analysed for verification of protocols [46].

3.2 Software Agents

A software agent is an autonomous program that operates on behalf of another entity. It has the ability to communicate, interact, and collaborate with other entities and may have the ability to learn, reason, adapt, and take initiative in pursuing goals. A mobile agent has the additional capability of being able to migrate under its own control within heterogeneous networks. Although the agent paradigm came into existence as a result of research within the AI community, both the AI community and computer science researchers have studied agents and agent systems for a number of years from their own different perspectives. Agents considered by the AI community have the following properties:

- autonomous
- proactive

- may be mobile (i.e. it may have the ability to move around an electronic network)
- can communicate with other agents including humans, and
- are most often intelligent; i.e. they have knowledge, and they can learn and reason with their own knowledge to perform complex tasks.

From a computer science systems point of view, agents are running code with data and state [47]. They are autonomous, proactive, interacting and may be mobile. Their functionality can be described in terms of human behaviours [48] such as beliefs, desires and intentions.

3.2.1 Relationship between Agents and Objects

There are a number of obvious similarities between agents and objects but there are significant differences also. A detailed discussion on this topic appears in [49] and a summary is reproduced below:

- Agents embody a stronger notion of autonomy than objects, and, in particular, they decide for themselves whether or not to perform an action on request from another agent;
- Agents are capable of flexible (reactive, pro-active, social) behavior, and the standard object model has nothing to say about such types of behavior;
- A multi-agent system is inherently multi-threaded, in that each agent is assumed to have at least one thread of control.

3.2.2 Verification and Validation Issues

The use of formal methods has recently been one of the most active areas of agent oriented software engineering research. Efforts have been made to apply formal methods to specify systems, for directly programming the systems, and for verification of

the systems, with only limited success [49].

It is widely recognized in software engineering community that interaction is one of the most complex characteristics of software systems. A system that changes its actions, outputs and conditions/status in response to stimuli from within or outside, is called a reactive system. The work carried out by Manna, Pnueli and colleagues [50] for specification and verification of reactive systems using temporal logic appears to be most relevant to multi-agent systems. Their work is based on the notion that the computations of reactive systems can be considered as infinite sequences which correspond to models for linear temporal logic. Temporal logic can be used both to specify systems, and to axiomatize programming languages. This axiomatization can then be used to systematically derive an abstract model of a program from the program text. Both the specification and the program model will then be encoded in temporal logic, and verification therefore becomes a proof problem in temporal logic. Comparatively little work has been carried out within the agent-based systems community on axiomatizing multiagent environments [49]. For agent systems, which fall into the category of Pnuelian reactive systems, refinement from abstract models to concrete implemetation is not so straightforward. This is because such systems must be specified in terms of their ongoing behavior - they cannot be specified simply in terms of pre- and post-conditions. In contrast to pre- and post-condition formalisms, it is not easy to determine what program structures are required to realize such specifications. As a consequence, researchers have only just begun to investigate refinement and design techniques for agent-based systems [49].

3.2.3 Formal Methods in Specification

Most of the research by the AI community is focused on the problem of conceptualizing agents i. e. formal representation of their properties, and reasoning about these properties. Attitudes that have been considered for representing agents can be divided into two important categories: information attitudes like beliefs and knowledge, and

pro-attitudes like desire, intention, and choice, etc. [51]. Information attributes represent an agent's knowledge or beliefs about the world, including itself and other agents. The pro-attitudes guide an agent's actions in a certain way. For example, "desire" represents objectives that the agent would like to accomplish, the "intentions" represent what the agent has chosen to do in order to achieve some of the objectives. Although it is not clear which combination of attitudes is most suitable to represent an agent, the most common representation involves at least one information attitude and one pro-attitude.

The most common approach for specifying agents is to treat them as intentional systems that have mental states like beliefs, desires and intentions. The problem is that rules of first order logic do not apply to intentional notions because they are referentially opaque. They set up opaque contexts, in which the standard substitution rules of first order logic do not apply as illustrated in the following example:

Let E and F be expressions and x be a variable. The notation $E(x := F)$ represents an expression that is the same as E except that all occurrences of variable x are replaced by F. According to Leibniz, for the two expressions X and Y:

$$\frac{X = Y}{E(x := X) = E(x := Y)}$$

However this axiom does not apply to intentional notions as shown in the following example from [51].

The staement: *Janine believes Cronos is the father of Zeus*, may be translated into first order logic as:

$$Bel(Janine, Father(Zeus, Cronos))$$

The constants *Zeus* and *Jupiter* by any reasonable interpretation, denote the same person: the supreme deity of classical world. Therefore, in first order logic we may write:

$$(Zeus = Jupitor)$$

The application of Leibniz's axiom will result in the following derivation:

$$Bel(Janine, Father(Jupitor, Cronos))$$

which is intutively invalid as believing that the father of *Zeus* is *Cronos* is not the same as believing that the father of *Jupitor* is *Cronos*.

As the classical logics are not suitable in their standard form for reasoning about intentional notions, alternative formalisms are required. Two basic approaches can be used to take care of problems in representing intentional notions by classical logic: the use of modal logic that contains non-functional modal operators, and the use of a meta-language. On the semantic front, the most common approach is the possible worlds semantics originally proposed by Hintikka [52] and now most commonly formulated in a normal modal logic using the technique developed by Kripke [51]. According to Hintikka, an agent's beliefs can be characterized as a set of possible worlds. This approach suffers from two problems: Rules developed using this notion require that an agent knows all valid formulae. This means, in addition to other things, it knows all prepositional tautologies, which is counter intuitive. The rules also require knowledge/belief be closed under logical consequence. These two problems together are referred to as the logical omniscience problem and make the possible worlds model unsuitable for resource bound agents. Some researchers have suggested modifications to the possible worlds model, for example by making a distinction between explicit and implicit beliefs [53]; however, others [54] have criticized such modifications. A number of meta-language formalisms have been proposed [55, 56]; however, meta-language formalisms have been criticized due to inconsistencies [51].

The above paragraphs provide a brief account of formalisms attempted to conceptualize an agent. However, a complete theory of agency must show how an agent's information and pro-attitudes are related, how the time and environment affect the cognitive state of an agent, and how an agent decides what action to take. A number of attempts have been made using different combinations of information and

pro-attitudes: for example Cohen and Levesque [57] used beliefs and goals whereas Rao and Georgeff [58] used beliefs, desires and intentions resulting in well known BDI agents that have been used by other researchers in their attempts to implement agents and multi-agent systems with varying success. Formalism has also been used to represent communication between agents based on speech act theory and has resulted in two related languages: the knowledge query and manipulation language (KQML) and the knowledge interchange format (KIF). Foundation of intelligent and physical agents (FIPA) has also developed an agent communication language (ACL).

3.2.4 From Theory to Implementation

Efforts have been made to construct computer systems satisfying properties specified by agent theories. The two main problems identified in [51] are: 1) that of translating the real world into an accurate, adequate symbolic description, in time for that description to be useful, and 2) that of how to symbolically represent information about complex real world entities and processes and how to get agents to reason with this information in time for the results to be useful. Despite much research, these two problems remain unsolved.

3.2.5 Alternative Approaches

There are so many unsolved problems related to representation of agents with symbolic AI that some researchers became disenchanted and questioned the viability of the whole paradigm. Most noted among them was Rodney Brooks [51] who argued that:

- Intelligent behaviour can be generated without explicit representation and explicit reasoning of the kind that symbolic AI proposes.
- Intelligence is an emergent property of certain complex systems.

- Intelligent behaviour arises as a result of an agent's interaction with its environment.

Brooks built a number of robots based on his so-called subsumption architecture, which is a hierarchy of task accomplishing behaviours. A major criticism of most of the formal or theoretical work is that while it is important and contributes to a solid underlying foundation for practical systems, it is not clear how to use this work to develop complete systems (no direction is provided as to how it may be used in the development of these systems [59]). Although several powerful formalisms exist, finding the right formalism is a nontrivial challenge as noted by Singh [47].

An agent from a CS perspective is often considered to be just a process [48], a piece of running code with data and state. The functionality of these agents can most often be described in terms of human behaviour. Agents are processes that are autonomous and pro-active (capable of making "their own" decisions when they like, within the limit of their design), interacting, and they may be mobile.

In short there is no agreed definition of what an agent is. The existing agent theories are divided broadly into two categories; the AI community normally follows the strong notion of agency whereas computer science researchers follow a weak notion of agency. The first approach considers an agent as a white box while the second considers it as a black box and defines an agent only in terms of observable properties such as autonomy, reactivity, pro-activity and social ability [60]. We follow the weak notion of agency in this research.

3.3 Unified Modeling Language (UML)

In general, most of the implemented multi-agent systems are based on principled but informal design methodologies; predominant among them is the use of Unified Modeling Language (UML) suitably adapted for agent-oriented design. The advantage of using UML is its familiarity to users of object oriented analysis and design. The

importance of dynamic analysis based on some formal model with powerful analysis techniques, especially at early stages of software development, has been steadily increasing in the recent past. UML does not provide a dynamic model to evaluate the system at an early stage. It lacks formal semantics and hence it is not possible to apply, directly, mathematical techniques on UML models for system validation [61]. In this research we propose to transform UML state diagrams to Object Coloured Petri Nets (OCPNs) and use the resulting Petri Net model for dynamic analysis to verify certain behavioural properties of the system. Details of the algorithm developed are given in Chapter 5. However, some of the terms relating to UML state diagrams that are used in the algorithm are described below:

3.3.1 Simple States

A simple state represents one of the finite number of abstract states in which the object modeled by the state diagram may find itself. It is a state of the object during which it satisfies some conditions, performs actions and waits for events. As shown in figure 3.2, such a state is represented by a rounded rectangle, usually has a name, and may have an entry action, a do-activity and an exit action.

3.3.2 Pseudo States

A state diagram starts with a pseudo initial state shown by a small solid circle. The solid circle is in fact marking the initial state, and that is why it is a pseudo state. A bull's eye circle represents the final pseudo state. A state diagram must have the initial pseudo state although the final pseudo state is optional. There could be other pseudo states such as history states.

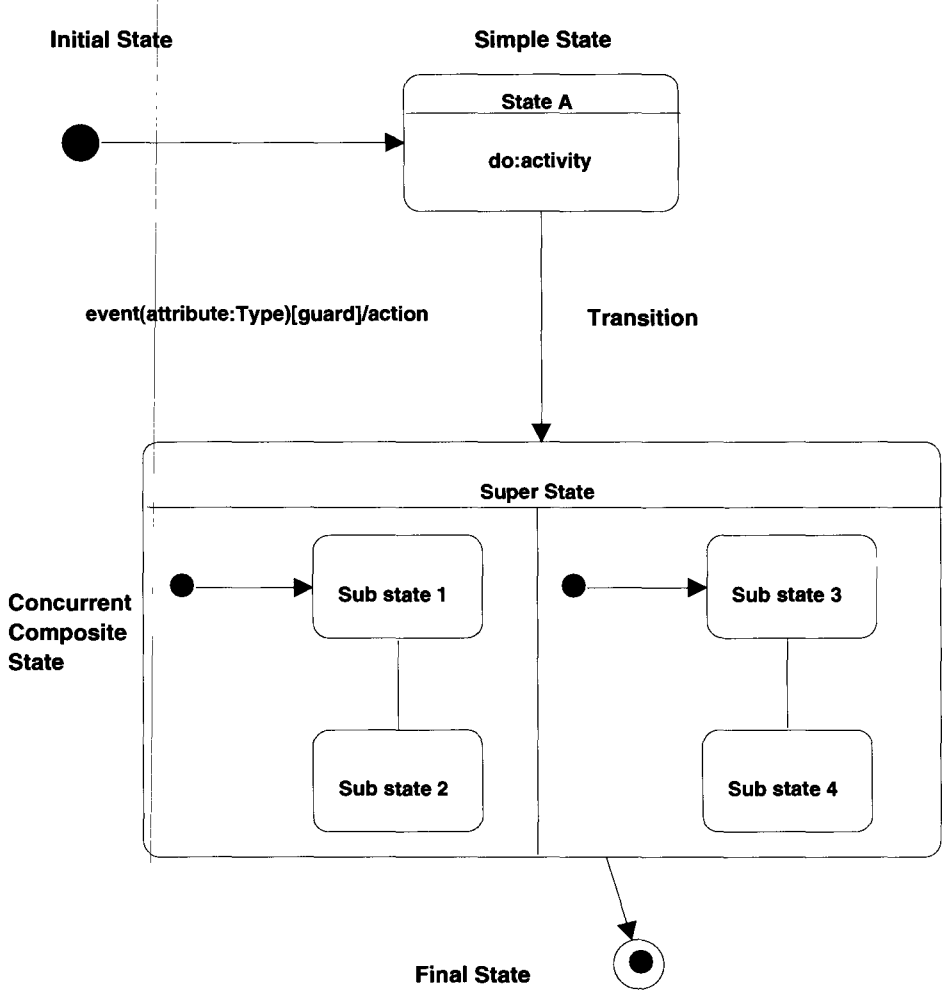

Figure 3.2: Different Types of States

3.3.3 Composite States

A composite state is composed of more than one sequential or concurrent sub-state. It is a convenient grouping of states and must be "flattened" to a set of simple states before a state diagram can be transformed to a Petri net. An algorithm to do so can be found in [62, 63]. In this research we assume that any composite state in the state diagram has been flattened.

3.3.4 Transitions

A transition represents an allowed change from a source state to a target state. Transitions from one state to another are represented by a directed edge. A transition may have an event, a guard, and an action associated with it. An event is the cause of a transition and is sometimes called a trigger. A guard is a Boolean expression, presented in square brackets, that prevents a transition from being taken unless the condition evaluates to true. An action is a function that represents the effect of a transition and is invoked on the object that owns the state machine as a result of the transition. The event, the guard and the action are shown as a label on the arrow representing the transition in the format: event[guard]/action. A state can source many different transitions and can be the target of many transitions. Instead of going to a different state, a transition may have the same source and target state. Such transitions represent situations where a message is received but does not result in a change of state. These transitions are called self-transitions. Transitions without an associated event are called triggerless transitions. An action resulting from a transition or a state entry/exit action can result in sending a message to another object. In such a case, the action is preceded by ^, e. g. ^ Target.Action. Communication between objects can be synchronous (blocking) or asynchronous (non-blocking). We are mainly interested in distributed systems and discuss asynchronous message passing only although the model allows synchronous communication also. An asynchronous call does not block the calling process and does not require an acknowledgement by a return value. The acknowledgement, if necessary, can be sent as another asynchronous message.

3.4 Modeling by Petri Nets

If the weak notion of agency is accepted and hence agent behaviour can be described by appropriate programs, then the experience of applying formal methods gained

in the theory of modeling and analysis of distributed systems can be used for the formal description and analysis of multi-agent systems [64]. Petri nets (PN) provide an excellent formal modeling tool for this purpose. Such a model gives both static as well as dynamic representations of systems. Different higher level Petri nets have been used for this purpose. E.g. CPN (Coloured Petri Nets) [65, 66], TPN (Timed Petri Nets) [67], SPN (Stochastic Petri Nets) [68], GSPN (Generalized Stochastic Petri Nets) [69, 70], Object Petri nets [71], and G-Nets [72] etc.

3.4.1 Object Oriented Petri Nets

The integration of object-oriented concepts with Petri Nets has drawn the attention of many researchers and a number of approaches such as OBJSA-nets [73], CO-OPN [74], OOPN [71], EOS [75] and OPN [76] have been proposed. Most of these approaches with the exception of the last two assume a fixed number of objects at the design stage. This approach presents a major hurdle in using the technique to develop software for complex systems, where objects need to be instantiated and deleted multiple times during the run time of the system and the number of objects of a particular type should not be constrained as a constant at the design stage [77]. Although all of these approaches result in Petri Nets that are behaviorally equivalent to CPN and other lower level Petri Nets, special purpose analysis tools for these nets are available at different stages of implementation [78].

3.4.2 Modeling a Multi-agent System using Petri Nets

Although Petri nets have been extensively used for modeling distributed systems, some researchers have only recently modeled agents and simple agent systems using Petri nets. It is possible to use a PN to model a single agent and the same PN can be used to model a complete multi-agent system, depending upon the semantics that describe the net behaviour (for example, by associating an agent with each transition). Another approach is to model different agents by separate Petri nets

and then the problem of combining these nets can be handled by following one of the several approaches in Petri net theory (e.g. place merging, transition merging or using hierarchical nets). It is not clear, in these approaches, how to accommodate agents that are spawned for a particular task and then terminate themselves after completion of a task assigned to them. One alternative is to represent an agent by a Petri net and then use this net as a token for the net representing the complete system. Another possibility is to represent the agent as a data structure and then use this data structure as a token that moves within a net representing a class to which this agent belongs. Details of these alternatives are discussed in the following subsections.

3.4.3 Nets within a Net

As mentioned earlier, in many of the object-oriented Petri net approaches, it is not clear how to accommodate objects that are created and destroyed several times during a run of the software and are, therefore not suitable for normal software development. For the purpose of the current research, we are interested in representing agents as objects that are spawned and de-spawned as required. As mentioned earlier, one of the alternatives is to represent an object by a token consisting of a Petri net. Valk [75, 79] proposed the concept of "Nets within a Net" in modeling software systems that use dynamic creation and destruction of objects/agents. In this approach, objects are represented by nets, which are token objects in a general system net. The complete system is modeled by a system net and one or more object nets. The creation of an object is then analogous to creation of a token as a result of firing a transition, and deleting of an object is the same as consumption of a token by firing a transition. Objects modeled by this approach have an external behaviour with respect to the base system and an internal behaviour as they change their internal state, when interacting with other objects or when they are subjected to system transactions. These objects can therefore adequately represent the autonomous agents that we use for the dynamic

performance support system. Further research indicated that, although a prototype tool called *Renew* has been developed for simulation of these Petri Nets, the analysis tools are not yet available. As our main purpose of modeling the proposed system with Petri nets is not only to be able to simulate the system but also to be able to verify its properties like fairness, liveness, boundedness and deadlock-freeness etc., it was decided not to pursue this approach any further.

3.4.4 Object Coloured Petri (OCP) Nets

Recently some researchers have noted that one of the most common problems in the area of object-oriented Petri nets is that in most cases previous experience is not taken into account in the development of novel approaches [77]. Proposals such as OOPr/T-Models [77], CCPNs [80] and OCP nets [81], consequently, build on existing work. We chose OCP nets for our work because they can be implemented using existing design, simulation and analysis tools for CPNs such as Design/CPN. We now present some details of these nets. OCP-nets adapt the concepts of object-oriented programming to Petri Nets and are formally defined in [81]. They can adequately model concurrency and they lend themselves to a prototype-based approach to software development. Just as the static structure of a system consists of classes in an object -oriented programming language, the static structure of an OCP net consists of Class Nets, each representing a class of the system. The dynamic structure consists of object nets, which are instances of the class nets. Objects communicate by exchange of tokens through communication channels. Both asynchronous as well as synchronous communication is supported using the concepts of fusion places [82] and synchronous channels [83] respectively. The class nets are suitably enhanced so that they behave as a set of object nets. The colours of the places in class nets are enhanced with the type OID of the object identifier so that a place in a class net can contain tokens representing different objects of the class. The class nets generate and delete these tokens on receipt of requests for services that are identified by the reserved identifiers

new and *del.*

3.5 Design/CPN

Several commercial as well as research tools are available for building, simulating and analyzing of Petri Net Models. A comprehensive overview is available at [84]. In this research, we used Design/CPN as the simulation and analysis tool. It is one of the most commonly used tools in Petri Net research. It was developed at the University of Aarhus in Denmark and consists of a graphical editor for constructing and manipulating CP nets, including syntax check for validating the resulting nets. A built-in simulator allows execution of CP nets with interactive monitoring and debugging capabilities. Large models can be organised into a hierarchy of nets. It also has an integrated analysis tool with a large number of built-in functions to verify desirable properties of a model. A 665-page reference manual available from the Design/CPN web site [82] describes various aspects of the tool in detail. Here we mention some of its basic features.

3.5.1 Language

Design/CPN uses the CPN ML language for data declarations, defining arc inscriptions and guards, and communication between a CP net and its environment. This language is based on Standard Meta Language (SML) - a well known functional programming language. SML is a special version of typed lambda calculus, therefore it is possible to define all kinds of mathematical functions as long as they are computable [82]. SML has been extended to CPN ML such that the language is easier to use even for people who are not familiar with all details of SML. Creating graphical CPN structure is equivalent to writing language statements in a program.

3.5.2 Data

CP nets use data types called colorsets and data objects known as tokens that are somewhat like objects in an object-oriented language. In this research we have used tokens as instances of different agent classes. All datatypes and variables in a CPN model are declared in a global declaration node. Figure 6.2 will show the global declaration node for our model.

3.5.3 Places

A place in a CP net is a graphical object usually represented by an ellipse that is used to hold zero or more tokens of a particular colorset. It has a colorset assigned to it that is one of the colorsets defined for the net in a global declaration node. All tokens in a place must be of the same color as that assigned to the place. It has an optional name to indicate what the place means as a part of a model. The name and colorset is displayed inside or next to a place. A group of tokens is represented by a multiset defined as follows [82]:

A multiset m, over a non-empty set $\mathbf{S}$, is a function $m \in [\mathbf{S} \rightarrow \mathbf{N}]$. The non-negative integer $m(s) \in \mathbf{N}$ is the number of appearances of the element s in the multiset m. The multiset m is usually represented by a formal sum:

$$\sum_{s \epsilon S} m(s)'s.$$

A set of all multi-sets over $\mathbf{S}$ is denoted by $\mathbf{S}_{MS}$. The non-negative integers $\{m(s) \mid s \in \mathbf{S}\}$ are called the coefficients of the multi-set m, and $m(s)$ is called the coefficient of s. An element $s \in \mathbf{S}$ is said to belong to the multi-set m iff $m(s) \neq 0$ and we write $s \in m$.

3.5.4 Markings

A multiset in a place is called marking of a place. Markings of all places in a net is called the state of the net. The state of a net before start of execution is called the initial state and markings of different places corresponding to an initial state are called initial markings. As the net executes, tokens are added or consumed from different places changing the state of the net. A state at any given time during execution of a net is called the current state and corresponding markings as the current markings. The initial marking and the current marking of a place are shown beside the place during execution of the net. For example, Figure 6.3 will show initial and current markings of places, *instance*1 and *instance*2 as $1'U(1)$ and $1'U(2)$ respectively.

3.5.5 Transitions

A transition represents an activity, the occurrence of which may change the number of tokens at one or more places. It is usually represented graphically by a rectangle and may have its name displayed inside or beside it. An arc provides a connection between a place and a transition. It is graphically represented by an arrow with a head indicating its direction. An arc that connects a place with a transition is called an input arc and the place that it connects is called the input place as opposed to the arc that connects a transition to a place, which is called an output arc with the place being called the output place. An arc may have an associated inscription which for an input arc inscription, is a multi-set expression that determines the number of tokens required to be present in an input place for a transition to enable. These tokens will be removed from the input place if the transition occurs. For an output arch inscription, it determines the number of tokens that will be put into the output place if the transition occurs.

3.5.6 Guards

A guard is a boolean expression associated with a transition that defines an additional constraint that must be fulfilled before a transition is enabled. Guards are shown inside square brackets next to associated transitions.

3.5.7 Fusion of Places

Design/CPN allows a user to specify that a set of places may be considered identical in a sense that all of them represent a single conceptual place even though they are drawn as individual places. When a token is added or removed at one of the places in a fusion set, an identical token is added or removed at all the places in the set.

3.6 Summary

In this chapter, we introduced main features of service oriented architecture and briefly discussed the strong and the weak notions of agency. An important feature of this research is the use of formal methods at the design phase in the development of agent based software systems. In this chapter, we examined the use of formal methods in research related to software agents and agent based systems. We pointed out the difficulties in using formal methods for development of complete systems. Our aim in this research, is to be able to carryout dynamic analysis of the proposed EPSS model at the design stage. We note that most of the implemented multi-agent systems use UML for design and analysis. Our initial design in this research is also based on UML. In ths chapter, we briefly introduced UML and some of the related terms that are subsequently used in this research. We also examined the possibility of using Petri nets for modeling multi-agent systems and briefly introduced OCPNs and presented a rationale for using this type of higher level Petri nets in this research. In the end we discussed some details of the Design/CPN tool used for simulation and analysis in this research.

In the next chapter, we discuss some of the design considerations and examine how the issue of flexibility in software, is tradionally handled. We then present a conceptual model of the proposed system and the initial UML based design.

Chapter 4

Design Considerations

Designing a software system that performs satisfactorily in a heterogeneous distributed environment is a complex task. In order to cope with this complexity, new paradigms are needed that facilitate the design of distributed and open systems.

4.1 Abstraction

Abstraction has played a key role in capabilities to develop software for complex systems. During the 70's the concept of abstract data types was introduced. Object Oriented Programming provided a new abstraction during the 1980's. Recent developments allow developers to view their systems with granularity larger than objects by thinking in terms of components that may consist of hundreds of objects. New abstractions such as aspects [85], actors [86], and agents [87] have been introduced. It is increasingly felt both within academia and industry, that software agents will be a key technology as computing systems become ever more distributed, interconnected, and open [88].

4.2 How to Achieve Flexibility?

Software flexibility refers to the ease with which a software system, or one or more of its comonents, may be modified in a timely and cost effective manner. One of the keys to achieving flexibility has always been the clear separation of different perspectives. Most computer applications - regardless of what they do or with which technology they are implemented - generally have three general areas of functionality:

1. Interfaces allow applications to communicate with users as well as with other applications and data resources. Traditionally, people interacted with computer applications using character terminals or graphical user interfaces (e.g., Microsoft Windows). Recently new interfaces such as telephones, web browsers, hand-held computers and wireless devices have been introduced.

2. Business rules support the business processes that are followed by organizations. They automate the process, defining what must be done and how it must be done. There is no standard format for specifying business rules. One might give a very complex procedure as a business rule to carry out a task, while someone else might divide it into several steps. It is however clear that as the business processes change, the business rules in the applications that support them must also be changed. An example of a business rule would be:

   > Issue a check IF (a) an invoice has been presented AND (b) the invoice is for work for which a purchase order was issued AND (c) the work has been performed AND (d) there is enough money in the bank to cover the check.

   Another example could be:

   > This student is eligible for early graduation IF (a) she or he has completed the required work AND (b) she or he has achieved a grade

point average of 3.0 AND (c) it is not yet time for her/him to graduate AND (d) she or he is at least 16 years old.

Business rules are processes followed when business events occur (i.e., business events are triggers for business rules). If business rules define what to do, business events define why it should be done.

3. Data access. Data access code automates the storing, searching, and retrieving of data by computer applications.

The ways in which these application functions are assembled, determines:

- The flexibility of the application.
- How quickly the application can be modified to support changes in business and technology.
- How easily the application interfaces with people and with other applications.

Middleware provides links among the functional levels of the application, as well as with the components supporting the application. Application architectures should be independent of any specific technology or set of development tools. Its components (the interfaces, business rules, and data access code), should be implementable with any development tool in any language on any platform supporting the business needs of the application.

4.2.1 Monolithic Applications

Monolithic applications are applications where the code that implements the business rules, data access, and user interface, are all tightly coupled together as part of a single, large computer program. A monolithic application is typically deployed on a single platform, often a mainframe or midrange computer. There are, however, examples of monolithic applications running on smaller systems – or even distributed

across multiple machines. The determining characteristic of a monolithic application is that the code is tightly coupled and highly interdependent.

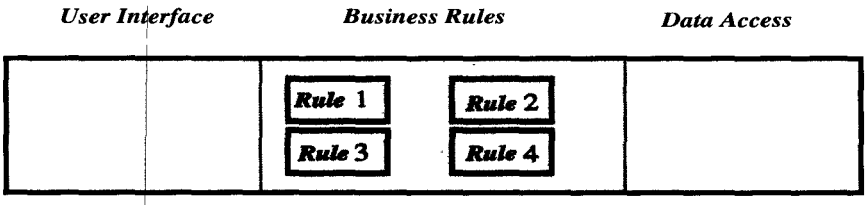

Figure 4.1: Monolithic Applications

Monolithic applications have several drawbacks:

- It is costly and time consuming to modify monolithic applications. Changing one piece of code that, for example, implements a business rule, accesses data, or provides an interface to users or other systems can have side effects on other code in the application. When any code in monolithic application changes, the entire application must be re-tested and re-deployed.

- It is difficult to integrate monolithic applications to share services and data. Most monolithic applications do not have well-defined interfaces that can be accessed by other applications or new user interfaces.

- There is little reuse of redundant code between monolithic applications, making them more expensive to build and maintain. Many monolithic applications contain functionality already replicated in other applications. Monolithic applications are slower and more costly to build because existing functionality must be reinvented many times. Monolithic applications are more expensive to operate, since the same data often have to be gathered, entered, and stored in many places.

- It is difficult to have monolithic applications communicate with other applications.
- There is little flexibility in where monolithic applications can be deployed. Most monolithic applications must be deployed on a single machine type, for example a mainframe. One of the reasons for doing this could be that the software code is tightly coupled to that machine type. Another reason could be the need to get enough processing capacity to process all parts of the application (i.e. the user interface, the business rules, and the data access code).

4.2.2 Two-tier Client/Server Applications

In order to achieve greater flexibility and to reduce unwanted side effects, client/server technology was adopted for new applications. Client/server applications are constructed of software "clients" that, in order to perform their required function, must request assistance - "service" - from other software components known as "servers." Middleware provides communication between the client and server. Early client/server applications used architectures dictated by the tools employed in their construction. As a result, most of the early applications used two-tier client/server architecture. The "tiers" of client/server applications refer to the number of executable components into which the application is partitioned, not to the number of platforms where the executables are deployed. Sometimes the tiers into which the application is partitioned is called "logical partitioning", and the number of physical platforms on which it is deployed is called "physical partitioning."

Fat Clients

In two-tier client/server architecture, application functionality is partitioned into two executable parts, or "tiers." In one model, one tier contains both the code that implements a graphical user interface (GUI) and the code that implements the business

rules. This tier executes on PCs or workstations and requests data from the second application tier, which usually executes on the machine where the application's data are stored. This model is referred to as two-tier, fat client, because, while the application is partitioned into two tiers of executable code, most of the application's code is contained in the tier executing on the workstations - the "fat client."

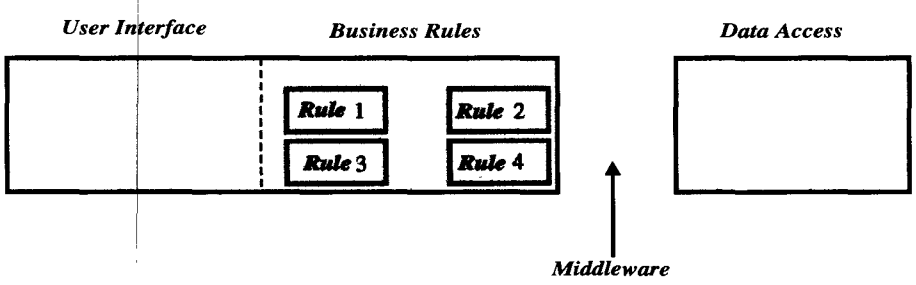

Figure 4.2: Two-tier Fat Client Applications

Since business rules are tightly integrated with user interface code, the code implementing the business rules must be deployed on the same platform(s) as the user interface. This means that the entire workstation-resident portion of the application must be re-deployed when either a business rule or the user interface changes. Whenever the number of workstations is high or the workstations are geographically dispersed, the maintenance costs for two-tier, fat client applications can escalate quickly.

Thin Clients

A second model for two-tier client/server applications has much of the code that implements the business rules tightly integrated with the data access code, sometimes in the form of database stored procedures and triggers. This model is called two-tier, fat server.

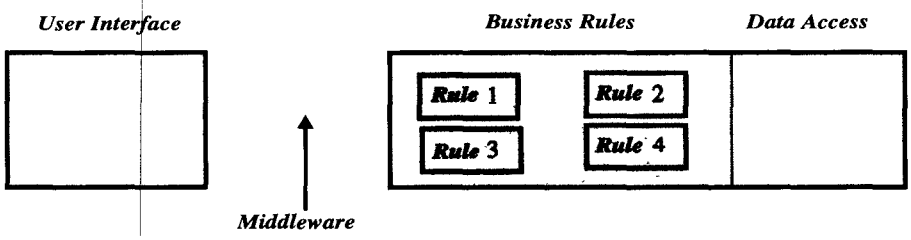

Figure 4.3: Two-tier Thin Client Applications

Since the business rules in two-tier applications are tightly integrated with either the user interface code or the data access code, two-tier client/server applications have the following drawbacks:

- Two-tier client/server applications are difficult and expensive to modify when business requirements change. The business rules still tend to be monolithic. Changing any business rule may, therefore, impact other business rules as well as the rest of the application.

- There is little reuse of redundant code in two-tier client/server applications. It is difficult to reuse business rules elsewhere (e.g., in other computer applications that require similar services or in batch processing that is part of the same application) if they are still tightly coupled to each other and to either the user interface (fat-client) or the data (fat-server).

- There is little flexibility in selecting the platforms where the two-tier client/server applications will be deployed. In two-tier, fat client applications, the business rules must execute on the same platform as the user interface because the code they are implemented in is tightly coupled with the interface. Likewise, in two-tier, fat server applications, the business rules can only execute on the machine

that hosts the database because they are implemented either with the database or inside the database.

- Two-tier client/server applications can be more difficult to manage than monolithic applications. Changes to either business rules or the GUI often means that the entire workstation-resident portion of the application must be redistributed and reinstalled on every workstation that uses the application. Such software distributions can be time-consuming, costly and logistically difficult to manage.

4.2.3 Three-tier Client/Server Applications

Three-tier client/server applications are partitioned into three executable tiers of code: the user interface, the business rules, and the data access software. This does not mean that the three tiers execute on three different platforms. Often, the business rule tier is deployed on the same platform as the data access tier, or on the same platform(s) as the user interface. Properly implemented three-tier client/server applications can achieve higher performance efficiency by providing more flexibility in where the application executables can be deployed.

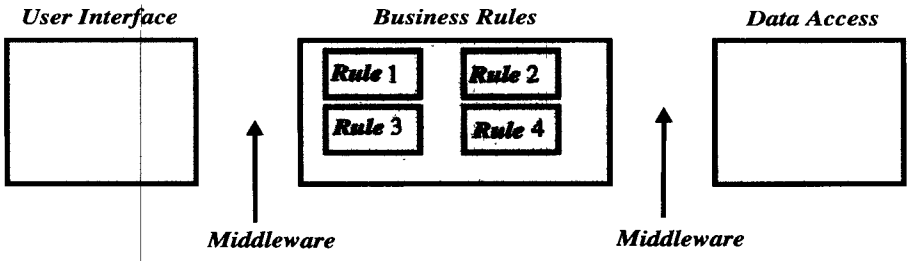

Figure 4.4: Three-tier Client/Server Applications

Three-tier client/server applications offer the following advantages:

- Three-tier client/server applications can be easier to modify to support changes in business rules.

- With three-tier client/server applications, there is less risk in modifying the code that implements any given business rule.
- Three-tier client/server applications can be made to support multiple user interfaces: character, graphical, web browser, telephones, and others.

Frequently the scope of these 3-tier architectures is too narrow, and is most frequently used in the context of a single application - enabling its parts to be physically deployed across the network. While this architecture provides advantages and flexibility, there is often some direct coupling between the client and the server. Different parts may often remain 'hard-wired' to one another with, for example:

- Single business processes dispersed across client and server, making them inseparable, and hence unusable in alternative situations.
- Related business rules distributed to client and server, often with the best intentions of improving performance, e.g. client side validation to reduce network overhead.
- Technology dependencies, e.g. specific database technology calls to the server are embedded in the client.

This means any changes made in the client usually require that corresponding changes be made in the server. In a rapidly changing world, this approach does not allow architectures and policies to be implemented that enable adaptable systems.

4.2.4 Web-enabled Applications

Web-enabled applications are a special case of client-server applications where the "client" is a standard Web browser like Netscape Communicator or Microsoft Internet Explorer. The browser serves as another type of user interface (thin client) in the 3-tier application. Use of a standard Web browser as the client offers the opportunity

to provide the user with a familiar, intuitive interface and significantly simplifies the process for developing and distributing the user interface.

4.3 Salient Features of the Design

The design of EPSS presented in this research, groups the information needs of employees according to business processes of an organization. For example, information needs, related to sales and distribution, are handled in one category and information needs related to plant maintenance in another category. We call the module of software handling each category, a support process (SP). That means there are support processes corresponding to different business processes that are responsible for providing related information to users of the EPSS. Each support process is partitioned into discrete units of functionality called "services." Each service implements a small set of related business rules or function points based on the information needs of employees. Whenever a business rule must be modified to support changing business requirements, only the service that implements the business rule needs modification, while the remainder of the application remains intact. Use of service-oriented architecture provides a loose coupling among different services such that the system is never out of operation while a new service is being added or an old service is being modified.

In order to achieve agility, services are implemented as software agents, which can be spawned or de-spawned according to requirements. User requests are passed on to special agents who obtain the location of services desired, from the published directory and communicate with the agents that implement these services to extract and deliver the desired information to the user. As the desired information is not hard coded in the software and is extracted on the fly by agents responsible for supplying such information, the information supplied to a user is always current and reflects any changes to the contents of information source. Agents can be mobile, if necessary, to

move from one source of data to another.

Common facilities required for the management of services, transport of messages and publishing the availability of services are grouped separately. These common services are implemented by middleware. The prototype discussed here uses Java Agent Development Framework (JADE) as the agent-oriented middleware to provide management and directory services.

For this design, a software agent is an autonomous program that operates on behalf of another entity. It has the ability to communicate, interact and collaborate with other entities, and may have the ability to adapt and take initiative in pursuing goals (within a predefined scope). It is created by another module in the application for a specific purpose. It can terminate itself once it has achieved its goal.

4.3.1 Anthropomorphic Model

We have created an anthropomorphic design that separates the common services such as processing requests from different agents, establishing communication links, spawning/de-spawning agents, tracking the activities and status of agents, and other administrative matters, from application specific services. The common services are the responsibility of two main modules called the *Director* and the *Request Supervisor*; the application-specific services are implemented by a number of agents that are created and destroyed according to the needs of the system. A conceptual view of the system is shown in Figure 4.5. It consists of a support process corresponding to each business process of an enterprise. A support process starts with the interaction of a user with the application through a GUI on his/her computer or laptop by selecting the desired support process. An interface displayed on the user's machine requests the selection of a process of interest. It is possible to choose more than one support process or more than one instance of a particular support process with different options for simultaneous access to different information. Selecting a support process with suitable options spawns a client agent local (CAL) that interacts with

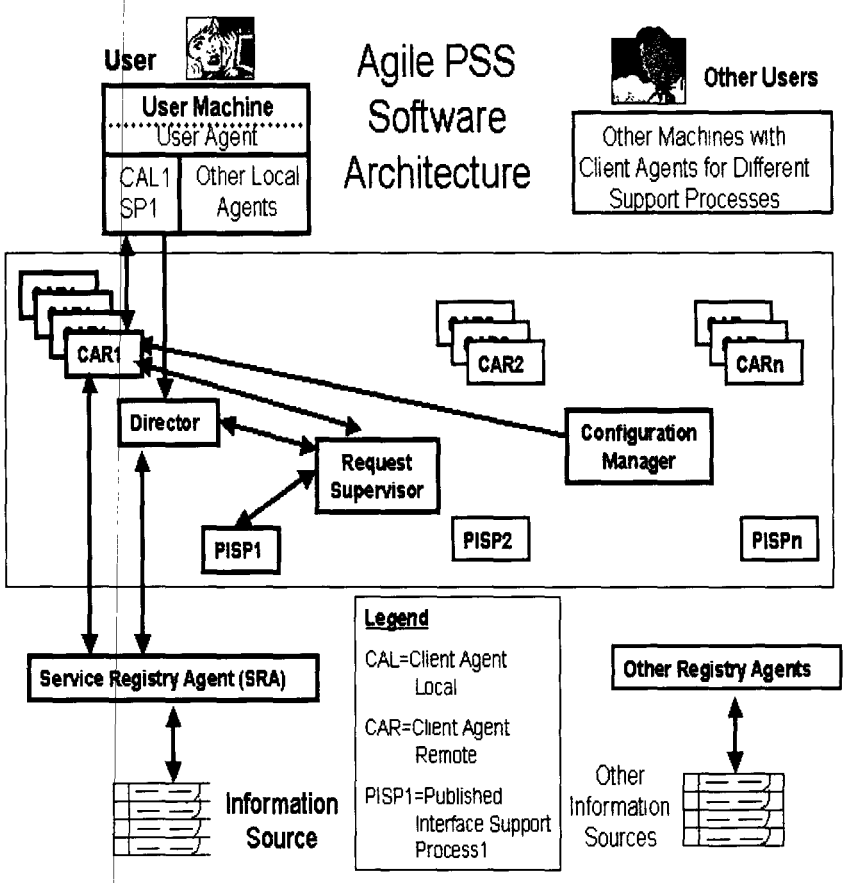

Figure 4.5: Anthropomorphic Model

the Director so that a client agent remote (CAR) is spawned. CAL and CAR then establish direct communication during which the service requested by the user is passed on to the CAR. On obtaining the relevant information, CAR next contacts the Request Supervisor (RS) to obtain the details about the published interface for a particular service related to the support process. With this information, CAR is able to contact the agent responsible for providing the service called the service-registry agent (SRA) and sends the request for the service to it. The desired information is extracted by the SRA and is passed on to the CAL through the CAR. The duties of various agents are detailed below:

User Agent (UA):

- Receives configurable options from the user and saves user queries for immediate or subsequent use.
- Contacts the platform director to request that a suitable client agent local (CAL) be spawned to take care of the current request of the user.

Client Agent Local (CAL):

- Contacts the platform director to request that a suitable client agent remote (CAR) be spawned to work cooperatively with CAL to fulfill the current request of the user.
- Passes clients' queries to CAR.
- Displays progress in processing of user request.
- Displays information for the user on receipt from CAR and requests termination of CAR.

Client Agent Remote (CAR):

- Communicates with request supervisor (RS) to obtain details about the published interface for a particular service in order to access it.
- Requests the published interface to provide the service.
- In case of disconnection with CAL, informs the 'director' and while waiting for resumption of communication, stores any information received from SRA for subsequent transmission.
- On completion of service, passes the information obtained to CAL.
- Terminates itself after CAL receives the information.

Service Registry Agent (SRA):

- Provides interface with the data source.
- Has access to all data in the data source.
- Registers the services that it can provide with the Request Supervisor.
- Has an integration layer to format data from different sources to a format suitable for the proposed system. (The data format within the system is always in the same format. Integration layers at SRAs or CALs will take care of any changes required at the source/destination.)

Request Supervisor (RS):

- Keeps track of the published sources that provide different services.
- At the request of CAR provides information on where a service is available and how to contact it. If multiple sources are available informs the requester about all sources.

Platform Director:

- Establishes and maintains all links among different agents.
- Spawns a CAL when requested by a UA.
- Spawns a CAR when requested by a CAL.
- Keeps track of all active agents.
- Updates records if an agent terminates.
- Receives information about any unplanned disconnection and tries to reestablish it.

- Provides and maintains all common services.
- In case of a large system, RS can be distributed and the director administers it.
- Allows access to middleware by authorized IT personnel.

4.4 Starting with a Specific Case

The goal of this research is to design a service oriented architecture for performance support systems that is easily adaptable to the changing needs of organizations, with very loose coupling among different modules. The architecture is able to handle a number of processes; however, in order to start with a specific case, we decided to work with one of the most common processes - sales and distribution. This process typically consists of the following activities:

1. Receive inquiries from customers and submit quotations.
2. Receive a purchase order.
3. Accept the purchase order after clarifications of technical and commercial conditions (if necessary) and issue an order confirmation to the customer.
4. Select a standard design or a standard design with minor modifications.
5. Get the selected design approved by customer's engineer.
6. Schedule manufacturing at the shop floor.
7. Test the product and arrange inspection by customer's engineer, if required.
8. Pack the product and send it to the finished goods store for onward delivery to the customer.
9. Raise an invoice and obtain payment according to agreed commercial conditions. (In some cases this step may overlap with above steps when, for example

progress payments are involved. For this simple case, we are assuming that full payment will be made on delivery of the product. After sales service, warranties and other possible variations in the process are also being ignored for the time being.)

The process outlined above involves a number of functional units of an organization such as:

(a) Sales Department: where sales agents interact with customers and receive inquiries.

(b) Estimation Department: for preparation of quotations.

(c) Design Department: for selecting a suitable design and obtaining technical approval from customer's engineer.

(d) Production Department: for producing the product as per approved design.

(e) Testing/inspection/quality control Department: for ensuring that the product meets the design specifications.

(f) Inventory Control Department: for safe keeping of the finished product.

(g) Shipping Department: for arranging shipment to the customer.

(h) Accounts Department: for raising the invoice according to agreed commercial conditions.

Each of the above departments may have its own computer system and its own way of keeping records of information. Supervisors/engineers/managers in different departments frequently need information that is available on the computer system

from another department, for efficient discharge of their duties and for planning and smooth running of their own sections/departments.

ERP (enterprise resource planning) systems provide one alternative to handle such needs, by consolidating data into one central database and discarding the existing legacy systems. However, according to some [89, 90], the main project aim for implementing ERP is reducing data redundancy and redundant data entry. Difficulties with customization have been cited as another factor contributing to the failure of ERP systems. It has been argued that some customization may involve changing of the ERP software structure, which may not be allowed and the re-engineering of business processes to fit the "industry standard" prescribed by the ERP systems may lead to a loss of competitive edge. Although costs of ERP systems have come down, it still is a major factor for many of the medium to small enterprises not being able to implement ERP solutions [89].

EPSS does not propose to replace any of the existing transaction processing systems and provides a flexible means to employees of a corporation for extracting desired information from different sources. The main focus of EPSS is not to generate reports based on transactions but to provide "how to" information to enable the employees do their job more efficiently. It extracts related information not only from databases but also from spreadsheets or text files. This means any changes to procedures or policies are available to employees without delay, improving their performance. As stated by Sleight [24]: *"An EPSS extends cognitive ability by abstracting the procedure or task from irrelevant details, and that extends memory by relieving people of the necessity of remembering details, or even of the necessity of learning. A familiar example of a job support tool is the list of instructions for pumping gasoline. It is not necessary to remember how to use the gas pump because the job support is always there, and only the details necessary to run the pump are included in the instructions."* In summary, some of the design considerations for an EPSS are:

- Requirements of different individuals are not the same.

- Procedures within similar functional units of various organizations are different.
- Processes constantly change to keep up with market demands and technological innovations resulting in modifications of procedures of which the employees need to be aware.
- The proposed service oriented architecture must provide easy and fast adaptability.

4.5 Agent Oriented Software Engineering (AOSE)

In order to design a system at the level of granularity of agents and to construct a model in terms of interacting agents, traditional tools are not sufficient. Most of the research related to analysis and design of agent oriented software development is based on "principled but informal" methodologies [49]. Most of the time they have used object oriented analysis and design methodologies as the basis and have proposed extensions or adaptations to make these methodologies applicable to agent oriented development [87, 49, 91, 92, 93]. In some cases adaptations of knowledge engineering techniques have also been proposed [49]. One of the main problems in using object-oriented approaches is that an agent is a much coarse-grained abstraction compared to an object. In fact, a multi-agent system may consist of several thousand objects but only a few agents. Object oriented methodologies also do not support certain concepts that are specific to agents like autonomy and pro-active and dynamic reactions. As UML is a de facto standard for object-oriented modeling, a number of attempts have been made to adapt the UML notations for agent oriented modeling [93, 94] in order to support the concepts of agent, ontology, and interaction protocol. However, a unified approach has not yet emerged. In the meantime a good model can be constructed for the architecture of an agent-based system using an existing mechanism of "stereotypes" in UML to associate an agent oriented semantic with class and collaboration diagrams [93]. This approach has been used in our model.

4.6 UML Model of the System

Rational Rose Professional edition was used to produce use-case models, use-case specifications, sequence diagrams, collaboration diagrams, class diagrams and state chart diagrams. These documents are reproduced in Appendix A. Instead of building a prototype based on the UML model, we wanted to investigate its dynamic properties at the design stage. However, because of the semi-formal nature of UML it is not amenable to dynamic analysis. We therefore looked at the possibilities of transforming the UML model to a formal model so that dynamic analysis could be carried out before coding the implementation of the model. Chapter 5 examines recent research to transform UML models to some kind of formal models amenable to dynamic analysis, and presents our algorithm to transform UML state diagrams to Object Coloured Petri Nets. It also gives examples to highlight how the algorithm can be extended from object-oriented systems to agent-oriented systems.

4.7 Summary

In this chapter, we examined some of the design considerations and discussed some of the approaches tradionally used to achieve flexibility. We then presented details of a conceptual model for the proposed EPSS and focussed our attention to a UML model for a prototype. Details of the UML model consisting of use-case model, use-case specifications, interaction diagrams, and class diagrams are given in Appendix A.

Since UML is based on semi-formal semantics, a UML model is not amenable to dynamic analysis and must be transformed into a formal model for analysis and verification of desirable properties of the system. The next chapter examines some of the approaches proposed by various researchers for such transformations. We then present one of the most important contributions of this research by giving details of an algorithm that transforms UML state diagrams to OCPN models, where dynamic analysis may be carried out.

Chapter 5

From UML to Petri Nets

Unified Modeling Language (UML) is the Object Management Group (OMG) standard for object oriented analysis and design and is widely used by software developers; however, it is based on semi-formal semantics and lacks analysis capabilities whereas Software Engineering (SE) practices require analysis and validation at an early stage in the software development process. The popularity of UML stems from its simplicity and graphical syntax. Therefore researchers have concentrated on improving its semantics to provide the software analyst with a dual approach of using UML to create different models that may be transformed into formal models for verification/validation. The development of approaches for analysis of UML models is a significant step for developers who routinely employ UML to create models for their systems. There is a strong interest among researchers to provide a formal foundation for UML, recognizing its popularity in the software industry. Earlier attempts concentrated on translating UML models to mathematical models using formal languages like Z, HOL and PVS [95]. In [96], the authors use vUML to translate UML state charts into PROMELA, which is the input language to the model checker SPIN. They do not seem to have investigated dynamic creation or deletion of objects although the current version of SPIN provides these facilities. Latella et. al. [1] propose a conversion of the hierarchical representation of UML state diagrams to extended hierarchical

automaton (EHA) as an intermediate step and then translate it to PROMELA. More recently it has been recognized that formal methods must be adapted such that developers without deep mathematical knowledge may also use them [95, 97]. Coloured Petri Nets [82] are well known for their formal foundations, their graphical appearance, their simulation and analysis capabilities, and their support for modeling of concurrent systems [77]. This has generated much interest in the translation of UML models to Petri Nets. Most of these proposals suggest algorithms based on a set of rules that can be followed step by step to translate UML models into Petri Nets. Merseguer et. al. [61, 98] present proposals for translating UML models to GSPNs for performance evaluation. Zhao et. al. [99] propose a set of rules using a series of graph transformation steps. Saldana [63] gives algorithms for translating UML state diagrams to Object Petri nets. We have followed some of their approaches in this research; however, they translate the UML diagrams into Object Petri Nets, which are based on a fixed number of objects that must be determined beforehand. This approach presents a major hurdle in using the technique to develop software for complex systems where objects may need to be instantiated and deleted multiple times during the run time of the system and the number of objects of a particular type should not be constrained as a constant value at the design stage [77]. Their Object Net Models (ONMs) representing individual objects are linked via an Intelligent Linking Place (ILP) but no details are given as to how the ILP handles communication between objects. It also does not bode well for extending such work to model agent systems, which must intercommunicate as well as be able to be spawned and de-spawned on demand during software execution. We propose a set of axioms based on strong intuitive motivations, to transform a UML state diagram to Object Coloured Petri Nets (OCPNs) [81]. These axioms lead to intermediate results [100] that are identical to those obtained by [1] using extended hierarchical automaton.

We note that an OCP-net for an object-oriented system can be created by first constructing an object net for each class. These object nets are then transformed into

class nets by adding facilities to create tokens corresponding to the requests for new objects and to delete/consume tokens corresponding to the requests for deleting the objects. The class nets are connected together through communication channels, to create a Petri net model for the system. Since our interest is mainly in distributed systems where communication between objects is by message passing, we implement asynchronous channels by creating sets of global fusion places. A set of global fusion places conceptually represents one place even though individual places may belong to different class nets. Addition or removal of a token at one place corresponds to addition or removal of identical tokens at all places in the set.

5.1 Overview of the Approach

The goal is to define a step-by-step method that transforms a UML behavioral specification, represented by UML state diagrams, to an OCPnet that represents the behavioral specifications of the system formally, so that the currently available tools for CPN may be used for simulation and analysis of the system. This is carried out in the following steps:

a) We assume that a state diagram exists for every class in the system and transform each state diagram to a corresponding OCPN. We call the net, so obtained, an object model.

b) Each object model is then converted into a class net by adding necessary facilities. Class nets are then integrated into an OCPN representing the system, by two sets of communication channels for asynchronous messages.

Different stages of the design and analysis phase are shown in Figure 5.1.

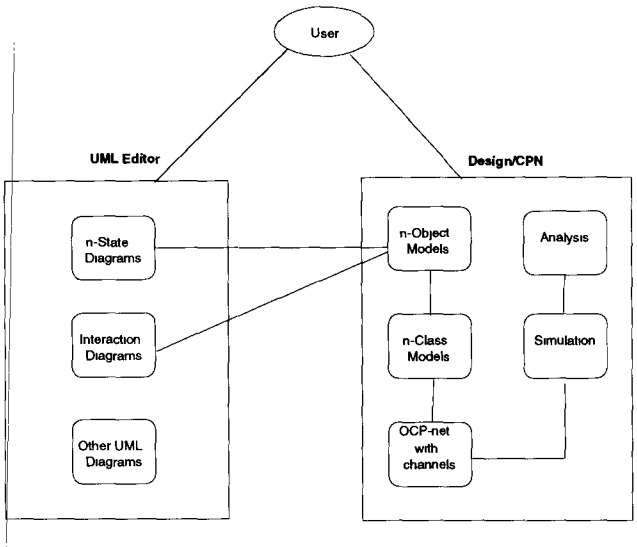

Figure 5.1: Overview of the Design and Analysis Process

5.2 Background

In order to spell out the rules for the conversion, it is necessary to understand different parts of a state diagram with their functions and how to map the functionality to a Petri net. A state diagram models the behavior of a single object. It specifies the possible abstract states of the instances of a class [101]. Some of the basic terms related to UML state diagrams were presented in Section 3.3. We now define the terms used in the algorithm and present the background and justification for some of the steps required to transform a UML state diagram into an OCPN.

5.2.1 Object Model (OM)

An Object Model is a tuple <BM, MP> where BM is a CPN that models the lifetime behavior of an object as specified by the corresponding UML state diagram and interaction diagrams and where MP = { *msg_in*, *msg_inv*, *msg_out*, *msg_rec* } is a set of four places. The place *msg_in* contains all tokens representing messages requesting some service of the current object. The place *msg_inv* contains all tokens

representing messages sent by the current object requesting some service. The place *msg_out* contains all tokens representing messages sent by the current object as an acknowledgement or a return value for a message received earlier from another object. The place *msg_rec* contains all tokens representing messages sent to the current object by another object as an acknowledgement or a return value for a message sent earlier by the current object.

5.2.2 Translation of UML State Diagrams to OCPN

We observe that UML transitions can be represented by Petri net transitions on a one-to-one basis. A UML transition indicates a possible change in the state of an object from the current state to the next. If the current state is represented by an input place and the next state by an output place of a Petri net, then the edge between the current state and the next state can be represented by a Petri net transition. A UML transition, however, may have a label indicating the event(s) that caused the transition, guards and actions, that must be accounted for while transforming it to a Petri net transition.

Transition Events

Events represent conditions that must be fulfilled for a transition from one state to the next; therefore, they are translated as input places for the corresponding Petri net transitions. A token in one of these places represents the occurrence of an event. However, if an event consists of a message received from another object, it is represented by the existence of a token in a special input place connected to the communication channel. This place is given the name *msg_in* if it contains tokens representing service requests to the currrent object by other objects. The place is named *msg_rec* if the tokens represent an acknowledgement or a return value as a result of an earlier message sent to another object by the current object.

Transition Actions

Actions represent the results of a transition and therefore correspond to Petri net output places. A token in one of these places indicates a request for the service implemented by a transition that has an input arc from this place. However, if an action consists of a message that invokes a service in another object, it is represented by the existence of a token in a special output place connected to the communication channel. This place is given the name *msg_inv* if it contains tokens representing outgoing messages that invoke services in other objects. The place is named *msg_out* if it contains tokens representing outgoing messages that send a return value or an acknowledgement to another object as a result of an earlier message received by the current object.

Guards, Entry and Exit Actions

Guards represent constraints that must be satisfied for the transition to take place and are translated to corresponding guards for the Petri net transitions. A UML state's exit action can be considered as a transition action on every outgoing transition from the state and an entry action is similar to a transition action on each of the incoming transitions to the state [101]. A self transition consists of an exit action followed by an entry action. Therefore the entry/exit actions of a state are translated in the same way as the transition action discussed above.

Parameters

If the invocation of a service needs parameters, they must be passed as part of the message, i.e. the message cannot be sent until the parameters become available. In other words, the Petri net transition cannot be enabled without input parameters. We transform the parameters to input places for the Petri net transition with initial markings of these places representing the parameter values. A parameter representing a return value is translated as an output place of the Petri net transition.

5.2.3 Class Net (CN)

A Class Net is an enhanced version of OM that meets the formal definition of class net given in [81]. It is obtained from the OM by adding:

1. Four transitions ($T_{in}, T_{inv}, T_{out}, T_{rec}$) and four places ($P_{in}, P_{inv}, P_{out}, P_{rec}$) to handle communication between objects.

2. Two transitions *Create and Delete* to handle creation and deletion of tokens representing objects of the class

3. A finite set of instance fusion sets, if necessary, for the class attributes.

We use the terms *input place, output place, input transition, output transition, input arc* and *output arc* as defined in [82], i.e. a node x is called an input node of another node y, iff there exists a directed arc from x to y. Analogously, a node x is called an output node of another node y, iff there exists a directed arc from y to x.

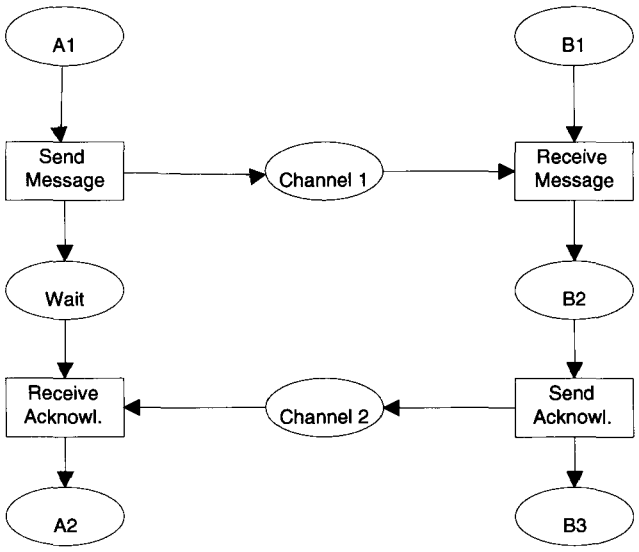

Figure 5.2: Communication Channels

5.2.4 Communication Between Objects

The class nets discussed above must be connected together so that messages may be passed between different objects. This is done by creating two channels, using the concept of global fusion places. A set of global fusion places, consisting of P_{in} and P_{inv} places in all class nets, is created to act as channel 1. Channel 2 consists of global fusion places P_{out} and P_{rec} of all classes. Here we have used the modeling guidelines proposed by [82] for asynchronous communication with acknowledgement, as shown in Figure 5.2. With this discussion, we are now ready to provide the algorithm.

5.3 Algorithm

Precondition: A UML2.0 state diagram is available for each class in the UML2.0 class model of the system that needs to be transformed into an OCPN and Design/CPN 4.0 or an equivalent software for graphically creating and analysing CPNs is also available.

Begin 1. For each state diagram

Begin Using Design/CPN

(a) Convert the state chart into an object model using Algorithm 1

(b) Transform the object model to a class net using Algorithm 2

End

2. Connect the class nets by a set of global fusion places defined as

$$Channel_1 = \left(\bigcup_{i=1}^{n} P_{in-i}\right) \cup \left(\bigcup_{i=1}^{n} P_{inv-i}\right)$$

$$Channel_2 = \left(\bigcup_{i=1}^{n} P_{out-i}\right) \cup \left(\bigcup_{i=1}^{n} P_{rec-i}\right)$$

Where n is the total number of class nets.

End

5.3.1 Algorithm 1

Converts a UML state diagram to OM based on the background and justification presented in Section 5.2.1.

Begin For each state diagram

1. If there are any composite states, "flatten" them to simple states
2. Examine the state diagram and assign a unique identifier to each state transition
3. Create two tables similar to Tables 5.1, 5.2 and fill in data for all state transitions.

Table 5.1: State Diagram Table A

State Transition#	Input State	Output State	Transition Events	Transition Guards

Table 5.2: State Diagram Table B

State Transition#	Input State Exit Action(s)	Output State Entry Action(s)	Transition Action(s)

4. Start with a new CPN page and create four places named *msg_in, msg_inv, msg_out and msg_rec* and position them close to the top, bottom, left and right of the page.
5. For each state transition
   (a) Create a CPN transition and transform the guard conditions to CPN guards
   (b) Create a CPN input place for the input state and a CPN output place for the output state. If a state already exists, do not duplicate it.

(c) **If** there is a state transition event

then If this is a local event create a CPN input place for it

ElseIf the event is a message received from another object requesting a service of this object

then create an input arc from *msg_in* place to this transition and another from a place containing a token of the object receiving the message.

ElseIf the event represents a returned value from a previous action

then draw an input arc from the *msg_rec* place to this transition and create an output place for the value returned. Also draw an input arc from a place containing a token of the object receiving the message, to this transition.

EndIf

EndIf

(d) **If** there is an action for the state transition or an entry action for the output state or an exit action for the input state

then If it is a local action create an output CPN place for it.

ElseIf this action is a message (M) for another object

then draw an output arc to *msg_inv* place from the transition. Create CPN input places for all parameters and draw arcs from these places to the transition. Specify initial markings for these places to represent the parameter values.

ElseIf an action results in sending of a return value or acknowledgement to another object

then draw an output arc to *msg_out* place.

EndIf

EndIf

End

5.3.2 Algorithm 2

Converts an OM to a CN as discussed in Section 5.2.3.

Begin

1. Create four CPN transitions named T_{in}, T_{inv}, T_{out} and T_{rec} and four CPN places named P_{in}, P_{inv}, P_{out} and P_{rec}
2. Draw an input arc to T_{inv} and T_{out} from *msg_inv and msg_out* places of the object net respectively
3. Draw output arcs from T_{inv} to P_{inv} and T_{out} to P_{out}
4. Draw an output arc from T_{in} and T_{rec} to *msg_in and msg_rec* places of the object net respectively
5. Draw input arcs to T_{in} from P_{in} and to T_{rec} from P_{rec}
6. Add facilities to create and destroy objects in the class by adding a *create* and a *delete* transition with input arcs from the *msg_in* place and in case of the create transition, an output arc to a place (AllIDS) that contains all object ids for the class and in case of the delete transition an input arc from this place.
7. Draw an input-output arc between T_{in} and AllIDS and another between T_{rec} and AllIDS to ensure that a message is passed on to an object only if it exists.
8. A create or delete message is handled by the class and it places a token representing the newly created object in the initial place of the net representing the class. In case of deletion, the token is consumed by the transition implementing this message.

End

5.3.3 Comments

Several examples of UML statecharts and corresponding Petri nets are reported in [63]. We tested the above algorithm on these statecharts and successfully transformed them to known Petri nets. The size of the Petri net obtained by application of this algorithm compares favourably with the size of the input UML statechart as discussed below:

The size of a PN can be measured by the number of transitions and the number of places [102]. There are a number of metrics for measurng the size and complexity of a UML statechart such as the number of entry actions, the numner of exit actions, the number of activities, the number of states, the number of transitions and the number of guards [103]. The algorithm transforms each UML statechart transition to a corresponding PN transition. The resulting PN transitions can sometimes be combined to reduce the total number of transitions by using simple reduction rules specified in [82]. This means the number of PN transitions resulting from the transformation are at most equal to the number of UML statechart transitions. The number of guards for the Petri net transitions equal the number of guards for UML transitions. Each event, entry/exit action, and simple state in the UML statechart is transformed into a place in the resulting PN. Events that are in the form of messages received from other objects and actions that result in sending messages to other objects are represented as tokens contained in special places that are not duplicated for every event/action of this type. This means that the total number of places in the resulting PN is at most equal to the total number of events, entry/exit actions, and simple states.

5.4 Illustrative Example

Application of the above algorithm is now illustrated by an example that consists of two classes: a user class acting as the root class and a DVD_Collection class whose objects have an integer attribute that holds the number of DVDs. This example has

been adapted from an example in [104, Page 148]. The User creates two objects of DVD_Collection type and stores the number of music DVDs in one object and the number of movie DVDs in the other. Figure 5.3 and figure 5.4 show the UML state diagrams for the DVD_Collection and the User classes respectively. Tables 5.3, 5.4 and 5.5 collect the information required by the algorithm from the two state diagrams. Figure 5.5 shows the collection object model and the figure 5.6 shows the user object model. Finally the figures 5.7 and 5.8 show the user class net and the collection class net respectively. The parts of these diagrams enclosed in a dashed box represent the corresponding object models. The complete CPN model was simulated using Design/CPN.

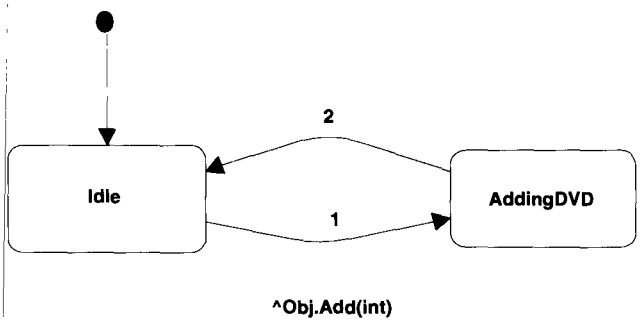

Figure 5.3: State Diagram of Collection Object

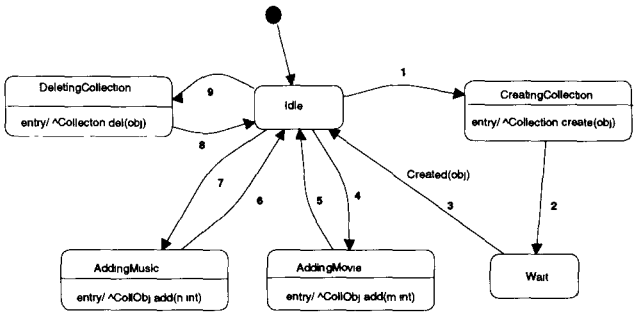

Figure 5.4: State Diagram of User Object

Table 5.3: Collection Object State Diagram Table A

State Transition#	Input State	Output State	Transition Events	Transition Guards
1	Idle	AddingDVD	^Obj Add(int)	Nil
2	AddingDVD	Idle	Nil	Nil

Table 5.4: User Object State Diagram Table A

State Transition#	Input State	Output State	Transition Events	Transition Guards
1	Idle	CreatingCollection	Nil	Nil
2	CreatingCollection	Wait	Nil	Nil
3	Wait	Idle	created(obj)	Nil
4	Idle	AddingMovie	Nil	Nil
5	AddingMovie	Idle	Nil	Nil
6	AddingMusic	Idle	Nil	Nil
7	Idle	Adding Music	Nil	Nil
8	DeletingCollection	Idle	Nil	Nil
9	Idle	DeletingCollection	Nil	Nil

Table 5.5: User Object State Diagram Table B

State Transition#	Input State Exit Action(s)	Output State Entry Action(s)	Transition Action(s)
1	Nil	^Collection Create(obj)	Nil
2	Nil	Nil	Nil
3	Nil	Nil	Nil
4	Nil	^CollectionObject Add(m int)	Nil
5	Nil	Nil	Nil
6	Nil	Nil	Nil
7	Nil	^CollectionObject.Add(n:int)	Nil
8	Nil	Nil	Nil
9	Nil	^Collection del(obj)	Nil

5.5 Handling of Composite States

UML state diagrams are based on state charts introduced by Harel [105] as a visual formalism for complex reactive systems. They extend state machines by hierarchy, concurrency, and communication. While inclusion of these features has helped greatly in specifying complex reactive systems, they interact in intricate and unexpected ways with the result that various formal semantics of state charts and state chart tools interpret their interaction in different ways or impose different constraints [106]. For

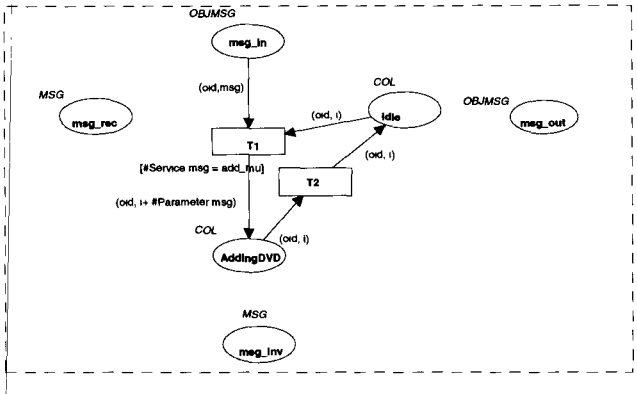

Figure 5.5: Collection Object Model

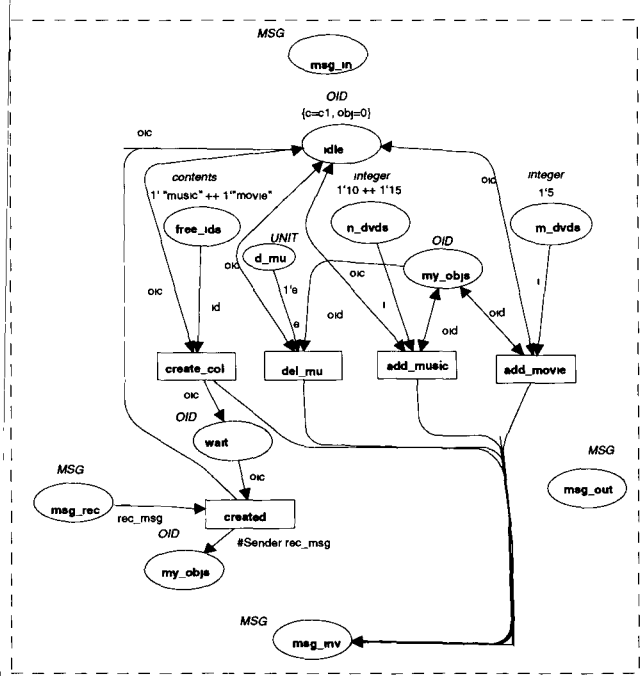

Figure 5.6: User Object Model

model checking of state charts, it is necessary to interpret the state chart model as a transition system. A number of different techniques, for this purpose, have been reviewed in [105]. These proposals can be classified into two groups: techniques that use a set of axioms to flatten any composite states in the state chart model and then translate the state chart descriptions directly into some kind of textual descriptions

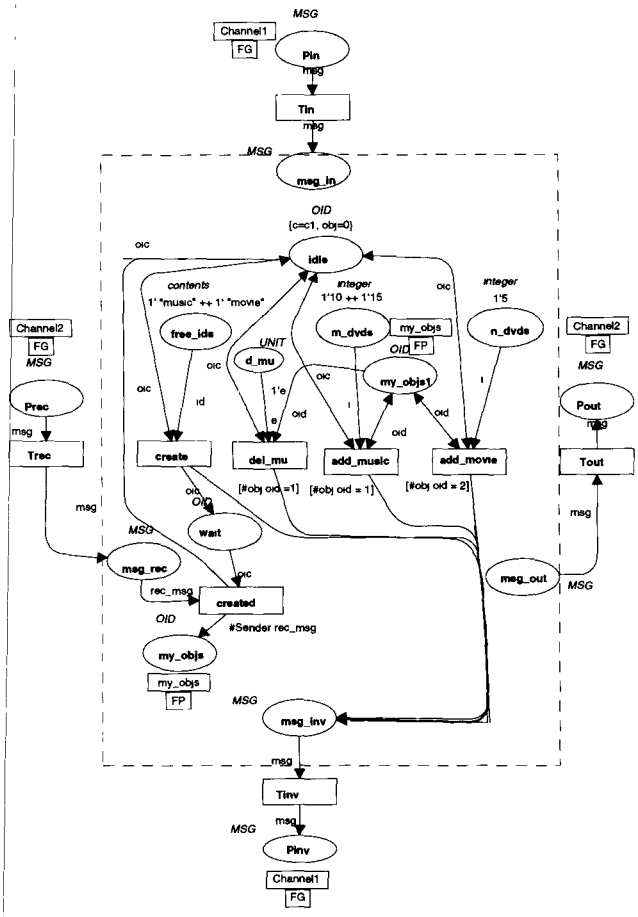

Figure 5.7: Class Net of Class User

of a formal language, and those that transform the state charts to a formal graphical structure such as an extended hierarchical automaton (EHA) and then use this structure to translate the state charts to a formal language such as PROMELA. The resulting interpretation of the state chart model as a transition system is next used as input to a model checker such as SPIN. In this research, the resulting transition system has been represented as a set of tables and the transformation of the state chart model into an Object Coloured Petri Nets (OCPN) model undertaken. The interpretation of the state chart model as a transition system assumed that any composite states in the state chart model have been flattened by using a set of axioms as

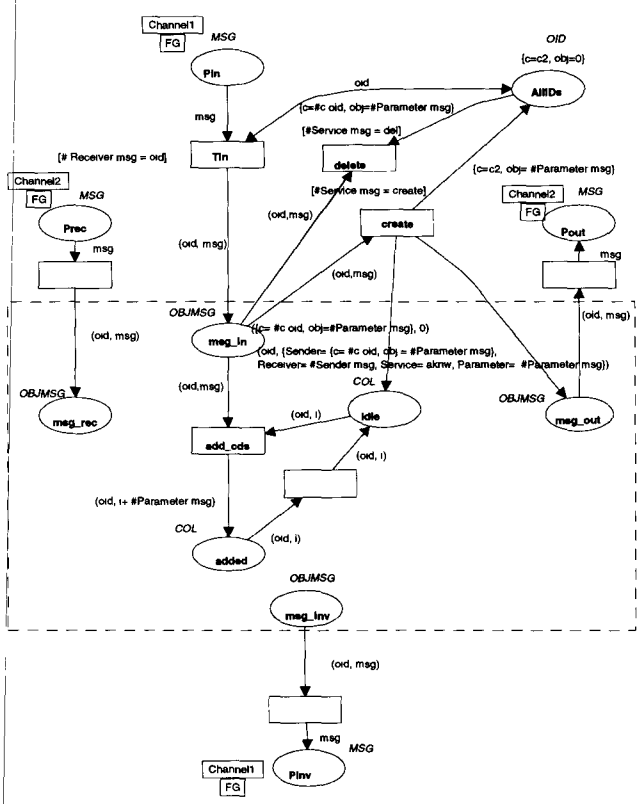

Figure 5.8: Class Net of Class DVD_Collection

proposed by [63]. We now show that following an alternate approach proposed by [1] instead of our algorithm results in the same transition system.

5.5.1 Use of EHA for Handling Composite States

Latella et. al. [1] use the state diagram shown in Figure 5.9 to discuss the translation of composite state diagrams to EHA. The diagram consists of three states s1, s2 and s3. The state s1 is a concurrent composite state consisting of concurrent states s4 and s5 which in turn are also composite states. The state s4 consists of s6 and s7 while s5 consists of s8 and s9. The state s7 is also a composite state consisting of s10 and s11. Although a transition can be labeled by a trigger event, a guard and a sequence of actions, the authors use only trigger/action pairs to label transitions.

At a given time the system can be in any of the following sets of states referred to as "configurations": {s1, s6, s8}, {s1, s6, s9}, {s1, s10, s8}, {s1, s11, s8}, {s1, s10, s9},{s1, s11, s9}, {s2}, {s3}. As a result of a trigger from the environment and provided the guard is satisfied, a transition can fire if and only if its source state is in the current configuration. If the transition is fired, the source state is left, the actions are executed, and the target state is entered. Inter-level transitions can be fork transitions that, in general, have more than one target state or join transitions that have more than one source state all of which must be in the current configuration for the transition to be enabled. The authors propose a syntactical translation of the statechart into a hierarchical automaton that describes the essential aspects of the statechart. An EHA consists of simple sequential automata related by a refinement function. A state is mapped via the refinement function into the set of automata that refine it. The statechart of Figure 5.9 is mapped into the hierarchical automaton of

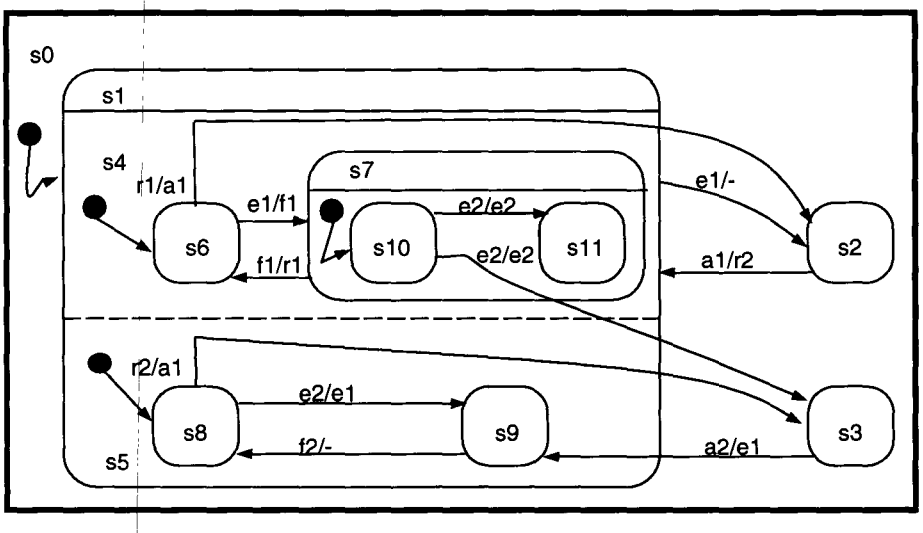

Figure 5.9: State Chart Example Reproduced From [1]

Figure 5.10 which is an alternate representation of the statechart. Initial states are shown by thick boxes and the refinement of a state into one or more sub-states is represented by the refinement function p; in the above example p s1 = {A1, A2}, p

s7 = {A3} and for any other state s, p s = 0. This refinement is shown by dashed arrows in Figure 5.10.

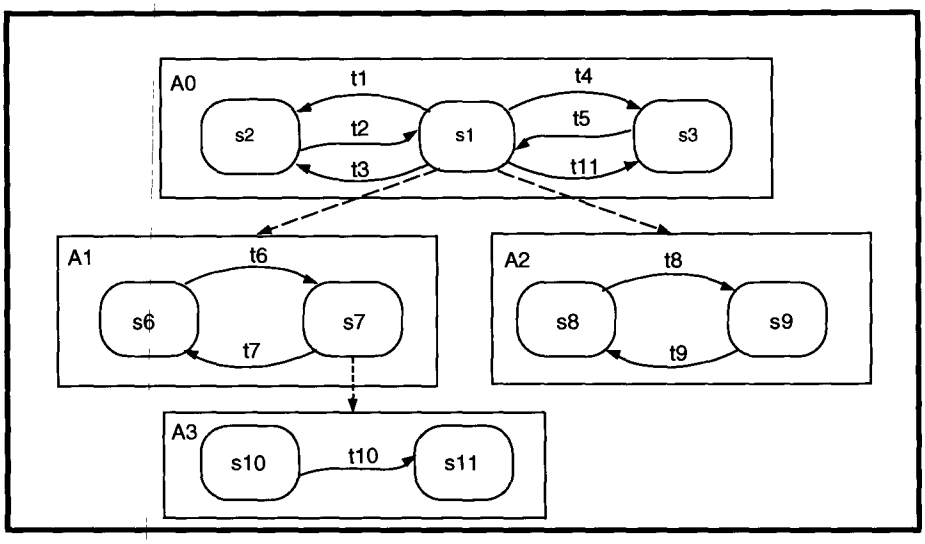

Figure 5.10: EHA Equivalent to State Chart in Figure 5.9 reproduced from [1]

Table 5.6: Transition System for the EHA of Figure 5.10

Transition	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11
SR	{s6}	0	0	{s8}	0	0	0	0	0	0	{s10}
EV	r1	a1	e1	r2	a2	e1	f1	e2	f2	e2	e2
AC	a1	r2	ϵ	a2	e1	f1	r1	e1	ϵ	e2	e2
TD	0	{s6, s8}	0	0	{s6, s9}	{s10}	0	0	0	0	0

The representation of the statechart of Figure 5.9 as a transition system is given in Table 5.6 where SR is the source restriction that indicates the actual origin of a transition, TD is the target determinator that lists all the basic states that must be reached when a transition is fired; EV is the event that triggers a transition and AC is the corresponding actions to be performed when the transition fires. The notation adopted by the authors is that when a source or a target state is part of a composite state, they are explicitly listed, however a basic source or target state is represented by a 0 in the table.

5.5.2 Use of Axioms for Handling Composite Sates

Saldana et. al. propose an algorithmic approach to flatten a composite UML statechart [63]. They further define a process for deriving a state transition model to handle the basic issues related to the hierarchical structure of composite states [107]. We used their algorithm to flatten the statechart of Figure 5.9 to that shown in Figure 5.11 where, to ensure clarity, the transition T4 represents a transition from any state to S2 as a result of trigger e1. We then used the algorithm proposed by us [108] to transform this statechart into a transition system represented by Table 5.7.

Table 5.7: Transition System for the Flattened State Chart Figure 5.11

Transition	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
SR	S6	S2	S2	Any	S8	S3	S3	S6	S10	S11	S8	S9	S10	S10
EV	r1	a1	a1	e1	r2	a2	a2	e1	f1	f1	e2	f2	e2	e2
AC	a1	r2	r2	Nil	a2	e1	e1	f1	r1	r1	e1	Nil	e2	e2
TD	S2	S6	S8	S2	S3	S6	S9	S10	S6	S6	S9	S8	S11	S3

Our table is more detailed because we treat each transition separately without regard to whether it is a simple or a complex transition because any composite states have already been flattened whereas Table 5.6 focuses on composite states.

5.5.3 Comparison of Results

In order to compare the results, we note that there are different types of transitions. A simple transition has only one source state and one target state and these are simple states. A complex transition on the other hand has a composite state as its source or the target state. A boundary transition is a complex transition that starts from or ends at, the boundary of a composite state. A cross-boundary transition, sometimes called an inter-level transition, is a complex transition that starts from, or ends at, one of the nested states of a composite state.

In case of a boundary transition, if the target is a concurrent composite state, all default states of concurrent regions are active when the transition fires. On the other hand, if the source of such a transition is a composite state, the transition can

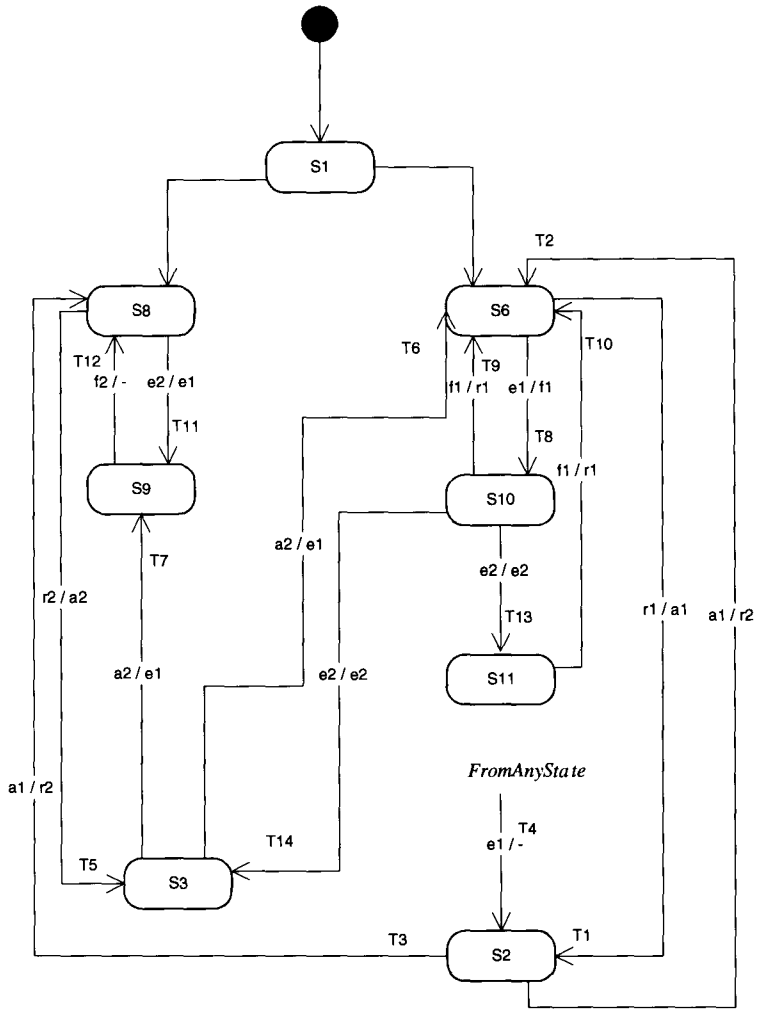

Figure 5.11: Flattened State Diagram

Table 5.8: Modified Table 5.7 According to Notations Discussed in Section 5.5.3

Transition	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14
SR	{S6}	Sb	Sb	Sb	{S8}	Sb	Sb	Sb	Sb	Sb	Sb	Sb	Sb	{S10}
EV	r1	a1	a1	e1	r2	a2	a2	e1	f1	f1	e2	f2	e2	e2
AC	a1	r2	r2	Nil	a2	e1	e1	f1	r1	r1	e1	Nil	e2	e2
TD	Sb	{S6}	{S8}	Sb	{S6}	{S9}	{S10}	Sb	Sb	Sb	Sb	Sb	Sb	Sb

fire with any of the nested states active. For a cross-boundary transition that targets a specific nested state of a concurrent region, the default states of other concurrent

Table 5.9: Result of Combining Transitions in Table 5.8 into Fork/Join Transitions

Transition	t1	t2 (T2+T3)	t3	t4	t5 (T6+T7)	t6	t7 (T9+T10)	t8	t9	t10	t11
SR	{S6}	Sb	Sb	{S8}	Sb	Sb	Sb	Sb	Sb	Sb	{S10}
EV	r1	a1	e1	r2	a2	e1	f1	e2	f2	e2	e2
AC	a1	r2	Nil	a2	e1	f1	r1	e1	Nil	e2	e2
TD	Sb	{S6, S8}	Sb	Sb	{S6, S9}	{S10}	Sb	Sb	Sb	Sb	Sb

regions are also active on firing of this transition. We adopt the following notation: When a source or a target state is a basic state, it is denoted by Sb. In case of cross-boundary transitions the specific nested state at which the transition starts or ends are listed using set notation. When more than one transition has the same source state but different target states, they are combined into one fork transition by showing the target states as a set of states. Transitions that have the same target state but different source states are combined into a join transition with the source states shown as a set of states. Following this notation we transform Table 5.7 into Table 5.8 and then to Table 5.9 which is essentially identical to Table 5.6 reported by Latella et. al.

5.6 Extension of the Algorithm to Software Agents

The behaviour of an agent can be described as a sequence of states that the agent is in, changing from one state to another, as response to external events consisting of interaction with other agents or with its environment. Each state may represent one of its activities including when it is idle waiting for something to happen in its environment. The change of state may be accompanied by the execution of an action that could consist of a local action by the agent or sending a message to another agent. Since we want to transform the UML model of an information system to a Petri net model in order to simulate the system and study its dynamic properties such as fairness, boundedness, liveness, deadlock-freeness and reversibility etc., we use a higher level abstraction of each agent in the system, similar to the concept of the

skeleton in [47], where an abstract state may correspond to a number of computation states considered alike for dynamic analysis of the system. Based on this abstraction, behaviour of an agent can be represented by the UML state diagram. For example, the UML representation of the behaviour of an information-processing agent is shown in Figure 5.12.

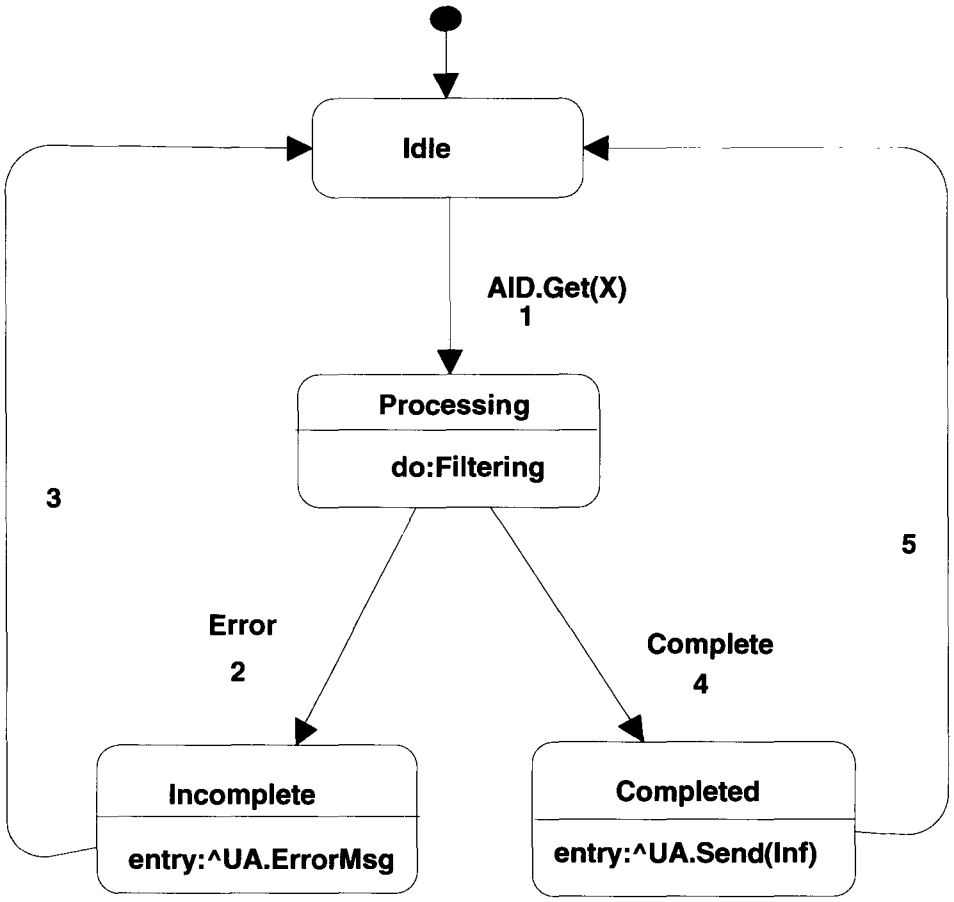

Figure 5.12: State Diagram of Information Agent

The algorithm is applied to agents in the following steps:

1. Different types of agents are represented by class diagrams using the mechanism

of "stereotypes" as discussed in Section 4.5.

2. UML state diagrams representing the behaviour of agents belonging to different classes are constructed at the level of abstraction discussed above.

3. The Object Model (OM) in the above algorithm is extended to the Agent Model: An agent Model is a tuple <BM, MP> where BM is a CPN that models the life-time behavior of an agent as specified by the corresponding UML state diagram with MP = { *msg_in, msg_inv, msg_out, msg_rec* }, a set of four places.

4. State diagram of each agent is transformed to an Agent Model by following the algorithm given in Section 5.3.

5. Each Agent Model is converted into a Class net by adding necessary facilities.

6. Class nets are integrated into an OCPN representing the system, by two sets of communication channels for asynchronous messages. These messages represent the internal events within an agent as well as the external events created by other agents in the system.

5.7 Summary

We began this chapter by examining some of the recent research aimed at transforming UML state diagrams to a model amenable to formal analysis. One of the most important contributions of this research is an algorithm to transform UML state diagrams to an OCPN model that was presented next. In order to confirm validity of our algorithm it was applied to a set of UML state diagrams used in recent research by Latella [1]. It was shown that our algorithm results in a transition system that is identical to that obtained by Latella et. al, who used extended hierarchical automaton (EHA) to obtain an intermediate transition system. The algorithm was then extended for application to software agents.

In the next chapter, we discuss simulation of the OCPN model obtained by transformation of UML state diagrams of EPSS using the algorithm presented in this chapter. We used the Design/CPN simulation tool and carried out analysis of the model by creating occurrence graphs for verification of some of its behavioural properties.

Chapter 6

Model Simulation And Analysis

In this chapter we give a brief description of the CP nets used for each type of agent class in the OCPN model of the EPSS. We then present analysis of the model carried out by building various occurrence graphs and by generating reports to verify some of the desirable properties of the model such as boundedness, liveness, fairness, deadlock-freeness and home properties.

6.1 The Model Hierarchy

The OCPN model of EPSS consists of CP nets distributed over six pages. Design/CPN has a facility to create a hierarchy of different nets which as a whole, model the EPSS. In order to keep track of the pages and their relationships to each other the model contains a special page called a hierarchy page as shown in Figure 6.1. In this page, every page in the model, including the hierarchy page (Hierarchy#10) is represented by a small oval called the page node. Each page node has a name and a page number. Design/CPN facilitates incremental development by executing only those parts of the net that appear on specially designated pages called "Prime" pages or in someway referenced by a prime page. A prime page has additional information about the page by designations "M" and "Prime" next to the page node. A net may

have any number of prime pages, however it must have at least one prime page to be able to execute.

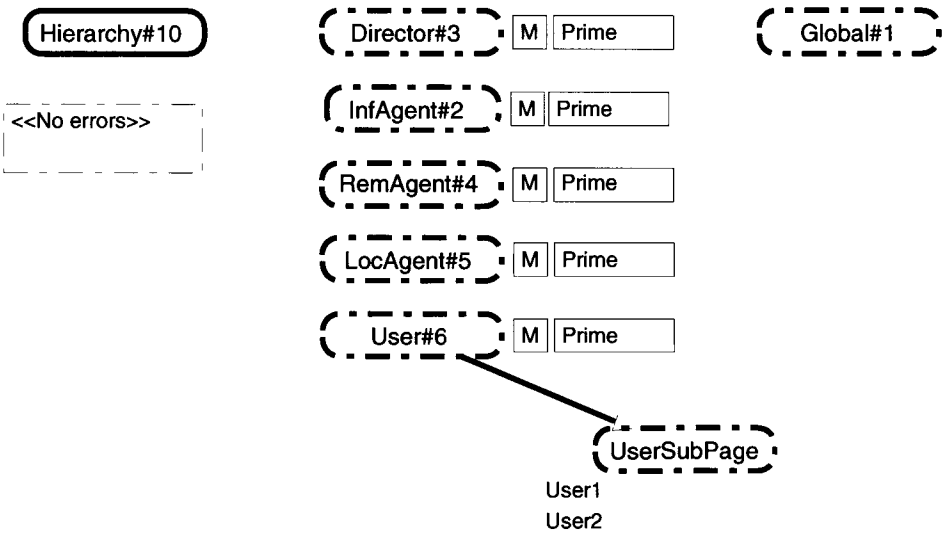

Figure 6.1: Hierarchy Page of the Model

The hierarchy page is also an active device by which the user can manipulate the pages. For example a page can be opened by double clicking on the corresponding page node or it can be deleted by deleting the corresponding page node. The function of different pages in the model, is as follows:

6.1.1 Global#1

Figure 6.2 is the global declaration page where all colorsets and variables are declared. The first line of the page ensures that the latest occurrence graph generation algorithm is used by the analysis tool and the second line takes care of a bug relating to representation of date that caused the tool to stop functioning on January 10, 2004.

As the EPSS can be used by any number of users simultaneously, a user number is represented by the indexed colorset U. A user action may result in one or more agents and an agent number is represented by the indexed colorset a, where the range

```
NewOGGeneration:= true;
use "/home/bokhari/tod.sml";


color PARAM =  int with 0..5;
color integer = int with 0..9;
val n =5 :PARAM;
val m = 2 :PARAM;


color contents = string;
color USER = index U with 0..m declare ms;
color AGNTN = index a with 0..n declare ms, index ;
color CID = with usr | car | cal | cia| cdr ;
color AID = record ur:USER * c:CID * ajd: AGNTN declare ms;
color MID = with crtcal | delcal | crtcar | delcar | crtifa | delifa | finfo | getinfo | error | aknw |
deltd |  crtd;
color COL = product AID * PARAM;

color MSG = record Sender:AID* Receiver:AID* Service:MID * Parameter:PARAM ;
color AGNMSG = product AID * MSG;
color UNIT = unit with e;
color AGENT =  record ag:AID *  inf :PARAM;
color FILT = with x | y;

val dir= {ur=U(0), c=cdr, ajd=a(0)} :AID;
val calc={ur=U(0), c=cal, ajd=a(0)} :AID;
val carc={ur=U(0), c=car, ajd=a(0)} :AID;
val ciac={ur=U(0), c=cia, ajd=a(0)} :AID;
var cl:CID;
var msg,rec_msg:MSG;
var id: AGNTN;
var in_msg:AGNMSG;
var out_msg: AGNMSG;
var aid, sid, aic, aid1, aid2:AID;
var i, k :PARAM;
var j :integer;
var u :USER;
```

Figure 6.2: Page Containing Declarations

of indices is 0 to m and 0 to n respectively. A class is represented by the colorset CID that can have one of the agent class $\{usr, car, cal, cia, cdr\}$ as its value. A particular

agent is identified by a composite colorset AID, which is a tuple consisting of a user number, class ID and an agent number. Colorset MID defines different types of messages that are exchanged between the agents. The message format consists of identifiers of the sending and receiving agents, type of message and a parameter that is used to specify additional information e.g. the criteria to be used to filter the information. Some constants and variables are also declared for use in different net inscriptions of the model.

6.1.2 User#6

Figures 6.3 and 6.4 show typical implementations of the User Interface and the User Agent class nets respectively. The system allows for one or more users to be logged into the system concurrently and each user can cause one or more Local Agents to be spawned concurrently by selecting different options presented by the User Agent, which is an instance of the User Agent Class. This can result in countless possibilities that are limited by the User Inteface net which, for the purpose of analysis, allows a limited number of users that can log into the system concurrently. Such users are represented by places of color USER and named Instance(n), where n is the user number. For example, the net shown in Figure 6.3 shows two such places making it possible for two users to log into the system concurrently. The number of Local Agents that a particular user can spawn, can be limited by inscriptions on the output arcs of the transition named 'login'. The output arc inscriptions in Figure 6.3 allow user U(1) to spawn two Local Agents while the user U(2) can spawn only one Local Agent.

The User Agent class net shown in Figure 6.4 is shared by each user currently logged into the system. This is implemented by building a hierarchical net using the concept of substitution transitions detailed in [82]. The transitions named User1 and User2 in Figure 6.3 represent the substitution transitions linking the User Interface to the User CLass net of Figure 6.4. This is highlighted in the diagram by the existence

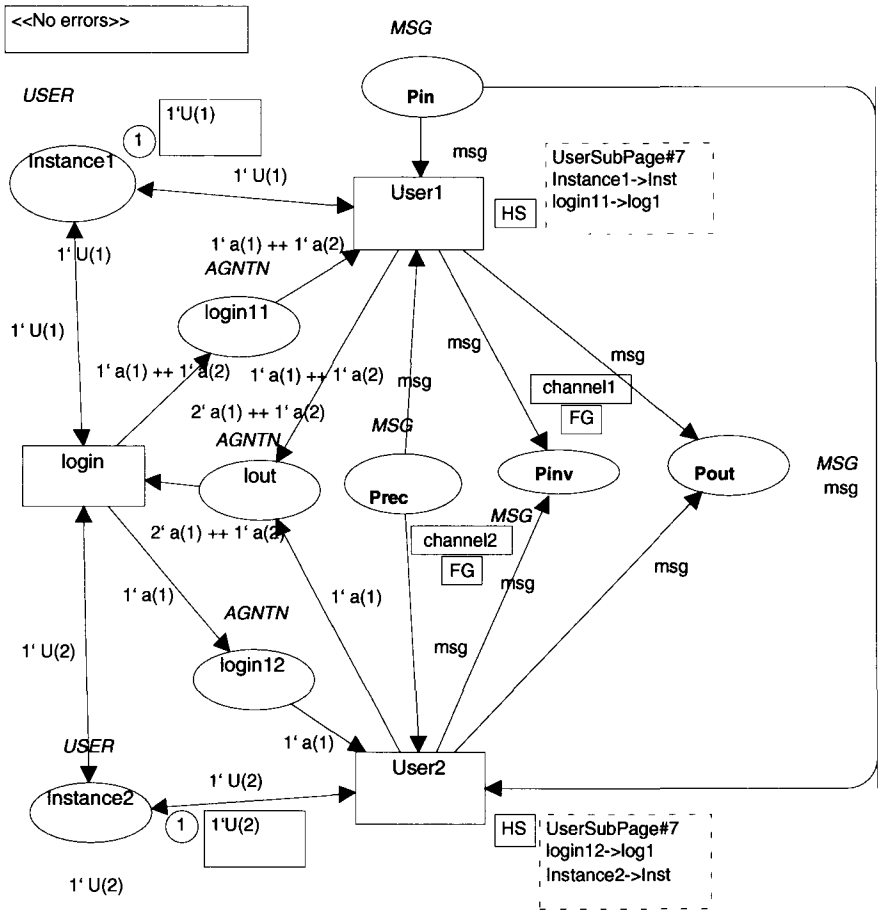

Figure 6.3: OCPN of the User Interface

of a small HS-tag adjascent to the transition. The dashed boxes beside the HS-tags define the details of the substitution and are called hierarchy inscriptions. The first line of the hierarchy inscription specifies the page that the substitution transition connects to, e. g. UserSubPage#7 in Figure 6.3. This page is called the subpage whereas the page containing the substitution transition is called the superpage. It can be seen in Figure 6.3 that the two substitution transitions share the same subpage.

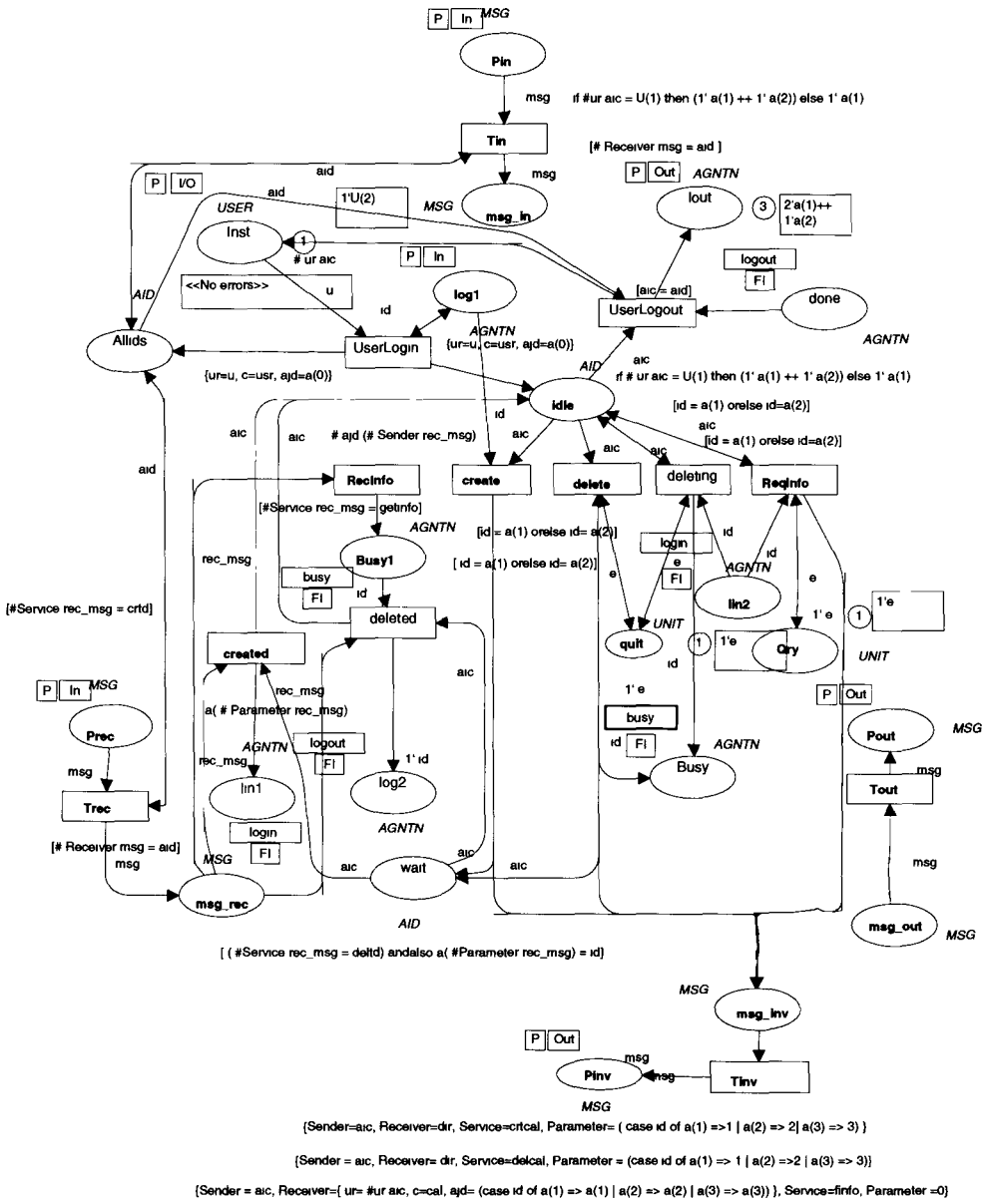

Figure 6.4: OCPN of the User Agent Class Net

This means that when the model is simulated there will be two page instances of the subpage each with their own private markings independent of the markings of the other instances. During simulation, the Design/CPN simulator has a separate window

for each page instance showing the marking of one page instance at a time and allowing the user to switch from one window to another. The remaining lines of the hierarchy inscription contain information on the port assignment and describe how a subpage is connected to the superpage. Each line relates a socket node on the superpage to a port node on the subpage. Port nodes on a subpage are identified by an In-tag, Out-tag or I/O-tag next to it. An In-tag beside a port node indicates that it must be related to a socket node on the superpage that is an input place for the substitution transition, an Out-tag indicates that the port node must have a corresponding socket node on the superpage that is an output place for the substitution transition. An I/O-tag indicates that the port has a socket node in the superpage that can be both an input as well as an output place for the substitution transition.

In Figure 6.4 the place named *log*1 is an input port that has the input socket *login*11 on the superpage for substitution transiton User1 and the input socket *login*12 for the substitution transition User2, the place *inst* is an I/O port connected to the corresponding sockets *Instance*1 and *Instance*2, the port *lout* is an output port connected to the socket named *lout* in the superpage and the ports P_{in}, P_{rec} are other examples of the input ports while P_{out} is another example of the output port. When a user logs into the system, it is represented by existence of a token in the socket node *Instance*1 (or *Instance*2) in the User Inerface net and that means a corresponding token is available in the port node *inst* in the User Agent class net. Also, occurrence of the transition *login* places tokens in the input socket *login*11 and that means these tokens are available in the input port *log*1. This causes the transition UserLogin of the User Agent Class to occur thereby placing tokens in the places $AllIDs$ and $Idle$ corresponding to the number of Local Agents a particular User Agent is allowed to spawn. A User Agent can now create a Local Agent, ask an existing Local Agent to get a particular information, delete an existing Local Agent or log out of the system. The rest of the User CLass Agnet net implements these choices.

6.1.3 LocalAgent#5

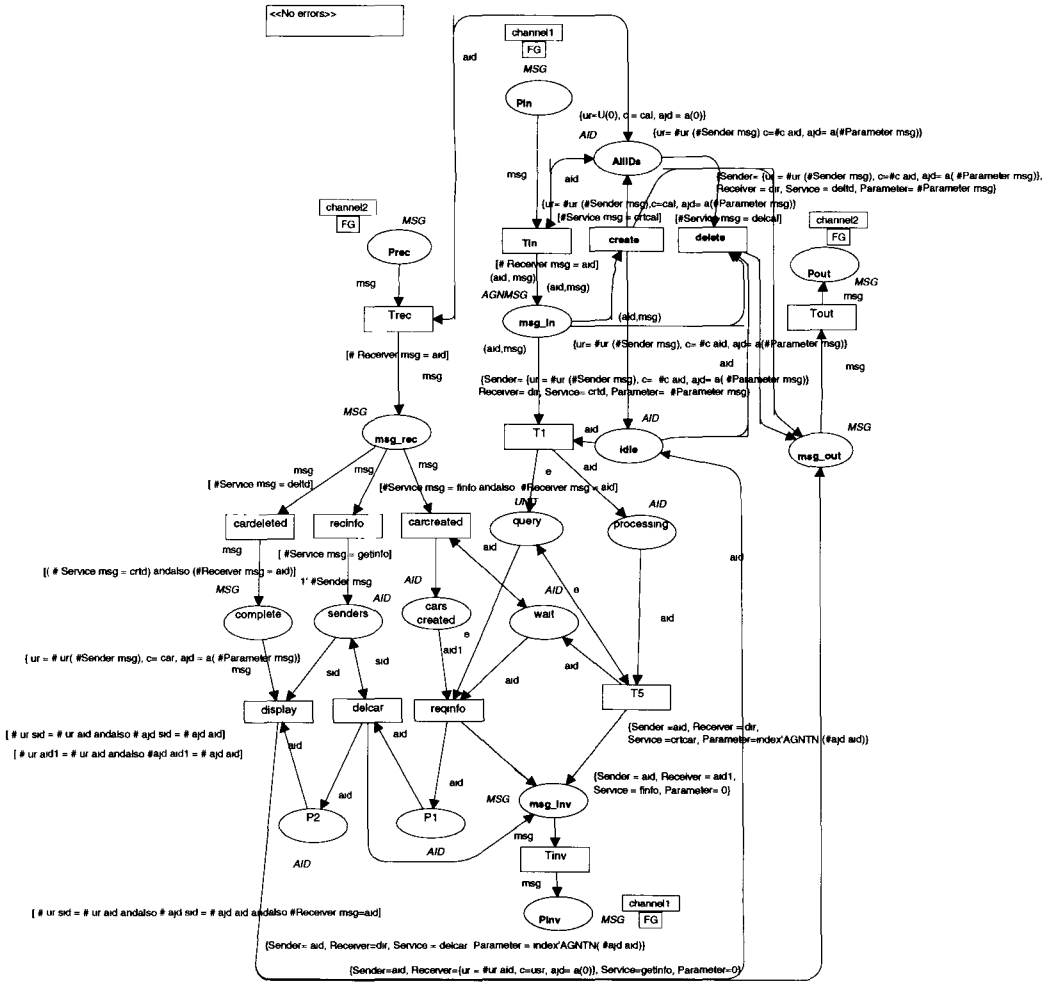

Figure 6.5: OCPN Class Net for the Local Agent

Figure 6.5 is an implementation of the Local Agent class net. The place $AllIDs$ has an initial marking of one token $\{ur = U(0), c = cal, ajd = a(0)\}$ indicating the existence of a class that is not associated to any user and with no agent instances. When the User Agent wants a Local Agent created, it requests the director to spawn it. The Director sends a message to the Local Agent class asking it to create an

instance of the Local Agent. The existence of an initial marking described above ensures that the message is received and acted upon by the Local Agent Class only, although it is broadcast on a channel shared by all agent classes in the system. The class net creates a token representating a Local Agent, places the token in the place named $Idle$ with a copy in the place $AllIDs$ and sends this agent's identifier to the User Agent through the Director. The newly created agent waits in $Idle$ state until a query is received from the User Agent when this agent moves on to the $Processing$ state and requests the Director to spawn a Remote Agent. On receipt of confirmation about the creation of a Remote Agent, this local agent communicates with it to get the information desired by the User Agent.

6.1.4 RemAgent#4

Figure 6.6 shows the Remote Agent class net. To begin with this class also has a single token $\{ur = U(0), c = car, ajd = a(0)\}$ indicating the existence of a Remote Agent class with no association with a particular user. This class creates tokens representing Remote Agents on receipt of requests from different Local Agents through the Director. A Remote Agent represented by a newly created token stays in the place $Idle$ while a message is sent to the requesting Local Agent about the Remote Agent's creation. The Local Agent sends details of the desired information to the newly created Remote Agent who gets the location of the Information Agent that provides the desired service, then communicates with it to get the information. It waits for a message from the Information Agent which could be the desired information or an error message if the information could not be found and after it passes on the contents of the message to the Local Agent, it returns to the $Idle$ state. As soon as the Local Agent gets the message from the Remote Agent, it asks the Director to delete the Remote Agent. The Director sends a message to the Remote Agent class which deletes the specified Remote Agent by removing the corresponding tokens from the places $Idle$ and $AllIDs$.

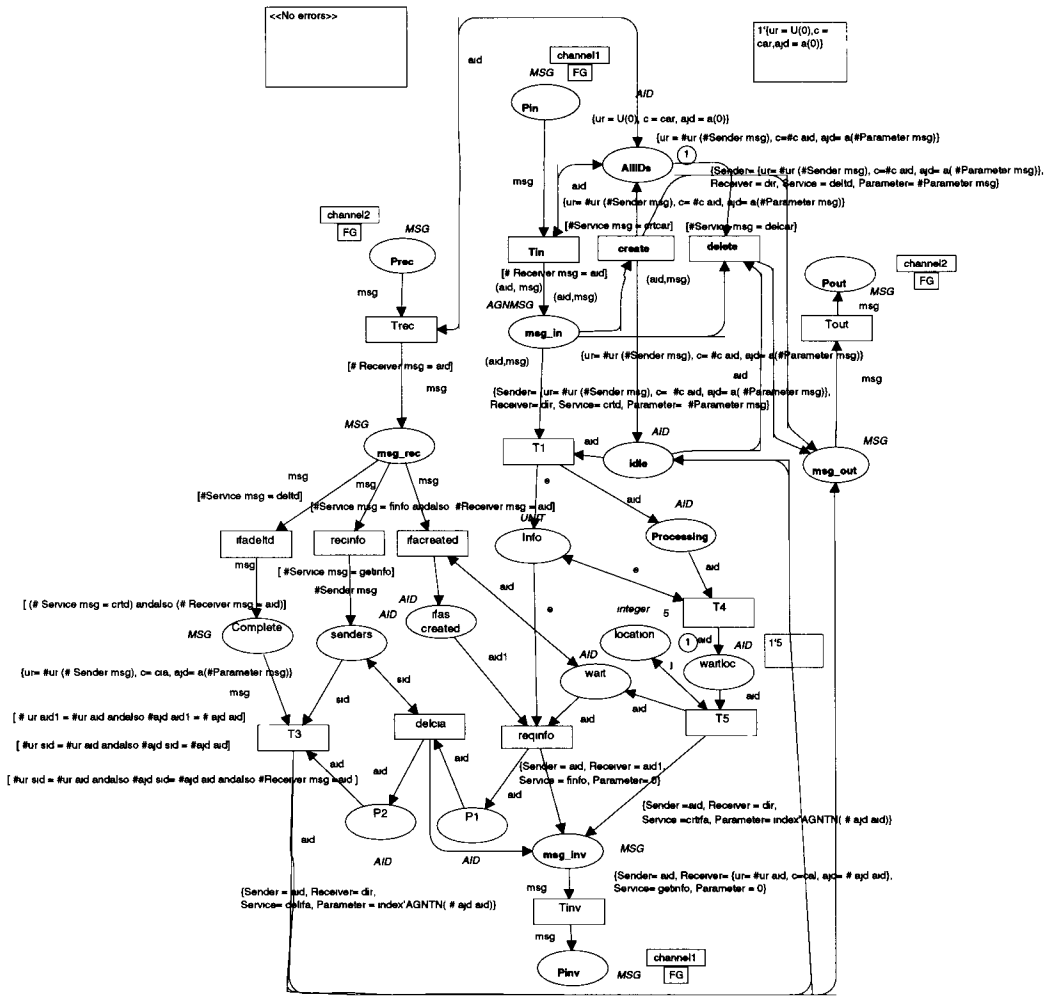

Figure 6.6: OCPN Class Net for the Remote Agent

6.1.5 InfAgent#3

Figure 6.7 is the Information Agent class net. The instances of this class interact with the source of information and based on the criteria received from the user through a Remote Agent, tries to find out the desired information. It then sends the retrieved information to the Remote Agent and if the information is not available, it sends

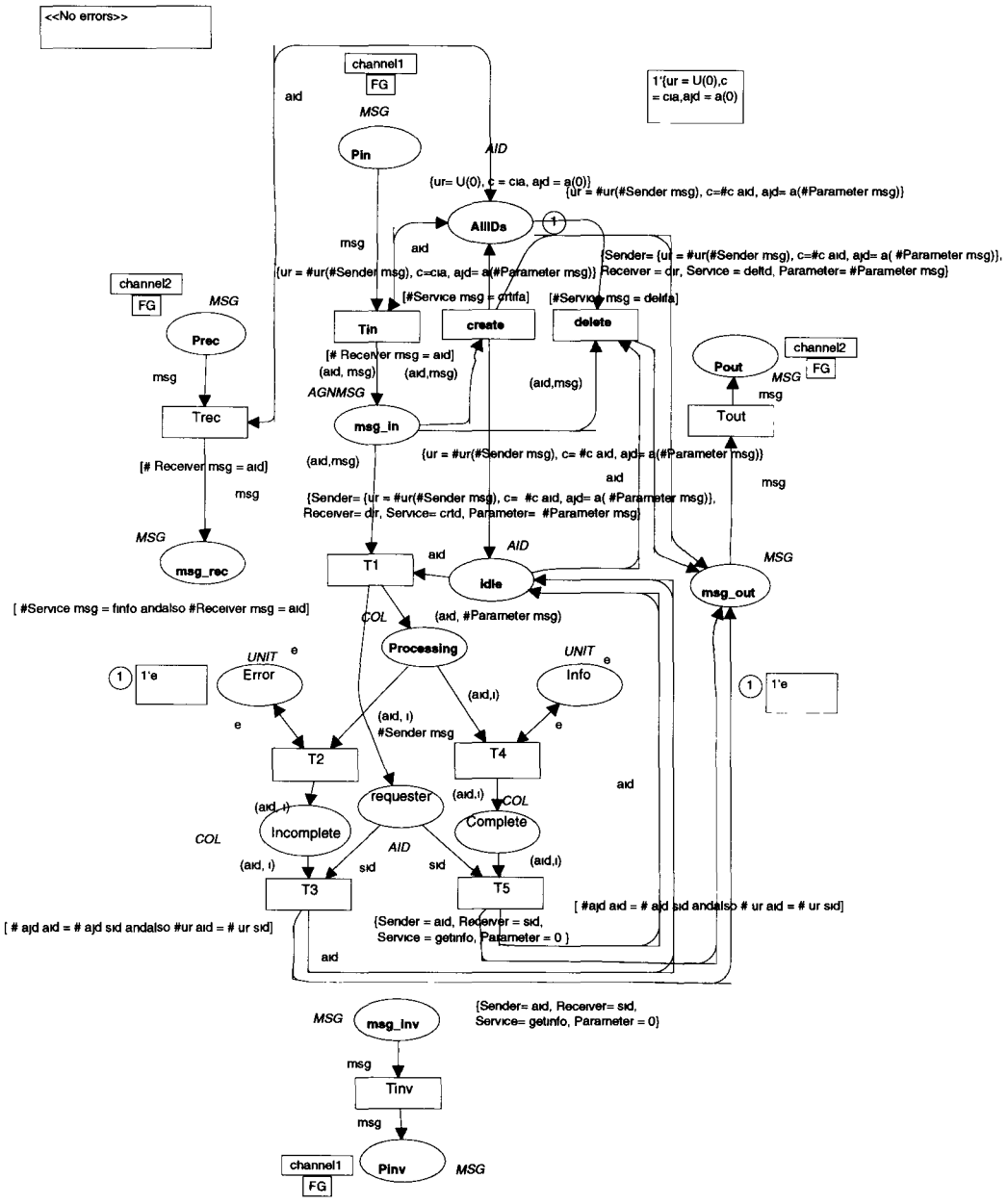

Figure 6.7: OCPN Class Net for the Information Agent

an error message back to the Remote Agent. Initially the class has just one token $\{ur = U(0), c = cia, ajd = a(0)\}$ representing the class itself. On receipt of a message

from a Remote Agent through the Director, the class creates a token representing an Information Agent and puts it in the places $Idle$ as well as $AllIDs$. The existence of a token in the place $AllIds$ ensures that a message is received by and acted upon by a specific instance of an agent represented by a token that is identified by an instance of a User Agent, an agent class and an agent number. As soon as the Remote Agent that requested the creation of the Information Agent in the first place, receives the desired information, it asks the Director to delete the Information Agent. The Director sends a message to the Information Agent class and the agent is deleted by removal of specific tokens identifying a particular agent from the places $Idle$ and $AllIDs$.

6.1.6 Director#2

The agent class net that represents the Director is shown in Figure 6.8. This agent coordinates the activities of all other agents. It also has just one token $\{ur = U(0), c = cdr, ajd = a(0)\}$ when the system first starts. Whenever an agent needs the services of another agent it requests the Director to spawn such an agent. An agent that is no longer required is also deleted through the Director. The Director keeps a record of all agents present in the system at a given time by storing a token corresponding to an agent when it is created at the place $AllIDs$ and by removing the corresponding token when an agent is deleted. It plays a crucial role in keeping track of the associations of different agents in the system. It ensures, on the creation of a new agent, that it has a unique identifier that associates the agent with a particular User Agent and a particular Local Agent working for this User Agent. It also ensues that when a request to delete an agent is received the specific agent is suitably identified by the class responsible for deleting the agent.

6.1.7 Communication Channels

Communication between different agents is implemented using the guidelines discussed in Chapter 5. The places P_{in} and P_{inv} of all class nets are members of a place

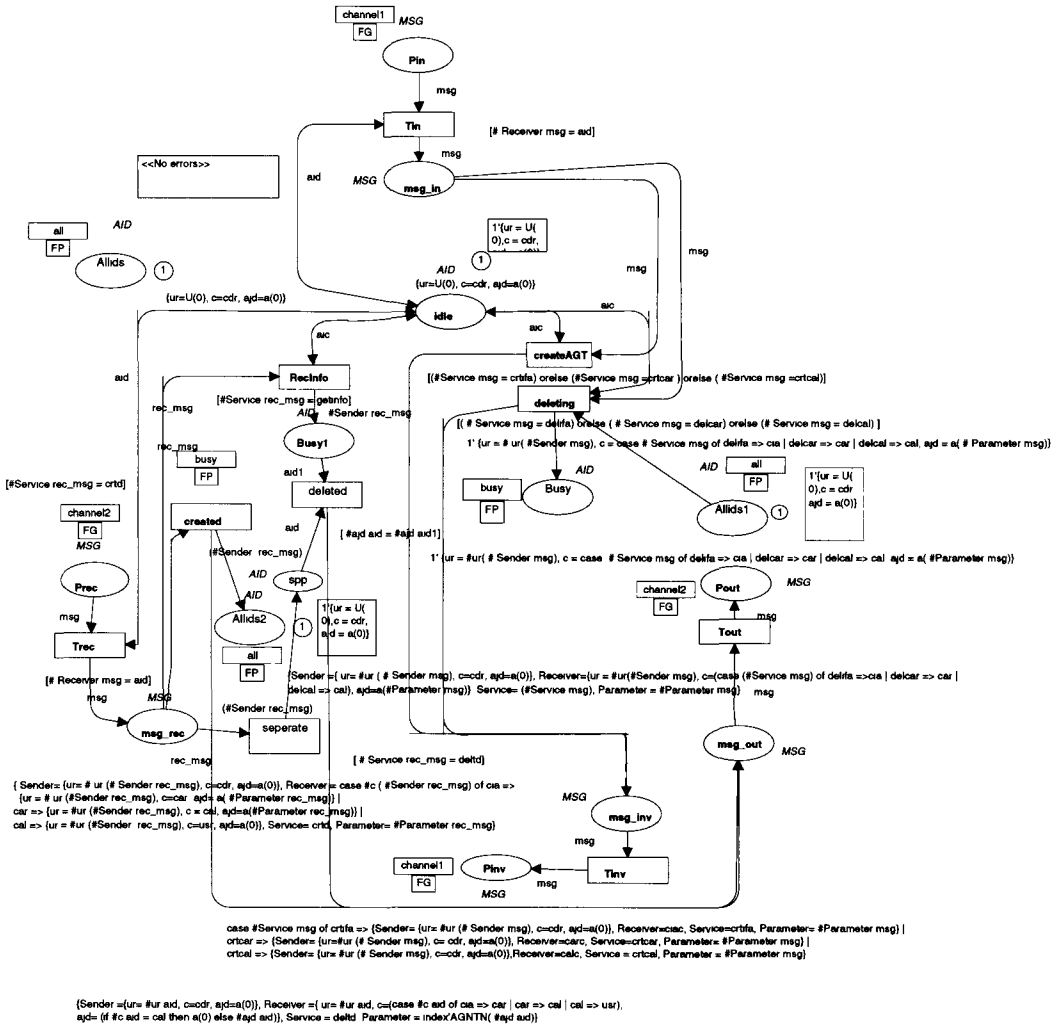

Figure 6.8: OCPN Class Net for the Director Agent

fusion set named *channel*1 and the places P_{out} and P_{rec} constitute a fusion set named *channel*2. Simulations were performed using the Design/CPN simulator to validate the model and to conduct initial analysis. First the interactive (single step) simulation was used to investigate if an execution of the model results in desired states.

Initial markings were changed to simulate more than one user with each user having one or more Local Agents. The simulator randomly picks up which user takes what action and graphically displays markings after each step. This provided insight into the operation of the system. After the model was validated, the behaviour of the system was investigated using automatic simulation. The number of steps for automatic simulation can be selected for its termination and were gradually increased to 1 million steps in order to check repeatability of operations.

6.2 Analysis of the Model

State space methods are one of the main approaches for checking correctness of a concurrent system. They are suitable for automatic analysis and verification of a system's behaviour. In their basic form, they construct a structure that consists of all states that a system can reach, and all transitions that the system can make between those states. This structure is called a state space, occurrence graph, reachability graph or reachability tree. The Occurrence Graph tool of Design/CPN allows a fully automated construction and analysis of the occurrence graph of a CP net. It is possible to specify whether a complete occurrence graph is to be constructed or or only a part of it, by using stop and branching criteria. The stop criteria is used to stop the construction of the graph when a certain amount of real time has been used or when a certain number of nodes have been constructed. The branching criteria imply that it is possible to develop some of the immediate successors of a given marking if desired. Once an occurrence graph has been constructed, the tool generates a standard report providing information about all standard CPN properties.

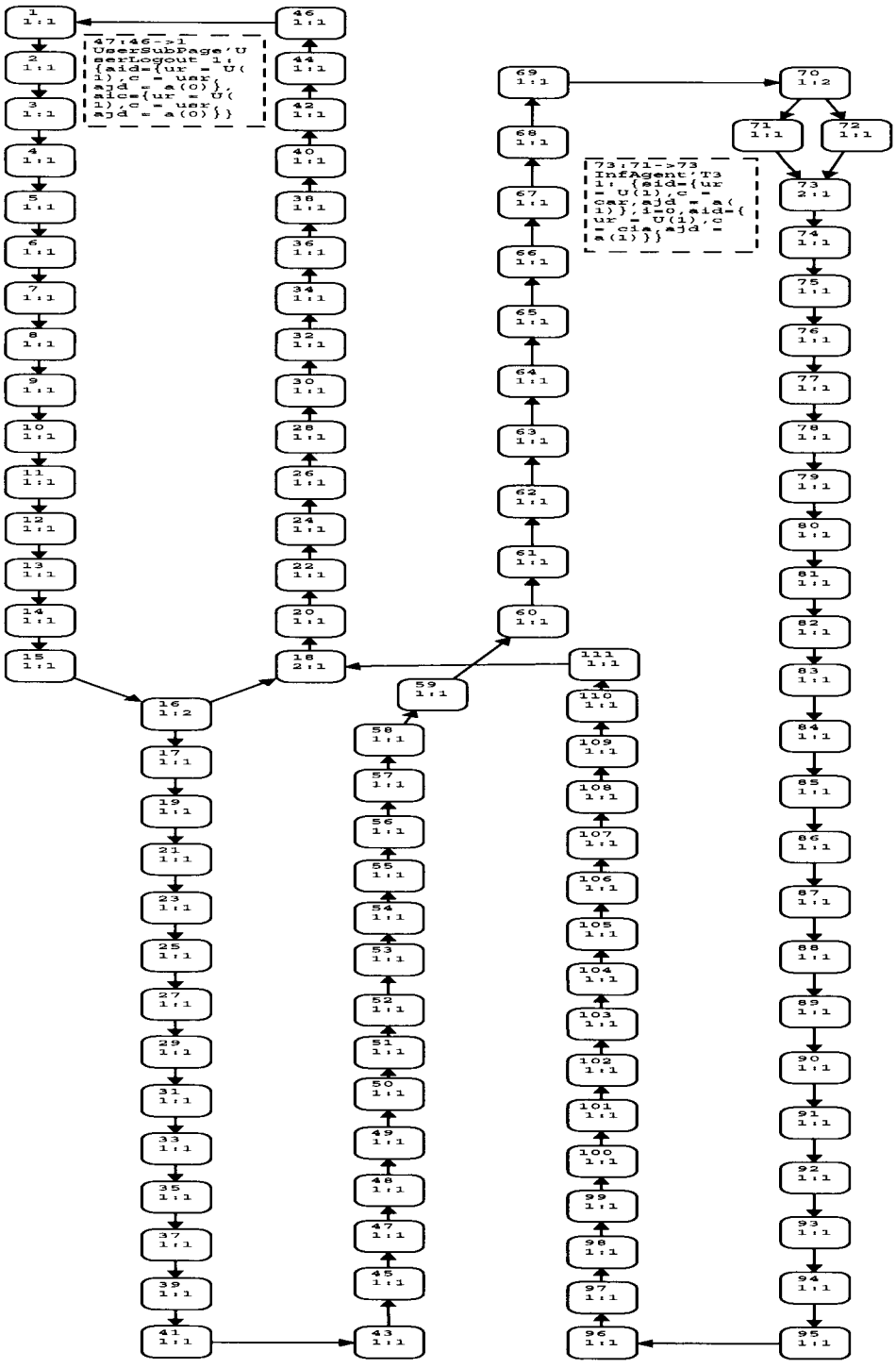

Figure 6.9: Occurrence Graph of the Basic Model

6.3 Generation of the Occurrence Graphs

A number of occurrence graphs corresponding to different initial conditions were generated, however here we discuss four cases:

1. the basic case, where it is assumed that there is one user and only one agent of each type is spawned in the system.

2. the second case extends the basic case by allowing the user to spawn two agents of each type in the system.

3. the third case extends the basic case by allowing two users to log in concurrently and have one agent of each type in the system.

4. the fourth case corresponds to one user with three agents of each type.

Figure 6.9 shows the occurrence graph for the basic case. There are 111 reachable markings (or nodes), represented by rounded boxes in the figure. Each marking has an identification number located at the top. Also, there are two numbers separated by a colon that represent the number of input and output arcs, respectively. The node that has the identification number 1 in the figure is the initial marking. It is possible to see details of each marking if desired, in which case the details relating to a particular node are shown in a dashed box next to it. The details of binding elements, represented by arcs of the graph, can also be shown on the arcs, which we have done for some of them in Figure 6.9. Also included is the identification number of the arc (located in the upper left corner) and the related node numbers ($n1 \rightarrow n2$, indicates that the marking $n2$ is reached from marking $n1$ when the binding element occurs). For example, the binding element 47, represented by arc number 47 in the figure shows a transition from the current markings represented by the node 46 to those represented by the node 1.

The occurrence graph for the second case has 11,258 reachable markings and although we were able to construct a full occurrence graph we show only the initial

and the final part of the graph in Figure 6.10 as the complete graph is too large and complex for a meaningful presentation here. We calculated the full occurrence graph for the third case also, which has 16,491 reachable markings and its starting and ending part is shown in Figure 6.11. A full occurrence graph for case 4 could not be generated due to limitations of the software system and/or the computer system available for this research.

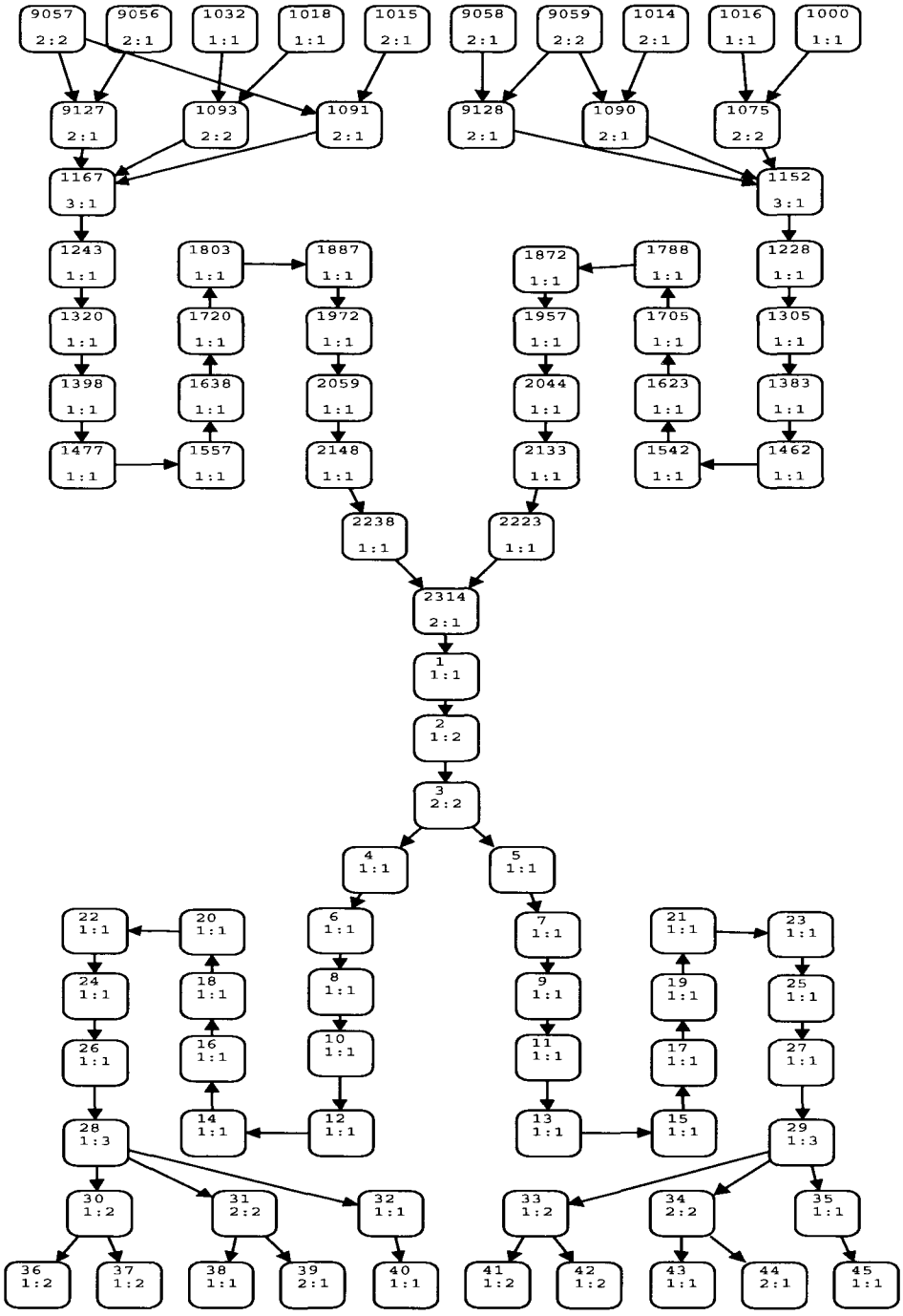

Figure 6.10: Part of the Occurrence Graph (One User, Two Agents of Each Type)

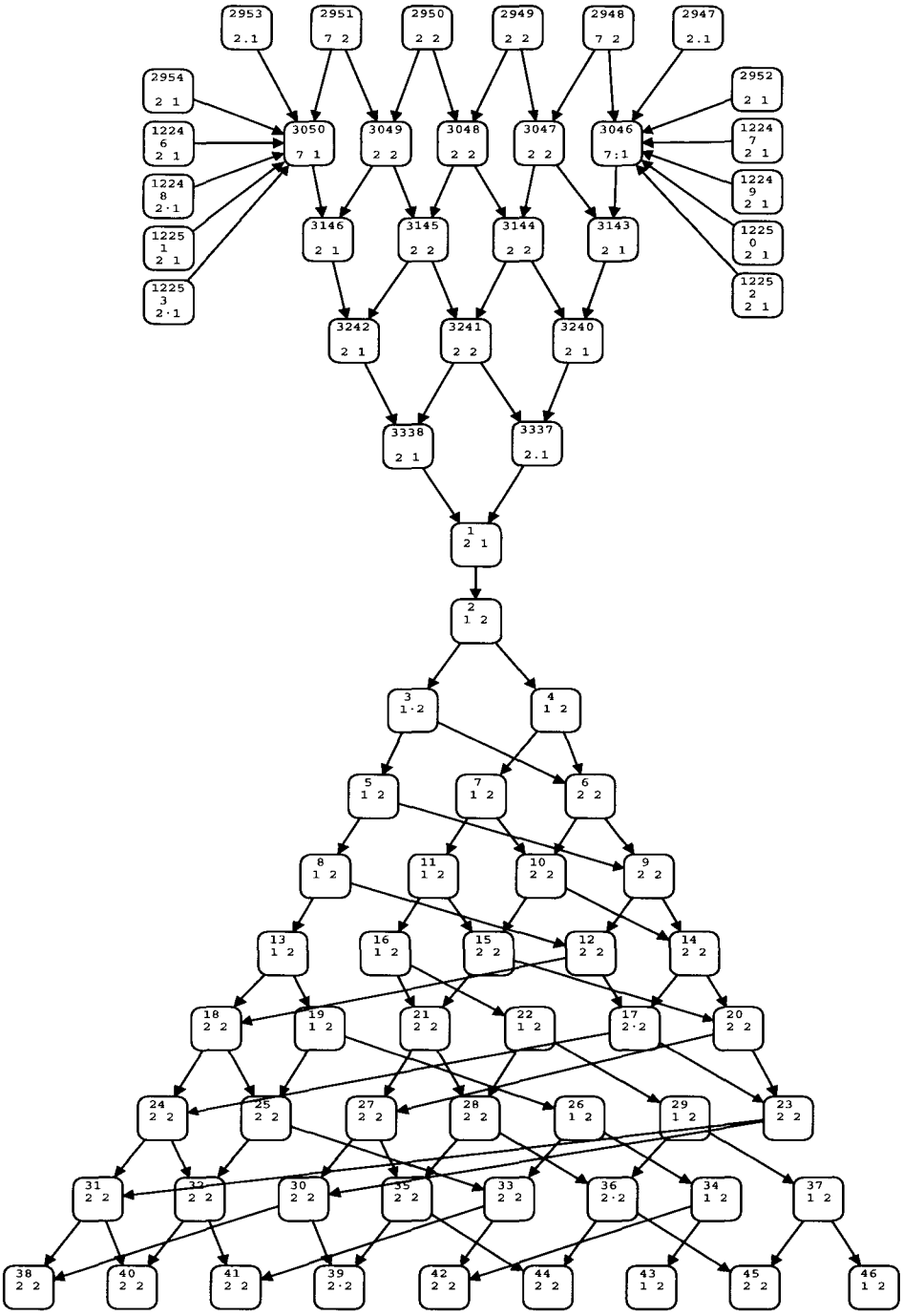

Figure 6.11: Part of the Occurrence Graph (Two Users, One Agent of Each Type)

6.3.1 Strongly Connected Components

A strongly connected component (of a directed graph) is a maximal subgraph in which all nodes are reachable from each other [82]. It is possible to calculate a strongly connected component SCC) graph for every occurrence graph generated in Design/CPN. Such a graph has a node for each strongly connected component and an arc for each occurrence graph arc starting in one component and ending in another. An SCC without incoming arcs is called the initial SCC, and a SCC without outgoing arcs is called a terminal SCC. For a cyclic CP net the SCC-graph is usually much smaller and often consists of only one node while the occurrence graph may consist of several thousand nodes. Some of the behavioural properties can be investigated very efficiently by using the SCC-graph. For example the existence of home markings can be determined by counting the number of terminal SCCs. By checking whether a transition instance appears in each terminal SCC, it can be determined if the instance is live or not. The model under discussion is completely cyclic as depicted by just one SCC component for various occurrence graphs constructed. This means that all mrkings including the initial marking are home markings as discussed further in section 6.5.3.

6.4 Behavioural Properties of the Model

Behavoural properties that describe the expected behaviour of the model are introduced in an informal way in this section. Formal definitions and further details can be found in [82]. Once the occurrence graph and the associated SCC-graph have been generated by the Occurrence Graph Tool of Design/CPN, a standard report can be generated. This report consists of a text file that contains key information about the occurrence graph and the dynamic properties that can be deduced from it. We now present different parts of the reports for the three occuurrence graphs discussed above. In oder to save space, essential features of the three reports have been combined in a

tabular form.

6.4.1 Statistics

This part of the report contains statistical information about the size of the occurrence graph and the size of the SCC-graph and is shown below:

```
Statistics
----------------------------------------------------------------------

Case 1              Case 2              Case 3              Case 4
Occurrence Graph    Occurrence Graph    Occurrence Graph    Occurrence Graph
 Nodes:  111         Nodes:  11258       Nodes:  16941       Nodes:886729
 Arcs:   113         Arcs:   22534       Arcs:   34483       Arcs:2615164
 Secs:   0           Secs:   23          Secs:   35          Secs:54160
 Status: Full        Status: Full        Status: Full        Status:Partial

Scc Graph           Scc Graph           Scc Graph           Scc Graph
 Nodes:  1           Nodes:  1           Nodes:  1           N/A
 Arcs:   0           Arcs:   0           Arcs:   0           N/A
 Secs:   0           Secs:   2           Secs:   2           N/A
```

It can be seen from this part of the report that we have calulated the entire graph in less than a second for case 1 and it contains 111 nodes and 113 arcs and that there is only one strongly connected component. This shows that every marking can be reached from any other marking in the model. For case 2, the occurrence graph has 11,258 nodes and 22,534 arcs and the graph for the case 3 consists of 16,941 nodes with 34,483 arcs. The SCC-graph for these two cases also has only one node. For case 4 only a partial graph was generated with 886,729 nodes and 2,615,164 arcs in 54,160 seconds, however the report generated for this case did confirm that all places

in the OCPN model for this case also were within the expected bounds as can be seen in part of the report discussed in the next section.

6.4.2 Boundedness

The next part of the occurrence graph report deals with boundedness properties that characterise the finiteness of the state space. The bound of a place is the maximum number of tokens that it may contain. A place is bounded if its bound is finite. A Petri net model is bounded if each place in it is bounded [79]. The report shows the best upper and lower integer bounds as well as the best upper and lower multiset bounds. The upper and lower integer bounds indicate the maximal and minimal number of tokens at individual places as shown in Tables 6.1, 6.2 and 6.3 below:

Table 6.1: Boundedness Properties

	Case 1		Case 2		Case 3		Case 4	
Best Integers Bounds	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower
Director'AllIDs 1	4	1	7	1	7	1	10	1
Director'AllIDs1 1	4	1	7	1	7	1	10	1
Director'AllIDs2 1	4	1	7	1	7	1	10	1
Director'Busy 1	1	0	2	0	2	0	3	0
Director'Busy1 1	1	0	2	0	2	0	3	0
Director'Pin 1	1	0	2	0	2	0	3	0
Director'Pinv 1	1	0	2	0	2	0	3	0
Director'Pout 1	1	0	2	0	2	0	3	0
Director'Prec 1	1	0	2	0	2	0	3	0
Director'idle 1	1	1	1	1	1		1	1
Director'msg_in 1	1	0	2	0	2	0	3	0
Director'msg_inv 1	1	0	2	0	2	0	3	0
Director'msg_out 1	1	0	2	0	2	0	3	0
Director'msg_rec 1	1	0	2	0	2	0	3	0
Director'spp 1	1	0	2	0	2	0	3	0
InfAgent'AllIDs 1	2	1	3	1	3	1	4	1
InfAgent'Complete 1	1	0	2	0	2	0	3	0
InfAgent'Error 1	1	1	1	1	1	1	1	1
InfAgent'Incomplete 1	1	0	2	0	2	0	3	0
InfAgent'Info 1	1	1	1	1	1	1	1	1
InfAgent'Pin 1	1	0	2	0	2	0	3	0
InfAgent'Pinv 1	1	0	2	0	2	0	3	0
InfAgent'Pout 1	1	0	2	0	2	0	3	0
InfAgent'Prec 1	1	0	2	0	2	0	3	0
InfAgent'Processing 1	1	0	2	0	2	0	3	0
InfAgent'idle 1	1	0	2	0	2	0	3	0
InfAgent'msg_in 1	1	0	2	0	2	0	3	0

Table 6.2: Boundedness Properties Continued ..

	Case 1		Case 2		Case 3		Case 4	
Best Integers Bounds	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower
InfAgent'msg_inv 1	0	0	2	0	2	0	0	0
InfAgent'msg_out 1	1	0	2	0	2	0	3	0
InfAgent'msg_rec 1	0	0	0	0	0	0	0	0
InfAgent'requester 1	1	0	2	0	2	0	3	0
LocAgent'AllIDs 1	2	1	3	1	3	1	4	1
LocAgent'P1 1	1	0	2	0	2	0	3	0
LocAgent'P2 1	1	0	2	0	2	0	2	0
LocAgent'Pin 1	1	0	2	0	2	0	3	0
LocAgent'Pinv 1	1	0	2	0	2	0	3	0
LocAgent'Pout 1	1	0	2	0	2	0	3	0
LocAgent'Prec 1	1	0	2	0	2	0	3	0
LocAgent'cars 1	1	0	2	0	2	0	3	0
LocAgent'complete 1	1	0	2	0	2	0	1	0
LocAgent'idle 1	1	0	2	0	2	0	3	0
LocAgent'msg_in 1	1	0	2	0	2	0	3	0
LocAgent'msg_inv 1	1	0	2	0	2	0	3	0
LocAgent'msg_out 1	1	0	2	0	2	0	3	0
LocAgent'msg_rec 1	1	0	2	0	2	0	3	0
LocAgent'processing 1	1	0	2	0	2	0	3	0
LocAgent'query 1	1	0	2	0	2	0	3	0
LocAgent'senders 1	1	0	2	0	2	0	2	0
LocAgent'wait 1	1	0	2	0	2	0	3	0
UserSubPage'AllIDs 1	1	0	1	0	1	0	1	0
UserSubPage'AllIDs 2	0	0	0	0	1	0	1	0
UserSubPage'Busy 1	1	0	2	0	1	0	2	0
UserSubPage'Busy1 1	1	0	2	0	1	0	2	0
UserSubPage'Busy1 2	0	0	0	0	1	0	1	0
UserSubPage'Busy 2	0	0	0	0	1	0	1	0
UserSubPage'Qry 1	1	1	1	1	1	1	1	1
UserSubPage'Qry 2	1	1	1	1	1	1	1	1
UserSubPage'done 1	1	0	2	0	1	0	2	0
UserSubPage'done 2	0	0	0	0	1	0	1	0
UserSubPage'idle 1	1	0	1	0	1	0	1	0
UserSubPage'idle 2	0	0	0	0	1	0	1	0
UserSubPage'lin1 1	1	0	2	0	1	0	2	0
UserSubPage'lin1 2	0	0	0	0	1	0	1	0
UserSubPage'lin2 1	1	0	2	0	1	0	2	0
UserSubPage'lin2 2	0	0	0	0	1	0	1	0
UserSubPage'log2 1	1	0	2	0	1	0	2	0
UserSubPage'log2 2	0	0	0	0	1	0	1	0
UserSubPage'msg_in 1	0	0	0	0	0	0	0	0
UserSubPage'msg_in 2	0	0	0	0	0	0	0	0
UserSubPage'msg_inv 1	1	0	2	0	1	0	2	0
UserSubPage'msg_inv 2	0	0	0	0	1	0	1	0
UserSubPage'msg_out 1	0	0	0	0	0	0	0	0
UserSubPage'msg_out 2	0	0	0	0	0	0	0	0
UserSubPage'msg_rec 1	1	0	2	0	1	0	2	0
UserSubPage'msg_rec 2	0	0	0	0	1	0	1	0

Table 6.3: Boundedness Properties Continued ...

	Case 1		Case 2		Case 3		Case 4	
Best Integers Bounds	Upper	Lower	Upper	Lower	Upper	Lower	Upper	Lower
UserSubPage'quit 1	1	1	1	1	1	1	1	1
UserSubPage'quit 2	1	1	1	1	1	1	1	1
UserSubPage'wait 1	1	0	1	0	1	0	1	0
UserSubPage'wait 2	0	0	0	0	1	0	1	0
RemAgent'AllIDs 1	2	1	3	1	3	1	4	1
RemAgent'Complete 1	1	0	2	0	2	0	2	0
RemAgent'Info 1	1	0	2	0	2	0	3	0
RemAgent'P1 1	1	0	2	0	2	0	3	0
RemAgent'P2 1	1	0	2	0	2	0	2	0
RemAgent'Pin 1	1	0	2	0	2	0	3	0
RemAgent'Pinv 1	1	0	2	0	2	0	3	0
RemAgent'Pout 1	1	0	2	0	2	0	3	0
RemAgent'Prec 1	1	0	2	0	2	0	3	0
RemAgent'Processing 1	1	0	2	0	2	0	3	0
RemAgent'idle 1	1	0	2	0	2	0	3	0
RemAgent'ifas 1	1	0	2	0	2	0	3	0
RemAgent'location 1	1	1	1	1	1	1	1	1
RemAgent'msg_in 1	1	0	2	0	2	0	3	0
RemAgent'msg_inv 1	1	0	2	0	2	0	3	0
RemAgent'msg_out 1	1	0	2	0	2	0	3	0
RemAgent'msg_rec 1	1	0	2	0	2	0	3	0
RemAgent'senders 1	1	0	2	0	2	0	2	0
RemAgent'wait 1	1	0	2	0	2	0	3	0
RemAgent'waitloc 1	1	0	2	0	2	0	3	0
usr'Instance1 1	1	0	1	0	1	0	1	0
usr'Instance2 1	1	1	1	1	1	0	1	0
usr'Pin 1	0	0	0	0	0	0	0	0
usr'Pinv 1	1	0	2	0	2	0	3	0
usr'Pout 1	0	0	0	0	0	0	0	0
usr'Prec 1	1	0	2	0	2	0	3	0
usr'login11 1	1	0	2	0	1	0	2	0
usr'login12 1	0	0	0	0	1	0	1	0
usr'lout 1	1	0	2	0	2	0	3	0

It can be seen that the place $AllIDs$ in the Director class net has 1 to 4 tokens in case 1: 1 token representing the Director Class and the 3 tokens representing the Local Agent, the Remote Agent and the Information Agent spawned by the Director. This matches with the corresponding information at the places $AllIDs$ of InfAgent, LocAgent and RemAgent each of which has 1 to 2 tokens: 1 token representing the corresponding class net and the other representing the instance of an agent. The same place has 1 to 7 tokens in case 2 as well as case 3: 1 token for the Director class and the remaining 6 tokens for 2 agent instances of each type matching the

munber of tokens representing instances of InfAgent, LocAgent and RemAgent at their respective places named as $AllIDs$. It is also noted that although there are two instances of UserSubPage with corresponding places $AllIDs1$ and $AllIDs2$, the place $AllIDs2$ is not active as it shows a zero token in both the upper and the lower integer bounds both in case 1 as well as case 2 however, it shows 1 token in case 3. This is expected because only one user is logged in for the occurrence graph in case 1 and 2 but 2 users are concurrently logged in for the occurrence graph in case 3. It is also noted that most of the places in case 2 and case 3 have identical number of tokens except for the places in the UserSubPage. This shows that most of the interactions in the model are symmetrical. Finally in case 4 the place $AllIDs$ in the Director class net has 1 to 10 tokens, which is exactly as expected as there are 9 agents of different types in the system for this case and one token is for the Director class net. Although it was not possible to construct the full occurrence graph in this case, it is clear that the system behaviour is in line with expectations and all places are bounded since the maximum number of tokens that can be present at $AllIDs$ place of the Director class net is equal to $3n + 1$ where n is the number of local agent instances in the system.

The multiset bounds provide information about the values of the tokens that the places can carry. In order to save space, only the multiset bounds for the place $AllIDs$ of the Director are shown below; complete details are available in [109].

```
Case 1
  Best Upper Multi-set Bounds
  Director'AllIDs 1   1`{ur = U(0),c = cdr,ajd = a(0)}++
                      1`{ur = U(1),c = car,ajd = a(1)}++
                      1`{ur = U(1),c = cal,ajd = a(1)}++
                      1`{ur = U(1),c = cia,ajd = a(1)}
Case 2
  Best Upper Multi-set Bounds
  Director'AllIDs 1  1`{ur = U(0),c = cdr,ajd = a(0)}++
```

```
                    1`{ur = U(1),c = car,ajd = a(1)}++
                    1`{ur = U(1),c = car,ajd = a(2)}++
                    1`{ur = U(1),c = cal,ajd = a(1)}++
                    1`{ur = U(1),c = cal,ajd = a(2)}++
                    1`{ur = U(1),c = cia,ajd = a(1)}++
                    1`{ur = U(1),c = cia,ajd = a(2)}
Case 3
  Best Upper Multi-set Bounds
  Director'AllIDs 1  1`{ur = U(0),c = cdr,ajd = a(0)}++
                    1`{ur = U(1),c = car,ajd = a(1)}++
                    1`{ur = U(1),c = cal,ajd = a(1)}++
                    1`{ur = U(1),c = cia,ajd = a(1)}++
                    1`{ur = U(2),c = car,ajd = a(1)}++
                    1`{ur = U(2),c = cal,ajd = a(1)}++
                    1`{ur = U(2),c = cia,ajd = a(1)}
Case 1, 2 and 3
  Best Lower Multi-set Bounds
  Director'AllIDs 1   1`{ur = U(0),c = cdr,ajd = a(0)}
```

The upper multi-set bound of a place is defined as the smallest multi-set which is larger than or equal to all reachable markings of the place. The lower multi-set bound of a place is defined as the largest multi-set which is smaller than or equal to all reachable markings of the place [82]. In the model under discussion, the places that have AID as their colour have tokens that represent instances of agents and the places that have MSG as their colour have tokens that represent instances of different messages only. The lower multiset bounds show tokens only in those places that have initial markings. This information was very useful while debugging as it gives details of a particular instance of an agent or a message that may be causing some problem. These tokens can then be traced in different places to pin point and remove the bug.

6.4.3 Home Markings

A home marking is a marking that can be reached from all other reachable markings. The existence of a home marking indicates that it is *possible* to reach the home marking although it is not guaranteed. This means there may exist infinite occurrence sequences which do not contain the home marking. In the case of our model, all markings are home markings as shown in the next part of the report below:

```
Home Properties
------------------------------------------------------------------------

                  Case 1          Case 2          Case 3
  Home Markings:  All             All             All
```

This in fact is a much stronger property as it is not only *possible* to return to the initial marking, it will *always* happen. This means that when a user initiates a query it will always result in some message returned to the user. This establishes the soundness of the system.

6.4.4 Reversibility

This property characterises the recoverability of the initial marking from any reachable marking. Since the system has all markings as the home markings, it means the initial marking is also a home marking and the model is reversible, so it is always possible to return to the initial marking.

6.4.5 Deadlock-Freeness

The liveness property indicates that it is possible for each reachable marking to find an occurrence sequence starting at that marking. In other words, the transition never loses the possibility to fire. The liveness condition is stronger than deadlock-freeness and guarantees that there are no deadlocks. As shown in the next part of

the occurrence graph report, the analysis of our model shows that there are no dead markings or dead transitions.

```
Liveness Properties
------------------------------------------------------------------------

                                Case 1      Case2       Case3
  Dead Markings:                None        None        None
  Dead Transitions Instances:   None        None        None
  Live Transitions Instances:   All         All         All
```

6.4.6 Fairness

The final part of the report in Table 6.4 shows that all of the transitions in the model were either *impartial* or *fair* or *just*. Transitions that are not constrained to occur are *impartial*. In case of a conflict it is necessary to ensure that all conflicting transitions eventually occur, if the conflict appears again and again. *Fair* transitions eventually occur in an infinite occurrence sequence if they are enabled infinitely often. *Just* transitions eventually occur in an infinite occurrence sequence if they are persistently enabled [79].

6.4.7 State Space Explosion Problem

Although one of the main approaches to checking the correctness of a concurrent system is the space state method, it has a fundamental problem that almost any practical system of reasonable size results in a huge number of states. Often the size of state space tends to grow exponentially in the number of its processes and variables. Limiting the state space explosion is the topic of much research, however as pointed out by Jensen [110] although the occurrence graph of a CP-net grows very fast with an increase in the size of the colourseets involved, in practice it is often sufficient to consider rather small colour sets in order to verify the logical correctness

Table 6.4: Fairness Properties

Transition	Fairness	Transition	Fairness
Director'RecInfo 1	Fair	Director'Tin 1	Impartial
Director'Tinv 1	Impartial	Director'Tout 1	Impartial
Director'Trec 1	Impartial	Director'createAGT 1	Impartial
Director'created 1	Impartial	Director'deleted 1	Impartial
Director'deleting 1	Impartial	Director'seperate 1	Impartial
InfAgent'T1 1	Fair	InfAgent'T2 1	Just
InfAgent'T3 1	Fair	InfAgent'T4 1	Just
InfAgent'T5 1	Fair	InfAgent'Tin 1	Fair
InfAgent'Tinv 1	Fair	InfAgent'Tout 1	Fair
InfAgent'Trec 1	Fair	InfAgent'create 1	Fair
InfAgent'delete 1	Fair	LocAgent'T1 1	Fair
LocAgent'T5 1	Fair	LocAgent'Tin 1	Impartial
LocAgent'Tinv 1	Fair	LocAgent'Tout 1	Impartial
LocAgent'Trec 1	Fair	LocAgent'carcreated 1	Fair
LocAgent'cardeleted 1	Fair	LocAgent'create 1	Impartial
LocAgent'delcar 1	Fair	LocAgent'delete 1	Impartial
LocAgent'display 1	Fair	LocAgent'recinfo 1	Fair
LocAgent'reqinfo 1	Fair	UserSubPage'RecInfo 1	Fair
UserSubPage'RecInfo 2	Fair	UserSubPage'ReqInfo 1	Just
UserSubPage'ReqInfo 2	Fair	UserSubPage'Tin 1	Fair
UserSubPage'Tin 2	Fair	UserSubPage'Tinv 1	Impartial
UserSubPage'Tinv 2	Fair	UserSubPage'Tout 1	Fair
UserSubPage'Tout 2	Fair	UserSubPage'Trec 1	Impartial
UserSubPage'Trec 2	Fair	UserSubPage'UserLogin 1	Impartial
UserSubPage'UserLogin 2	Fair	UserSubPage'UserLogout 1	Impartial
UserSubPage'UserLogout 2	Fair	UserSubPage'create 1	Impartial
UserSubPage'create 2	Fair	UserSubPage'created 1	Impartial
UserSubPage'created 2	Fair	UserSubPage'delete 1	Impartial
UserSubPage'delete 2	Fair	UserSubPage'deleted 1	Impartial
UserSubPage'deleted 2	Fair	UserSubPage'deleting 1	Just
UserSubPage'deleting 2	Fair	RemAgent'T1 1	Fair
RemAgent'T3 1	Fair	RemAgent'T4 1	Fair
RemAgent'T5 1	Fair	RemAgent'Tin 1	Fair
RemAgent'Tinv 1	Fair	RemAgent'Tout 1	Fair
RemAgent'Trec 1	Fair	RemAgent'create 1	Fair
RemAgent'delcia 1	Fair	RemAgent'delete 1	Fair
RemAgent'ifacreated 1	Fair	RemAgent'ifadeltd 1	Fair
RemAgent'recinfo 1	Fair	RemAgent'reqinfo 1	Fair
usr'login 1	Impartial		

of a given CP-net. Having convinced ourselves that the EPSS model has the correct behaviour for the 3 to 9 agents active concurrently in the different cases discussed in this chapter, there is a strong likelihood that the design also works correctly for any larger number of agents. This, in fact proved correct when a prototype of the model was constructed and then tested with 50 agents concurrently active in the system

without any problems.

6.5 Summary

In this chapter, we presented details of the OCPN model of EPSS for simulation. Relevant details of various occurrence graphs and reports generated by Design/CPN were discussed. This analysis showed that the system has desirable properties such as boundedness, reversibility, liveness, deadlock-freeness, and fairness. This capability provides another important contribution of this research by bringing dynamic analysis to the design phase in the software development cycle.

In the next chapter we discuss some details related to implementation of a prototype.

Chapter 7

Implementation of Prototype

7.1 Selection of the Middleware

Much research activity related to software agents is reported in the current literature. Most of the researchers have produced some sort of multi-agent framework to act as a test bed for their ideas. Although a number of commercial agent construction tools are available we give a partial list of research tools that can be freely downloaded from various research sites. More detailed information is available at [111] and [112].

From the list shown in Table 7.1, the following systems were installed on our machines and various example agents were run on these systems.

FIPA-OS

The Foundation for Intelligent Physical Agents (FIPA) was formed in 1996 in Switzerland with a goal to produce software standards to enable interoperability between various Multi Agent Systems. FIPA have produced standards for agent communication (based on speech act theory), management, and service discovery (yellow and white pages). FIPA specifications focus on the interfaces for agent communication, specific internal agent architectures are not mandated. A number of independent implementations of the specifications have been built, which have been used to vali-

Table 7.1: A Partial List of Agent-Building Tools.

Research Organization	Project Name	Language Used
Carnegie Mellon University	RETSINA	C++
Stanford University	ProcessLink	Java
The University of Texas at Austin	Sensible Agents	Java, C, Lisp
MIT Artificial Intelligence Lab	Sodabot	Proprietary
SRI International	Open Agent Architecture	Java, C, Lisp
University of West Florida	NOMADS	Java
University of Massachusetts	Java Agent Framework	Java
Stanford University	JATLite	Java
University of Cincinnati	JAFMAS	Java
University of California San Diego	Infospiders	Java
The Media Lab, MIT	Hive	Java
University of Delaware	DECAF	Java
Dartmouth University	D'Agents	Tcl
Purdue University	BOND	Java
University of Toronto	Agent Building Shell	Proprietary
University College Dublin, Ireland	Agent Factory	VisualWorks
Air Force Institute of Technology	agentTool	Java
Foundation of Intelligent Physical Agents	FIPA-OS	Java
CSELT S.p.A., University of Parma	JADE	Java
British Telecommunications Labs	Zeus	Java
University of Parma	LEAP	Java
University of Bologna	SOMA	Java
University Wrzburg	SeSAm	Java
Technical University of Madrid	MAST	C++
University of Stuttgart	MOLE	Java
Columbia University	Mobiware	Java
Technische Universitat Berlin	JIAC	Java
University of Otago	JatLiteBean	Java
Technical University of Vienna	Gypsy	Java
Toshiba Corporation	Bee-gent	Java

date the standards. First released in August 1999, FIPA-OS is the world's first Open Source implementation of the FIPA standards developed by Nortel Networks UK. It is a component-based toolkit that can enable rapid development of FIPA-compliant agents. It is available in two flavors: A standard FIPA-OS based on Java 2 and a MicroFIPA-OS based on Java 1.1 but designed to execute on PDA's. The installation of FIPA-OS on a Windows 95 machine required a number of changes in different batch files. However it ran smoothly on a Windows 2000 machine. Example agents that sent a ping message to other agents running on the same machine or running on two different machines received responses from other agents successfully. It has a GUI to manage the activities of agents but does not provide many facilities. It uses RMI for intra-platform communication.

RETSINA

RETSINA is multi-agent system (MAS) developed at the Intelligent Software Agents Lab - The Robotics Institute - Carnegie Mellon University that supports communities of heterogeneous agents. The system uses C++ for coding and sockets for inter-agent communication. Each agent acts as a server that listens to messages at a particular port. An agent naming service (ANS) is used to locate the address of an agent along with a port number. It also has a Matchmaker that serves as a "yellow pages" of agent capabilities, matching service providers with service requestors based on agent capability descriptions. RETSINA was installed on two Windows 2000 machines and a number of example agents were successfully run. Although the documentation provides good tutorials and source code for examples, it does not provide source code for the main engine (only dlls are provided). Also the software license is restricted and forbids installation on more than two machines or any modifications to the system.

JADE

JADE (Java Agent DEvelopment Framework) is a software framework to develop agent-based applications in compliance with the FIPA specifications for interoperable intelligent multi-agent systems. The goal is to simplify the development while ensuring standard compliance through a comprehensive set of system services and agents. JADE can then be considered an agent middleware that implements an Agent Platform and a development framework The agent platform can be distributed across machines (which do not even need to share the same OS) and the configuration can be controlled via a remote GUI. JADE is completely implemented in Java. It uses an agent communication language (ACL) that is a subset of the knowledge query and manipulation language (KQML) for messages. It uses a content language (CL) for the contents of messages. JADE was installed on two Windows 2000 machines. It uses the notion of a container that contains AMS (agent management system), DF (directory facilitator) and RMA (remote monitoring agent) and other agents. A number of containers can be created on a machine. In our software evaluation experiments up to four containers were created on one of the machines and two on the other machine. Messages were successfully exchanged between different agents. The platform provides a Graphical User Interface (GUI) for the remote management, monitoring and controlling of the status of agents, allowing, for example, the stopping and restarting of agents. The GUI also allows creating and starting the execution of an agent on a remote host, provided that an agent container is already running. JADE uses RMI for intra-platform communication and http or IIOP for inter-platform communication. It has a news group and regular support is provided to the users. It conforms to FIPA standards and issues a open GL licence. Both the source code and binaries are available and there are no restrictions on modifications or additions. We, therefore, decided to use JADE as the middleware for implementation of the prototype. It was noted that the following two extensions of JADE have also been implemented, and that boosted our confidence in JADE.

LEAP

Lightweight Extensible Agent Platform (LEAP) is a project based on JADE that has developed an agent platform, which is: lightweight, executable on small devices such as PDAs and phones; extensible, in size and functionality; supporting wireless communications and TCP/IP. The project is a joint effort of the University of Parma, Italy and a number of commercial organizations like Motorola, ADAC, BroadCom, BT, Telecom Italia Lab and Siemens.

ZEUS

This is a multi agent development environment based on JADE and developed by British Telecom Lab. It provides a highly graphical user interface and a library of software components and tools that facilitate the design, development and deployment of agent systems.

7.2 Platform-specific Model

The prototype was implemented in Java with JADE as the middleware. Figure 7.1 shows the platform specific model. The middleware resides on computer 0 as shown in the figure and provides management and communication services between agents. The agent management system can be requested to spawn a new agent or kill an existing one as dictated by needs of the system. It also has an agent that acts as the request supervisor - an agent that keeps track of all currently registered services and provides details of the location of a service to a requesting agent. CARs are also spawned on this machine as and when requested by various CALs. A service registry agent is associated with each data/information source covered by the system. A computer can have one or more data /information sources, for example in figure 7.1, computer 2 has two sources of information with one service registry agent (SRA) for each, while computer 3 has just one source of information along with its SRA. A

user can log into the system using a separate computer such as computer 4 shown in the diagram or he/she can use a computer with one of the information sources on it such as computer 3 and may have more than one local agent, if required as shown in the case of user 1.

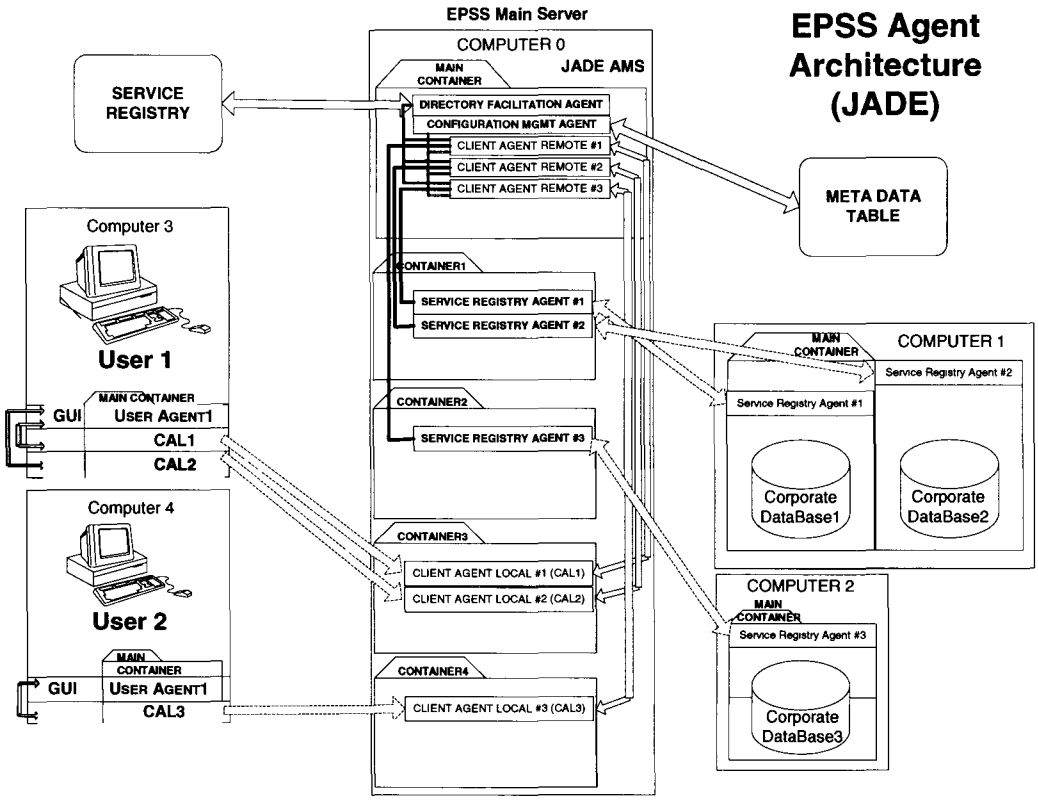

Figure 7.1: Platform Specific Model.

7.3 Salient Features of Working System

When a user logs in, a UA-gui is displayed as shown in figure 7.2. The user has two options: to configure the system for the kinds of queries the user would like to use, or to select a previously created query and request updated information corresponding to

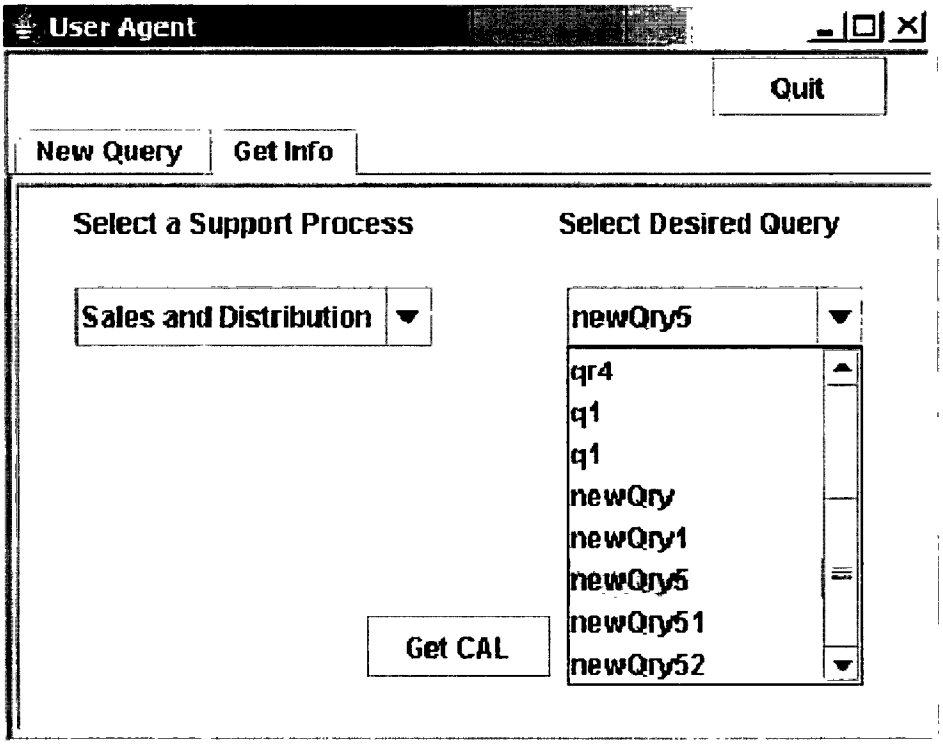

Figure 7.2: User Agent (UA).

it. The latter option results in spawning of a CAL. A graphical interface for this agent, shown in figure 7.3, displays the progress of search and the extracted information on its completion. Behind the scenes, CAL requests the agent management system to create a CAR and on receipt of confirmation that CAR has been created, sends a message to it passing details of criteria according to which information is desired. At this stage neither CAL nor CAR knows if the requested service is available and if so, where the information source is located. CAR tries to find from the Request Supervisor where it can find the desired service if it is available. If the service is not available, a message is sent to CAL to inform the user about it. If the service is available, CAR passes on the criteria to the relevant SRA who searches the desired information and sends results to CAR for onward passing to CAL to display it for the user. The source of information can have a number of different formats. It can

be in a database or in a worksheet or in a comma separated value (CSV) file. Once

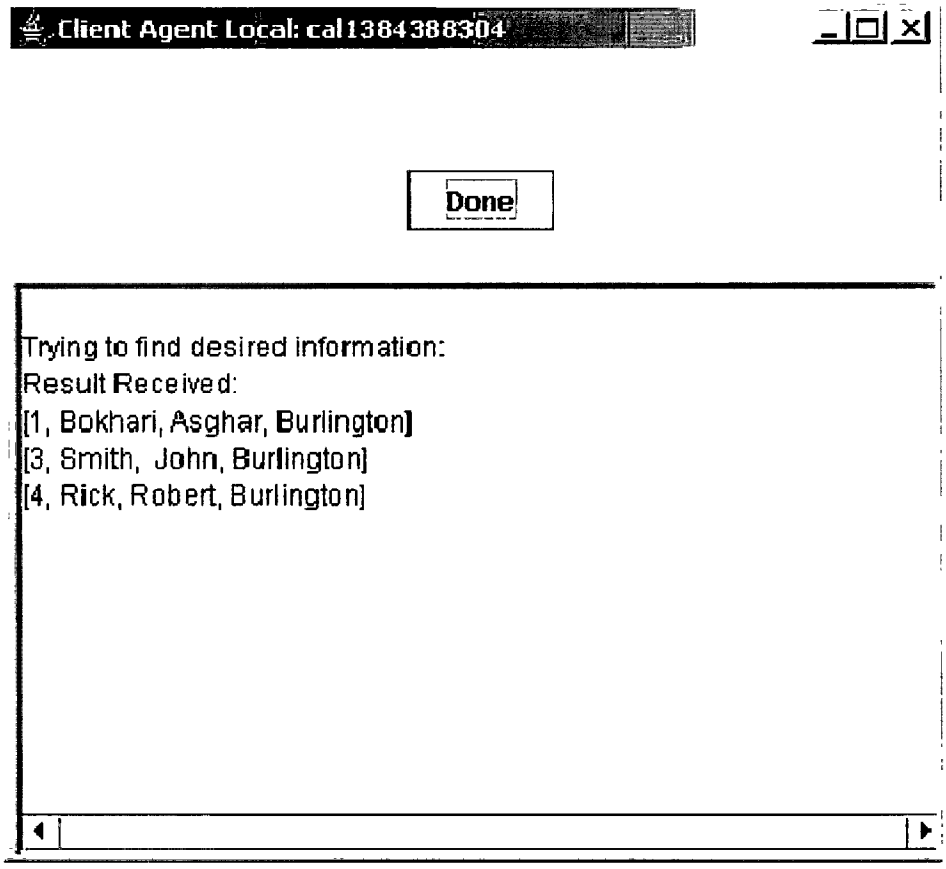

Figure 7.3: Client Agent Local (CAL).

CAL receives the desired information it requests the agent management system to kill CAR to save system resources. The user can also kill CAL after reading the displayed information.

7.3.1 Agent Communication

One of the basic properties of a multi-agent system is that different agents interact with one another by exchange of messages. JADE supports asynchronous message passing between agents. For this purpose each agent has a message queue and the

JADE runtime posts all messages for an agent into its message queue. Each time a message is posted the runtime also informs the agent about it, however if or when a particular message will be picked up from the queue for processing entirely depends on how an agent is programmed to handle it. Interaction between agents requires that they understand the messages passed on to them and that means a standard format and semantics must be followed for these messages. Agent communication has been a hot topic of research and has resulted in a number of agent communication languages. Most of these languages are based on the speech acts theory of Searle [113]. A first attempt that resulted in a standard agent communication language came forth from the ARPA knowledge sharing project and is called KQML (Knowledge Query and Manipulation Language) [114]. A recent attempt from the Foundation for Intelligent Physical Agents (FIPA) is called FIPA-ACL. JADE messages comply with (FIPA) standards that ensure communication with agents created in other languages and running on other platforms. ACL specifies the format of messages exchanged between agents.

7.3.2 Message Structure

An ACL message usually consists of:

- Sender
- Receiver
- Performative (INFORM, QUERY, PROPOSE etc.)
- Content (the main content of the message)

And may consist of one or more of the following attributes:

- ConversationID - links messages in same conversation
- Language - the language is used in the content

- Ontology - the ontology is used in the content
- Protocol - the protocol followed by conversation
- ReplyWith - a field to help distinguish answers
- InReplyTo - used to help distinguish answers
- ReplyBy - used to set a time limit on an answer

7.3.3 Content Language

In order to specify content of a message some kind of language is required. JADE supports three possibilities. The most primitive is to use simple strings as contents of messages. This method is useful only if the content of a message does not consist of abstract concepts, objects or data structures, i.e. the content does not need to be parsed to access its parts. The second approach is to transmit serialized java objects directly as content of a message. This requires that all agents are implemented in Java and in this method the contents are not human readable. The third and most versatile approach is to use a specific content language to code and decode a message in a standard FIPA format. Jade supports the FIPA-SL family of languages for this purpose.

7.3.4 Ontology

An ontology is a domain specific vocabulary for content elements, their semantics, their properties and relationships between them are defined by ontology. As described by Grimshaw [115] it is a description (like a formal specification of a program) of the concepts and relationships that can exist for an agent or a community of agents. Ontologies can be represented by a number of different languages; however at present JADE does not support these directly. Instead, ontologies are encoded as Java classes, either written by hand or generated automatically using tools like Protege [116].

7.3.5 Jade support for Ontology

Agents exchange information among themselves by ACL messages, however this information may have different representations inside an agent compared to that used in the message. For example, the message may contain some attributes of a Java object as a string, whereas inside the agent those attributes are part of an object. It is not convenient to represent them as strings inside the agents. This means that every time an agent needs to send some information to another agent, it must convert it from its internal representation to a string and the receiving agent must convert it back to the internal representation. Additionally the receiving agent needs to check that the received information is meaningful in that it complies with the rules of agreed ontology for communication between the two agents. JADE supports content languages and ontologies by providing facilities in the form of *jade.content* package to automatically make the above conversions and checks. According to JADE documentation, exploiting the JADE content language and ontology support included in the jade.content package, to make agents talk and reason about "things and facts" related to a given domain goes through the following steps.

1. Defining an ontology including the schemas for the types of predicate, agent action and concept that are pertinent to the addressed domain.

2. Developing proper Java classes for all types of predicate, agent action and concept in the ontology.

3. Selecting a suitable content language among those directly supported by JADE.

4. Registering the defined ontology and the selected content language to the agent.

5. Creating and handling content expression as Java objects that are instances of the classes developed in step 2 and let JADE translate these Java objects to/from strings or sequences of bytes that fit the content slot of ACLMessages.

7.3.6 EPSS-Ontology

For the implementation of prototype, we created a simple ontology named EPSS-Ontology described below:

```
public class EPSSOntology extends Ontology {
        // The name identifying this ontology
    public static final String ONTOLOGY_NAME = "EPSSOntology";
        // The singleton instance of this ontology
    private static Ontology instance = new EPSSOntology();
        // Method to access the singleton ontology object
    public static Ontology getInstance() { return instance; }
        // VOCABULARY
    public static final String RECORD = "Record";
    public static final String RECORD_FIELDS = "fields";
    public static final String DATA_SET = "DataSet";
    public static final String DATA_SET_RECORDS ="records";
    public static final String FIND_INFO = "FindInfo";
    public static final String FIND_INFO_PROCESS = "supportProcess";
    public static final String FIND_INFO_QUERY = "desiredQuery";
// Private constructor
    private EPSSOntology()
     { super(ONTOLOGY_NAME, BasicOntology.getInstance());
    try {
          // Add Concepts
          add(new ConceptSchema(RECORD), Record.class);
          add(new ConceptSchema(DATA_SET), DataSet.class);
          add(new AgentActionSchema(FIND_INFO), FindInfo.class);
          // Structure of the schema for the Record concept
          ConceptSchema cs = (ConceptSchema) getSchema(RECORD);
```

```
        cs.add(RECORD_FIELDS, (PrimitiveSchema)
              getSchema(BasicOntology.STRING),1, ObjectSchema.UNLIMITED);
        cs = (ConceptSchema) getSchema(DATA_SET);
        cs.add(DATA_SET_RECORDS, (ConceptSchema)
              getSchema(RECORD),1, ObjectSchema.UNLIMITED);
        //The RECORD slot has cardinality > 1
        AgentActionSchema as = (AgentActionSchema) getSchema(FIND_INFO);
        as.add(FIND_INFO_PROCESS, (PrimitiveSchema)
        getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
        as.add(FIND_INFO_QUERY, (PrimitiveSchema)
        getSchema(BasicOntology.STRING), ObjectSchema.MANDATORY);
      } catch (OntologyException oe) {
        oe.printStackTrace(); }
    }
}// End EPSSOntology
```

7.3.7 Interaction Protocol

Agent communication is usually a conversation between two roles, initiator and responder(s). Initiator may request the responder to do something. The responder may choose to refuse or may act on the request and in turn inform the initiator about the result of carrying out the request. FIPA have standardized protocols for different types of conversations such as REQUEST, PROPOSE, QUERY, CONTRACT NET, DUCH AUCTION etc. JADE supports FIPA Interaction protocols by a number of classes defined in the jade.proto package. We commonly used FIPA REQUEST protocol in implementing the prototype. A graphical description of this protocol is given in Figure 7.4.

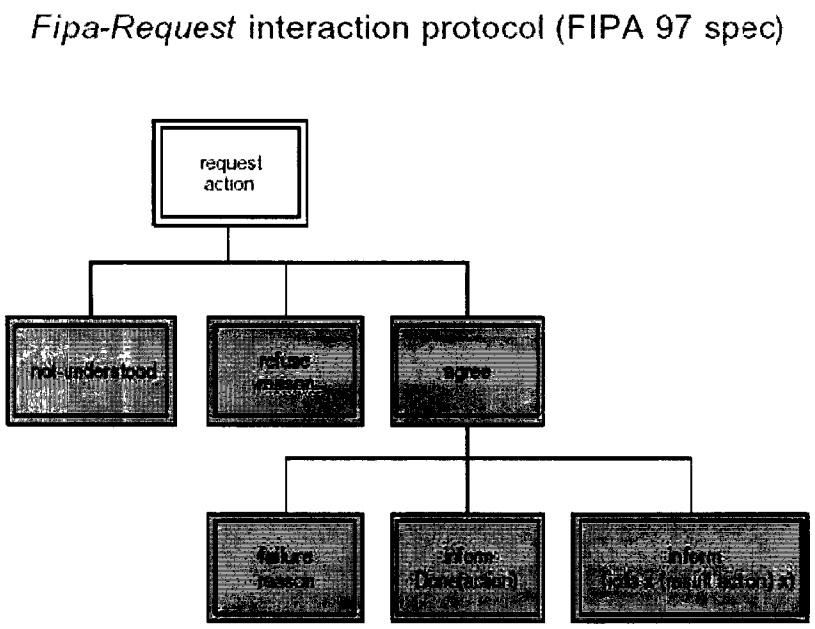

Figure 7.4: FIPA Request Protocol

7.4 Summary

In this chapter, we discussed some of the agent-development software available as free downloads from various research sites. We also discussed salient features of JADE that we have used as middleware for implementation of the prototype. A platform-specific model was presented and some of the main features of the prototype such as its working, agent communication, message structure, content language, ontology, and the FIPA request interaction protocol were discussed.

In the next chapter, we summerise the work documented in this thesis along with conclusions and future directions.

Chapter 8

Summary, Conclusions and Future Work

8.1 Summary

The goal of this research is to explore the design, technical feasibility and usability of an architecture for Electronic Performance Support Systems that promises to meet the challenges of rapid and continuous change in the business and industrial environments of the 21st century. As noted in Chapter 1, the conventional method of dealing with various changes by issuing a new release of EPSS, updated to incorporate these changes, cannot keep up with the rate of change in modern businesses. We have proposed a flexible and agile architecture that can continuously evolve without the need for significant developer intervention. We have focused on the design of an agile performance support system that extracts current information from sources distributed over the intranet of an organization on the fly when requested by a user. In this research, we followed a model driven approach to design an agile performance support system with these desirable features by combining SOA and software agents. Users of this system can expect to obtain updated information in time to perform efficiently without waiting for a new release of the software. It is also possible to add

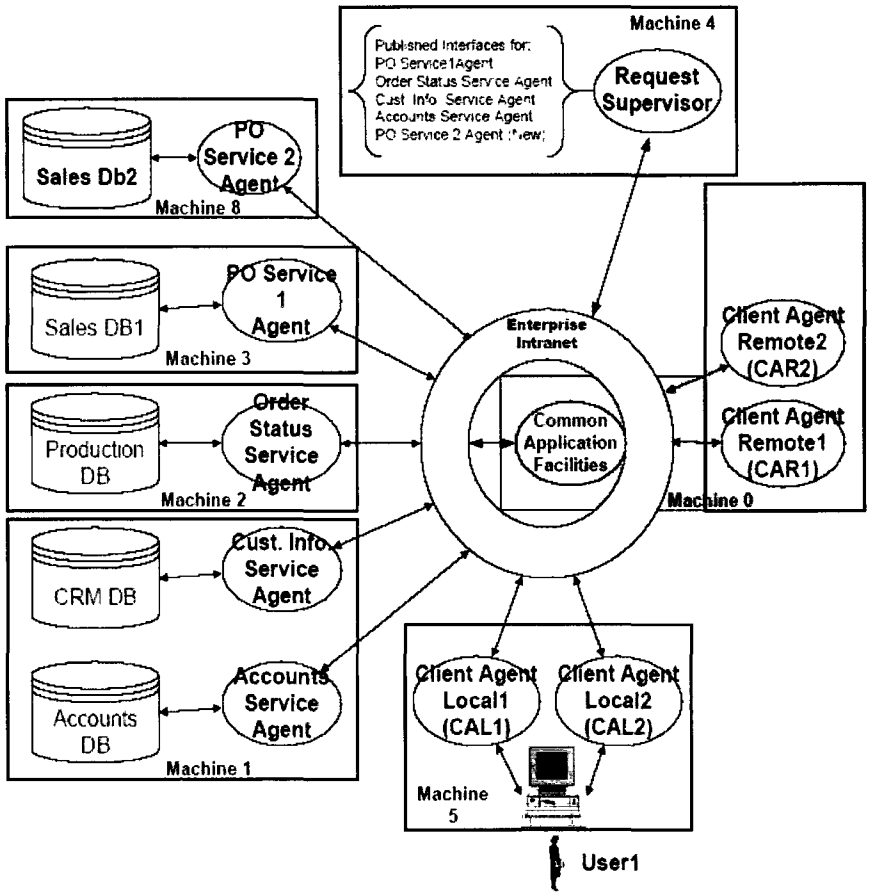

Figure 8.1: How Agility can be Achieved.

new sources of information or new user requirements for information with minimum disruption to the system.

Figure 8.1 shows how the system architecture enables information updates to occur without requiring any changes to be made to its central code and how a new source of information or a new service is added without affecting the rest of the application. A user request is handled by a Client Agent Local (CAL) that requests the middleware to spawn a Client Agent Remote (CAR) and passes on the user request to it. CAR interacts with the Request Supervisor to obtain the location of

the relevant service agent and communicates with it. The service agent searches for the latest information that is passed on to the user through CAR and CAL. As the information is not hard coded into the software, the user always accesses the current updated information.

The anthropomorphic model of the system handles the availability of a new source of information, perhaps due to certain changes in business rules, in a way analogous to how a corporation would handle the need for additional expertise by hiring a new employee who either has the requisite knowledge or can be trained to acquire the aditional expertise before assigning him/her the reponsibility of providing addtional expertise. Note that during this time, the main operations of the corporate entity remain untouched operationally. In case a new source of information needs to be added to the system, for example, Sales DB 2 in figure 8.1, all that is required is to write code for an agent that provides a related service based on the new source of information and to add its interface in the list of the Request Supervisor. The new agent can be tested independently before its interface is added into the list. The same logic applies if some additional service is required based on an existing source of information.

Two of the main activities involved in the development of a software system are the software requirement specification and the software design [8]. Whereas the current software engineering practice requires the requirements specification phase to ensure unambiguous documentation of the desired behaviour of the system, it requires the design activity to translate the requirements into a representation of the software that can be assessed for quality before coding begins. It is widely recognised that the development of a complex software system not only requires a complete, consistent and unambiguous specification, but also a correct design that meets certain requirements. This observation motivated the initial work in this research by exploring the use of formal methods in concurrent object-oriented design, and further derived our model-based approach for development of agent-oriented software systems. There

has been much activity to formally define agents and their behaviour. Also, formal methods have been frequently used in requirements specification and analysis phase, however their use to create formal models at the design stage is rare [117]. Traditionally UML has been the tool of choice for design and analysis of such systems. UML has no facilities for dynamic analysis whereas Software Engineering practices expect analysis and verification of desirable properties at an early stage in the software development process. The main contribution of this research is the development of an algorithm to transform a UML state diagram representing the behaviour of an agent class at a suitable level of abstraction, to an OCPN class net. We have also shown that an agent based software system can be modelled using CP nets by transforming the state diagrams representing the behaviour of all agent classes in the system to OCPN class nets and then combining these class nets using communication channels into one CP net model of the system. Such a model can then be subjected to validation and verification. Validation is an activity during the analysis of the design that is used to convince others (and ourselves) about the soundness of the system's behaviour, whereas verification is an activity to rigorously prove that a formal design has a formally stated property. The two common approaches to verification are model checking that involves exhaustively checking that a property holds in a fnite representation of the design, such as a state space and theorem proving that involves automatic or semi-automatic proof search [118]. We have used the simulator of Design/CPN for validation and its occurrence graph tool with associated reports for verification of properties such as reachability, liveness, deadlock-freeness and fairness as detailed in chapter 6. There are a number of other formal methods that can be applied for model checking as mentioned in chapter 5. However, CP nets are more focused on pragmatism than most other commonly known formal methods and are one of the most commonly applied kinds of Petri Nets among practitioners [118]. This popularity stems from their graphical nature and good tool support.

We would like to mention here that the dynamic analysis of the model not only

provided insight into the working of the actual system, it also helped in identifying and rectifying potential problems particularly when both the manual as well as automatic simulations were running smoothly but generation of the occurrence graph showed that some of the places in the system were unbounded. An investigation revealed that, since the occurrence graph examines all possible bindings occurring randomly, it was possible for a user to kill a Local agent before the results of a query were received from other agents thus leaving the corresponding Remote agent without an agent to respond to. Suitable changes were made to ensure that a Local agent can only be killed when the corresponding Remote agent has first been killed. In order to identify which Remote agent is spawned on the request of which Local agent, each agent in the system is uniquely identified by a *Colour* consisting of the user Id, the class Id and the agent number. It is not possible to reuse the agent number of an agent that was previously in the system and was killed after it performed its task, because the JVM loads the class representing several agents only once and keeps track of all the agent-identifiers used in a particular session. Therefore, we randomly generate agent identifiers during run time of the system so that different instances of agents can be created and deleted as required.

The use of JAVA as the programming language for implementation of the system ensures that the User Agents as well as the Local Agents can run on any platform that has Java runtime available, without the need to recompile the code. We used Linux, MS Windowes 2000, MS WindowsXP and Sun Solaris operating systems on different machines with data soucrces in different formats e. g. as spreadsheets, databases and CSV files. The compliance of FIPA standards by different agents means that agents implemented with middleware other than JADE can easily communicate with the system. It is interesting to note that current literature, while discussing scalability issues of SOA, points out that scalability can be achieved by simple web server scalability options if the services are stateless, autonomous, short running and non-CPU intensive [119]. We have used software agents to implement services that

have all these characteristics making the architecture easily scalable.

It may be noted that according to Web Services Architecture W3C Working Group [120] "Using Web services technologies to implement a distributed system does not magically turn a distributed object architecture into an SOA. Nor are Web services technologies necessarily the best choice for implementing SOAs – if the necessary infrastructure and expertise are in place to use COM or CORBA as the implementation technology and there is no requirement for platform neutrality, using SOAP/WSDL may not add enough benefits to justify their costs in performance, etc.

In general SOA and Web services are most appropriate for applications:

- That must operate over the Internet where reliability and speed cannot be guaranteed;
- Where there is no ability to manage deployment so that all requesters and providers are upgraded at once;
- Where components of the distributed system run on different platforms and vendor products;
- Where an existing application needs to be exposed for use over a network, and can be wrapped as a Web service."

In contrast to this view, we have designed via this research an architecture for an agile performance support system for the employees of an organization that will run on its intranet and not the Internet. All services in the system are owned by the same organization. They are all implemented with the JAVA programming language. By the use of the middleware JADE and implementation of services as software agents, the specific aspects outlined in the W3C document with respect to Web Services, in our case, need not be considered.

8.2 Conclusion

In this thesis we have proposed a software architecture for Electronic Performance Support Systems and have demonstrated that it results in an agile EPSS that provides its users updated information without waiting for a new release of the software. We have also shown, in section 8.1, that new sources of information or new user requirements for information could be added with minimum disruption to the system. The research reported in this thesis presents a complete development strategy based on a model driven approach. The research was not limited to just a conceptual or platform specific model but a prototype was also successfully implemented. It was demonstrated that with the use of higher-level Petri nets it is feasible to integrate formal methods for dynamic analysis of a complex system at the design stage in the software development cycle. A unique characteristic of this research is the demonstration that when such a system is designed with appropriate tools and transformations, its complexities can be dealt with at an early stage with an architecture that possesses the desirable dynamic properties in accordance with appropriate software engineering practices. As part of this process we proposed an algorithm to transform UML state diagrams to Object Coloured Petri (OCP) nets. This provides the software analyst with a dual approach of using UML to create different models as usual and to create formal models for dynamic analysis by transforming UML state diagrams. This algorithm was extended for application to software agents and it was shown how an existing Petri net tool (Design/CPN) can be used to detect design errors, and how model checking techniques can support the verification of some key behavioral properties of agent-oriented software systems. We have demonstrated that the requirements of current software engineering practices can be fulfilled while developing an Electronic Performance Support System by following the steps listed below:

- Create a UML use case model for the application being designed.

- List the services that the system is expected to provide to its users by examining the use case model.
- Implement services through software agents located at the sources of relevant information.
- Identify any common facilities that may be required for management or communication between different agents and for making the interface of these agents publicly available. Implement these common facilities by middleware.
- Create UML class diagrams as well as interaction diagrams.
- Create a UML state diagram at a suitable level of abstraction, for each class representing an agent.
- Convert the state diagrams to OCPN class nets.
- Combine the class nets to obtain an OCPN model for the system.
- Simulate the OCPN model using Design/CPN.
- Carryout dynamic analysis by using the simulation and the occurrence graph tools of Design/CPN.
- Construct an application specific model.
- Develop the prototype using a suitable programming language.

8.3 Future Work

We were able to generate a partial graph with 886729 nodes and 2615164 arcs that represents a very satisfactory result using the current version of the Occurrence Graph Tool [121]. Future versions of the tool are expected to be able to handle much larger

graphs and it will be interesting to attempt to generate full occurrence graphs with 10 or more agents concurrently present in the system.

Using the prototype, we were able to create up to 50 local agents on different machines used by clients limited by the available RAM. It will be interesting to study the relationship between complexity and performance of the system, in terms of time constraints, with increases in the number of users and corresponding agents. This will require more powerful machines with large RAM and a newer version of the Design/CPN Tools software.

For the prototype, a user interface was created with just enough capability to handle simple queries. A number of improvements are possible to handle more complex user requirements. The current version uses a text file to provide options to the user for building a query. It may be possible to use the system to obtain meta information about the data available in each information source in the system dynamically and use this information to build queries. This will also mean that the meta information will be automatically updated if a new source of information is added to the system.

The prototype can extract information provided by one service at a given time. If a user requires information that is scattered over several information sources that have different services for information extraction, separate queries will need to be generated for each service. It may be possible to modify the user interface such that a user request is automatically parsed by it and sent to different Service Registry Agents. The results can then be synthesized before presentation to the user.

Knowledge Management (KM) is viewed as one of the most important topics for academic research and industrial practice encompassing a variety of disciplines like general management information systems development, information resource management, decision support systems, artificial intelligence, and human resource management [122, 123]. An Electronic Performance Support System (EPSS) is a good candidate to act as a vehicle for dissemination of the right knowledge to the right

person at the right time, provided it can dynamically adapt to the rapid changes in the knowledge contents of the organization. This opens up a whole new avenue of research where the use of strong notion of agency may be explored so that the agents may have the capabilities to dynamically search the knowledge sources according to specific knowledge representations used in different sources.

Bibliography

[1] Latella Diego, Majzik Istvan, and Massink Mieke, "Automatic verifcation of a behavioural subset of UML statechart diagrams using the spin model-checker," *Formal Aspects of Computing*, vol. 11, pp. 637–664, 1999.

[2] Bezanson W R, "Performance support: Online, integrated documentation and training," in *Proceedings of the 13th annual international conference on Systems documentation: emerging from chaos: solutions for the growing complexity of our jobs*, February 1995, pp. 1–10.

[3] Dickleman Gary J, "Performance support in internet time: The state of the practice on the eve of the new millennium," *http://www.pcd-innovations.com/PsinInternetTime/PSinInternetTime.pdf (Accessed 03/26/2006)*, pp. 1–14, 2000.

[4] Kasvi Jyrki J J and Vartiainen Matti, "Performance improvement on the shop floor," *Performance Improvement*, pp. 40–46, July 2000.

[5] Malcolm S, "Where epss will go from here," *Training*, pp. 64–69, March 1998.

[6] Gates Bill, *Business At the Speed of Thought*, Warner Books Inc., New York, 1999.

[7] Giorgini J, "Modeling deployment and mobility issues in multiagent systems using auml," in *Giorgini P and Mller J P and Odell J, (Eds.): AOSE 2003, LNCS 2935, pp. 69-84*, Springer Verlag, Berlin Heidelberg, 2004.

[8] Xu Haiping, "A model-based approach for development of multi-agent software systems," Ph.D. dissertation, University of Illinois at Chicago, USA, 2003.

[9] Gery G, *Electronic Performance Support Systems*, Ziff Communications, Cambridge, MA, 1991.

[10] Marion C, "What is performance centered design?," http://www.chesco.com/~cmarion/PCD/WhatIsPCD.html (Accessed 09/29/2006).

[11] Gery Associates, "Performance centered design competition," http://www.pcd-innovations.com/samples.htm (Accessed 01/09/2007).

[12] McCaffery J, "What is performance improvement?," http://www.jhpiego.org/global/pi.htm (Accessed 09/29/2006).

[13] Greenough R, "Electronic performance support systems," http://www.drive.cranfield.ac.uk/rgreenough.htm (Accessed 09/29/2006).

[14] Platt E, "Developing e-learning and performance support systems," http://www.cherryleaf.com/developing_performance_support_ documentation.htm (Accessed 09/29/2006).

[15] Malcolm S E, "Reengineering corporate training," *Training*, pp. 57–61, 1992.

[16] Malcolm S E, "Case study: Low-budget reusable flash card shell," http://www.performance-vision.com/articles/case-screen-saver.htm (Accessed 09/29/2006).

[17] Malcolm S E, "Case study: Performance support for product configuration," http://www.performance-vision.com/articles/case-howell.htm (Accessed 09/29/2006).

[18] Malcolm S E, "Case study: Performance support for management decision-making," http://www.pcd-innovations.com/what_is_epss.htm (Accessed 09/29/2006).

[19] Carr C, "A new horizon for expert systems," *AI expert*, pp. 44–49, 1992.

[20] Rosenberg M J, "Performance technology, performance support and future of training: A commentary," *Performance Improvement Quarterly*, vol. 8, no. 1, pp. 94–99, 1995.

[21] Raybould B, "Making a case for epss," *Innovations in Education and Training International*, vol. 32, no. 1, pp. 65–69, 1995.

[22] Cole K, Fischer O, and Saltzman F, "Just in time knowledge delivery," *Communications of the ACM*, vol. 40, no. 7, pp. 49–53, July 1997.

[23] Kilby T, "What is WBPSS?," http://www.webbasedtraining.com/primer_whatiswbpss.aspx (Accessed 10/02/2006).

[24] Sleight D A, "What is electronic performance support and what isn't?," http://www.msu.edu/ sleightd/epssyn.html (Accessed 10/02/2006).

[25] Johannes C, "Electronic performance support: Appropriate technology for the development of middle management in developing countries," http://hagar.up.ac.za/catts/abc/epss.html (Accessed 09/29/2006).

[26] Raybould B, "Performance support engineering: An emerging developing methodology for enabling organizational learning," *Performance Improvement Quarterly*, vol. 8, no. 1, pp. 7–22, 1995.

[27] Zopler A, "First cousins once removed: Knowledge management and performance support," http://www.pcd-innovations.com/what_is_epss.htm (Accessed 10/02/2006).

[28] Elsenheimer J, "The performance support bridge to knowledge management," http://www.pcd-innovations.com/what_is_epss.htm (Accessed 10/02/2006).

[29] Poehlman W F S, Garland W J, Bokhari A A, Wilson R J, and Baesten C W, "Performance support systems and artificial intelligence considerations," in *Proc. International Nuclear Congress - INC93*, October 1993.

[30] Poehlman W F S, "The OPUS approach to domain/software/hardware mapping," in *Workshop on Performance Support Systems, McMaster University, Hamilton, Ontario, Canada*, June 15-17 1994.

[31] Wilson R J, Bokhari A A, Garland W J, Poehlman W F S, and Beasten C W, "The design and implementation of an operator's performance support system," in *Proc. International Nuclear Congress - INC93*, October 1993.

[32] Miller B, "EPSS: Expanding the perspective," http://www.pcd-innovations .com/what_is_epss.htm (Accessed 10/02/2006).

[33] Sleight D A, "Use of just-in-time performance support tools for learning on demand in a work practice," http://www.msu.edu/ sleightd/jit.html (Accessed 10/02/2006).

[34] Sleight D A, "Types of electronic support systems: Their characteristics and range of design," http://www.pcd-innovations.com/what_is_epss.htm (Accessed 10/02/2006).

[35] Sen Mehmet, "Distributed open asynchronous information access environment.," Ph.D. dissertation, Saracuse University New York, USA, http://aspen.ucs. indiana.edu/collabtools/senthesisdraft.doc (Accessed 10/02/2006), 2000.

[36] Brusilovsky P and Cooper D W, "ADAPTS: Adaptive hypermedia for a web-based performance support system," in *Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW*, Toronto, Canada, May 11-14, 1999, pp. 41–47.

[37] Banerji A, "Designing electronic performance support systems," Ph.D. dissertation, University of Teesside, UK, http://homepage.ntlworld .com/philip.barker2/ISRG/banerji.htm, (Accessed 10/02/2006), 1995.

[38] Rhodes B J, "Just-in-time information retrieval," Ph.D. dissertation, Massachusetts Institute of Technology, http://www.bradleyrhodes.com/Papers/rhodes-phd-JITIR.pdf, (Accessed 10/08/2006), 2000.

[39] Stevens M, "The service-oriented approach," http://www.developer .com/services/article.php/10928_1010451_2 (Accessed 10/02/2006).

[40] Wikipedia, "Service oriented architecture," http://en.wikipedia.org /wiki/Service-oriented-architecture (Accessed 01/09/2007).

[41] Choreography-Working-Group W3C, "From WS choreography model overview editor's draft 24 march 2004," http://www.w3.org/2002/ws/chor/edcopies/model/ModelOverview.html (Accessed 01/09/2007).

[42] Chris Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[43] William B. Bradley and David P. Maher, "The nemo p2p service orchestration framework," in *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9*, Washington, DC, USA, 2004, p. 90290.3, IEEE Computer Society.

[44] Foster H, Uchitel S, Magee J, and Kramer J, "Leveraging eclipse for integrated model-based engineering of web service compositions," in *eclipse '05: Proceedings of the 2005 OOPSLA workshop on Eclipse technology eXchange*, New York, NY, USA, 2005, pp. 95–99, ACM Press.

[45] Michael Stollberg, Dumitru Roman, and Juan Miguel Gomez, "A mediated approach towards web service choreography," in *Proceedings of the workshop on Semantic Web Services: Preparing to Meet the World of Business Applications held at the 3rd International Semantic Web Conference*, Hiroshima, Japan, 11 2004.

[46] Carbone M, Honda K, Yoshida N, Milner R, Brown G, and Ross-Talbot S, "A theoretical basis of communication-centred concurrent programming," http://www.w3.org/2002/ws/chor/edcopies/theory/note.pdf (Accessed 01/09/2007).

[47] Singh M, "Synthesizing coordination requirements for heterogeneous autonomous agents," *Autonomous agents and multiagent systems*, vol. 3 No. 2, pp. 107–132, 2000.

[48] Wijngaards N J E, Overeinder B J, van Steen M, and Brazier E M T, "Supporting internet-scale multi-agent systems," *Data and Knowledge Engineering*, vol. 41, pp. 229–245, 2002.

[49] Wooldridge M and Ciancarini P, "Agent-oriented software engineering:the sate of the art," in *Proc. 1st Int. W/S (AOSE-2000)*, Springer-Verlag: Berlin, Germany, 2000.

[50] Zohar Manna and Amir Pnueli, *Temporal verification of reactive systems: safety*, Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[51] Wooldridg M and Jennings N, "Intelligent agents: Theory and practice," *The Knowledge Engineering Review*, vol. 10(2), pp. 115–152, 1995.

[52] Hintikka J, *Knowledge and Belief: An Introduction to the Logic of the Two Notions*, Cornell University Press, Ithaca New York, 1962.

[53] Levasque H J, Cohen P R, and Nunes J H, "On acting together," in *In Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90), pp. 94-99*, Boston MA, 1990.

[54] Reichgelt H, "Knowledge for reasoning about knowledge and belief," *The Knowledge Engineering Review*, vol. 4(2), pp. 119–139, 1989.

[55] Konolige K, "A first order formalization of knowledge and action for a multi-agent planning system," in *In Hays, J.E., Michie D., Pao Y.eds Machine Intelligence 10, pp. 41-72*, Ellis Horwood, Chichester, England, July 1999.

[56] Hass A, "A syntactic theory of belief and action," *Artificial Intelligence*, vol. 28(3), pp. 245–292, 1986.

[57] Cohon P R and Levesque H J, "Intention is choice with commitment," *Artificial Intelligence*, vol. 42, pp. 213–261, 1990.

[58] Rao A S and Georgeff M P, "A model theoretical approach to the verification of situated reasoning systems," in *In Proceedings of Thirteen International Joint Conference on Artificial Intelligence (IJCAI), pp. 318-324*, Chambery, France, August 28-September 3 1993.

[59] Sampath P, "Modeling multi-agent reactive systems," in *International Conference on Logic Programming (ICLP), LNCS Vol. 2401, pp. 476*, Springer Verlag, Berlin Heidelberg, 2002.

[60] Lind J, "Issues in agent-oriented software engineering," in *Ciancarini P and Wooldridge M J, (Eds.): AOSE 2000, LNCS 1957, pp. 45-58*, Springer Verlag, Berlin Heidelberg, 2001.

[61] Bernardi S, Merseguer J, and Donatelli S, "From UML sequence diagrams and statecharts to analysable Petri net models," in *Proceedings of the third*

International Workshop on Software and Performance (WOSP'02), pp. 35-43, Rome Italy, July 2002.

[62] Hu Z and Shatz Sol M, "Mapping UML diagrams to a Petri net notation for system simulation," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 213-219*, Banff, Canada, June 2004.

[63] Saldana J A and Shatz Sol M, "Formalization of object behavior and interactions from UML models," *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, vol. 11 No. 6, pp. 643–673, 2001.

[64] Lomazova I A, "Multi-agent systems and petri nets," in *Proceedings of International Workshop on Distributed Artificial Intelligence and Multi-Agent Systems (DAIMAS97), pp. 147-152*, St. Petersberg, Russia, 1997.

[65] Bakam I, Kordon F, LePage C, and Cois F, "Using coloured petri nets for the study of an hunting management system," in *LNAI (Lecture Notes on Artificial Intelligence) 1871, pp. 123-132*, Springer Verlag, Berlin Heidelberg, 2001.

[66] Patrick C K and Hung M, "Modeling e-negotiation activities with petri nets," in *Proceedings of the 35th Hawaii International Conference on System Sciences, pp. 379-88*, 2002.

[67] Zuberek W M, "Timed petri nets and preliminary performance evaluation," in *ISCA '80: Proceedings of the 7th annual symposium on Computer Architecture, pp. 88–96*, ACM Press New York, USA, 1980.

[68] Ballarini P, S Donatelli, and G Franceschinis, "Parametric stochastic well-formed nets and compositional modeling," in *In Nielsen M., Simpson D. (Eds.): Lecture Notes in Computer Science (LNCS) 1825, pp. 43-62*, Springer Verlag, Berlin Heidelberg, 2000.

[69] Chiola G, Marsan M A, Balbo G, and Conte G, "Generalized stochastic petri nets: A definition at the net level and its implications," *IEEE Trans. Softw. Eng.*, vol. 19, no. 2, pp. 89–107, 1993.

[70] Marsan M A, Conte G, and Balbo G, "A class of generalized stochastic petri nets for the performance evaluation of multiprocessor systems," *ACM Trans. Comput. Syst.*, vol. 2, no. 2, pp. 93–122, 1984.

[71] Bohuslov K, "Object-oriented Petri nets and their application and type analysis, in: Information technologies and control," *Information Technologies and Control, Vol. 1, No. 1, Sofia, BG*, vol. Vol. 1, No. 1, pp. 27–31, 2003.

[72] Perkusich A and deFigueiredo J, "G-nets: A petri net based approach for logical and timing analysis of complex software systems," *Journal of Systems and Software*, vol. 39, no. 1, pp. 39–59, 1997.

[73] Battiston E, De Cindio F, and Mauri G, "A class of high level nets having objects as domains," in *G. Rozenberg (Eds.):Advances in Petri Nets, Lecture Notes in Computer Science (LNCS 340)*, BSpringer Verlag, Berlin Heidelberg, 1988.

[74] Buchs D and Guelfi N, "A concurrent object oriented Petri nets approach for system specification," in *Application and Theory of Petri Nets, Proceedings of the 12th International Conference (ICATPN'91), pp. 432-454*, Gjern, Denmark, 1991.

[75] Valk R, "Petri nets as token objects: An introduction to elementry object nets," in *J. Desel, M. Silva (Eds.): Proceedings of the International Conference on Application and Theory of Petri Nets (ICATPN'98), pp. 1-24*, Springer Verlag, Berlin Heidelberg, 1998.

[76] Lakos C, "Object oriented modeling with object Petri nets," in *Agha et*

al.(Eds.):Concurrent OOP and PN, LNCS 2001, pp. 1-37, Springer Verlag, Berlin Heidelberg, 2001.

[77] Philippi S, "Seamless object-oriented software development on a formal base," in *Proceedings of the Workshop on Software Engineering and Petri Nets, 21st International Conference on Application and Theory of Petri Nets*, June 2000.

[78] Baresi L., "Some preliminary hints on formalizing UML with object Petri nets," in *Integrated Design and Process Technology, IDPT-2002*, June 2002.

[79] Girault C. and Valk G., *Petri Nets for System Engineering*, vol. 1, Springer Verlag, Berlin, 2003.

[80] Barros J. P. and Gomes P., "On the use of coloured Petri nets for object oriented design," in *Proceedings of the International Conference on Application and Theory of Petri Nets (ICATPN'98),Lecture Notes in Computer Science (LNCS 3099)*, Springer Verlag Berlin Heidelberg, 2004.

[81] Maier C and Moldt D, "Object coloured Petri nets - a formal technique for object oriented modelling," in *Agha et al.(Eds.):Concurrent OOP and PN, LNCS 2001, pp. 406-427*, Springer Verlag, Berlin Heidelberg, 2001.

[82] Jensen K, *Coloured Petri Nets Basic Concepts, Analysis Methods and Practical Use*, vol. 1, Springer Verlag, Berlin Heidelberg New York, 2nd edition, 1997.

[83] Christensen S and Hansen N D, "Coloured Petri nets extended with channels for synchronous communication," in *International Conference on the Application and Theory of Petri Nets, Lecture Notes in Computer Science (LNCS 815), pp. 159-178*, Springer Verlag, Berlin Heidelberg, 1994.

[84] Germany University of Hamburg, "Petri nets tools database quick overview," http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/quick.html (Accessed 11/18/2006).

[85] Agha G, "Abstracting interaction patterns: A programming paradigm for open distributed systems," in *Najm E and Stefani J B, Eds, Formal Methods for Open Object-based Distributed Systems*, Chapman and Hall, New York, 1997.

[86] Beraldi R and Nigro L, "Distributed simulation of timed petri nets: A modular approach using actors and time warp," *IEEE Concurrency*, vol. 7, no. 4, pp. 52–62, 1999.

[87] Wooldridge M, Jennings N R, and Kinny D, "A methodology for agent-oriented analysis sand design," in *Proc. 3rd Annual Conf. On Autonomous Agents*, 1999.

[88] Jennings N R, Sycara K P, and Wooldridge M, "A roadmap of agent research and development," *Autonomous Agents and Multi-Agent Systems*, vol. 1, pp. 7–38, 1999.

[89] Exforsys Inc., "The advantages and disadvantages of erp," http://www.exforsys.com/content/view/2478/1133/ (Accessed 01/12/2007), 2006.

[90] Davenport T H, "Putting the enterprise into the enterprise system," *Harvard Business Review*, pp. 121–131, July - August 1998.

[91] Kinny D Georgeff M and Rao A, "A methodology and modelling technique for systems of bdi agents," in *W. Van de Velde and J. W. Perram, Eds, Agents Breaking Away: Proceedings of the Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World, (LNAI Volume 1038), pp. 56-71*, Springer-Verlag, Berlin, Germany, 1996.

[92] Odell J, Parunak H, and Bauer B, "Representing agent interaction protocols in uml," in *P. Ciancarini and M. Wooldridge, Eds, Agent-Oriented Software Engineering, Proc. 1st Int. W/s (AOSE-2000)*, Springer-Verlag, Berlin Germany, 2000.

[93] Elizabeth A K, "Agent software engineering with role modelling," in *P. Ciancarini and M. Wooldridge, Eds, Agent-Oriented Software Engineering: Proceedings of the First International Workshop (AOSE-2000)*, Springer-Verlag, Berlin Germany, 2000.

[94] Papasimeon M and Heinze C, "Extending the uml for designing jack agents," in *In Proceedings of the Australian Software Engineering Conference (ASWEC01), pp. 89-97*, Canberra, Australia, August 26-27 2001.

[95] Rysavy O, "A survey on approaches to formal representation of UML," Tech. Rep., Brno University of Technology, Czech Republic, 2003.

[96] Lilius J and Paltor I P, "Formalizing UML state machines for model checking," in *Robert France and Burnhard Rumpe (Eds.):UML'99, Lecture Notes in Computer Science (LNCS 1723), pp. 430-444*, Springer Verlag, Berlin Heidelberg, 1999.

[97] Varro N, "A formal semantics of uml statecharts by model transition systems," in *ICGT '02: Proc. 1st Int. Conf. on Graph Transformation, pp. 378-392*, Springer-Verlag, London UK, 2002.

[98] Merseguer J, Campos J, Bernardi S, and Donatelli S, "A compositional semantics for UML state machines aimed at performance evaluation," in *Proceedings of the sixth International Workshop on Discrete Event Systems (WODES'02), pp. 295-302*, Zaragoza, Spain, October 2002.

[99] Zhao Yu and Fan Y, "Towards formal verification of UML diagrams based on graph transformation," in *Proceedings of the IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-East'04)*, 2004.

[100] Bokhari Asghar and Poehlman Skip, "Formalization of uml statecharts: Approaches for handling composite states.," Tech. Rep. CAS-05-07-SP, McMaster University, Computing and Software Department, 2005.

[101] Deacon J, "A developer's guide to UML 2," http://www .john-deacon.net/UML/UML_Appendix/Generated /UML_Appendix.asp (Accessed 12/08/2006).

[102] Morasca S and di Milano P, "Measuring attributes of concurrent software specifications in petri nets," in *Proceedings of the Sixth International Metrics Symposium (METRICS'99), pp. 100-110*, 1999.

[103] Genero M, Miranda D, and Piattini M, "Defining metrics for uml statechart diagrams in a methodological way," in *Jeusfeld M A and Pastor O (Eds.): ER 2003 Workshops, LNCS 2814, pp. 118-128.*, Springer Verlag, Berlin Heidelberg, 2003.

[104] Lewis J and Loftus W, *JAVA Software Solutions: foundations of program design*, Addison Wesley Longman, Inc., USA, 2nd edition, 1998.

[105] Bhaduri P and Ramesh S, "Model checking of statechart models. survey and research direction," http://arxiv.org/abs/cs.SE/0407038 (Accessed 12/08/2006).

[106] Sekerinski E and Zurob R, "istate: A statechart translator," in *M. Gogolla and C. Kobryn (Eds.): UML 2001, LNCS 2185, pp. 376-390*, Springer-Verlag, Berlin Heidelberg, 2001.

[107] Hu Z and Shatz S M, "Explicit modeling of semantics associated with composite states in uml statecharts," *Intl Journal of Automated Software Engineering*, vol. 13, no. 4, pp. 423–467, 2006.

[108] Bokhari Asghar and Poehlman Skip, "Translation of uml models to object coloured petri nets with a view to analysis," in *Proceedings of the Eighteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'06) pp. 568-571*, San Francisco, CA USA, July 5-7 2006.

[109] Bokhari Asghar and Poehlman Skip, "Design/CPN analysis reports for EPSS," Tech. Rep. CAS-06-10-SP, McMaster University, Computing and Software Department, 2006.

[110] Jensen K., "Distributed data base," http://www.daimi.au.dk/designCPN/exam /Examples/DistributedDataBase/DistributedDataBase.pdf (Accessed 12/08/2006).

[111] "Agent construction tools," http://www.paichai.ac.kr/ habin/research/agent-dev-tool.htm, (Accessed 10/14/2006).

[112] "Europeon co-ordination action for agent-based computing," http://eprints.agentlink.org/view/type/software.html, (Accessed 10/14/2006).

[113] Searle John Rogers, *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, London, UK, 1969.

[114] Chaib-Draa B and Dignum F, "Trends in agent communication language," *Computational Intelligence*, vol. 2, no. 5, pp. 1–14, 2002.

[115] Grimshaw David, "Ontology," http://www.ryerson.ca/ dgrimsha/courses /cps720_02/courseTopics.html, (Accessed 10/14/2006).

[116] Cliffe O, "Ontology support in jade," http://agents .cs.bath.ac.uk/cm30174/jade/content_languages.shtml, (Accessed 10/14/2006).

[117] Saldana J A and Shatz Sol M, "UML diagrams to object Petri net models: An approach for modelling and analysis," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), pp. 102-110*, July 2000.

[118] Mortensen K H, "Coloured petri nets: a pragmatic formal method for designing and analysing distributed systems," Ph.D. dissertation, University of

Aarhus, Denmark, http://www.daimi.au.dk/PB/522/PB-522.ps.gz., (Accessed 12/10/2006), 1998.

[119] Anderson R W and Ciruli D, "Scaling sao with distributed computing," *Dr. Dobb's Journal*, pp. 22–26, November 2006.

[120] Architecture WG, "Web services architecture w3c working group note 11 february 2004," http://www.w3.org/TR/ws-arch/wsa.pdf (Accessed 01/09/2007).

[121] Arhus University Denmark, "Design/CPN and CPN tools," http://www.daimi.au.dk/CPnets/.

[122] Wang S and Ariguzo G, "Knowledge management through the development of information schema," *Information and Management*, vol. 41, Issue 4, pp. 445–456, 2004.

[123] Bokhari Asghar and Poehlman Skip, "Performance support system: a knowledge management tool," in *Proceedings of the 26th McMaster World Congress*, Hamilton, Ontario, Canada, January 19-21, 2005.

Appendix A

UML Design

A.1 Use-case Model

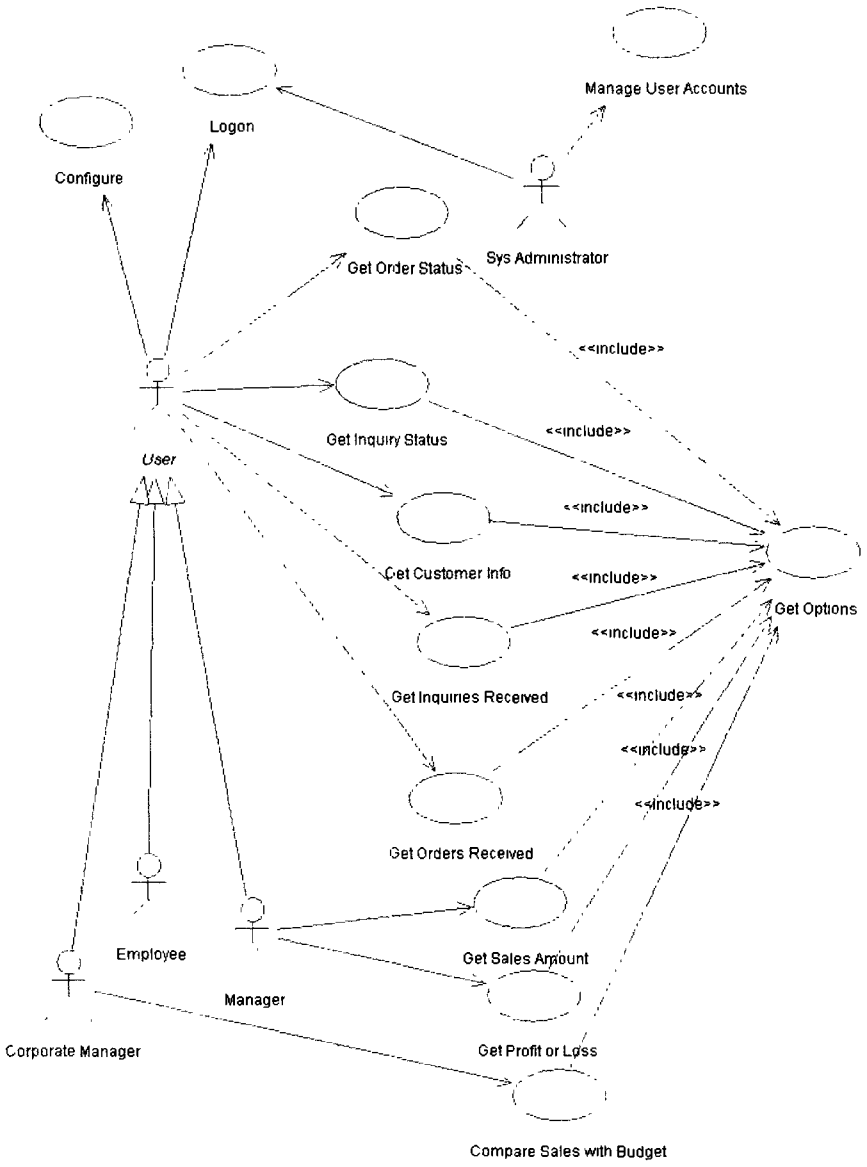

Figure A.1: Use Case Model

A.1.1 Use-case Specifications

SOPSS	Version: 2002
Login Use Case Specification	Issue Date: 10/29/06
Login	

1. Login

1.1 Brief Description

This use case describes how a user logs into the EPSS.

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to Login to the System.

1 The system requests that the actor enter his/her name and password

2 The actor enters his/her name and password

3 The system validates the entered name and password and logs the actor into the system.

4. If the user is a system administrator an additional option 'create user accounts' is presented.

5. If this is the first login by the user then configurations options are presented as detailed in configure use case

6 The system presents an options screen according to the preferences of a particular user if previously configured in the 'configure' use case.

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid name and/or password, the system displays an error message The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

1.3 Special Requirements

None

1.4 Pre-Conditions

The user must have a valid user account and pass word.

1.5 Post-Conditions

If the use case was successful, the actor is now logged into the system If not, the system state is unchanged

1.6 Extension Points

None

Figure A.2: Logon Use Case Specifications

SODSS	Version. 2002
Create User Account Use Case Specification	Issue Date: 10/29/06
Accounts.doc	

1. Create User Account

1.1 Brief Description

This use case describes how a system administrator creates new user accounts or deletes the ones not required

1.2 Flow of Events

1.2.1 Basic Flow

1. This use case starts when the actor wishes to create a new user account or delete an existing one
2. The system requests the actor to specify the function he/she wants to perform If "create a new account" is selected, Create New Account sub-flow is executed. If "delete account" is selected, Delete Account sub-flow is executed.

1 2 1 1 Create New Account

1. The system requests that the actor enter the name and password of new user
2. The actor enters the name and password.
3. The system validates the entered name and password creates a new user account.

1 2 1.2 Delete Account

1. For deleting an existing account, the system asks the actor to enter the user id.
2. The actor enters the user id and selects delete option.
3. The system deletes the user account

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid name and/or password, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

1.3 Special Requirements

None

1.4 Pre-Conditions

The actor must be logged in as administrator.

1.5 Post-Conditions

If the use case was successful, the new user account is created If not, the system state is unchanged

1.6 Extension Points

None

Page 1 of 1

Figure A.3: Create User Account Use Case Specifications

SODSS	Version: 2002
Configure Use Case Specification	Issue Date: 10/29/06
Configure.doc	

1. Configure

1.1 Brief Description

This use case describes how a user configures different options to use the system

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor selects an option to configure the System

1 The system requests the actor to choose a process relating to options he/she wants to configure.

2 The actor selects a process from a list presented to him/her

3 The system presents all the information available relating to the process in a list, from which the actor can choose items of interest.

4. The system requests the actor to choose a name for the inquiry

5 The actor enters the name and chooses the 'save' option

6. The name is shown in a list under the process name

7 The actor is given an oppourtunity to select another process for configuration or to terminate the use case

8 If the actor wants to continue configuration, steps 1 to 6 are repeated.

9 If the actor opts to end configuration, the use case terminates

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid name, the system displays an error message The actor can choose to either return to the beginning of the **Basic Flow** or cancel the configuration, at which point the use case ends

1.3 Special Requirements

None.

1.4 Pre-Conditions

None.

1.5 Post-Conditions

If the use case was successful, the actor preferences are saved for future use of the system. If not, the system state is unchanged

1.6 Extension Points

None

Figure A.4: Configure Use Case Specifications

SOPSS	Version. 2002
Compare Sales with Budget Use Case Specification	Issue Date: 10/29/06
CompareSalesBudget	

1. Compare Sales with Budget

1.1 Brief Description

This use case describes how a user can compare sales during a period with budget for the same period.

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get a comparison of actual sales during a specified time period with those projected in the budget for the same period

1 The system requests that the actor enters start date of the period.

2 The actor enters the start date

3. The system requests that the actor enters the end date of the period

4. The actor enters the end date

5 The system validates the entered dates and access rights of the actor and displays a comparison of actual sales and budgeted sales during the specified period

1.2.2 Alternative Flows

1.2 2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid date or he/she does not have access rights to this information, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends

1.3 Special Requirements

None.

1.4 Pre-Conditions

The actor must have access rights to this information (Corporate Manager)

1.5 Post-Conditions

If the use case was successful, a comparison of actual and budgeted sales over the desired period is displayed. If not, the system state is unchanged.

1.6 Extension Points

None.

Figure A.5: Compare Sales with Budget Use Case Specifications

SOPSS	Version. 2002
Get Customer Info Use Case Specification	Issue Date: 10/29/06
CustomerInfo	

1. Get Customer Info

1.1 Brief Description

This use case describes how a user can get the current status of an order previously confirmed by a customer

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get some information relating to a customer.

1. The system requests that the actor enter the customer number.
2. The actor enters the customer number.
3. The system checks the validity of customer number and asks the user to select type of information desired (Details, Pending Inquiries, Pending Orders, Credit Status or Other).
4. The actor selects an option
5. The system displays the desired information.

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid customer number, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the operation, at which point the use case ends

1.3 Special Requirements

None

1.4 Pre-Conditions

None

1.5 Post-Conditions

If the use case was successful, the desired information about a customer is displayed. If not, the system state is unchanged

1.6 Extension Points

None

Figure A.6: Get Customer Information Use Case Specifications

SODSS	Version: 2002
Get Inquiries Received Use Case Specification	Issue Date: 10/29/06
InquiriesReceived	

1. Get Inquiries Received

1.1 Brief Description

This use case describes how a user can get a list of all inquiries received during a particular time period.

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get information about inquiries received during certain time period.

1 The system requests that the actor enter the start date

2 The actor enters the start date

3. The system requests that the actor enter the end date

4. The actor enters the end date.

5 The system checks the validity of dates and displays a list of all inquiries received during the period covered by the start date and the end date

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid date, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the operation, at which point the use case ends

1.3 Special Requirements

None

1.4 Pre-Conditions

None.

1.5 Post-Conditions

If the use case was successful, a list of all inquiries received during the time period specified is displayed If not, the system state is unchanged

1.6 Extension Points

None.

Figure A.7: Get Inqueries Received Use Case Specifications

SOPSS	Version: 2002
Get Inquiry Status Use Case Specification	Issue Date: 10/29/06
InquiryStatus	

1. Get Inquiry Status

1.1 Brief Description

This use case describes how a user can get the current status of an inquiry previously submitted by a customer

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get the current status of an inquiry

1. The system requests that the actor enter the inquiry number
2. The actor enters the inquiry number.
3. The system checks the validity of inquiry number and displays the status if the number is valid.

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid inquiry number, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the operation, at which point the use case ends.

1.3 Special Requirements

None

1.4 Pre-Conditions

None

1.5 Post-Conditions

If the use case was successful, the current status of the inquiry is displayed If not, the system state is unchanged

1.6 Extension Points

None

Figure A.8: Get Inqueries Status Use Case Specifications

SODSS	Version: 2002
Get Orders Received Use Case Specification	Issue Date: 10/29/06
OrdersReceived	

1. Get Orders Received

1.1 Brief Description

This use case describes how a user can get a report on all orders received during a specified period.

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get the details of all orders received during a given time period

1. The system requests that the actor enter the start date
2. The actor enters the start date
3. The system requests the actor to enter the end date
4. The actor enters the end date
5. The system displays a list of all orders received during the period covered by the start and end dates.

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid date, the system displays an error message. The actor can choose to either return to the beginning of the **Basic Flow** or cancel the operation, at which point the use case ends

1.3 Special Requirements

None.

1.4 Pre-Conditions

None.

1.5 Post-Conditions

If the use case was successful, a list of all orders received in the desired time period is displayed. If not, the system state is unchanged.

1.6 Extension Points

None

Figure A.9: Get Orders Received Use Case Specifications

SOPSS	Version: 2002
Get Order Status Use Case Specification	Issue Date: 10/29/06
OrderStatus	

1. Get Order Status

1.1 Brief Description

This use case describes how a user can get the current status of an order previously confirmed by a customer.

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get the current status of an order.

1 The system requests that the actor enter the order number

2 The actor enters the order number

3 The system checks the validity of order number and displays the status if the number is valid

1.2.2 Alternative Flows

1.2.2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid order number, the system displays an error message The actor can choose to either return to the beginning of the **Basic Flow** or cancel the operation, at which point the use case ends

1.3 Special Requirements

None

1.4 Pre-Conditions

None

1.5 Post-Conditions

If the use case was successful, the current status of the order is displayed. If not, the system state is unchanged

1.6 Extension Points

None.

Figure A.10: Get Orders Status Use Case Specifications

SOPSS	Version: 2002
Get Sales Amount Use Case Specification	Issue Date: 10/29/06
SalesAmount	

1. Get Sales Amount

1.1 Brief Description

This use case describes how a user can get total sales amount in a specified time period

1.2 Flow of Events

1.2.1 Basic Flow

This use case starts when the actor wishes to get total sales figures for a time period.

1 The system requests that the actor enters the start date for the period

2 The actor enters the start date.

3 The system requests that the actor enters the end date for the period.

4 The actor enters the end date.

5 The system validates the entered dates and the access rights of the actor and displays the total sales amount for the desired period

1.2.2 Alternative Flows

1.2 2.1 Invalid Name/Password

If in the **Basic Flow**, the actor enters an invalid date or he/she is does not have access rights to this information, the system displays an error message The actor can choose to either return to the beginning of the **Basic Flow** or cancel the login, at which point the use case ends.

1.3 Special Requirements

None.

1.4 Pre-Conditions

The actor must have access rights to this information (Manager level)

1.5 Post-Conditions

If the use case was successful, the actor gets the total sales figures for the time period desired by him. If not, the system state is unchanged.

1.6 Extension Points

None

Figure A.11: Get Sales Amount Use Case Specifications

A.2 Class Diagrams

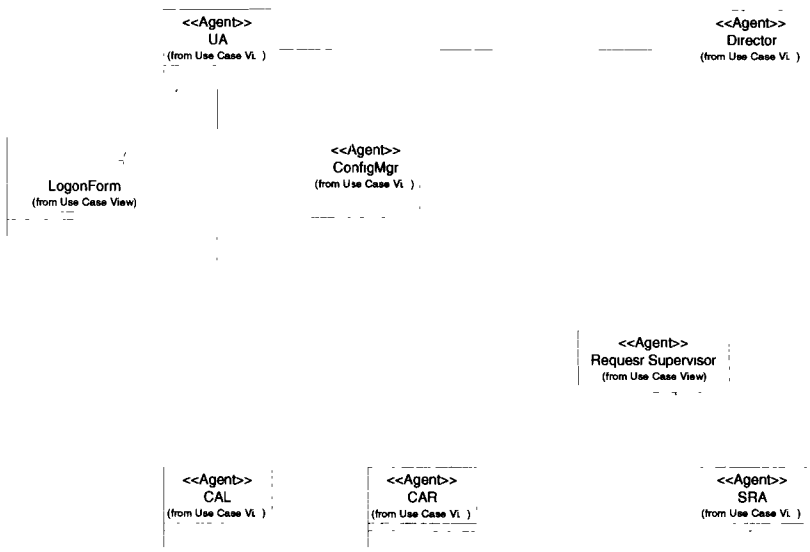

Figure A.12: Logical Model of the EPSS

A.3 Interaction Diagrams

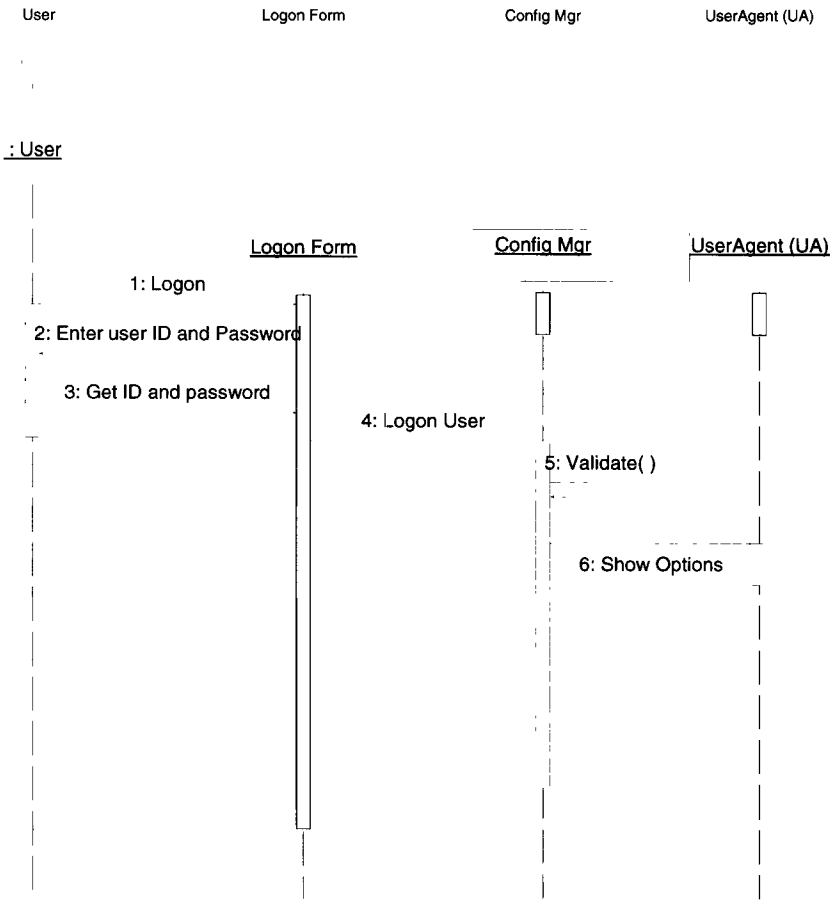

Figure A.13: Logon Use Case Sequence Diagram

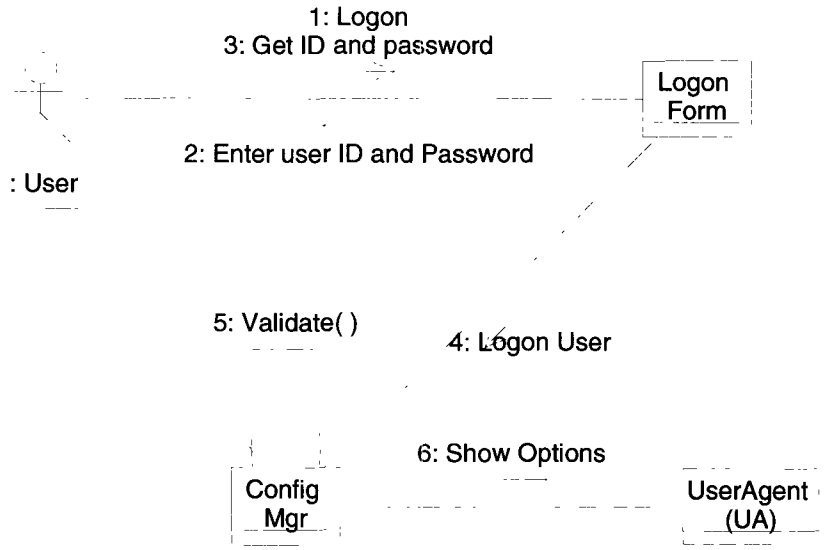

Figure A.14: Logon Use Case Collaboration Diagram

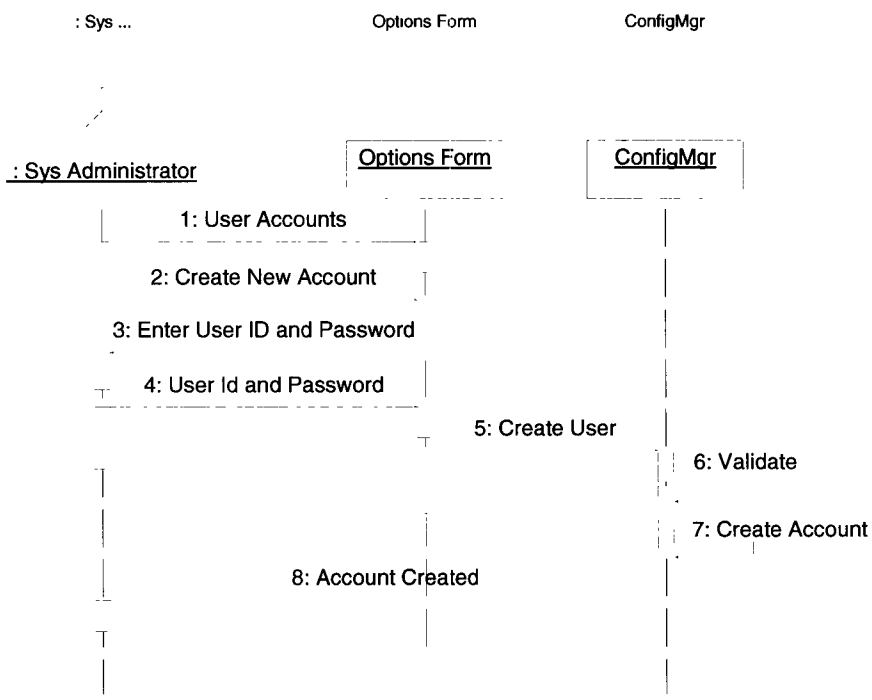

Figure A.15: Create User Account Use Case Sequence Diagram

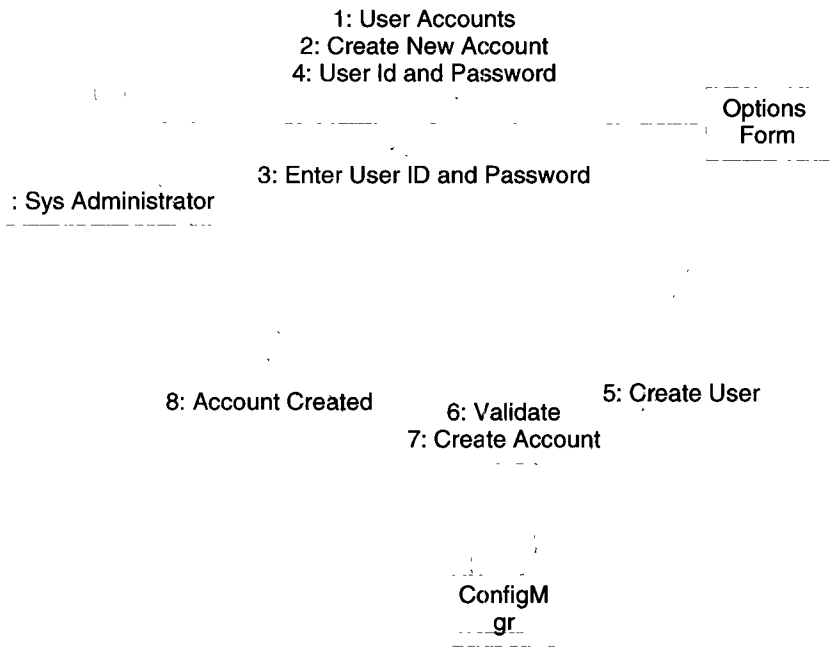

Figure A.16: Create User Account Use Case Collaboration Diagram

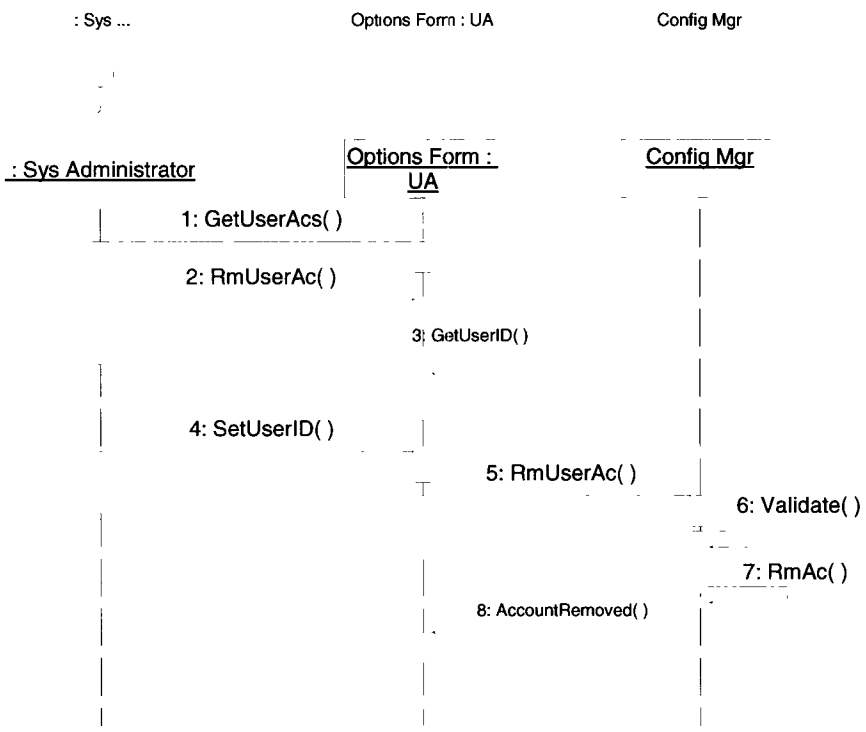

Figure A.17: Remove User Use Case Sequence Diagram

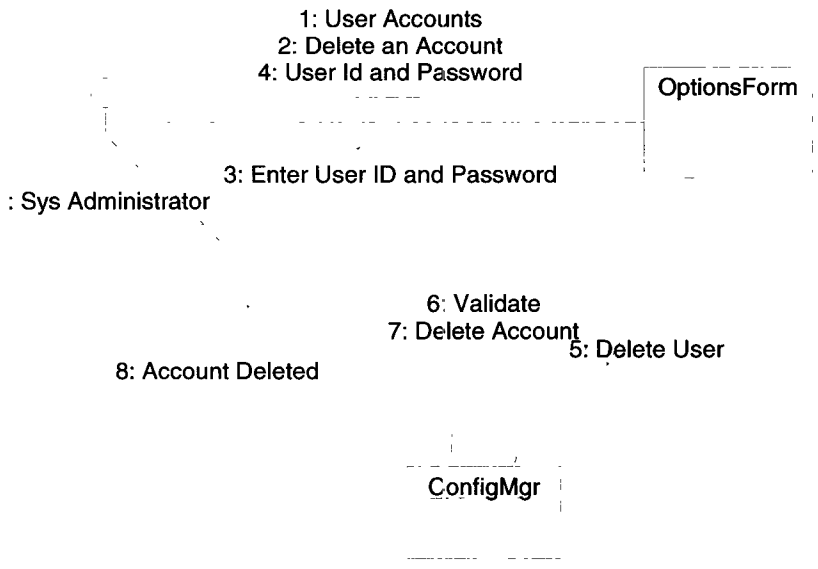

Figure A.18: Remove User Use Case Collaboration Diagram

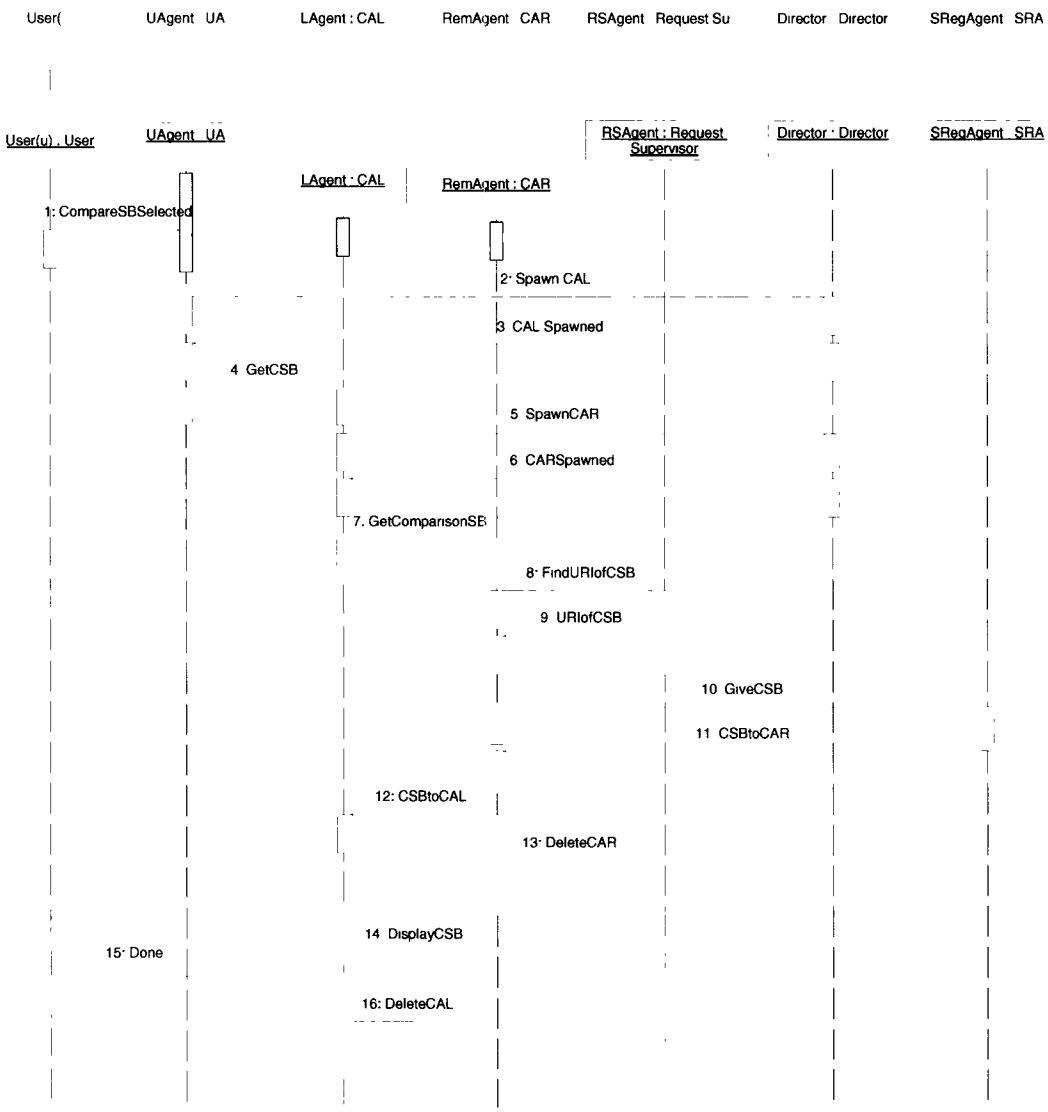

Figure A.19: Compare Sales with Buget Use Case Sequence Diagram

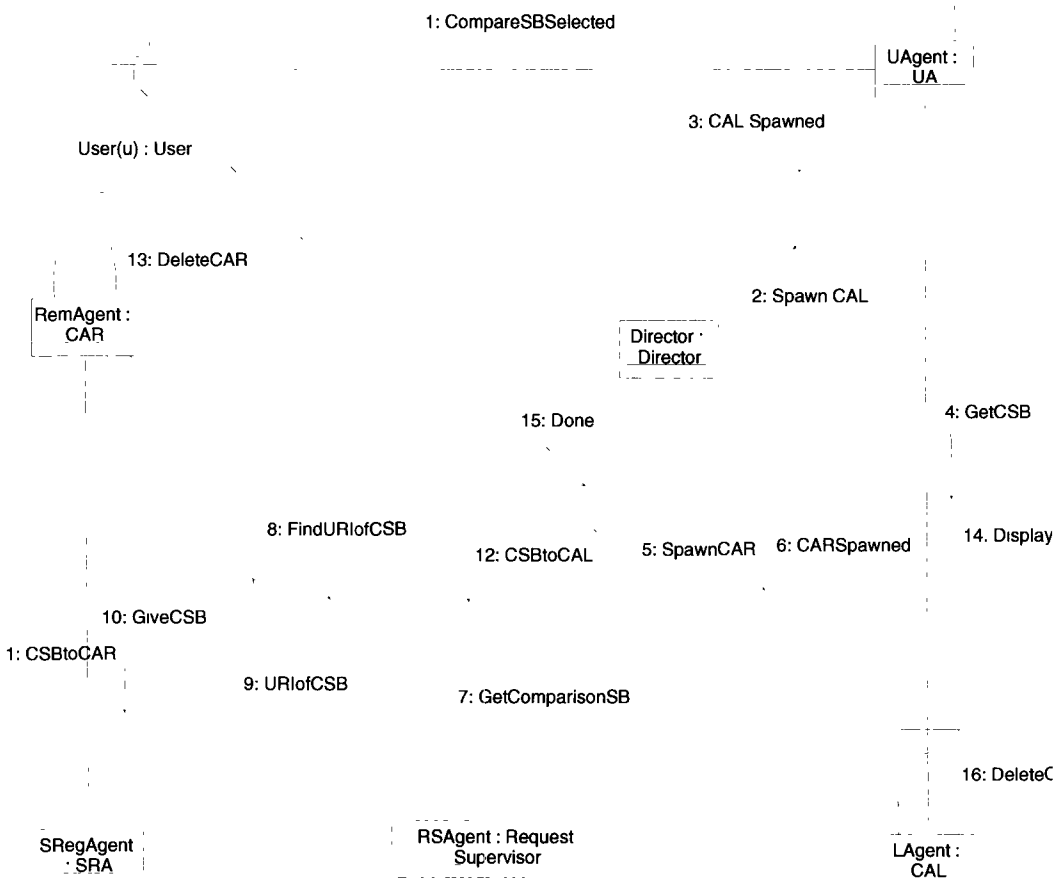

Figure A.20: Compare Sales with Buget Use Case Collaboration Diagram

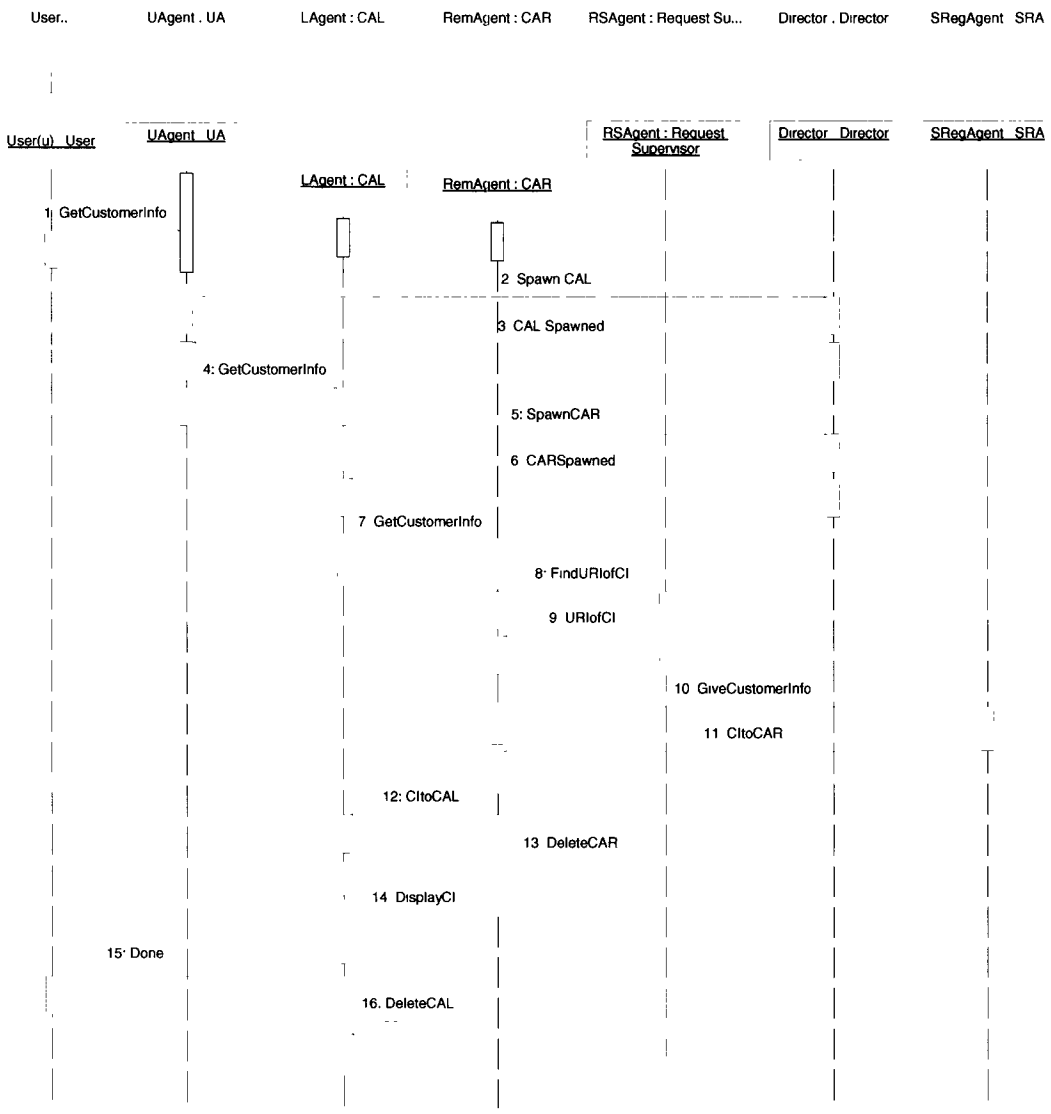

Figure A.21: Get Customer Info Use Case Sequence Diagram

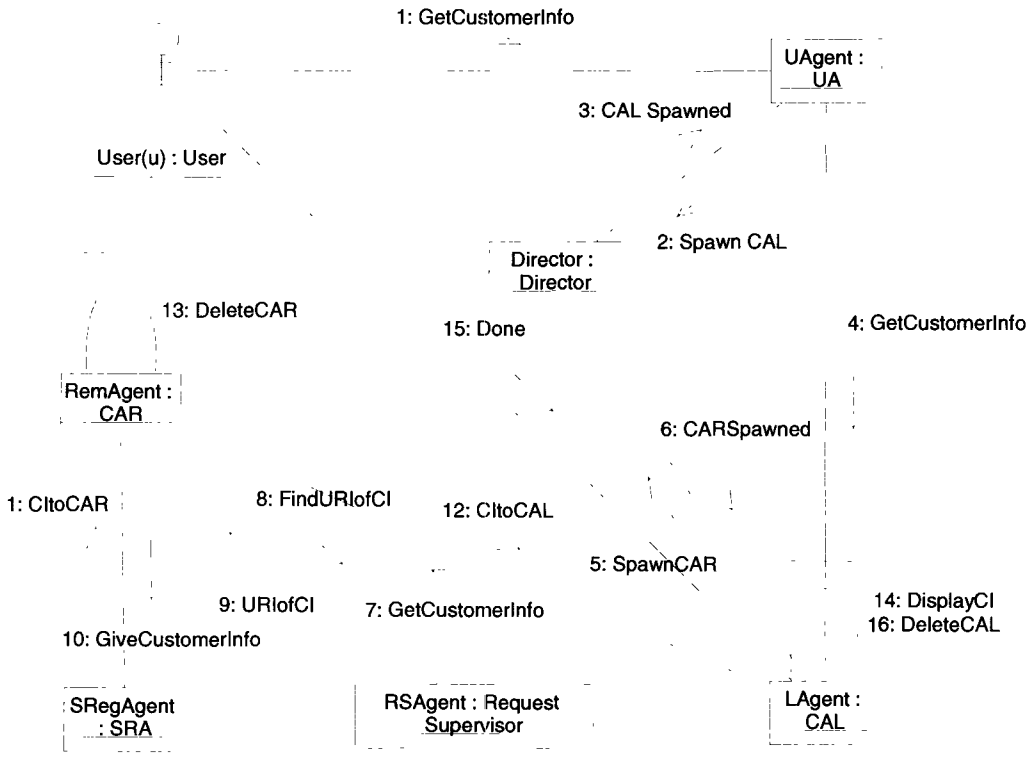

Figure A.22: Get Customer Info Use Case Collaboration Diagram

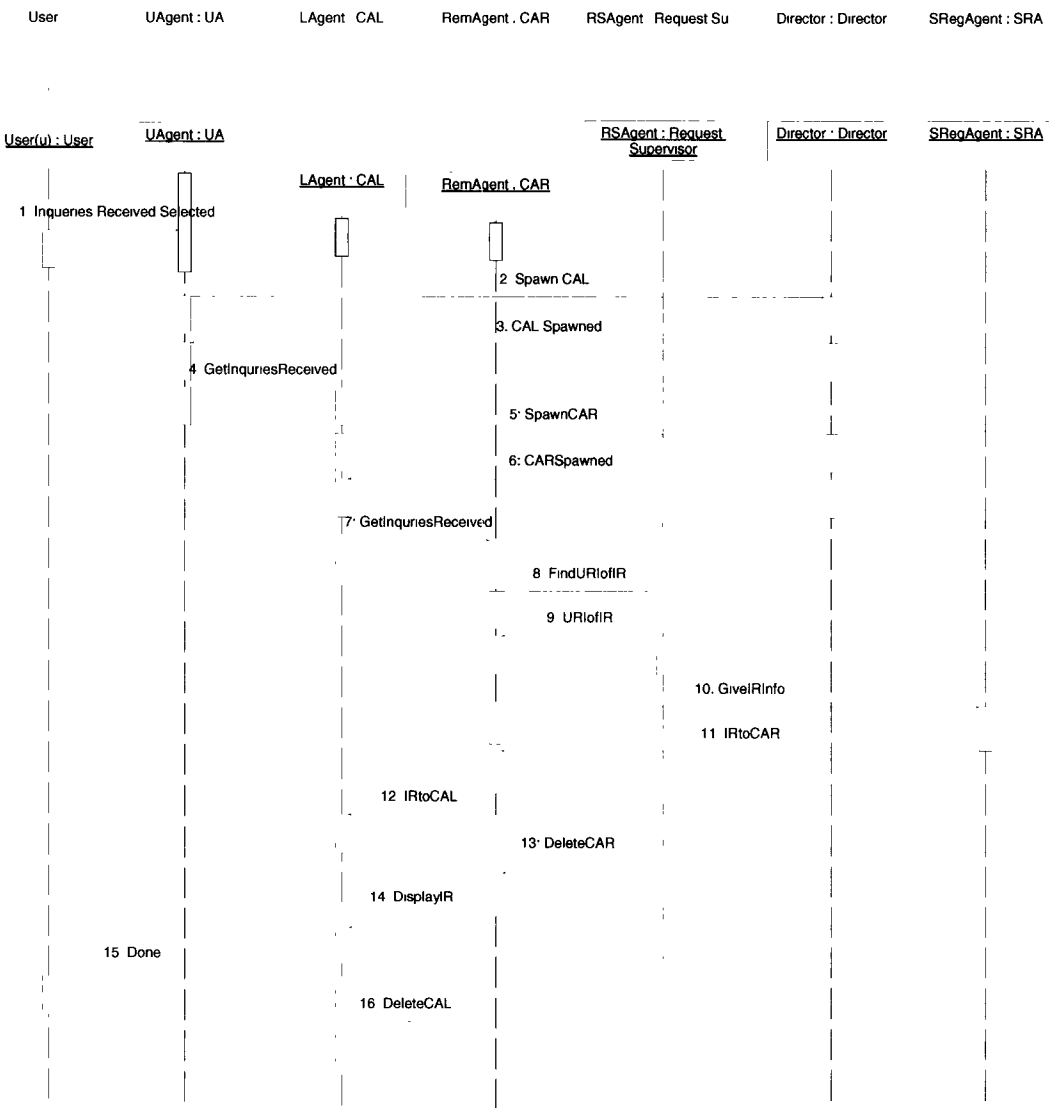

Figure A.23: Get Enquries Received Use Case Sequence Diagram

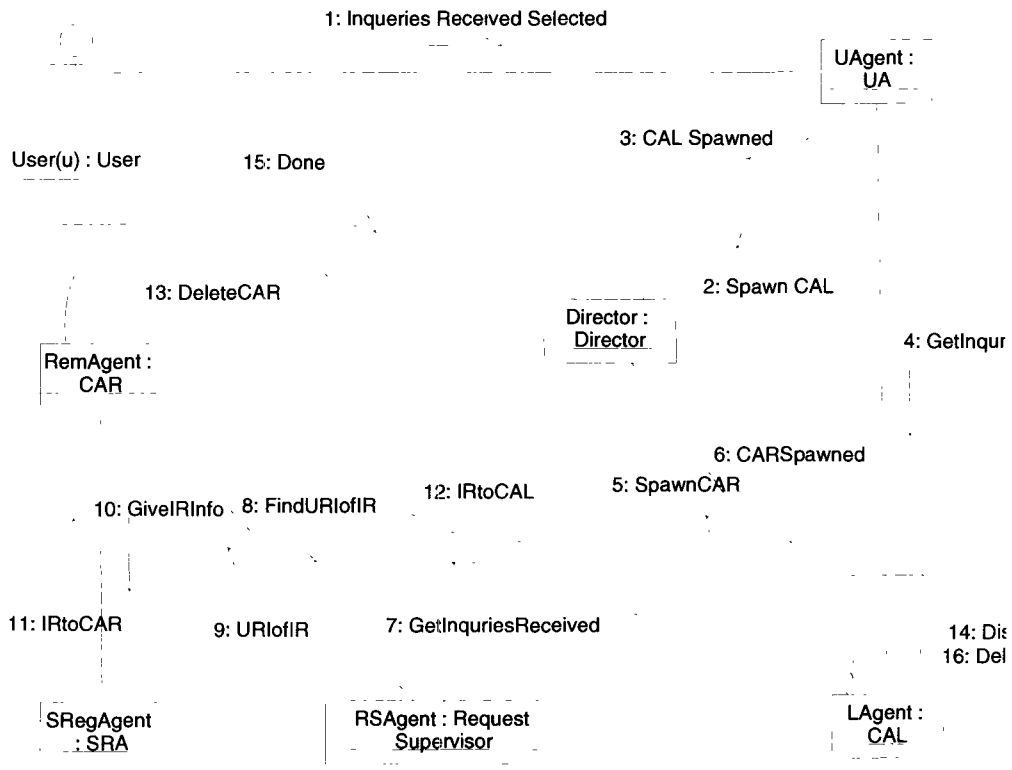

Figure A.24: Get Enquries Received Use Case Collaboration Diagram

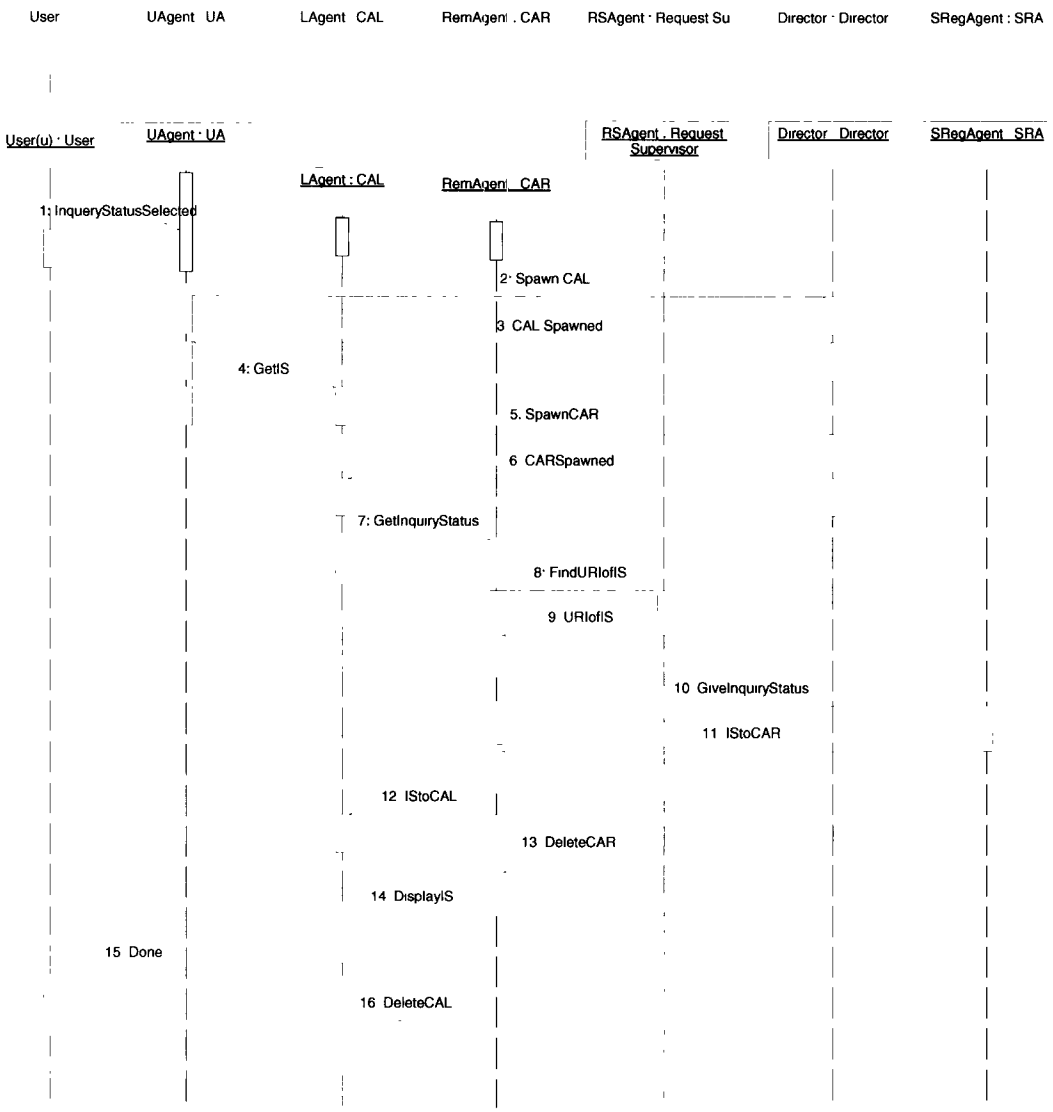

Figure A.25: Get Enquries Status Use Case Sequence Diagram

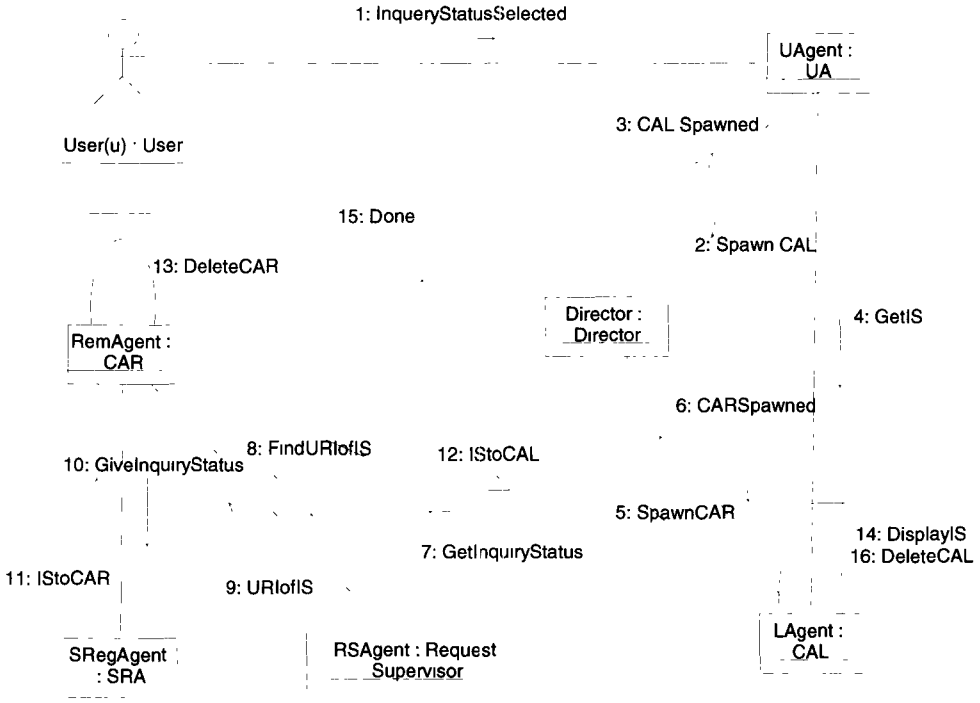

Figure A.26: Get Inqueries Status Use Case Collaboration Diagram

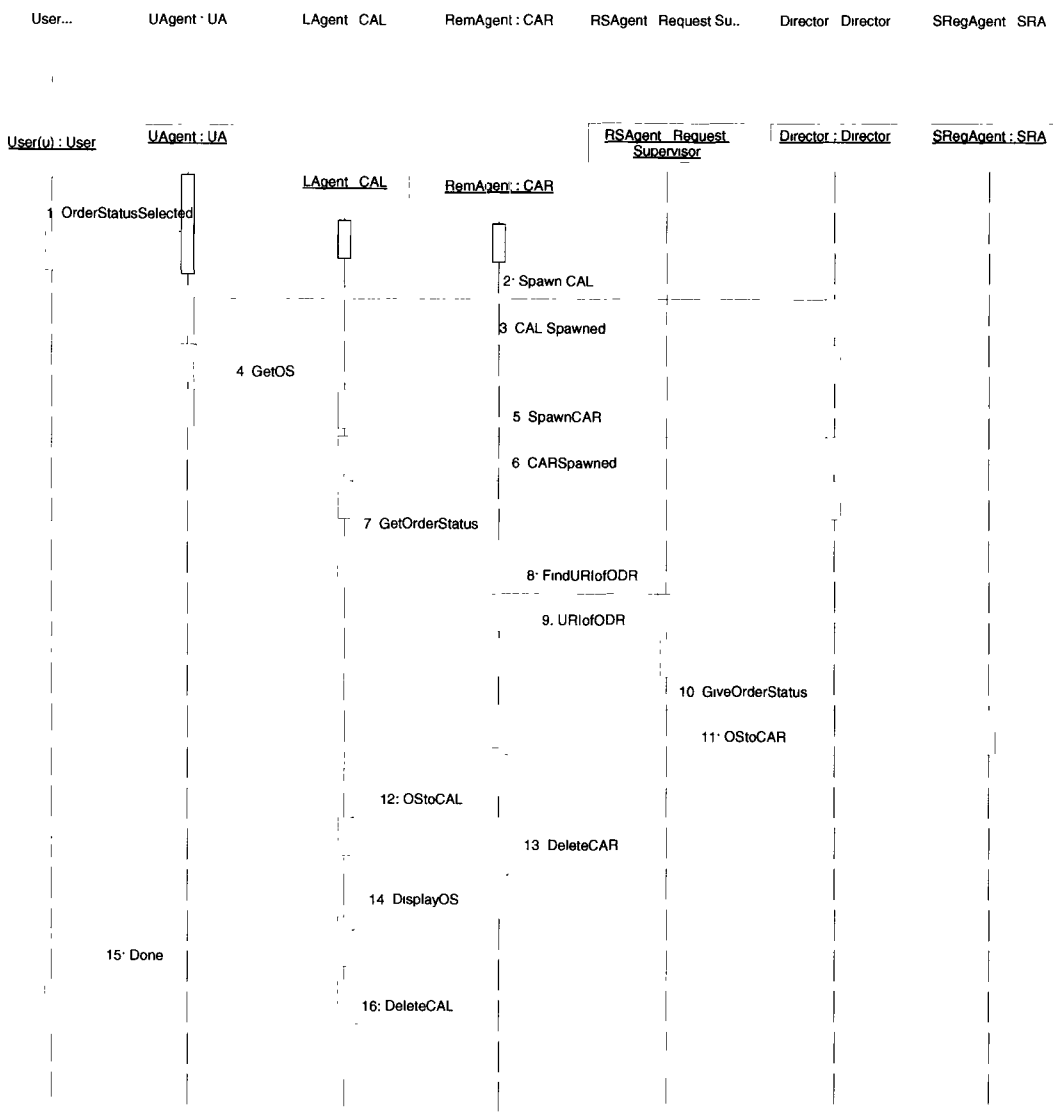

Figure A.27: Get Order Status Use Case Sequence Diagram

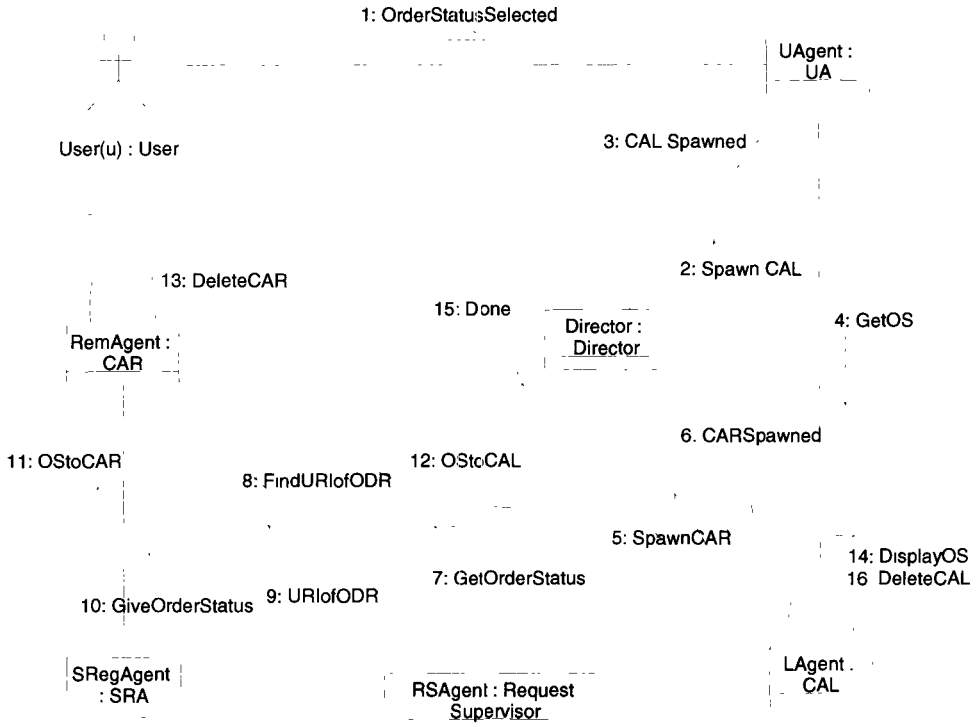

Figure A.28: Get Order Status Use Case Collaboration Diagram

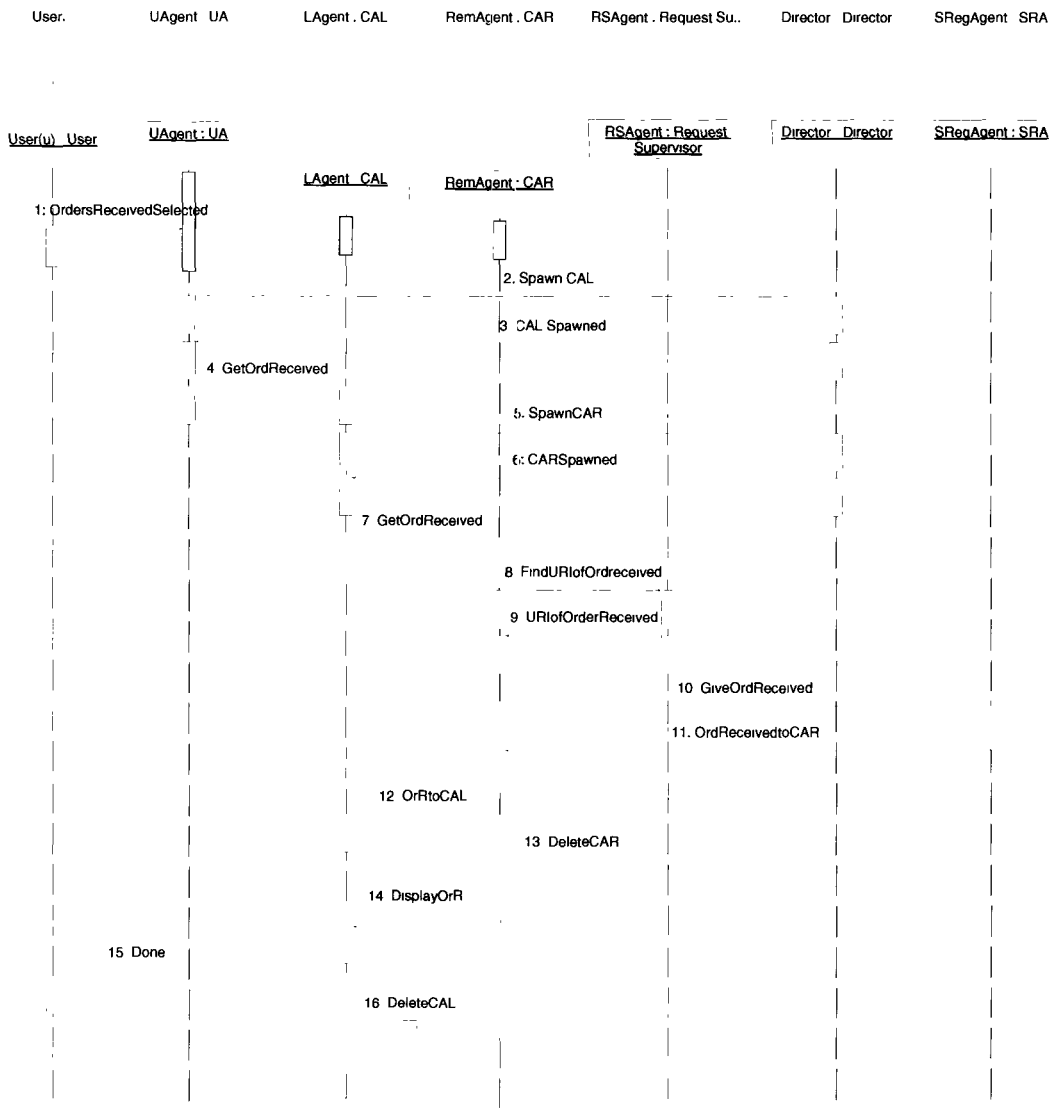

Figure A.29: Get Orders Received Use Case Sequence Diagram

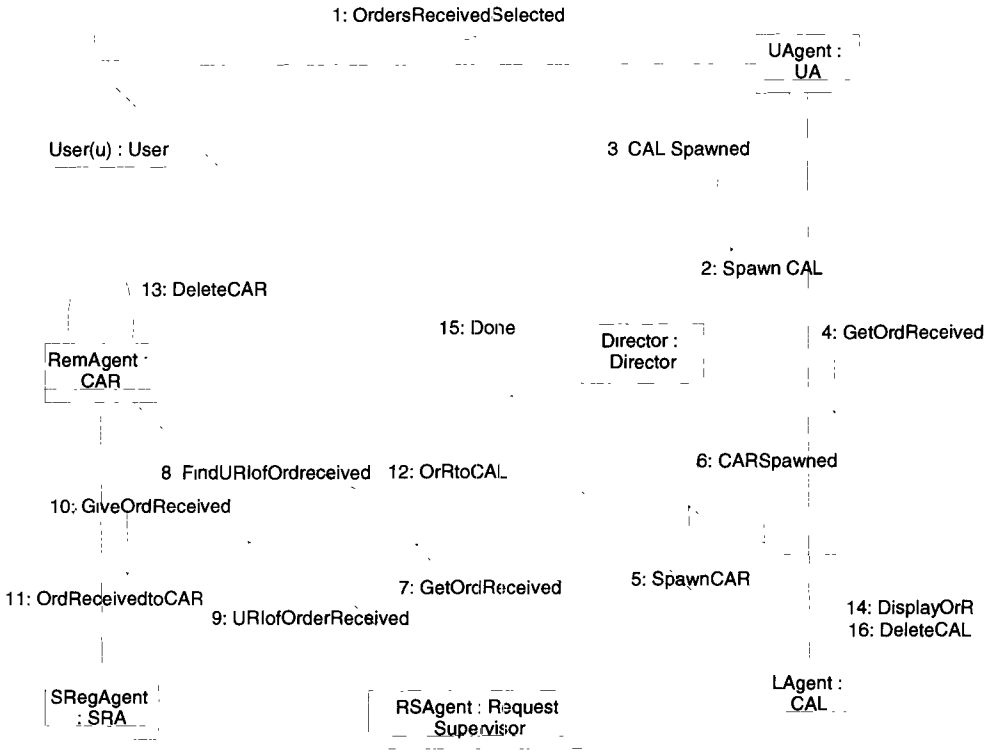

Figure A.30: Get Orders Received Use Case Collaboration Diagram

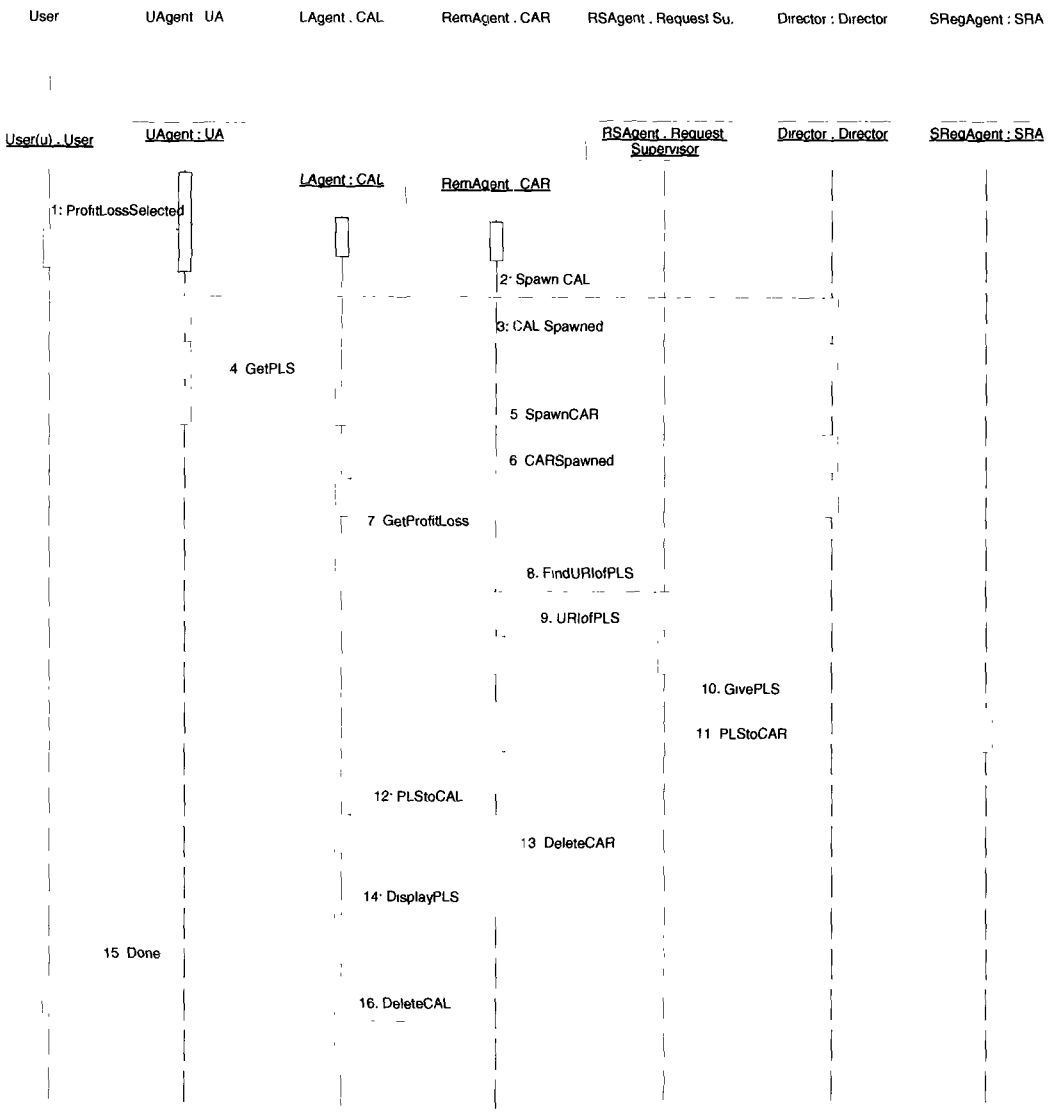

Figure A.31: Get Profit Loss Use Case Sequence Diagram

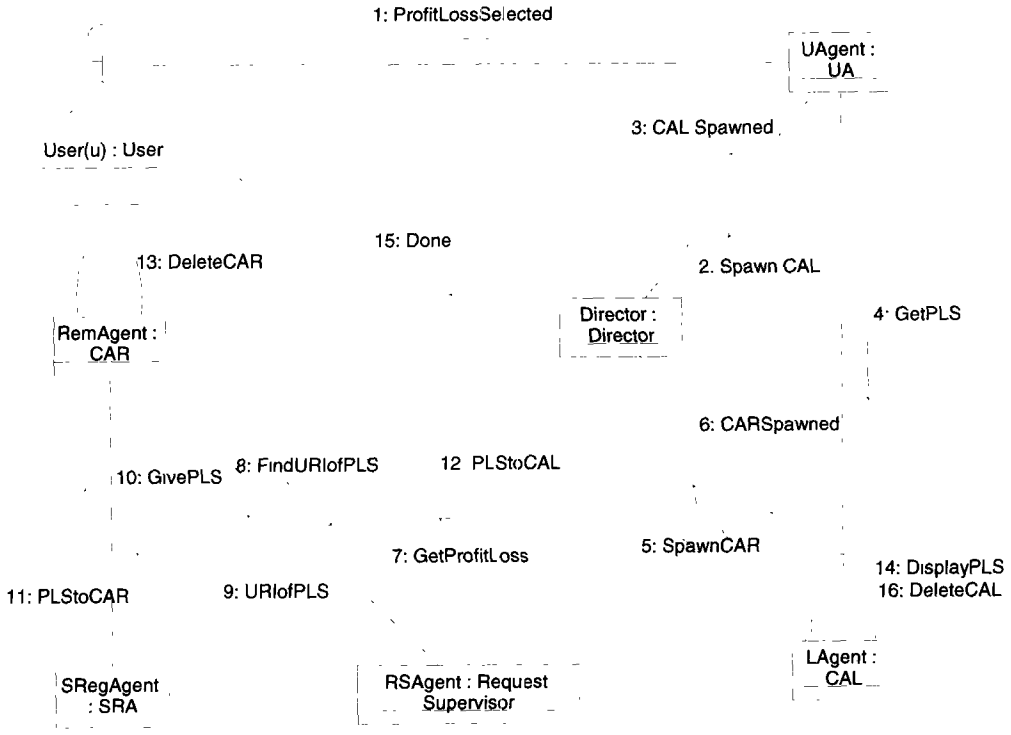

Figure A.32: Get Profit Loss Use Case Collaboration Diagram

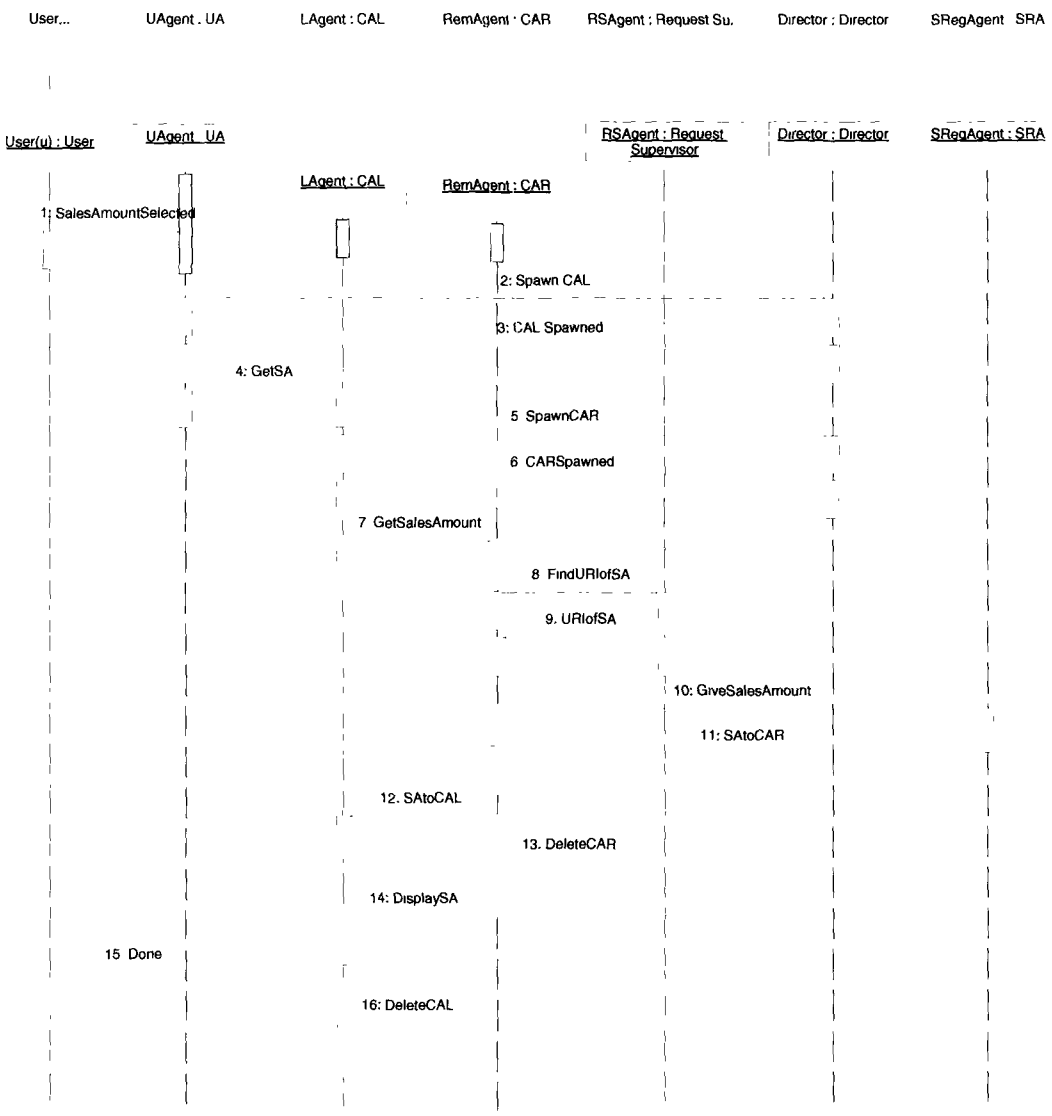

Figure A.33: Get Sales Amount Use Case Sequence Diagram

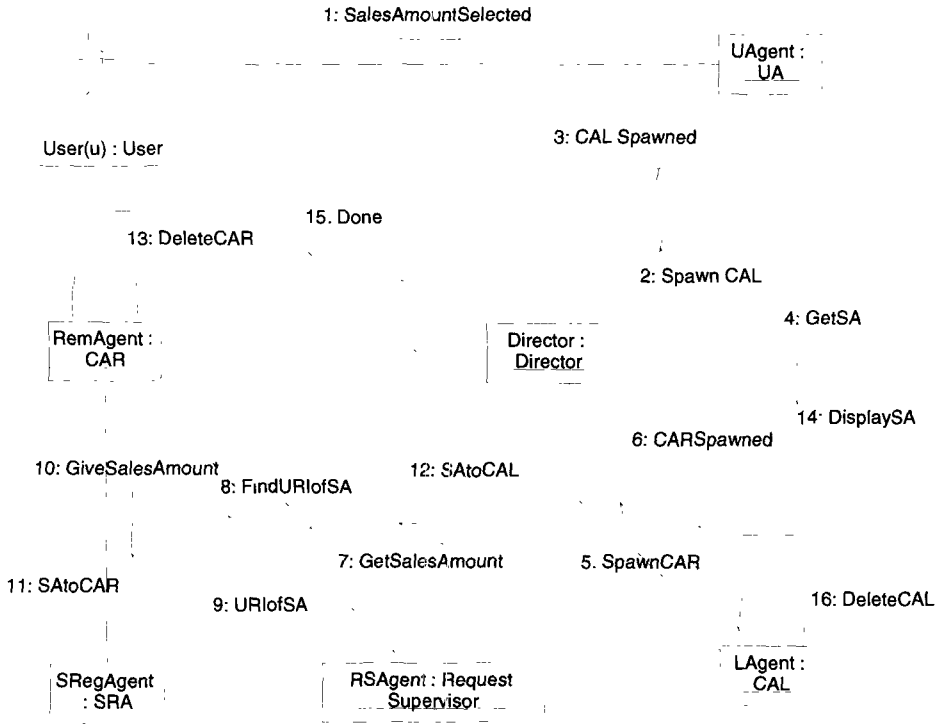

Figure A.34: Get Sales Amount Use Case Collaboration Diagram