A Study on Healthcare Workflow

By
Sumagna Patnaik

Department of Computer and Information Sciences
School of Mathematics and Computer & Information Sciences
University of Hyderabad
Hyderabad - 500046
INDIA

February, 2009

A
Thesis
on

A Study on Healthcare Workflow

by

Sumagna Patnaik

submitted in partial fulfillment of the requirements for the award of Degree of Doctor of Philosophy

Department of Computer and Information Sciences
School of Mathematics and Computer & Information Sciences
University of Hyderabad
Hyderabad - 500046
INDIA

February, 2009

Department of Computer and Information Sciences
School of Mathematics and Computer & Information Sciences
University of Hyderabad
Hyderabad - 500046
INDIA

CERTIFICATE

This is to certify that this thesis work entitled "**A Study on Healthcare Workflow**" being submitted by **Sumagna Patnaik** (Reg. No. 02MCPC03), in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy (Computer Science)** of the University of Hyderabad, Hyderabad, is a bonafide record of the work carried out by her under my supervision.

The research work submitted in this thesis is not submitted to any other University or Institution for the award of any degree or diploma.

Prof. Hrushikesha Mohanty
Supervisor,
Department of CIS,
University of Hyderabad,
Hyderabad–500046

Prof. Arun Agarwal
Head,
Department of CIS,
University of Hyderabad,
Hyderabad–500046

Prof. T. Amaranath
Dean,
School of MCIS,
University of Hyderabad,
Hyderabad–500046

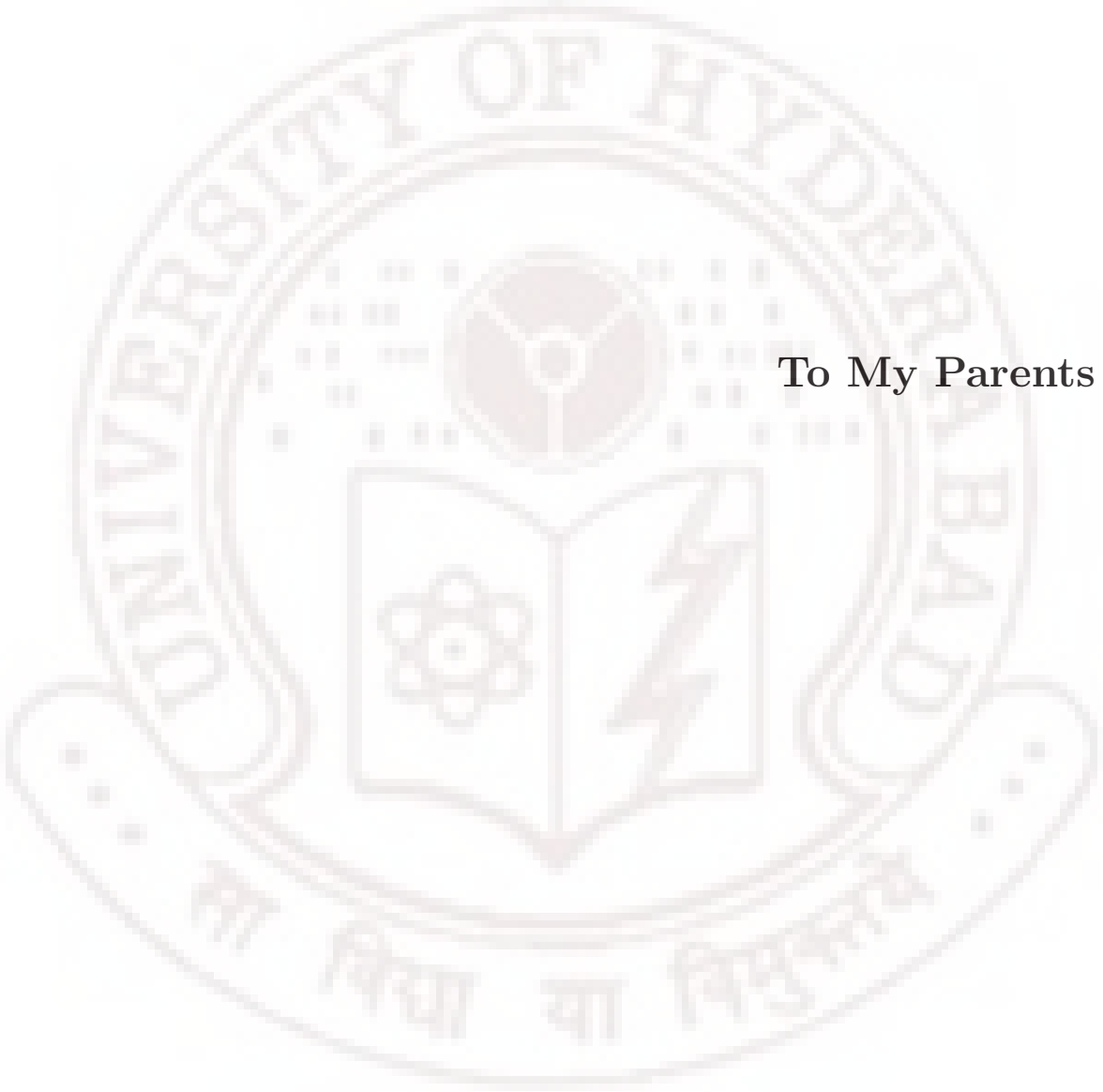
DECLARATION

I, **Sumagna Patnaik**, hereby declare that the work presented in this thesis entitled **"A Study on Healthcare Workflow"**, has been carried out by me under the supervision and guidance of **Prof. Hrushikesha Mohanty**, Department of Computer and Information Sciences, School of Mathematics and Computer & Information Sciences, University of Hyderabad, Hyderabad - 500046, India.

This research work is submitted as per the ordinances of the University, in partial fulfillment of the requirements for the award of the degree of **Doctor of Philosophy (Computer Science)** of the University of Hyderabad, Hyderabad.

I declare, to the best of my knowledge, that either full or part of this thesis has not been or is being concurrently submitted for the award of any other degree/diploma to any other Institute/University.

Sumagna Patnaik
Regd. No. 02MCPC03



To My Parents

ABSTRACT

Workflow technology is being researched in healthcare domain to manage complex tasks involved in treating a patient particularly for the one suffering from complex ailments. Currently corporate healthcare units have started in making use of workflow technology to deliver smart services to patients. But, thesis is limited to management issues e.g. healthcare resource management. In this work we focus on modeling a workflow that is patient-centric. It integrates the roles of doctor(s) as well as healthcare staffs (diagnostic staff, nurses etc.) associated in treatment of a patient. We propose a model called Treatflow to specify a treatment by sequencing treatment activities as required to treat a patient. Structured design specifies a system diagrammatically while formal methods use mathematical techniques. In order to avail advantages due to both the approaches, we propose a hybrid approach to specify healthcare workflow. Primitives with mathematical rules are proposed for Treatflow design and verification.

Models for healthcare processes could be complex like the processes observed in business domains. Modeling healthcare activities at one level could be complex and cognitively loaded. Hence we propose a modular approach to healthcare process specification. We have several operators to facilitate modular composition of healthcare workflow. Using these operators a medical practitioner can specify a treatment plan in an expression. We also propose rewriting rules to generate alternate equivalent treatment plans to facilitate decision making by doctors,patients as well as healthcare managers.

Healthcare workflow as directly deals with human life must specify precautions to exception i.e.undesired situations during its executions. We analyze the domain and present a comprehensive view on genesis of exceptions and corresponding actions. Treatflow Management System (TFMS) aims at automating a treatment process for creation, monitoring and termination of a Treatflow with maintenance of healthcare history. We propose an architecture to implement Treatflow Management System. A requirement analysis for such a system considering users viz. (doctors, patients and staffs) and their roles in treatment process have been discussed. Some of the results due to this research are reported in [[65],[64],[63],[55]].

Acknowledgments

My deepest gratitude to my supervisor **Prof.Hrushikesha Mohanty**, Department of Computer and Information Sciences, School of Mathematics and Computer & Information Sciences, University of Hyderabad for his valuable advice, constant guidance and encouragement during the entire period of my work. He has been a mentor, a coach and a guide to help me to get exposure to other institutes and universities also. I have learnt a lot from him lest it would not have been possible for me to see the final form of my thesis. His constant motivation filled me with the courage to strive through one of the most challenging period of my life making it one of my most joyful and memorable time as well. I express my deep sense of gratitude to my supervisor for taking interest in me and advising me a path to follow which would help me in building a successful career.

I am immensely thankful to **Prof.Arun Agarwal**, Head, Department of Computer and Information Sciences for providing me the necessary infrastructure to carry out this work. I would like to express my sincere thanks to **Prof.T Amaranath**,Dean, School of Mathematics and Computer & Information Sciences for his cooperation. I also like to put on record the valuable advices suggested by **Dr.Chakravarthy Bhagvati** and **Dr.S Durga Bhavani** as members of the doctoral review committee. I am indebted to all of them.

I wish to offer my sincere thanks to the staff of Artificial Intelligence Lab for allowing me to use required equipment whenever I needed.

I am sincerely thankful to Dr.Alok Kumar Pattanayak, B.H.E.L. General Hospital, Hyderabad for providing necessary data and case studies.

My heartfelt thanks goes to Mrs.Anjali Mohanty for her unconditional support and blessings. Special thanks to my daughter (Ruchi), family members who have been waiting for this day. In all crisis their whole-hearted love & support was extremely helpful to complete this thesis.

I also would like to thank the management of St.Ann's P.G. College, Mehdipatnam, Hyderabad for permitting me to carry out my doctoral work.

Although there are lots of other people whom I wish to thank and there are many of them whom I know I have missed ...

Sumagna Patnaik

Contents

1 Introduction **1**
- 1.1 Motivation . . . 5
- 1.2 Issues for Investigation . . . 5
- 1.3 Dissertation Outline . . . 8

2 Survey **10**
- 2.1 Introduction . . . 10
- 2.2 Workflow and its Management . . . 11
- 2.3 Workflow Modeling and its Features . . . 13
- 2.4 Workflow Modeling Techniques . . . 16
  - 2.4.1 Formal Methods in Workflow . . . 16
  - 2.4.2 Workflow Modeling with EPCs . . . 20
  - 2.4.3 Workflow Modeling with Petrinet . . . 22
  - 2.4.4 Workflow Modeling with UML . . . 27
  - 2.4.5 A Comparison . . . 32
- 2.5 Workflow Analysis . . . 32
- 2.6 Expected Features . . . 38
- 2.7 Healthcare Workflow . . . 40
  - 2.7.1 Features of Healthcare Workflow . . . 41
  - 2.7.2 Models . . . 42
  - 2.7.3 Verification . . . 49
- 2.8 Summary . . . 51

3 Modeling Treatflow Design Primitives and Assertions **53**
- 3.1 Introduction . . . 53
- 3.2 Specifying a Treatflow . . . 55
- 3.3 Modeling Treatflow Activities . . . 57
- 3.4 Treatflow Verification . . . 72
  - 3.4.1 Verification algorithm . . . 80
- 3.5 Modularization of Treatment . . . 86
  - 3.5.1 Motivating example . . . 88
  - 3.5.2 Module Specification: . . . 89
  - 3.5.3 Module Composition: . . . 92
  - 3.5.4 Rewriting of a Treatment plan . . . 96
- 3.6 Exception Management in case of Treatment . . . 99

3.6.1 Exception in Treatflow . . . 101
3.6.2 Exception Analysis . . . 104
3.6.3 Specification of Exception . . . 112
3.6.4 Exception Processing . . . 115
3.7 Summary . . . 118

4 Software System Architecture on Treatment Plan in Healthcare System 120
4.1 Introduction . . . 120
4.2 Related Work . . . 121
4.3 Requirement Analysis . . . 123
4.4 System Architecture . . . 131
4.5 Module Design . . . 135
4.6 System Behavioral Analysis . . . 139
4.7 Summary . . . 147

5 Conclusions & Future Work 149
5.1 Summary of the Work Done . . . 150
5.2 Future Work . . . 152

List of Figures

2.1 Sequence and Parallel . . . 15
2.2 Choice and Iterative . . . 15
2.3 Workflow Modeling using EPCs . . . 21
2.4 Classical Petrinet . . . 22
2.5 Dynamic Behavior of Classical Petrinet . . . 22
2.6 Workflow Modeling using Petrinets . . . 24
2.7 Colored Petrinet . . . 24
2.8 Time Petrinet . . . 26
2.9 Hierarchical Petrinet . . . 27
2.10 Workflow Modeling using Activity Diagram . . . 28
2.11 Workflow Modeling using Statechart and State transition diagram 30
2.12 Example for FSM . . . 31
2.13 Healthcare in WF-net . . . 44
2.14 Petrinets based Healthcare Workflow . . . 45

3.1 Treatflow Diagram . . . 58
3.2 Linear Activities . . . 62
3.3 Parallel Activities . . . 63
3.4 Split and Merge Activities . . . 64
3.5 Repeating Activities . . . 67
3.6 Nested Repeating Activities . . . 68
3.7 Choice Activities . . . 69
3.8 Cooperative and Supportive Activities . . . 71
3.9 Treatflow Graph . . . 74
3.10 Treatflow Graph for Fever Treatment . . . 74
3.11 Incomplete Disorder in a Treatflow Graph . . . 75
3.12 Synchronization Disorder in a Treatflow Graph . . . 76
3.13 Deadlock Disorder in a Treatflow Graph . . . 77
3.14 Retention Conflict in Treatflow . . . 79
3.15 Contextual conflict in Treatflow . . . 80
3.16 Expectation conflict in Treatflow . . . 81
3.17 Treatflow Modules . . . 89
3.18 Exceptions in case of Treatflow . . . 103
3.19 Exception Categorization . . . 111
3.20 Processing of Exceptions . . . 116
3.21 Logging of Exceptions . . . 116

4.1 Use case Diagram for Patient 124
4.2 Activity Diagrams for Patient 126
4.3 Use case Diagram for Doctor 128
4.4 Activity Diagrams for Doctor 129
4.5 Use case Diagram for Staff 130
4.6 Activity Diagrams for Staff 131
4.7 A Software System Architecture for Treatflow Management System 132
4.8 Representation of Module 132
4.9 Class Diagram for Treatflow 136
4.10 Class Diagram for Authentication 137
4.11 Class Diagram for Administration 138
4.12 Class Diagram for Treatplan 139
4.13 Class Diagram for Treat_Management 140
4.14 Class Diagram for Handle_Exigencies 141
4.15 Class Diagram for Role_Based_Enquiry 142
4.16 Interaction Diagram for Authentication requirement 142
4.17 Interaction Diagram for Handle_Exigencies requirement 143
4.18 Interaction Diagram for Make_Treatment requirement 143
4.19 Interaction Diagram for Role Based Enquiry 144
4.20 Interaction Diagram for Recording 144
4.21 Interaction Diagram for Registration 145
4.22 Interaction Diagram for Treatflow 145
4.23 Interaction Diagram for Update Status 146
4.24 Interaction Diagram for Update Treatment 147

List of Tables

2.1 Workflow Modeling Techniques 31

3.1 Operators and Symbols 96
3.2 Operators and Properties 96
3.3 Exception Analysis 112
3.4 Exceptional Events with Actions 113
3.5 Types of Actions 113

Chapter 1

Introduction

Business objectives are met by successful execution of several business processes. Business processes are represented as sets of tasks, where each task carries out a host of well defined activities. A task includes logical steps of actions that contribute towards the business goals. A sequence of such logical steps make a Workflow. So, a workflow can be defined as a process consisting of a number of activities that need to be coordinated to achieve a particular business goal. For example, business processes in manufacturing and banking domains can be modeled as workflows. A workflow management system (WFMS) is a system that supports process specification, enactment, monitoring and coordination. Execution of a workflow is based on business logic as specified in workflow model.

Being able to change a workflow model or to create a new one quickly provides competitive advantage to organization to carry out business process efficiently. .In order to facilitate workflow modeling we require a convention to represent workflows. A workflow designer follows different modeling approaches to model business processes using software tools. In general, the workflow modeling approaches fall into two categories: structured and formal. Structured approach follows a systematic way to compose business processes and the structured design is modular. Each module is represented by a graphic notation and modules are connected by edges. Further a module can be decomposed into submodules

and so on to atomic activities. In a workflow atomic activities are connected by directed edges. The direction of an edge shows the control flow during workflow executions. Thus this model provides a blue print of workflow system design. Such modeling approaches are popular among software engineers because of their visual appeals. A designer intuitively verifies workflow system design by going through design diagrams. In contrast to the structured approach, a formal approach insists on mathematical statements on business process behaviors and interactions. The use of a formal approach helps the workflow designers to detect inconsistency, reduce ambiguity in the development and reason about the system. Hence, the use of formal methods is being seriously promoted among the designers of workflow systems.

Of late, workflow technology is being experimented in healthcare domain to manage complex tasks involved in treating a patient particularly for the one suffering from complex ailments. Corporate healthcare organizations intend to make use of workflow technology to deliver smart services to patient. However, it is observed that the classical workflow concepts being used in manufacturing domain are not readily usable in healthcare domain due to the specialties and challenges the domain projects.

In order to avail advantages from both the structured and the formal approaches, we propose a hybrid approach to specify workflow in case of treatment in healthcare. Some primitives with mathematical rules are identified for workflow design. Firstly a structured method is proposed to specify treatment by sequencing treatment activities. For healthcare workflow modeling, one can make use of the modeling primitives like linear, parallel, choice, repeating, nested repeating, split & merge, supportive and cooperative to specify treatment activities. These sequencing of treatment activities can be termed as *Treatflow*. A Treatflow is similar to traditional workflow used in manufacturing and other domain. Further,

it is augmented by special primitives viz.*supportive,cooperative* those are found useful to model activities in treating a patient. Secondly for each design primitive, mathematical specification is proposed to detect inconsistency, reduce ambiguity in patient treatment.

Once a Treatflow is specified, it is essential to verify the flow before making it operational as some mistakes might have crept in inadvertently while it is being composed by a doctor-a non-computer science professional. Traditionally, verification of workflows deals with structural verification comprising a study on lack of synchronization and deadlock detection [86][2]. In case of Treatflow verification we have discussed various verification issues viz. incompleteness, lack of synchronization, deadlock, retention conflict, contextual conflict, expectation conflict and temporal conflict keeping the relevance of healthcare domain in view.

Treatment in healthcare deals with complex ailments of patients. Modeling Treatflow for all these treatments becomes complex to design, verify and follow. This necessitates modular design of healthcare activities. Building a modular system is appreciated for its uses in enhancing system understanding and for its ability in dealing with large and complex system i.e. by making a system modular thus scalable. In healthcare a treatment for a patient can be composed of several treatment modules where a treatment module is specified with several treatment activities. A specification of a Treatment Module needs to provide comprehensive view that can be useful for automation of Treatflow activities and also enables a practitioner and a patient to have a quick view on a treatment. Ideally, a Treatment Module needs to specify conditions at which it can be applied to a patient. After specifying treatment modules, these modules can be composed to form a treatment plan termed as *Treatplan*. For composition purpose several composition operators e.g. linear, parallel, choice, supportive and cooperative have been defined .Therefore, a prescription written by a doctor can be specified by an

expression of treatment modules. Due to non-availability of resources or incompatibility of treatments, there is a need to change a Treatplan and to write an equivalent Treatplan. This can be done with the help of rewriting rules as well as properties of composition operators. Treatment in healthcare needs to be extremely sensitive for safety of a patient i,e an exception in health parameter needs to be treated. So exception handling is a necessary part of treatment workflow. Studying exceptions in healthcare workflow and exploring how such exceptions can be handled promptly and appropriately within the workflow system are to be studied to improve healthcare quality and efficiency.

The proposed Treatflow Management System aims at automating treatment process, it is different from existing healthcare systems which focus mainly on patient registration, bill collections, maintaining the history of medication undergone by patients i.e especially on issues related to hospital management and EPR: electronic patient record management. We propose a system where the authenticated users (doctor, staff and patient) can use this system, On registration of a patient, a doctor can make use of the proposed system to synthesize a treatment process by choosing treatment modules for treating a patient. And then a synthesized treatment process is instantiated; that means the patient is subjected to the treatment. A patient can record his/her health status as well as health related exigencies. Using the system, the doctor can issue health advisory to the patient in case of exigencies and can change the treatment process, if necessary. Users of the system can access treatment related information based on their roles. The main objective of Treatflow Management System (TFMS) is not only for automating the existing treatment process but also for bringing transparency in the treatment process.

1.1 Motivation

To our knowledge no effort has been made to model a treatment. Instead current work concentrates on modeling workflows on hospital management issues like personnel scheduling, resource procurement, ward management etc. In this work we focus on modeling a workflow that is patient-centric integrating the roles of doctors as well as healthcare staffs (diagnostic staff, nurses etc.) involved in treatment of a patient. In order to model treatment of a patient suffering from ailment(s), we have extended primitives of activity diagram for making it expressive to model workflows in healthcare domain.

1.2 Issues for Investigation

This thesis first focuses not only in differentiating workflow being practiced in manufacturing industry with the workflow of healthcare domain, but also identifies exciting features in healthcare workflow. The *issues investigated* in this thesis include Treatment Modeling, Treatflow Verification, Modularization of Treatment, Exception Management, Architecture for TFMS. Below, we deal upon each issue individually.

- **Treatment Modeling**

  Structured design specifies a healthcare system diagrammatically while formal methods use mathematical techniques. In order to avail advantages from both the approaches, we propose a hybrid approach to specify Treatflow in healthcare. In order to design Treatflow, primitives like linear, parallel, split and merge, repeating, nested repeating and choice are identified. For each primitive mathematical rules are specified so that the approach not only helps in Treatflow design but also for its verification. A resultant model is not only structured but also equipped with rigor that makes the model safe

for its use in healthcare domain that deals with human.

- **Treatflow Verification**

  Once a Treatflow is specified, it is essential to verify a Treatflow model before making it operational. Treatflow verification is required in order to ensure the safety of a treatment. For the purpose of verification Treatflow is represented as a graph comprising of nodes and edges. Where a node represents a treatment and an directed edge represents control in treatment execution. Keeping healthcare domain in view verification issues like lack of synchronization, deadlock detection, incompleteness check, retention conflict, contextual conflict, expectation conflict and delay conflict etc must be taken into consideration.

- **Treatment Modularization**

  Modeling a treatment for a patient suffering from multiple ailments or for a chronic patient becomes very critical to understand and to transmit it to supporting healthcare staff. The issue is similar to the representation of a design of a complex software system for which modularization has been recognized as a solution [82].This has motivated modularization of treatment specification. The safeguards in modules must be specified which can alert medico, nursing staff as well as patients on prescription and administration of wrong medications. In case of incompatibility of treatments doctors may need to generate alternate Treatplans to facilitate decision making by doctors,patients as well as healthcare managers. These requirements motivate us on composition of a Treatplan with treatment modules as well as rewriting rules for an alternative Treatplan.

- **Exception Management**

  Healthcare treatment often requires human participation of doctors, patients

and collaborating staff ranging from paramedical to administrative personnel. Though each of them play well defined roles still human actions are liable to lapses. Patient treatment involves life risks; hence healthcare workflow management should have means to deal with these lapses. These factors emphasize the necessity of exception management in healthcare domain.

- **Architecture for TFMS**

  In order to automate treatment process which is different from current healthcare systems that focus especially on hospital management and EPR: electronic patient record management, we propose TFMS (Treatflow Management System). The purpose of TFMS is to specify and verify a Treatflow as well as to execute, monitor and handle exceptions while treating a patient. This system is to be used as an aid by different actors to participate in treatment process. We feel, such a system will be helpful for both doctors and patients in managing treatment process. On registration of a patient symptoms must be recorded by an authorized administrative staff. A doctor can plan a treatment for the patient by selecting treatment modules from a treatment repository. Such a repository could be made available by doctors recording their experiences and practices. A patient is subjected to treatment. While undergoing treatment a patient can provide information on his/her health related exigencies, update health status and can access treatment related informations. After going through patient's health status and health related exigencies, a doctor can change a treatment. This motivates us to propose a system architecture which aims at addressing these requirements.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows.

Chapter 2 first describes a survey of the background that is necessary to understand the notion of workflow and its management. Workflow modeling and its features are described in 2.3. Several workflow modeling techniques are presented in section 2.4. In order to achieve correctness,efficiency and effectiveness of business process supported by workflow management system, it is necessary to analyze a workflow process definition before it is put into production which is discussed in section 2.5. Some of the exciting features of workflow are described in section 2.6. Healthcare workflow and its features as well as healthcare workflow modeling are discussed in section 2.7. Finally some of the proposed methods for modeling healthcare workflow are presented.

Chapter 3 presents a hybrid approach to model healthcare workflow systems. Firstly it describes about modeling treatment activities and its related work in healthcare domain. To specify a Treatflow in healthcare domain with the help of a case study is discussed in section 3.2. In order to model a Treatflow using primitives like linear, parallel, choice, repeating, nested repeating and cooperative and supportive, a visual object have been identified so that workflows can be presented and viewed diagrammatically. Mathematical rules for each design primitive have also been defined in section 3.3. In order to verify a Treatflow, several verification issues are discussed in section 3.4. With the help of a verification algorithm various verification issues are discussed considering Treatflow as a graph.

Complex health problem treatment is modularized not only for administering the treatment but also to monitor and to modify if necessary. These requirements motivate us to plan a treatment in modular form which is discussed

in section 3.5. In order to motivate readers on modularization, this section describes this concept with the help of a case study. Treatment module can not only be specified and composed to form a treatment plan called *Treatplan* but also can generate an equivalent treatment plan using rewriting rules which is also discussed in this section. Exception management in treatment process is discussed in section 3.6 and presented a comprehensive framework to deal with it. In this section we analyzed the domain and presented a comprehensive view on genesis of exceptions and their corresponding actions.

Chapter 4 discusses about Treatflow Management System (TFMS) which aims at automating treatment process. Here we propose a framework to implement Treatflow Management System. A requirement analysis for such a system considering users viz. (doctors, patients and staffs) and their roles in treatment process has been discussed in section 4.3. System architecture and its component are discussed in section 4.4 A broad picture on design of each module with the help of class diagrams is discussed in 4.5. Analysis is done with the help of interaction diagram. Based on each user requirements how the modules interact among themselves is discussed in section 4.6

Chapter 5 gives a brief conclusion, summarizing the work described in this thesis and some direction for future work.

Chapter 2

Survey

This chapter surveys basic workflow modeling features, various workflow modeling techniques and verification of workflow management systems. We review some of the expected features of workflow. It also identifies the basic issues of research interests associated with workflow in healthcare. Some research related to modeling and verification in case of healthcare workflow are also discussed.

2.1 Introduction

An organization performs several business processes to achieve its goals. A process is carried out by people/systems where each of them carries out tasks assigned to. For example, in an assembly sub-floor technicians engaged in making a product, participate in a definite sequence. A defined order of working with an objective to produce an object (to realize a business goal) is termed as a workflow. For example, a business process "Issue a Loan" may consist of several tasks like: reviewing an application for completeness and accuracy, executing a credit check, creating a new loan contract, sending it to the client, checking the contract form on return, setting up of payment schedule and finally issuing payment cheques. Sometimes a workflow of a business process may span over several organizations. This happens for the case where multiple agencies are involved in making a product (say aircraft manufacturing). In early days, assigning tasks to people and monitoring them were carried out manually. Of course, such manually maintained business

process is prone to human error. Later, at the advent of technology, automation has been brought in to manage workflow. At the beginning, a-piece-meal approach was adopted to automate certain activities like material management, personnel management. Later in several areas like distributed computing, DBMS, communication technologies, there has been attempt in total automation of organization activities. This has resulted in systems called Workflow Management Systems (WFMS).

In the following, firstly workflow, its management and workflow modeling features are discussed in section 2.3. Different workflow modeling techniques, formal methods in workflow as well as comparison among them, are presented in section 2.4. Analysis of workflow is discussed in section 2.5. Some of the expected features of workflow are identified in section 2.6. Details about healthcare workflow is presented in section 2.7. Finally a summary of the chapter is given.

2.2 Workflow and its Management

Workflow is defined as the automation of a business process, in whole or parts, during which documents, information or tasks are passed from one participant to another for action according to a set of procedural rules. Therefore workflow is a sequence of tasks organized to accomplish some business process. These business processes can be executed in sequence or parallel to accomplish the goals of an organization. According to the needs, a workflow can be started manually or automatically at a given time or situation. If there is a problem in executing a particular process, workflow tools have the capability to suspend the workflow and then the workflow can be started once again to run the process. It also defines the order of task invocation or condition under which task must be invoked, task synchronization and information flow. Patient treatment in a hospital, banking

services, manufacturing and catalog ordering processes and many more activities in todays business life can be modeled as a workflow.

Workflow in a business process can be managed by a workflow management system. In general term a Workflow Management System (WFMS) is a computer software that supports, controls, manages and collaborates among business processes through automatic flow of tasks, electronic information, and documents [5]. It also has a provision to integrate with other workflow systems. A WFMS can be used as a tool to model, specify, analyze and execute a workflow [35]. It also helps organizations to schedule their underlying processes and to carry out activities effectively and efficiently. Currently, most of the business houses carry out their activities at geographically distributed places. A distributed WFMS is a natural choice to automate business activities of such business houses[94] .

Workflow management system consists of two basic components: First component is the workflow modeling component or *buildtime system* which enables administrators or analysts to define processes and activities, analyze and assign them to people. It also provides graphical modeling tools that help designers to design, test and validate workflow processes. Second component is the workflow execution component, sometimes called *runtime system* composed of software which are responsible for creating and controlling instances of business processes at run time. It helps in coordinating and performing the processes and activities [8] and provides an execution interface for end-users.

Workflow software records information of business process execution into a repository. Using workflow monitoring tools information like starting and finishing time, status (started, stopped, succeeded), errors and exceptions handled in business processes can be easily retrieved from repository by querying for further analysis. Workflows can also be used to notify the process status through email to the top most executives of the company or whoever in need of it.

Descriptive level of WFMS deals with describing the modeling aspect of workflow system. It describes what a WFMS will do, which procedure will it implement/monitor and what data it stores, for which organization WFMS is a part of and which type of user-interface will it present and so on. *Execution level* of WFMS is further decomposed into enactment and management. *Enactment* deals with the effective execution of activity specified at the process description i,e what activities are being performed and who is performing the activity. *Management* level has privilege access to information about current and past work cases, statistical information about users, processes, activities and so on [7].

Usually workflow management has been process focused i,e coordinating activities of people working on a common task or project. Once a process is defined, workflow management system makes sure that activities occur in proper sequence and the users are informed. A business process is a collection of activities designed to produce a specific outputs for a particular customer. It implies a strong emphasis on how the work is done in an organization instead of focusing on what. Therefore a process has a specific ordering of work activities with a beginning and an end with clearly defined inputs and outputs. Modeling is an efficient and effective way to represent the need of an organization. It provides information to members of an organization to understand and communicate business rules and processes. In the next section we will discuss about workflow modeling and its features.

2.3 Workflow Modeling and its Features

Modeling is an efficient and effective way to represent the need of an organization. Workflow involves a set of activities to be carried out in order to meet some predefined requirements of an organization. Workflow modeling is a key activity

during process improvement because techniques provide an effective means of analyzing existing workflow and comparing proposed improvements [80]. Workflow processes are marked by three dimensions: (1) control-flow dimension (2) resource dimension and (3) case dimension. Control dimension is concerned only with partial ordering of tasks. Tasks need to be executed are identified and the routing of cases along with these tasks is determined i,e conditional, sequence, parallel and iterative routings are the typical structures specified in control-flow dimension as shown in Figure 2.1 and Figure 2.2. In resource dimension, resources (human or device) are classified by identifying their roles. Lastly in case dimension workflow is concerned with individual case which are executed according to process definition by the proper resource dimension [84].

Therefore workflow modeling is nothing but the representation of activities that compose workflow processes, their execution sequence and relationships, agents responsible for their execution, and resources that are used during execution [59]. In an organization workflow management which is already existed, but recently workflow modeling has become the main focal point of researchers. Researchers have proposed different modeling techniques. Several other techniques such as flowcharts, state transition diagram, data flow diagram have been proposed to represent business process, but some of the conventional modeling techniques have some drawbacks: lack of formal semantics and absence of powerful analysis methods and tools [90].

Workflow Modeling Features The following perspectives are relevant for workflow modeling and execution [19] [88].

- Control-flow (Process) perspectives: Describes the activities and their order of execution e.g sequential, parallelism, choice and join synchronization
- Data(Information) perspectives: Deals with business and processing data on control perspectives

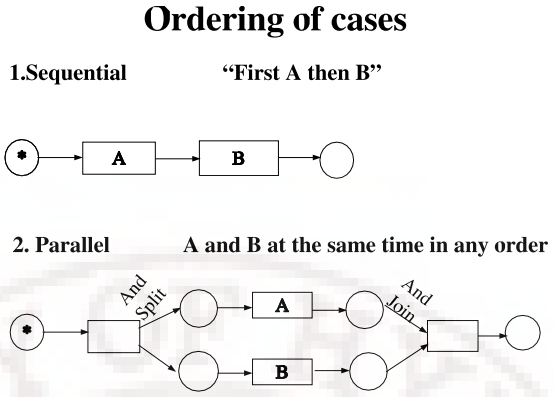


Figure 2.1: Sequence and Parallel

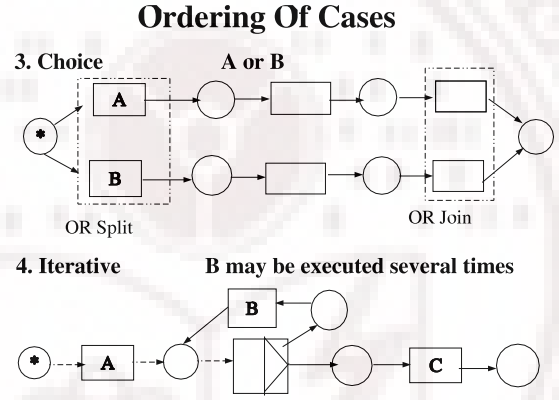


Figure 2.2: Choice and Iterative

- Resource (Organization) perspectives: Provides an organizational structure in the form of human and device roles responsible for executing the activities
- Operational (Application) perspectives: Described the elementary action executed by activities.

In a workflow model the following minimal information are associated with an activity [62] .

- Pre and post conditions (conditions to be met before and after an activity)
- Who has control over the activity (role)
- Which other activities are required to complete the activity

- Input/output of an activity

Considering features of workflow modeling, several workflow modeling techniques are available which will be discussed in the next section.

2.4 Workflow Modeling Techniques

Workflow models explain the activity interdependency and timing, branching and merging of process flow, choice, looping and parallelism in much greater detail. Workflow Model is a collection of several activities where each activity is represented by a box with a verb describing an activity. For example, Check Credit History, Process an Order, Ship an Order are some of the examples for these [89]. In general, activities take inputs and transform them into outputs and the relationship between these activities is represented by arrows. The main objects or components in workflow modeling are activities or processes, one or more junctions and arrows. Junctions are nothing but a small box that enable branching or joining of processes. Several workflow modeling techniques are available, for example Petrinet, UML, Statechart, EPC and Formal Method etc..

2.4.1 Formal Methods in Workflow

Formal models are useful for specifying workflow properties and correctness study. For example, for specifying a property that needs to hold good during progress of workflow execution, one can make use of formal approach to state the property unambiguously because of its mathematical preciseness. In case of structured based modeling, specifying such property is cumbersome. Recognizing strength of formal approach we will review some of the recent works in this area. Before proceeding we will dwell upon two important publications from literature that laid emphasis on formal approach.

It has been shown that how mathematics can provide scientific foundation for modeling, describing and developing methods in software engineering [12]. It argues that while diagrammatic models provide syntactic approach to model still the approach lacks semantics to describe system properties. The author puts forward a mathematical approach so that a description model diagram can be expressed mathematically, and then to express relations among the models mathematically. Thus, diagrams together can project aspects of software systems that remain unspecified and individual models project system view in a discreet way. Here, the author shows refinement of a mathematical model is possible to specify system at different levels of abstraction.

Further, another paper [14] advocates the use of formal method in specifying a program. It identifies elements of structured design like assignment, if, if-else-if and while; and for each mathematical specification that mainly uses pre- and post conditions is proposed. It shows that a program can be thought of as a composition of blocks where each block is composed with a set of statements. Structured design primitives can be used to show the composition blocks as well as composition of each block. It shows a process following which one can refine a block into a set of statements which again can be specified by design structures accompanied by mathematical specifications. The paper shows that, a program not only can be well-defined by mathematical specifications but also can be verified.

There has also been some work reported on formal specification of workflow in wider context business process. In a recent work [93] CSP: Communicating Sequential Process has been used to specify complex workflow systems. A process-algebraic approach is suggested to model and verify workflow processes. Authors propose CSP process model to specify and verify the model by comparing model behavior to expected behavior of a workflow process. Process refinement algebra is used to define design and development of a workflow. Scalability of the

approach is shown by providing rules to integrate several workflows for building a complex workflow system. In order to demonstrate the strength of the model the authors have proposed a formal semantics for BPMN: Business Process modeling Notation in CSP so that workflow designer can use BPMN for design and at the same time can make use of corresponding CSP model for verification. The paper indicates that for verification several available CSP based model checkers like FDR can be used.

In another work [61][4] CTL: Computational Tree Logic has been proposed to model workflows. A workflow is considered as a collection of events and associated actions. It does not consider structure to model workflow. Rather, it models execution of a workflow as processing of events. At a given time during execution, system selects the event that is to be executed by asking available formal automata where each automation models a dependency among events. Thus all the automata for a work process define a partial order among events. And the order is to be maintained during execution of the workflow. Thus CTL based formal modeling of workflow is not only useful to specify a workflow but also to verify. However, one to one relation between dependency and automata, projects the requirement of a large number of automata for a moderately complex system. Scheduling an event by coordinating a large number of events becomes computationally expensive. Transaction logic to model workflow is proposed [11]. It proposes atomic formula to compose transactions of a workflow system. There are a number of different flavors of TR; all these flavors come with a model theory for the full logic; and a sound-and-complete proof theory for the logic programming fragment. The proof theory for the logic programming fragment can be used to execute programs. The flavors differ in the operations for composing atomic operations into complex operations. In all these flavors, atomic formulas (predicates) represent elementary actions or queries on a database. Sequential Transaction

Logic: For composing atomic formulae into complex ones, TR uses the usual classical operators ($\neg, \wedge, \vee, \leftarrow$) and two additional operators: serial, conjuction, denoted $\otimes$ and a modal operator for executional possibility, denoted as $\diamond$ .These operators mean the following:

1. serial conjuction: $\alpha \otimes \beta$ means first execute α and then execute β. M,$\pi \models \alpha \otimes \beta$ if and only if M,$\pi_1 \models \alpha$ and M,$\pi_2 \models \alpha$ and $\pi = \pi_1 \bullet \pi_2$. Here M is a path structure which determines the truth value of each formula on a path (indicating whether the formula can execute along a multipath), π is a path in M, and $\bullet$ denotes the concatenation of two paths.

2. classical conjunction: $\alpha \wedge \beta$ if and only if M, $\pi \models \alpha$ and M,$\pi \models \beta$.

3. classical disjunction: $\alpha \vee \beta$ means execute either α or β M,$\pi \models \alpha \vee \beta$ if and only if M, $\pi \models \alpha$ or M,$\pi \models \beta$

4. negation: $\neg\alpha$ means execute anything but α

5. subprocedure: $\alpha \leftarrow \beta$ means that to execute α one can execute β; M,$\pi \models \alpha$ if and only if $\alpha \leftarrow \beta$ is in M and M,$\pi \models \beta$.

A workflow is modeled as a set of TR formulas that describe dependency among tasks. Tasks are modeled using predicates, they can be either atomic or a subprocedure i.e a composition of atomic formulas. This allows modeling of workflow activities by using compositional operators like sequence, choice or concurrence. Workflow execution depends on satisfaction of constraints associated to each activity. Thus like CTL approach this Transactional logic approach becomes computationally expensive.

2.4.2 Workflow Modeling with EPCs

The main elements of event-driven process chain are functions, events and logical connectors. A function corresponds to an activity (task, process step) needs to be executed. Functions can be represented using hexagons. Event describes a situation before or after a function is executed which can be represented by rectangles. Functions are linked by events. An event may correspond to post-condition of one function and act as a pre-condition of another function. Logical connectors can be used to connect the activities and events. The control flows of a business process can be described by using three types of connectors like $\wedge$ (and), XOR (exclusive-or) and $\vee$ (OR).

EPC can be mathematically defined as

an EPC is of five tuples (E,F,C,T,A)

- E is a finite set of events
- F is a finite set of functions
- C is a finite set of logical connectors
- $T \in C \rightarrow \ \wedge$, XOR,$\vee$ is a function which maps each connector into a connector type.
- $A \subseteq (E\times F) \cup (F\times E) \cup (E\times C) \cup (C\times E) \cup (F\times C) \cup (C\times F) \cup (C\times C)$ is a set of arcs.

For example graphical representation of EPC as in Figure 2.3 for modeling the processing of a customer order. The process starts with an event of customer order received. First the customer order data is checked and the result of this function may be accepted or rejected. After the execution of function (customer compare order) XOR connector models the fact that either of the two events (customer order accepted or customer order rejected) holds. If the order is rejected then the

process stops. In case the order is accepted, availability is checked. If the article is not available then two functions are executed in parallel, purchase material and make production plan. If either the event article available and finished goods holds then the function ship order can be executed. One of the main advantage of

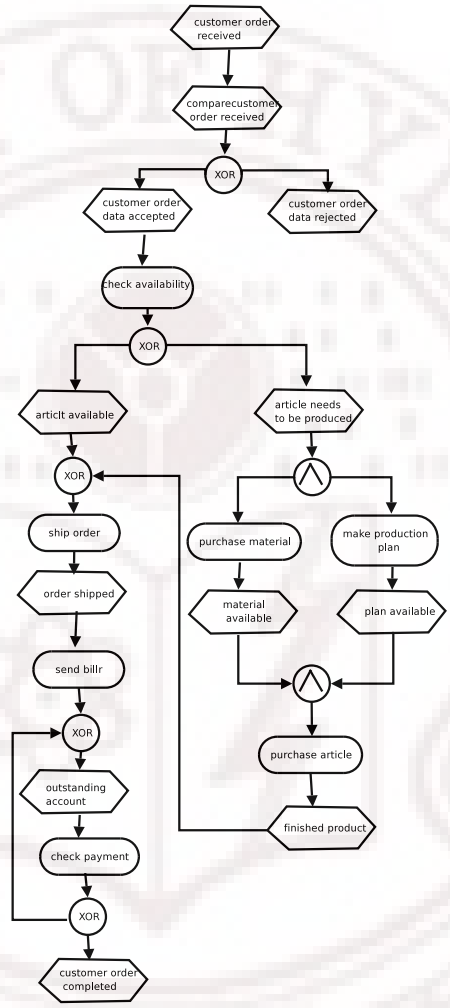


Figure 2.3: Workflow Modeling using EPCs

using EPCs is that it is easily understandable for end users [66] for its graphical representation. EPC is often used for capturing and discussing business processes with people who have never been trained in any kind of modeling technique (i.e worker on the shop floor). EPC creates a common platform for communication and the analysis of ideas [73]. .

2.4.3 Workflow Modeling with Petrinet

In order to implement a workflow system, it is first necessary to find a suitable means of designing and modeling a workflow process. Petrinets are class of modeling tools originated from the early work of Carl Adam Petri. Petrinets have well defined mathematical foundation and easy to use graphical feature. Classical Petrinet model has been used in many application areas, for example communication protocol, flexible manufacturing system and distributed information system. Among other modeling techniques, Petrinets are mainly used for quantitative and qualitative analysis of workflow and workflow system. The objective of qualitative system is to prove that the model is valid [40]. Classical petrinet is a directed

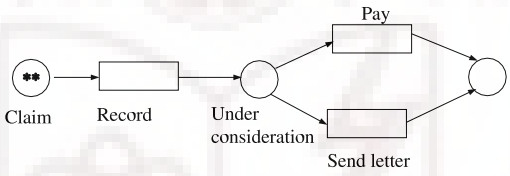


Figure 2.4: Classical Petrinet

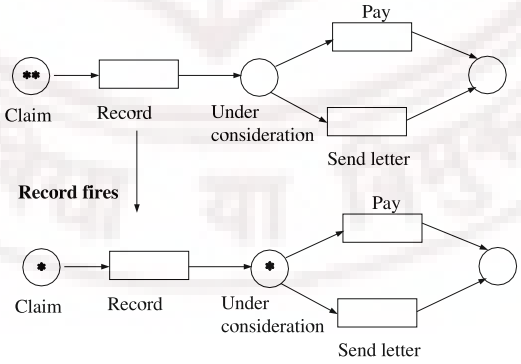


Figure 2.5: Dynamic Behavior of Classical Petrinet

graph with two types of nodes called places and transitions. An arc connects from place to transition and from transition to place. Formally a Petrinet is a tuple

PN = { P,T,F, M_0} where P is a set of places, T is a set of transitions and F = {P × T } U { T × P } flow relation between place to transition and from transition to place and M_0 is the initial marker

While representing graphically as in Figure 2.4 petrinet consists of three structural components: places, transition and arcs where places are drawn as circles and transitions are drawn as rectangles and arc as arrows. It can also be used to model asynchronous system with concurrency. The transitions represent events and the places represent states or conditions. Inputs and outputs are allowed only between places and transitions: one cannot go directly from one place to another without a transition. The dynamic behavior of a system can be represented using tokens as shown in Figure 2.5.

Tokens are represented as black dots in places. A transition is "active" when each of its input places contains a token and each active transition in the diagram is "fired" by removing one token from each input place and generating one in each output place. For example consider an automobile insurance company as shown in Figure 2.6. Where a workflow consists of five tasks submit-claim, check-insurance, contact-garage, send-letter and pay-damage. It is assumed that two tasks contact-garage and check-insurance may be performed simultaneously in order to determine whether the claim is justified. If the claim is justified, the damage is paid otherwise a rejection letter is sent to the claimant. However classical petrinets modeling for real system is found to be very large and complex and is inadequate to model complex industrial system. To solve these problems high level petrinets i.e extension to petrinet model have been proposed by authors [54]. Therefore an extended version of classical petrinet model has been used to model industrial system [81]. Three extensions are colour, time and hierarchy.

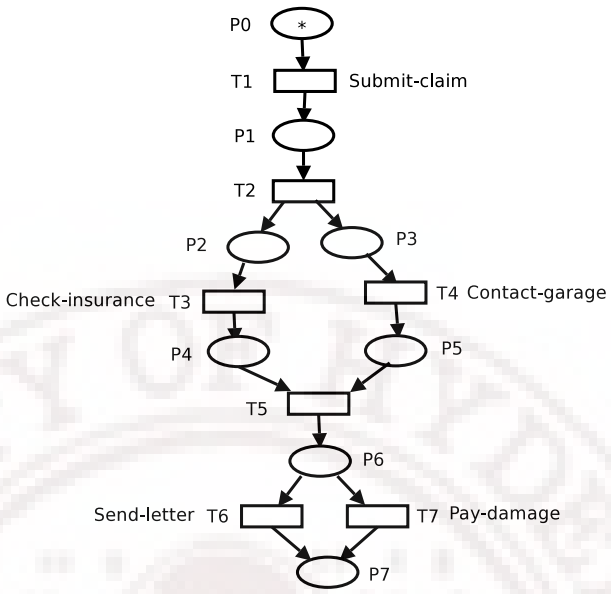


Figure 2.6: Workflow Modeling using Petrinets

Colored Petrinet

In classical petrinet model it is impossible to distinguish between two tokens in same place. For example in case of two insurance claims in order to incorporate the separate characteristics of the two claims in a model, it is necessary to include policy number, name of the policy holder etc. Classical petrinet is extended using color in order to enable coupling of characteristics with corresponding token. In a colored petrinet a transition can only be enabled if there is a token at each of its input points and pre-condition must be met for values of token to be consumed as shown in Figure 2.7. In contrast to classical petrinet, the result of color exten-

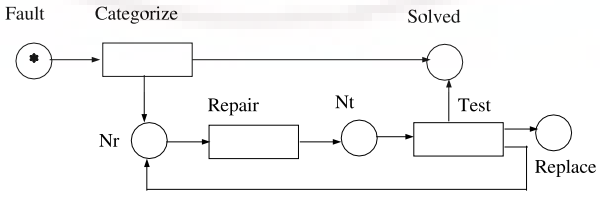


Figure 2.7: Colored Petrinet

sion is that graphical representation of colored petrinet can no longer contain all

information. For transition following factors must be ascertained.

- Whether there is a pre-condition and post-condition must be specified precisely.
- Number of tokens produced on each firing depend on value of tokens consumed.
- Value of tokens produced depend on value of tokens consumed.

Tokens often represent object (i,e resource, goods or human) are colored which facilitates the modeling for objects having attributes. A good modeling method is essential in order to simplify the workflow processes and to integrate with other application in case of critical business process. To model family of workflow processes WFCP-net (Workflow net based on coloured Petrinet) is a petrinet based analysis technique to verify the workflow process specification [86]. WF-net allows users to find control flow errors [46].

Timed Petrinet

Since classical petrinet is not capable of handling time. By adding time extension to classical petrinet it is able to model the temporal behavior of system (i,e to model duration and delay). Using time extension tokens can receive time stamp as well as a value. This indicates time from which token is available. Timing can be added to a petrinet by naming all the places and transitions, and drawing up a table with minimum and maximum times for each transition to occur based on the time of arrival of the tokens at its input place. If a transition fires and tokens are produced, each of these given a time stamp equal to or greater than the time of firing. Time stamp of produced token is equal to the time of firing plus the delay which is determined by firing transitions. For example as in Figure 2.8 with initial marking at time t=1,t_1 starts firing within a time less than t=2. If it does

not fire within that time it cannot fire any more because at time t=2, t_2 start firing having higher priority than t_1.

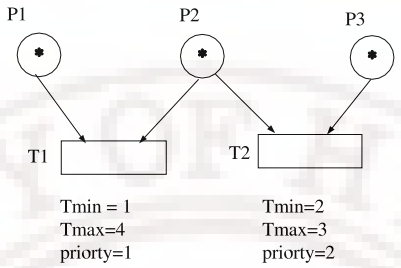


Figure 2.8: Time Petrinet

Hierarchical Petrinet

Using color and time extension though we are able to describe very complex processes, but they usually do not provide proper reflection of process. With 'Hierarchy' extension it is possible to add structure to petrinet model. Because a process can be constructed from sub-processes, which in turn also be constructed from sub-processes, therefore it is possible to structure a complex process hierarchically as shown in Figure 2.9. HiWorD (Hierarchical Workflow modeling) is a prototype with the simulation capability. It models the business process using petrinet in a hierarchical manner [9]. High level petrinets have been used in many application areas for example communication protocol, flexible manufacturing system, computer system, production system, administrative system, real time system, workflow system and distributed information system.

Reasons for Modeling in terms of High level Petrinet : There are few advantages for workflow modeling using high level petrinet. These are as follows.

- Reason for using petrinet based workflow management system is that it is easy to understand the graphical nature of petrinet and business logic can be represented formally [21].

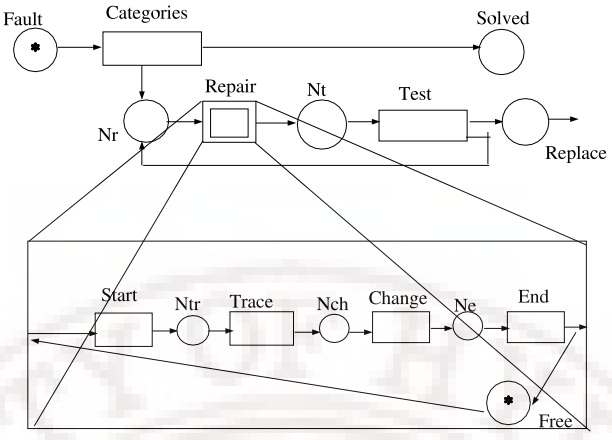


Figure 2.9: Hierarchical Petrinet

- It is state based instead of event based.
- Petrinet based modeling a system can be analyzed using petrinet theory .

2.4.4 Workflow Modeling with UML

UML supports several different models of a system through different diagrams. Some of the diagrams describes the static information about the objects of the system whereas other concentrate on dynamic aspects of how the system behaves when it runs [79]. UML activity diagram and statechart has been discussed in this section.

Workflow Modeling with Activity Diagram

Major constructs for workflow modeling are sequence, parallel path, alternate path and iteration. An activity diagram is a directed graph consists of nodes and directed edges. An activity diagram models the behavior of a system where node represents state and edge represents transition [23]. Constructs for creating activity diagram are

- Start: can be used for beginning of a process
- End: indicates the end of a process
- Fork: can be used for splitting a process into several parallel execution paths
- Decision activity: can be used for providing alternative paths
- Iteration: can be modeled by joining two decisions.

As in Figure 2.10 for example in a manufacturing company first an order is received by the action receive_order. The diamond shape represents a decision node and then the execution of fill_order depend on the condition of acception and rejection. The fork splits the path of control flow into two. Then in the left side action like produce_goods followed by ship_goods gets executed and simultaneously on the right side two actions like send_invoices followed by receive_payment are executed. After the receive_payment, the behavior pay() is invoked. When they both have completed their execution, join (next black bar) may takes place and after that, the action close_order takes place. Among the set of UML diagrams used for dy-

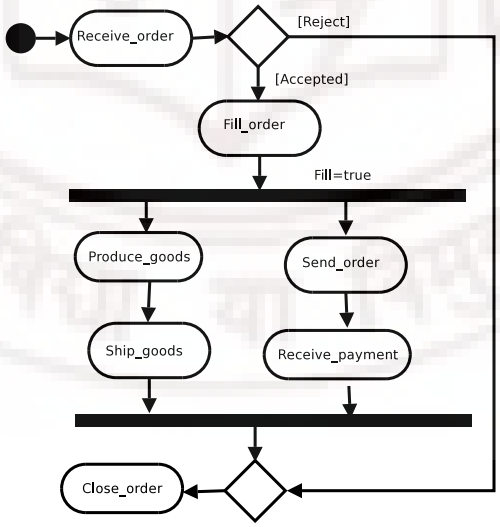


Figure 2.10: Workflow Modeling using Activity Diagram

namic aspects, activity diagram (AD) deals with Business Process (BP) but lacks

well defined semantics which has been taken care of by the finite state processes (FSP) discussed in [74]. Compared to EPC, in case of business process modeling there are some problems with activity diagrams. Not all logical operators for splitting and joining the control flows can be modeled in a straightforward way. UML does not consider activity diagram for object external flow. UML activity diagrams are to succeed as a standard for organizational process modeling. In the context of workflow specification the strong points of activity diagrams are that they support signal sending and receiving at the conceptual level and they support both waiting and receiving state. Activity diagrams provide a mechanism for decomposing the activity into sub-activities, but some of their constructs lack a precise syntax and semantics [24]. An extension to UML activity diagram called WAD (Workflow Activity Diagram) is used to describe the business process in production systems which applies the C-WF model concepts. The C-WF model represents the structural and functional enterprise objects involved in business processes [70]. In [53] a new approach has been discussed to do the analysis of workflow by modeling workflow using activity diagram and then converting the activity diagram to petrinet for analyzing formally.

Workflow Modeling with Statechart

Statecharts are graphic-oriented and their visual appeal leads to consider for representing reactive system. They are the extension of state-transition diagram. The basic elements of statecharts to represent a system are state, event, condition, action, transition, expression, variable and label. A business workflow can be formally represented by a statechart. An activity of a workflow whether it is automated or manual, is represented by a state in the statechart. An activity is being carried out only if the system is in the state that corresponds to that activity. Transitions between activities are thus modeled as transition between

states in the statechart which are triggered by events and guarded by conditions. Events can be external or internal. When the system is in the source state, event must occur and the condition must be true for the transition to take place [57]. States of a statechart are organized as a tree like structure, where root of the tree is called root state of the statechart, the children of a state are its immediate sub-states and the leaves of the tree are the basic states. In Figure 2.11

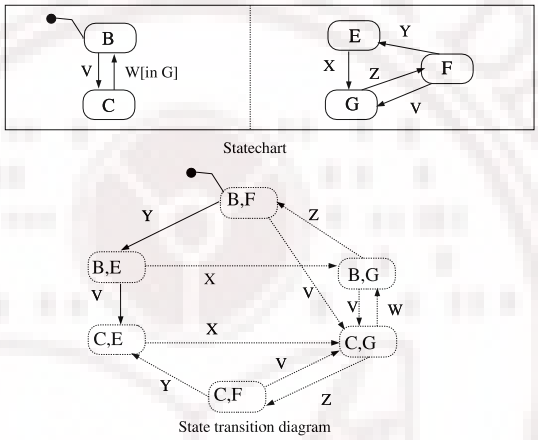


Figure 2.11: Workflow Modeling using Statechart and State transition diagram

a statechart and its equivalent state transition diagram are represented to show the problem of exponential growth of states in ordinary state transition diagram. We can easily model concurrency in a system by making use of orthogonal "and" states in a statechart. On the other hand to perform the same modeling in the equivalent state transition diagram we require six states [57]. Extend Statecharts creates a alternative modeling technique that deals with the concurrency, parallelism, simultaneous use of resources in a clear and better way [16]. To specify a real time system, time semantics is introduced into UML class and statechart diagrams [75]. Time semantic allows to verify the properties of real time system by means of timed computation tree logic. Statecharts is a language that extends the notation of FSM (Finite State Machine) with the concepts of concurrency, hierarchy and broadcast communication. FSM transitions are labeled by pairs,

where the first pair is referred to as trigger and consists of positive and negative signals (an event) and the second element is referred to as an action. FSM states are either basic states, OR-states and AND-states [49] . In Figure 2.12 shows a purchase order business process provided by a vendor using FSM [92]. The process starts with a purchase order (c#v#PO) message followed by a delivery (c#v#Delivery) message and either a credit card payment(c#v#ccPay) or an invoice payment (c#v#InvoicePay) message. In case the ordered product is not in stock, the vendor may reject a purchase order by sending a no stock available (c#v#noStock) message.

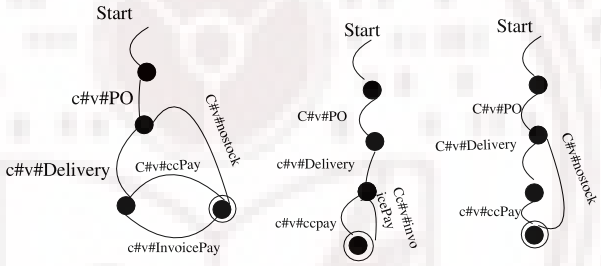


Figure 2.12: Example for FSM

Models ↓ Features →	Controlflow perspectives	Resource perspectives	Data perspectives	Time perspectives	Workflow primitives
Petrinet (Classical) [82] ,[53]	√	×	×	×	√
Petrinet High level[40] ,[46],[76]	√	√	√	√	√
UML Activity diagram[24],[70] ,[22] ,[26]	√	√	√		√
UML Usecase diagram[66]	×	√		×	×
UMLsequence diagram[66]	×	×	√		
UML Statechart diagram[66]	√	×	√	√	√
EPC diagram [20] ,[85]	√	×	×	×	√

Table 2.1: Workflow Modeling Techniques

2.4.5 A Comparison

As discussed in previous section several types of models viz. finite state machine, petrinet, event processing chain etc are used for workflow specification. Analysis has been done in Table.2.1 considering all workflow modeling technique with respect to different perspectives. Apart from analysis, here in this section we have done a comparative study among the modeling techniques. State based models are popular for its structure and simulation purpose, but different formal models are being investigated for specification. Though formalism is often used in pursuit of absolute correctness and safety, it can guarantee neither. Formal technologies do not offer practical solutions for wide spread use in industry. While in case of FSM, intertask communication for multiple FSM is difficult to depict. Although event process chain have become widespread process modeling technique, they suffer from a serious drawback i,e neither syntax nor the semantic of an event process chain are well defined [20]. Though FSM as well as Petrinet are very effective to specify dynamic behavior of a system, still their uses become difficult as well as cumbersome to specify a complex system composed of several FSMs or Petrinets. It has been observed that activity diagram is widely used to model workflow diagrams because of its natural capability to describe sequence of actions and also for its visual appealing.

2.5 Workflow Analysis

Workflow management systems facilitate the everyday operation of business processes. In contrast to traditional information systems, they attempt to support frequent changes of the workflows at hand. Correctness, efficiency and effectiveness of business process supported by workflow management system are vital for organization. Therefore it is necessary to analyze a workflow process definition

before it is put into production. Basically there are three types of analysis.

- validation : testing whether the workflow behaves as expected.
- verification : establishing the correctness of a workflow
- performance analysis : evaluating the ability to meet requirements with respect to throughput times, service level and resource utilization.

Workflow Verification

To support the automation of business processes, it should be specified using a language called workflow specification language. The result of a workflow specification language is called workflow specification which describes various aspects of workflow. Building workflow specifications is a complex and error-prone process, especially for large-scale workflows. It is likely to introduce inconsistencies or errors in workflow specifications. Such inconsistencies or errors may lead to incorrect execution of some or all workflow cases. Therefore, workflow specifications should be verified to ensure that they are correct to implement the corresponding business process objective [45]. The workflow verification aims at establishing the correctness of workflow specifications. Verification issues like structural constraints, temporal constraints, resource constraints and authorization constraints can be investigated [27]. Some specific questions to be addressed in the context of workflow correctness includes

- How can it be determined whether a step in a workflow be successful?
- What is the effect of interleaving of states from different workflows?
- When one or more steps in a workflow fail, what happens to that workflow and other workflows those have accessed data produced by the failed workflow?

- What happens to a workflow when one or more of WFMS components fails?

In the following sub-sections verification of different modeling techniques are discussed in detail.

Verification of Petrinet Models

Given a petrinet (P,T,F) and a state M, we have the following notations $M_1 \longrightarrow M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2. $M_1 \longrightarrow M_2$: there is a transition t such that $M_1 \ \underline{t} \ M_2$ $M_1 \longrightarrow M_n$ the firing sequence $= t_1\ t_2\ t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n. Set of intermediate states are $M_1, \ldots M_{n-1}$ i,e $M_1 \longrightarrow M_2 \longrightarrow M_3 \ldots \longrightarrow M_n$

- Reachability: In a Petrinet PN a state M_n is called reachable from M_1 ($M_1 \longrightarrow M_n$) iff there is a firing sequence σ with an initial state M. A state $M^{'}$ is a reachable state of (PN,M) iff $M \rightarrow M^{'}$
- Liveness: A Petrinet (PN,M) is live iff for every reachable state $M^{'}$ and every transition t there is a state $M^{"}$ reachable from $M^{'}$ which enables t.
- Boundedness: A Petrinet (PN,M) is bounded iff for each place P there is natural number n such that for every reachable state the number of tokens in P is less than n.
- Strongly connected: A Petrinet is strongly connected iff for every pair of nodes (places, transitions) x and y there is directed path from x to y.
- Free choice: A Petrinet is a free choice Petrinet iff, for every two transitions t_1 and t_2, $^* t_1 \cap {}^* t_2 \neq \phi$ implies $^* t_1 = {}^* t_2$ where *t denote set of input places for transition t.
- Soundness: Procedure modeled by WF-net (A Petrinet which models the control flow dimensions of a workflow) PN = (P,T,F) is sound if and only if

- for every state M reachable from i, there exists a firing sequence leading from state M to state o. Formally $\forall_M$ (i$\rightarrow$ M) $\Rightarrow$ (M $\rightarrow$ o)
- State o is the only state reachable from state i with at least one token in place o . Formally $\forall_M$ (i $\rightarrow$ M $\wedge$ M $\geq$ o) $\Rightarrow$ (M = o)
- There are no dead transitions in (PN, i)

A class of petrinets so called BP net suitable for representation, validation and verification for the construction and analysis of business process in case of workflow management is discussed in [82]. In this case correctness of these petrinet representation, for example soundness property is verified using polynomial time. Whereas to completely specify a business process allocating resources to tasks has not been discussed in this paper. Analysis method based on Petrinet is discussed in [84], Here petrinet is used to specify partial ordering of tasks (considering control-flow perspectives). It is possible in mapping task structure onto WF-nets which allows for verification of task structure using Petrinet based analysis technique [87]. Though workflow procedure (specifies set of tasks required to process cases successfully) is supported by WFMS, therefore to completely specify workflow process management of resources is also important [83]. Verification of process control aspects in conceptual workflow specification [2] and correctness verification of synchronization based workflow model [39] are presented. A spin model checker is used to check correctness of synchronization based workflow. Spin is popular open source software tool. Compared with current verification methods, semantic deduce based workflow verification method not only can detect deadlock and lack of synchronization, semantic errors, but also these are simple for computing [36]

Verification of UML Models

Certain properties can be discussed for the verification of UML whether certain states may or may not be reached. For instance will the terminal state be reached

at all circumstances can formally defined as

$$\forall_{m \in R_N(\overline{m})} : \underline{m} \in R_N(m)$$

where R_N(m) denotes the set of markings of N reachable from m, and $\overline{m}$ and $\underline{m}$ are the initial and final marking of N respectively.

- Absence of deadlocks may be verified by ensuring that

  $$\forall_{m \in R_N(m)} : \exists_{t \in T_N} : m\underline{t}$$

  where m$\underline{t}$ is the Petrinet notation to express that transition t is activated in m.

- If the order is filled whether the respective goods be shipped eventually may be expressed as $\forall_{m \in R_N(m)} : m\underline{fill\_order} \Longrightarrow \exists_{m' \in R_N}(m) : m'\underline{ship\_goodsorder}$

These properties are structurally rather similar, so that translating intuitive property into petrinet terminology and interpreting their results for the UML activity diagram is quite easy. Formal semantics of activity diagram which is based on colored Petrinet and covers control flow, concurrency and data flow [78]. By formalizing UML activity diagrams using Finite State Process(FSP), a given UML activity diagram specification can be analyzed by checking its equivalent FSP description using LTSA model checker [74]. LTSA can provide a compositional analysis of concurrent aspects of business process in order to check safety and liveness properties. C-Wf model which is an extension of CIMOSA can be used to model an enterprise considering the concepts required in a workflow model [71]. C-Wf reference model defines elements required to support a resource allocation process as well as basic requirement for monitoring a business process execution and considers both temporal and synchronism aspects involved in the allocation process.

Verification for EPC Models

Verification issues for event driven process chain are as follows.

- Regular: An event driven process chain is regular if and only if
  - EPC has two special events e_{start} and e_{final}. Event e_{start} is a source node :$^*e_{start} = \phi$ and e_{final} is a sink node : $e_{final}{}^* = \phi$
  - Every node n $\in$ N is on a path from e_{start} to e_{final}.
- Soundness property: A regular EPC is sound if and only if
  - process terminates properly for any case (i.e termination is guaranteed) for every state M reachable from initial state (i,e where event e_{start} is the only event that holds), there exists a firing sequence leading from state M to final state (i,e the state where the event e_{final} is the only event that holds).
  - there are no dead function (i,e for each function there is a firing sequence which executes f). There is no dangling references and deadlock and livelock are absent
  - completeness preserving: Non-Petrinet based approaches have been proposed for the verification of informal modeling techniques. Most of the modeling techniques are graphic based. Using graph-reduction to reduce the complexity of verification problem in such a way that correctness is not violated by the reduction. If the EPC is correct before reduction then the result after the reduction is also correct and if the EPC is not correct before reduction then the result after reduction is not correct [91] .

Although EPCs have become a wide spread modeling technique, they suffer form serious drawback i,e neither the syntax nor the semantics of EPCs are well defined.

The problem of EPC i,e formal semantics can be tackled by mapping EPCs to Petrinet. Petrinets have formal semantics and abundance analysis techniques. As a result the approach of mapping EPCs gives a formal semantics to EPCs. Correctness of EPCs can be checked in polynomial time by using petrinet based analysis technique [20]. Method for correctness verification of synchronization based workflow model has been discussed in [39]. Based on WF-net soundness and relaxed soundeness have been defined. There is no sound formal semantics for EPC which is fully compliant with the informal semantics [85].

2.6 Expected Features

Process oriented WFMS applications are currently reliable and secure if and only if business process is well structured and there is no need of adhoc deviation or dynamic extension at run time. Changes in a business process arise due to three reasons [42].

- Process improvement: performing the same business process with increased efficiency
- Process innovation: performing the business process in a different way.
- Process adaptation: adapting the process for unforeseen changes e.g handling a special case.

Reasons and requirements of workflow adaptations or ad-hoc derivation during runtime [95] [77].

1. Dynamic refinement : unavailability of a complete specification.
2. User involvement : decision making of user need to be considered.

3. Unpredictable events : due to some external factors, user intervention or timeout

4. Erroneous situations : system failure or erroneous operation.

Therefore changing a workflow definition is required due to some new requirements or exceptional situations [68]. There are two types of approaches emerged in this regard .

- Dynamic evolution: Dynamic evolution enables modification of workflow even during execution of workflow instances. Dynamic evolution copes with the problem of changing workflow definition due to new requirements or some exceptional situations. Changes can be evolutionary changes and ad-hoc changes. Evolutionary changes consists of structural modification that should be propagated to all new workflow instances as well as to the already existing ones. Ad-hoc changes can affect only a selected set of workflow instances. Changes may result from errors, rare situations or for a special requirement for a customer. Ad-hoc changes are typically supported by adaptive workflow system as a special case of evolutionary changes.

- Open point flexibility: Open point based flexibility is achieved by ensuring that there are a number of execution paths in the workflow definition and key decision point is well represented. It supports loose modeling of workflows, postponing complete specification to runtime.

Changes can be made either to a process or to an available resource or to a resource allocation mechanism. There are some policies which have been devised to manage workflow instance evolution [69] [42].

- Flush: Current workflow instances are allowed to complete according to the old process model and new instances can start following the new model.

- Migrate: Execution of workflow instances continues while the new changes are integrated into the process.
- Abort: All workflow instances of old schema are aborted.
- Adapt: The process must be altered for individual instances in order to accommodate some exceptional cases
- Build: The whole process can be rebuilt at runtime so that appropriate process model that correspond to the particular situations at hand can be created.

2.7 Healthcare Workflow

Healthcare is the prevention, treatment, management of illness and the preservation of mental and physical well-being through the services offered by medical, nursing, and allied health professions [37]. The organized provision of such services may constitute a healthcare system. Before the term "healthcare" became popular, english-speakers referred to medicine or to the health sector and spoke of the treatment and prevention of illness and disease. Patient visits various organizations or units within an organization to get proper diagnosis and treatment. Workflow management in healthcare technology is not an easy task. The role of healthcare workflow management by use of IT is to adjust the contributions of organizations or units in terms of timing, quality and functionality [43]. Healthcare is characterized by close collaboration and sharing of information among actors i,e doctors, nurses, technicians and patients, who cooperate for patient care at different times being at different places. Supporting distributed and cooperative work and sharing secure patient information is possible among healthcare personnel from different remotely located sites, He@lthcoop -a web based system enables

gathering, storing and accessing all patient clinical and personnel data anytime from anywhere [52] .

2.7.1 Features of Healthcare Workflow

In case of business process the objectives of modeling are to improve the efficiency of the organization, lower costs, reduce inventory, improve product delivery time, and to promote innovation. In healthcare industry we try not only to improve efficiency but also to improve safety of healthcare delivery. It is possible to achieve through the process of re-engineering workflow in healthcare industry. So can we literally translate workflow concepts used in business and manufacturing industry into healthcare industry? Healthcare is unique for the following reasons.

- In healthcare, a single unit is responsible for the total outcome of a patient. However during the care process number of requests are made to other areas of hospital to provide services such as laboratory tests, pathology tests, radiology tests, and specialist consultation etc.

- A common problem in healthcare is that a very few channel of communications are used for all information irrespective of urgency.

- One of the main reasons for modeling healthcare processes is to target points in the workflow for intervention.

- The clinical context of a process is the variable complexity of each patient. The variability between patients means that the same process may have different outcomes and there may be a need for various strategies to improve the health outcomes for each patient.

- Healthcare workflow that models a treatment activity can be modified at the level of the schema and it may also change during healthcare treatments

depending on unpredictable events, for instance new data on patients which cause modification of clinical process.

2.7.2 Models

To implement the features of healthcare workflow, modeling should have

1. Business orientation : Healthcare workflow is not only process oriented i,e how, but also resource oriented i,e by whom.

2. Concurrency : One or more processes can be executed simultaneously. For ex: CBP study, X-Ray and MRI scan can be executed concurrently.

3. Temporal behavior Timing is an important factor in healthcare process where a process has to be executed within a specific period of time in case of urgency.

4. Adaptive : We need a workflow model that supports evaluation as well as ad-hoc modification of process instances. Ad-hoc modifications allow users to adjust a particular process instance to specific circumstances so that this ad-hoc modification of process instance can be adapted..

5. Collaborative: The complexity of specialized knowledge and practices in the healthcare domain require active collaboration among different categories of healthcare providers such as nurses and medical specialists to facilitate coordination, information sharing and communication.

6. Scalability: In healthcare there may be many possible cases and corresponding therapies. Therefore modeling should be able to support high workloads.

7. Traceability: Refers to the ability to describe and follow the life of a conceptual or physical artifact on both forward and backward direction [94].

8. Activity crediting : In healthcare we faced with situations where more than one illness occurring simultaneously in a patient which require cooperation. Rather than patient repeat the same test for two different workflows, this can be done by merging two tasks into single task.

Modeling helps us to improve patient safety. Using product flow and information flow we can do the analysis regarding whether the needed parts arrive to the needed place at the prescribed time and how any change in the process affect the large system. Modeling of a HCU (Health Care Unit) contain four components [47] .

- Static (such as clinical data)
- Dynamic (temporal/transactional aspects such as activities in a hospital when a patient arrives)
- Operational (organizational aspect such as medical staff structure)
- Managerial (planning and control aspects such as resource allocation)

In case of workflow and information flow if there is delay in transferring data then there is definitely potential delay for the treatment of patient. The negative impact of both patient and system has been examined as a whole [44].Time management in workflow processes is crucial in determining and controlling the business activities. Integration of time management with workflow involves assignment of deadlines, synchronization of constraints, calculation of overall process duration and checking of timing inconsistencies. We need workflow representation language that is state based rather than event based to perform these studies.

Petrinet based Modeling

Petrinets became popular since they allow the graphic representation of computational structures and formal proofs of properties. Using a clinical healthcare

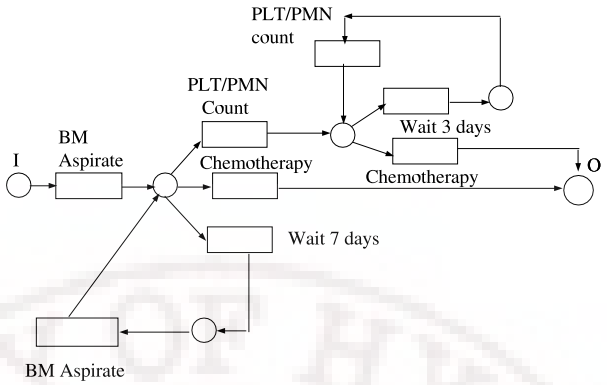


Figure 2.13: Healthcare in WF-net

process case study [76] illustrates the model using extend WF-nets called Time Wf-nets as shown in Figure 2.13. In this case sharing of resources at different times can be modeled. This model permits the modeling of shared resources. Each patient is competing for the limited resource(nurse) required for the task. Time constraint in this model ensures that patient 1 is to put in a request and this complete the procedure first before patient 2 does. Modeling is based on extended petrinet [18] where task corresponds to a transition and workflow state corresponds to a place as shown in Figure 2.14. Primitives are introduced to simplify visual representation in order to avoid more complex petrinet representation. There are two types of compound tasks: sub-workflow and aggregate has been discussed. Sub-workflow results in launching a new instance. Aggregate represents a group of tasks to form a higher level abstraction. Each place is represented by a circle. Each task is represented by a rectangle and sub-workflow is represented by a rectangle with double border. A set of operators for modifying an existing workflow schema has been discussed. In case of healthcare task ordering means sometimes a set of sequential tasks needs to be undertaken in a specific order. Other features encountered in healthcare workflow are the requirements to specify pre-requisite, co-requisite or post-requisite activity constraints and resource assignment also been discussed. For example the diagram illustrates how

to manage the blood pressure (BP) sub-workflow for patients with different BP readings. Petrinet representation have been chosen for clarity by not explicitly depicting intermediate states. In this case hypertension sub-workflow composed of number of sub-workflows, one of which progressive therapy is expanded and illustrated. Adaptability and sharing of resources has not been discussed in this paper. Operating room processes are modeled in terms petrinets so that soundness of a procedure can be checked using polynomial time algorithm [6]. This paper shows study of WF nets with shared resources (WFRS) can constitute a valuable source to improve sharing resource and to reduce the waiting period for patient.

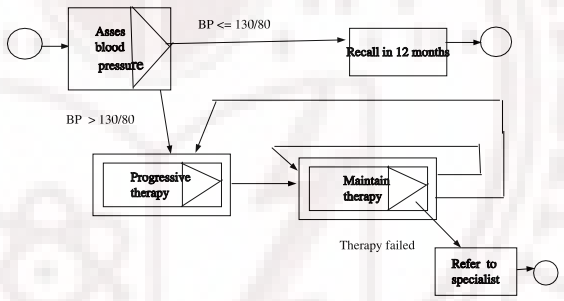


Figure 2.14: Petrinets based Healthcare Workflow

Formal Methods in Healthcare

Though very few in number, there has been some work emphasizing the formal approach for engineering of healthcare workflow. Some of them we will review here to project the current trend of research. A very recent work [13] presents a method for conducting cancer research. Here model driven architecture for cancer research information system shows that several types of users like patients, para-medical staff, administrative staff and doctors can interact with the system for seamless operations. As this system requires interactions among several participants and eventually becomes fairly complex, researchers propose to take the

help of formal methods for precise specification. They have used Z as formal specification language to formally specify system architecture. It is realized that many hidden issues were not only revealed but also defined precisely due to uses of formal methods.

Description logic has been used in [17] to specify healthcare workflow and particularly to verify liveness and safety properties. Authors propose off-line verification of specifications so that runtime verification of system does not require increasing number of rules to apply. Runtime verification is necessary to manage safety conditions in case of any unseen situations arise during treatment of patients. Description logic knowledge base consists of TBox - Terminology knowledge base comprises of unary and binary predicates; and ABox - Assertion identifying individuals in TBox. Description logic is used to state the status of workflow while it's under execution. Adaption to unforeseen situations during workflow execution is managed by a theorem prover by executing assertions that are triggered due to addition of new data to workflow. Authors also use temporal description logic to specify long lived workflow transactions and query on it. This is useful for healthcare as some treatment continues for a long period. Formal methods are used in managing medical protocol. While treating patients medical practitioners follow strict protocol as per medical guidelines. Adherence to medical guidelines is must for efficacy as well as safety of treatment. In case of serious and complicated ailments adherence to such protocol is cumbersome. Initially assuring correctness as well as completeness of such protocols are essential. Authors achieve it by integrating formal methods to lifecycle of guidelines. They have shown that model checking is useful to verify medical protocols. Thus, the results appeared in the paper confirms the trend of using formal methods in healthcare domain and has motivated us to carry out study on this aspect. However, there has not been enough research on formal methods for healthcare workflow. As

the requirements in healthcare automation increase, now there has been growing interest in application of formal methods in development of healthcare workflow systems.

UML diagrams based modeling

ATERUS model is used in healthcare structure to represent clinical and managerial activities [32]. This model uses hyper graph to describe the activities graphically and data of those activities is described by textual representation. In this case state diagram is used to control the activity. This conceptual representation become more complex while handling clinical complications which can be managed by exception handling. States which represent exceptions such as suspended, cancelled or aborted can be represented by state diagrams. In order to fulfill the requirement of healthcare workflow ATERUS model has the following features.

- Abstraction: Through a process of top down refinement this model is based on a description at different levels of abstraction of activities.
- Activity interaction: Concurrent, iteration, suspension or start etc are the primitives used in this model.
- Delegation: How to assign tasks to activities and what resources are necessary to execute activities.
- User orientation: User has a graphical representation of process.
- Partial definition: This model allows user to define part of the entire process in order to start the process execution.
- Monitoring of process execution: It helps in understanding whole process i,e what has happened and what may happen next.

In healthcare workflow management urgent request and critical messages has to be delivered timely [15]. Alert Management System (AMS) is proposed to handle this urgent constraint. Alert conceptual model that can capture both data and process integration represented using **UML** class diagram and workflow of the medical house call center is represented by **UML** Activity diagram.

The purpose of graphical representation of our model is that they are easier to communicate and therefore it is easy for people to understand in solving problem. Modeling the process and information flow between all of the stakeholders involved in the healthcare of a patient can be done using sequence diagram. From the workflow sequence model, it is possible to identify high-risk processes and therefore it can be most beneficial to reduce risk and improve patient safety. The workflow sequence diagram is derived from the Unified Modeling Language (UML),a set of diagrams and semantics used in software engineering. In complex system, sequence diagrams are one of the graphical representations developed for interaction of messages among entities. In sequence diagram entities in a process to be modeled are represented on the top of the diagram. For example in healthcare processes entities like unit, individual organization etc can be modeled. Constructs of Sequence Diagram for modeling are the following.

- Participating objects
- Depicting the sequencing of object operations

For the episode of care to be modeled, there are different active participants which can be represented using hollow rectangles. In workflow sequence diagrams solid arrows can be used to represent requests for information and dashed arrows for new information received. Gap between information request and information deliver can easily verify whether there is delay in information or information lost[48] .

EPCs based Modeling

Business process modeling approach has been done with selected results from the analysis of Patient Care Process(PCP) in neuroscience ward of public hospital [31].This analytic modeling technique is based on a software tool -the Architecture of Integrated Information System(ARIS). ARIS allows the presentation of PCP (in terms of object i,e data, information source, staff member with particular skill, etc) linked into Event Driven process Chain(EPCs). Using ARIS concept, every dimensions(called view) of PCP is built into a business process model. These includes

- data view (patient record, pathology request and oral report)
- organizational view (categories of staff in a hospital involved in PCP)
- activity view (tasks performed by staff)
- output view (results from the performed tasks)
- process view

ARIS concept allows linking a chain of activities to a particular organizational goal. In ARIS framework there are no restrictions in adding diversified list of personal group specific goal if it was observed that a individual or a group behavior may affect PCP. ARIS model involves alternative chain of activities to achieve the same objectives, for example informal connection with staff from different organizational unit to improve the efficiency of operation.

2.7.3 Verification

Business process is modeled in terms of petrinets [6]. Soundness verification of the underlying business process has been done with the help of a case study work-flow net with shared resources is used to analyze operating room processes and to

support re-engineering. E-commerce transaction executed by the WFMS, is represented using graphical model [41]. For calculating completion time and cost while executing a workflow, which is possible by providing modeling support for alternative paths in case of workflow. Scheduling algorithm is designed to meet user requirements on completion time and cost of execution. How to utilize workflow control structure augmented with information about timing behavior to calculate what delay to expect when execution of work item is postponed to certain dates. In general the issues to be verified in the healthcare model are as follows

- Soundness verification : means there is no deadlock or no unreachable tasks
- Correctness verification of synchronization based workflow
- Checking for both liveness and safety properties of workflow during runtime
- Temporal consistency checking
- Speeding up workflow instances by exploiting alternate paths
- Calculation of delay time when execution of a work item is postponed to certain date to execute a specific work item in case of emergency

A temporal interval as an execution duration is assigned to every workflow task. Though the real time taken by the task is unpredictable, so it may be between the specified bounds. Extending workflow nets (WF-nets) with time intervals and the new nets are known as Time WF-nets (TWF-nets). Properties of workflow modeled in TWF-nets can be verified in terms of safety, liveness and soundness using clinical heathcare process as a case study [76]. Presenting hospital cases as processes and then representing business processes has benn done using workflow mapping to petrinet [6]. Possibilities of performance improvement of whole system is presented by introducing the balance between specialization and generalization, centralization and decentralization of resource classes while preserving soundness.

In this case Patient Workflow Management System(PWFMS) is considered to illustrate the safety of TWF nets. Each task is broken down into sequential sub-tasks where each task require a resource and a specific job description. Resource manager controls the allocation of resources. Consider two parallel processes of two patients awaiting for the same procedure where there is one nurse (resource) available. Adding time constraint to provide time safety which is analogous to boundedness of petrinet in such a way that Patient 1 to put in a request and complete the procedure before Patient 2 does. Hence PWFMS permits sharing of resources. Certain behavioral properties of workflow processes modeled in TWF-net can also be verified. Based on this duration constraints, algorithm to calculate the longest/shortest process instance in a workflow graph were formulated. Using verification algorithm temporal requirement and inconsistencies can be checked. But deadlock, liveness and fairness has not been analyzed.

2.8 Summary

In this section we have reviewed some important aspects of workflow modeling techniques. For the sake of completeness we have reviewed some major works on workflow modeling. Then we have reviewed workflow models for healthcare domain. Broadly, the survey has touched upon two different kinds of models viz. structural and formal. Various structural models viz. petrinet, UML diagrams used for modeling several aspects of workflows both in business as well as healt-care domains are reviewed in detail. Similarly, formal models based on descriptive logic, temporal logic, Z etc. proposed for modeling workflows in both business as well as healthcare domains are also reviewed. It has been observed that though formal method is well known for rigor due its mathematical preciseness its uses have been limited because of its unfamiliarity in user community for its shyness to

mathematical rigor. Users from different domains are not usually trained in mathematics required to write formal specification and to prove model correctness that needs expertise in discrete mathematics. Hence there is quest for a method that can retain utility of formal method and at the same time will ease its complexity for users for wide spread use. This has resulted in light weight formal method that is gaining momentum among researchers as well as users. RAISE [30] is a formal specification language designed for industrial uses. It advocates rigor in specification but downplays rigor in correctness proving with the help of axiom based study on system behavior. Elsewhere, we have made use of this method in development of mobile computing applications and found it satisfactory [1].

In an early work in this direction [25] the term *light weight* methods indicate that the methods can be used to perform partial analysis on partial specifications, without a commitment to developing and base lining complete, consistent formal specifications. Based on this observation, here we propose to develop a light weight formal specification approach to model workflow for healthcare domain particularly for automating treatment process. Following this line of approach for formal specification, here we propose to use predicate logic for specifying healthcare workflow. An early work [67] proposes the uses of predicate logic for specifying software systems. It has shown that a variant of predicate logic is able to state partial functions and can be useful to specify software systems. Taking a clue from it, we have proposed a method to specify healthcare workflow using pre and post conditions defined on work entities. In addition, the proposed method intends to provide means for *flexibility* in modifying a workflow, *traceability*, *transparency* and managing *anamoly* in healthcare workflow executions and assigning *roles and responsibilities* for actors associated with healthcare. The next chapter presents a technique for baseline modeling of healthcare workflow.

Chapter 3

Modeling Treatflow Design Primitives and Assertions

1 2 3

This chapter identifies some of the workflow primitives used for healthcare workflow design. Primitives for structured workflow design are specified mathematically and observations are made by analyzing these specifications. In this chapter we present a verification algorithm which is used to study the correctness issues in a healthcare workflow design. Here we propose a modular approach to healthcare process specification. We have defined several operators to facilitate modular composition. Using these operators a medical practitioner can specify a treatment plan in an expression. We have also proposed rewriting rules to generate alternate equivalent treatment plans to facilitate decision making by doctors,patients as well as healthcare managers In this chapter we analyze the domain and present a comprehensive view on genesis of exceptions and corresponding actions

3.1 Introduction

Business objectives are met by successful executions of several business processes. A workflow is an ordered sequence of execution of business processes. It describes the exchange of process data, the control flow and the synchronization of process executions, the services provided by business processes as well as the system re-

[1]This work has been published in proceedings of sixth international conference on information and technology,CIT 2003, Bhuabaneshwar, Tata McGrawHill Pages 91-96, 2003.

[2]Primitives for structured workflow Design: A Mathematical Specification and Analysis published in proceeding of ninth international conference on information and technology, Bhubaneshwar, Proc. IEEE Computer Society, Pages 299-300, 2006.

[3]This work has been published in proceeding of the 11th international conference on information technology, Bhubaneswar, Proc.IEEE Computer Society, pages 215-220, 2008.

quirements. Workflow system is regarded as an important means of increasing the productivity of enterprises since it crosses organizational boundaries and facilitates the exchange of information among the users. Hence, workflow technology is crucial to automate business processes and their re-engineering. The workflow technology is applied to wide range of applications including office automation, healthcare, insurance, manufacturing and e-governance. In order to facilitate the use of this workflow technology, generic workflow management systems are being developed. Thus workflow management systems are software applications that support the design of workflows, their executions as well as monitoring.

A workflow designer follows different modeling approaches to model business processes using software tools. In general, the workflow modeling approaches fall into two categories: structured and formal. Structured approach follows a systematic way to compose business processes and the structured design is modular. Each module is represented by a graphic notation and modules are connected by edges. Further a module can be decomposed into submodules or atomic actions. In a workflow atomic actions are connected by directed edges. The direction of an edge shows the control flow during workflow executions. Thus this model provides a blue print of workflow system design. Such modeling approaches are popular among software engineers because of their visual appeals. A designer intuitively verifies workflow system design by going through design diagrams. In contrast to the structured approach, a formal approach insists on mathematical statements on business process behaviors and interactions. The use of a formal approach helps the workflow designers to detect inconsistency, reduce ambiguity in the development and reason about the system. Hence, the use of formal methods is being seriously promoted among the designers of workflow systems.

Models for healthcare processes could be complex like the processes observed in business domains. Modeling healthcare activities in one level could be

complex and cognitively loaded. Hence we propose a modular approach to healthcare process specification. We have defined several operators to facilitate modular composition. Using these operators a medical practitioner can specify a treatment plan in an expression. We have also proposed rewriting rules to generate alternate equivalent treatment plans to facilitate decision making by doctors,patients as well as healthcare managers. Though workflow technology is being used for automating enterprise activities but its uses in healthcare domain is at nascent stage. As healthcare workflow is directly deals with human life, therefore it must take precautions to undesired situations during its executions. In this chapter we analyze the domain and present a comprehensive view on genesis of exceptions and corresponding actions.

Following section discuss about Treatflow specification. The section 3.3 provides modeling of Treatflow activities. Section 3.4 presents Treatflow verification issues and with respect to that an algorithm is also presented. Modularization of treatment is discussed in section 3.5 and section 3.6 presents exception management in case of treatment. Finally we give a summary of the chapter.

3.2 Specifying a Treatflow

We proposed a model called *Treatflow* to specify a treatment by sequencing treatment activities as required to treat a patient. A doctor can make use of the modeling primitives to design a patient specific Treatflow. A *Treatflow* though similar to traditional workflow used in manufacturing domain still is augmented by special primitives viz. *supportive,cooperative* that are found useful to model activities in treating a patient. In addition, the traditionally known primitives viz. linear, parallel, split & merge, repeating, nested repeating and choice can also be used in modeling.

In this section specification of patient specific treatment activities in healthcare are discussed with the help of primitives. A hospital example is given below. In order to motivate readers we analyze Treatflow activities of a hospital and show that our primitives can be used to specify Treatflow.

Case history: *A 60 year obese single old man with history of smoking and strong family history of CHD (Coronary Heart Disease). The patient notices sudden chest pain in early hours of morning. He rushes to the emergency department of a hospital. Administration and preliminary clinical examination by G.P.(General Practitioner) reveals that the person may be suffering from either cardiac ailment or acid peptic disease. The patient complains of shortness of breath and mild sweating and chest pain radiating to abdomen.*

Situation-1: *Routine blood samples are sent to the department of pathology for investigations e.g. CBP(Complete Blood Picture), Lipid Profile and ECG(Electro Cardiogram). Resting ECG shows changes in some leads. Trop-T (Troponin-T is a cardiac enzyme marker which shows injury to cardiac muscle at an earliest stage) test at the bed side is positive. The patient is shifted to cardiac ICU (Intensive Care Unit). The doctor prescribes dispirin 300mg tablet, clopidrogel 300mg tablet and 1 sorbitrate 10mg tablet and the doctor sends for cardiac enzyme profile. He is put in oxygen inhalation and continous cardiac monitoring. The patient is subjected to pre-operative checkup for operation in OT(Operation Theater) Pre-anesthetic checkup, peri and post operative recovery is done by anesthetist. After operation (Coronary angiogram) the patient is shifted to post-operative ward and is prescribed drugs and the patient is discharged after observing for an week.*

Situation-2: *ECG is normal. Then the patient is investigated in the line of Acid Peptic Disease(APD). The emergency doctor finds out an episode of binge drinking in the previous night. Intravenous ranitidine and anti spasmodic medication is*

given. To counteract the side effects of disprin bolus dose of intravenous omeprazole and liquid sucralgel given. The patient improves and is observed for few days before discharging. When the symptoms are not abated, then upper-GI (Gastro Intestinal) endoscopy is done. If the upper-GI endoscopy is positive then prescribe drug for Duodenal Ulcer(DU). The patient improves and is advised a course of drugs for a period of six months.

The above scenario that takes place in a hospital can be specified in terms of Treatflow activities. These Treatflow activities can be specified in several ways namely linear, parallel, split & merge, repeating, nested repeating, cooperative and supportive etc. Further we present the analysis in structural form. We represent a Treatflow graph comprises of nodes and edges. In this case nodes are represented by rectangle. And split node and And merge node also represent with the help of a rectangle. Whereas in case of And split there is one incoming edge and more than one outgoing edges and in And merge there are more than one incoming edges and one outgoing edge. The same is true for Or split and Or merge, but the Or split and merge is represented with the help of a circle. But in case of linear it can have at most one incoming edge and at most one outgoing edge. The flow of execution is shown by directed edges.

3.3 Modeling Treatflow Activities

A graphical representation of a Treatflow in Figure.. 3.1 shows the activities with respect to the said case study. Though pictorial description of treatment activities is easy to understand still not useful to reason about. For example, an analyst may be interested in tracing the completeness of a treatment using a given Treatflow. Particularly, the automation of a reasoning process requires process specification of treatment activities. In order to facilitate the process in this section we have

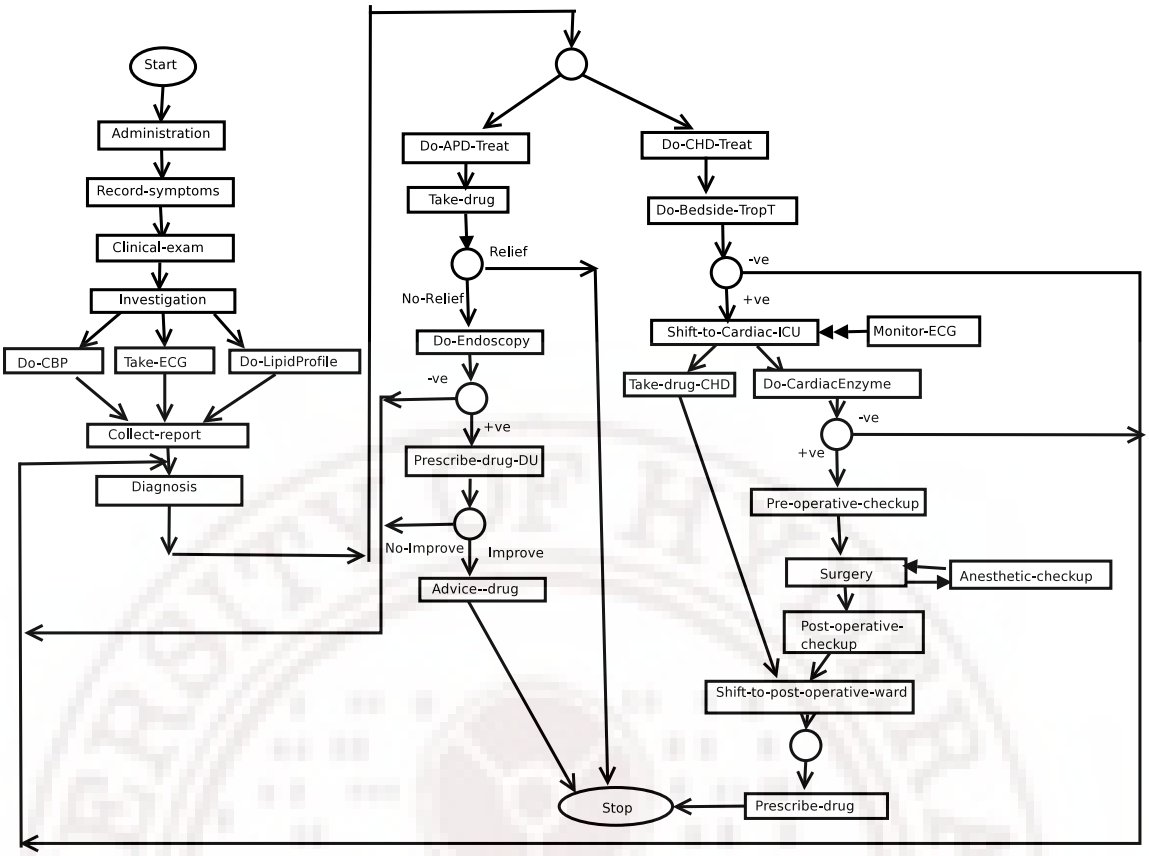


Figure 3.1: Treatflow Diagram

identified the types of activities and discuss on their concrete specification and behavioral correctness. Before discussing on individual category of primitive we present a generic syntax to describe a Treatflow activity. The syntax of a Treatflow activity specification is as follows:

Treatment activity :: <name>

Status :: {<active/waiting/start/end>}

Attributes ::

Resource-required :: {<name> <description>}

Operation-time :: {<time>}

Observation :: {<show(list-of-attributes)>}

Hiding :: {<hide(list-of-attributes)>}

Record-attribute :: { <name> <value> }

Prec :: {< predicates > }

Postc :: {< predicates > }

Safety-condition :: {<predicates> }

Sideeffect :: {<attribute,description>}

Patient-complaint :: {<attribute,description >}

Description :: {do <name> with <Prec> and <Postc> for <time> }

End

A unique *name* is given to each treatment activity. Once an activity is started, activity *status* gives a clear picture about activity either it is active or waiting before it has to come to an end state. An activity can assume different states e.g. active, waiting, start and end. Each attribute has a name and associated value(s). At different states during life cycle of an activity, its attributes can assume different values. In order to execute, an activity may need some resource which is represented by *resource-required*. Specific time period is allotted with the help of *operation-time* for the execution of an activity. Some treatment related data has to be recorded with the help of *record-attribute* while an activity is under execution.

During execution patient must be kept under *observation*. In order to start, an activity depends on satisfiability of Prec (pre-condition) and successful execution satisfies Postc (post-condition). Safety measure for each activity is taken into consideration with *safety-condition*. *Patient-complaint* has to be maintained for each activity in order to make change in the treatment. Each activity has to be executed with corresponding Prec and Postc for the specified period of time is described by *description*. Input and output attributes associated to an activity are also defined. Further we model a new idea on visibility of attributes to a patient. In a Treatflow activity, a patient is particularly directed to make note of some attributes specifying health parameters. And some are kept hidden for the

patient with the help of *hiding* attribute.

Later, it is shown that this specification of a Treatflow helps in analyzing and reasoning of an activity. For illustration we present an instantiation of an activity as follows.

Treatment-Activity :: <Take-drug>

Status :: {<active>}

Attribute ::

Resource-required :: {<drug> <Disprin 300mg,Clopidrogel 300mg,Sorbitrate 10mg>}

Operation-time :: {<24 hours>}

Observation :: {<chestpain,sweating,change-in-ECG>}

Hiding :: {<ECG-report>}

Record-attribute :: [<ECG> <reading>]

Prec :: {< is-chestpain(yes) > $\wedge$ < is-sweating(yes) > < change-ECG(yes) > }

Postc:: {< is-chestpain(no) > $\vee$ < is-sweating(no) > < change-ECG(no) > }

Safety-condition :: {<is-normal(BP)> }

Sideeffect :: {<headache,severe>}

Patient-complaint :: {<headache,severe>}

Description :: [do <Take-Drug> with <is-chestpain(yes) > $\wedge$ < is-sweating(yes)> and <is-chestpain(no) > $\vee$ < is-sweating(no) > for < 24 hours>]

End

In order to model Treatflow, we have identified several execution control primitive. In this section each of these primitive is discussed with the help of an example.

- **Linear Activities**

  A linear chain depicts a serial execution of activities with a start and a end

activity. For example Administration, Record-symptom and Clinical-exam can be performed in sequence, with Administration and Clinical-exam as the start and the end activities as shown in Figure.3.2. These two end points provides a boundary between which activities involved in a linear chain gets executed .If a set s with activities (p,q,r) form a linear chain LC then the elements of the set are ordered. Thus

LC(s) = p,q,r

Where p and r are start and end activities

$p \rightarrow q \rightarrow r$

that q is succeeded by p and preceded by r.

Without loss of generality two features are introduced viz: Prec and Postc for each activity stating pre-condition and post-condition respectively.

For an activity p, p.Prec and p.Postc indicate pre-condition and post-condition respectively. For example if p and q are in sequence i,e (p,q) then

p.Postc$\Longrightarrow q.Prec$

Execution of an activity depends on satisfiability of pre-condition and successful execution satisfies post-condition. As the activities are executed serially pre-condition of an activity is satisfied by post-condition of its previous activity.

$$[\text{p.Prec}]\text{p} \Longrightarrow p.Postc$$
$$[\text{p.Postc}] \Longrightarrow q.Prec$$
$$[\text{q.Prec}]\text{q} \Longrightarrow q.Postc$$

Similarly an activity q is said to be linear to its preceding activity p when

q.StartTime $\geq$ p.EndTime

p.EndTime $\geq$ p.StartTime

and pictorially shown as p → q

On execution of activity p, its status assumes Terminate i,e p.Status = Terminate and the activity q is started. Thus

$$\text{p.Status} = \text{Terminate} \Longrightarrow \text{q.Status} = \text{Active}$$

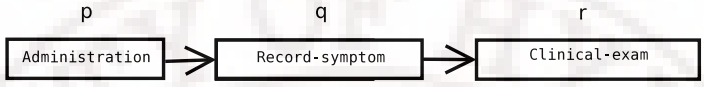


Figure 3.2: Linear Activities

The **dynamic behavior** of linear activities p → q can be verified by making use of their pre- and post-condition specifications i,e p → q execution is correct if

p.Postc ⟹ q.Prec

when p.Postc is the post-condition of the activity p and q.Prec is the pre-condition of the activity q.

- **Parallel Activities**

  In contrast to linear execution of activities, a set of activities can be carried out simultaneously for saving time or as required by a treatment procedure. A set of activities P $=\{p_i\}_{i=1}^{n}$ are said to be parallel then all of them are in active state in a given time t i,e

$$^{t}p_1 \parallel {}^{t}p_2 \ldots \parallel {}^{t}p_n \equiv \{t \in T \mid \forall i \in n \quad {}^{t}p_i.State = Active\}.$$

  Where T is the time period spanning total execution period of all the activities, $^{t}p_i.State$ indicates the state of activity p_i at time t .A set of activities can be in parallel for a time duration. In Figure.3.3 two activities Take-drug-CHD and Do-CardiacEnzyme are specified as parallel i,e a patient has to go for CHD treatment and at the same time different treatment

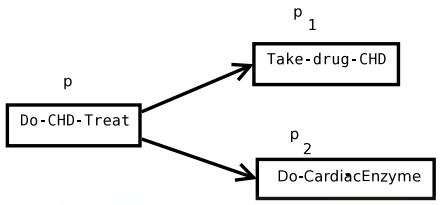


Figure 3.3: Parallel Activities

activities like taking drug for CHD (Take-drug-CHD) and require test (Do-CardiacEnzyme) can be carried out. An activity $p_i \in P$ obviously starts after the execution of its just preceding activity as discussed in case of linear activity. The interesting points in parallel treatment activities are conditional compatibility and non-race condition on resource requirements. The following rules are proposed for verification of **dynamic behavior** of parallel activities. If $p_i \parallel p_j$ i,e activities p_i and p_j are parallel treatments

$$p_i \nrightarrow p_j.Inv \wedge p_j \nrightarrow p_i.Inv \tag{3.1a}$$

and

$$p_i.Resource \cap p_j.Resource = \emptyset \tag{3.1b}$$

where $p_i \nrightarrow p_j.Inv$ means execution of p_i does not make $p_j.Inv$ untrue i,e non-interference of treatments.

$p_i.Inv$: health safety conditions associated to treatment p_i

$p_i.Resource$: Resource required for treatment p_i (Note: Excludes sharable resources and resources that can be made multiple copies)

- **Split & Merge Activities**

  Split and Merge are pseudo activities as these do not have tangible implementation. Rather these are the abstractions defined over a set of parallel activities. A set of parallel activities while originate concurrently we specify the point of origination as pseudo activities of sort 'Split' and common point

they terminate is specified as a pseudo activity of sort 'Merge'. Each Split or Merge activity can be meaningfully labeled but do not have specific details as we see in case of an activity investigation in Figure. 3.4 as because a pseudo activity do not have an implementation. For speedy execution an

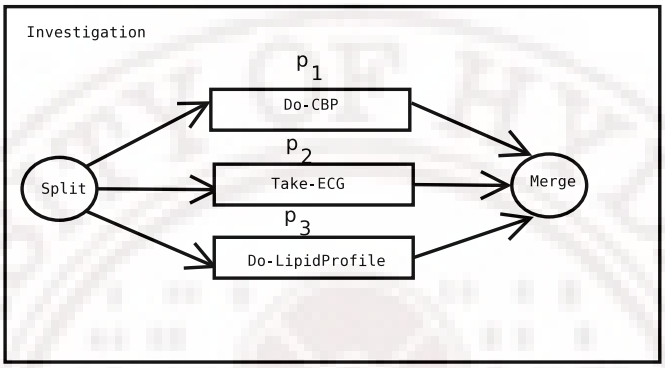


Figure 3.4: Split and Merge Activities

activity may split to several activities and they can run in parallel or in sequence and on their termination merge into an another activity. For example the process of Investigation is splitted into activities such as Do-CBP, Do-LipidProfile and Take-ECG and after execution again merged to the activity Collect-report where split and merge are the pseudo states. Let x,p,y are three activities being executed in sequence and activity p can split to p_1, p_2, p_3 activities that run in parallel. On execution of x,

$$x.Postc \Rightarrow p.Prec$$

Say p is split to three activities at time *ts* so that start times (stimes) of an activity is *ts* so as per definition

$$\text{split(p)} = p_1, p_2, p_3$$
$$\text{stime (split(p))} = \text{ts}$$
$$\Rightarrow \text{stime } (p_1) = \text{stime } (p_2) = \text{stime } (p_3)$$

The activities $p_1 \ldots p_3$ are executed on parallel and may end at different times say end time (i.e. etime). For generality, let's assume etimes are different. So

$$\text{etime}(p_1)=t_{e1}$$
$$\text{etime}(p_2)=t_{e2}$$
$$\text{etime}(p_3)=t_{e3}$$

On completion they merge (their outputs are merged) to provide the output of p and their cumulative post-conditions satisfy pre-condition of the succeeding process y. These definitions can be formally written as
A set of activities $\{p_i\}_{i=1}^{m}$ succeeding to an activity x, are said to be split activities when

$$^{t}p_1 \parallel {}^{t}p_2 \ldots \parallel {}^{t}p_m$$
and
$$x.EndTime \leq p_i.StartTime \; \forall i = 1 \ldots m$$

An example of split activities can be seen in Figure. 3.4. Investigation activity is splitted into three implementation activities viz Do-CBP, Do-LipidProfile and Take-ECG which are carried out in parallel. The following properties of parallel activities qualify a merge point.

1. $p_i.State$ = Terminate $\forall i = 1 \ldots m$
2. $\forall i, p_i$.EndTime $\leq$ y.StartTime

where y is the activity that succeeds to the merge point. For example, on termination three parallel activities Do-CBP, Do-LipidProfile and Take-ECG and then Collect-report is carried out. So,the StartTime for the activity

succeeding to parallel activities should be greater than equal to the latest EndTime all at which all the parallel activities must have been terminated. **Dynamic behavior** of activities at split point should not be such that their pre-conditions contradicts to each other i.e

$$\alpha \in p_i.Prec \Longrightarrow \neg\alpha \notin p_j.Prec, i \neq j \tag{3.2}$$

where α is a predicate and $p_i.Prec$ is set of predicates of p_i as its pre-conditions. After split, the parallel activities satisfy the properties discussed before. The dynamic behavior of activities at merge point should satisfy the following condition. Post-conditions of these merging activities should not be in conflict i.e.

$$\alpha \in p_i.Postc \Longrightarrow \neg\alpha \notin p_j.Postc \tag{3.3}$$

Rules 1-3 can be used to verify dynamic behavior of parallel treatment activities.

- **Repeating Activities**

  Alike looping in programming language, a set of activities may be executed repeatedly for some duration or until a given condition is satisfied. Repetitive execution of certain activities can also be found in healthcare workflow. For example, a patient may be advised to take antibiotics for three days or to undertake an exercise till the normality is achieved. An example can be seen in Figure. 3.5. This illustrates the need for modeling activities that are to be executed repetitively for specified time period (T) or until certain loop conditions (Lc) are satisfied. Formally a set of repetitive activities P = $\{p_i\}_{i=1}^{n}$ with p_1 the start activity and p_n the last should satisfy

$$p_1.\text{Prec} \wedge p_i.State = Active_{i=1}^{n} \wedge !(p_n.\text{Postc})$$

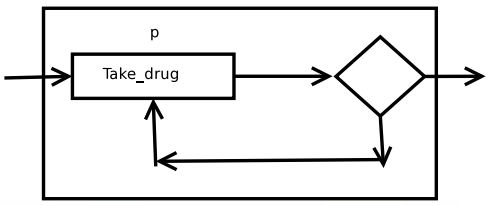


Figure 3.5: Repeating Activities

The execution of a loop is said to be correct when

$$\text{P.Postc} \Longrightarrow \text{P.Lc}$$

i,e The post-conditions due to set of activities P implies the satisfiability of loop condition of P i,e P.Lc . The termination condition of a set of repeating activities can be specified by p_n.Postc i.e post-condition of the last activity or by a time period T specifying that all the activities in P are to be executed T times.

The **dynamic behavior** of a set of repeating activities can be verified by the rules

- if $p_1.Prec$ true then $\forall p_i \in$ P, p_i.State=Active
- if $p_n.Postc$ true then $\forall p_n \in$ P, p_n.State=Terminate
- if No-of-Execution(P) = T then $\forall p_i \in$ P, p_i.State=Terminate

No-of-Execution() is a function that returns the number of times P is executed.

Treatment for a particular ailment in general follow a generic procedure. Still it has to be person specific, as a patient may respond to some abnormal behavior based on his/her personal constitutions. A treatment needs to respond to such events. In order to enable Treatflow specifying such requirements we have added three primitives viz Nested Repeating, Cooperative and Supportive.

- **Nested Repeating Activities**

  While a loop contains a set of activities, a nested loop contains a set of loops where loops are ordered from the inner-most loop to the outer-most loop. Conceptually the concept is similar to nested looping constructs found in a programming language. We have seen the utility of nested loop in Treatflow modeling as shown in Figure.3.6. For example, some people may

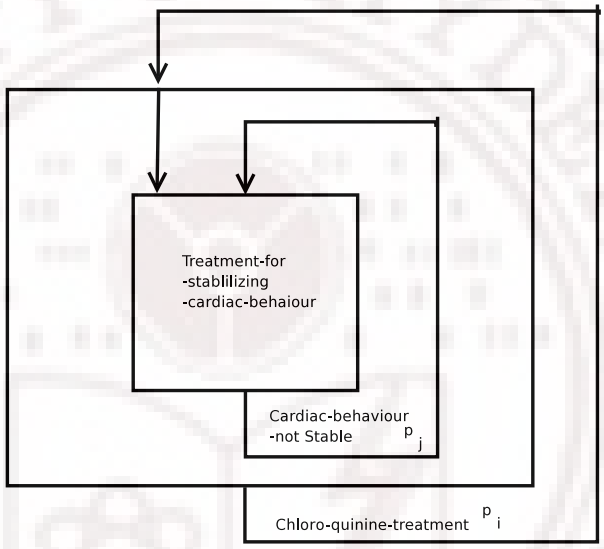


Figure 3.6: Nested Repeating Activities

develop cardiac behavior due to chloro-quinine treatment. In such case, chloro-quinine treatment is to be suspended for cardiac treatment. For a successful completion of the treatment, chloro-quinine treatment is to be resumed. Such a treatment pattern is modeled as nested repeating activities. Assume two activities p_i and p_j are nested and they form outer-loop and inner-loop respectively. The rules specifying such activities are as follows:

1. $\exists p_i \mid$ C-in(p_j,p_i)
2. $\exists$ e $\mid$ active-in(e,p_i) $\wedge$ triggers(e,p_j)
3. $\forall\ p_i\ \exists\ p_i$.Postc or p_i.T

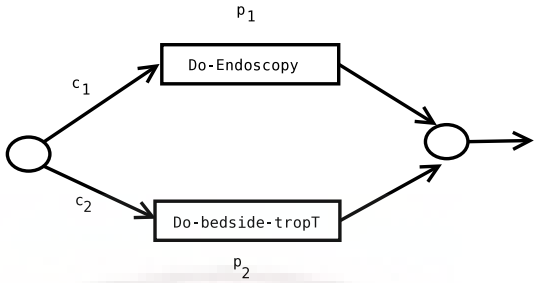


Figure 3.7: Choice Activities

The relation *C-in* specifies that the activity p_j is contained in the activity p_i. The first rule defines the outer and the inner loop. The second rule defines the event e on being active or appearing during execution of p_i may trigger execution of p_j.The third rule ensures termination of activities in nested loop. The **dynamic behavior** of nested repeating activities p_i and p_j when p_i contains p_j are :

1. p_j.State=Active $\Longrightarrow$ p_i.State=Suspend
2. p_j.State=Terminate $\Longrightarrow$ p_i.State=Active
3. p_i.StartTime $<$ p_j.StartTime and
4. p_i.EndTime $>$ p_j.EndTime

The above rule summarily describes that activity in outer-loop is suspended until the nested activity is terminated and then the execution of the outer loop is resumed. The temporal relations stated in the third rule discretized the execution time intervals for both the activities. Analysis of nested repeating activities exhibits mutually exclusion of the repeating activities.

- **Choice**

  As in Figure: 3.7 from the activity of Diagnosis either the activity of Do-Endoscopy or Do-bedside-Trop-T can be carried out depending on the choice.

Let p and q are the activities to be executed. An activity p can be decomposed into either p_1 or p_2 depending on the choice c_1 or c_2 respectively.

Before executing the activity p, the pre-condition of p must be true. The activity p is further refined to p_1 or p_2 under conditional choices. So for execution of p_1 and p_2 we need to extend pre-conditions as shown below.

$$[p.Prec \wedge c_1]p_1 \Rightarrow p_1.Postc$$

$$[p.Prec \wedge c_2]p_2 \Rightarrow p_2.Postc$$

$$p.Postc = p_1.Postc \cup p_2.Postc$$

$$p.Postc \Rightarrow q.Prec \text{ and [q.Prec] q} \Rightarrow \text{q.Postc}$$

The **dynamic behavior** for the choice activity is as follows.

As an activity succeeding to another activity with choice, it is possible to **trace** the choice path traversed during execution of the later process. Let q succeeds to p with choices p_1 and p_2 and the path through p_1 is traversed. Then according to definition we can infer the following.

$$\text{q.Prec} = \text{p.Postc}$$

$$\text{p.Postc} = p_1.\text{Postc}$$

$$\therefore \text{q.Prec} = p_1.\text{Postc}$$

Thus at q it can be decided that the process is reached traversing the path passing through p_1.

- **Cooperative & Supportive Activities**

  A set of parallel activities are cooperative / supportive when they interact among themselves during their execution. There are two-way interactions among cooperative activities while one way interaction exists among

supportive activities as shown in Figure. 3.8. For example, the activity Operation-OT (Operation Theater) consists of two parallel activities viz. Anaesthetic-checkup and surgery. During operation two-way interactions exist to increase / decrease level of anaesthesia depending on the requirements demanded by surgery. Also depending on status due to Anaesthetic-checkup, Surgery is guided. The rules that govern such activities are as follows. When p and q are cooperative then at time(s) during their execution one requests another for some information m and eventually the later gets a reply from the former. The rule also tells that activities can not terminate before the scheduled pairs of Request-Reply events take place.

$$(p, q, Cooperative) \Longrightarrow \{\exists t, t^{'} \mid Request(p, q, m, t) \wedge Reply(q, p, m, t^{'}) \wedge (t^{'} > t) \wedge (p, q, Parallel)\}$$

A set of parallel activities are said to be supportive when one way interaction exists during their execution. An activity is termed to be supportive when it facilitates functioning of another activity. The concept of support means enabling pre-conditions or mitigating undesired effects due to an activity. CHD

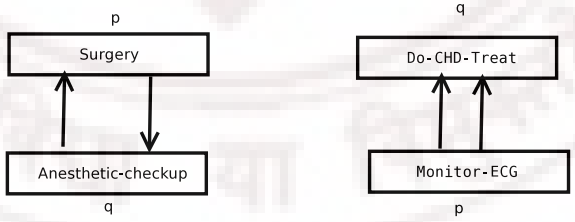


Figure 3.8: Cooperative and Supportive Activities

treatment is a good example of supportive activities. The CHD-treatment activity consists of activities like Do-CHD-Treat and Monitor-ECG . The information from Monitor-ECG supports Do-CHD-Treat. An activity p supports an activity q by passing information (Inform) m to q at different times

during their execution.

$$(p, q, Supportive) \Longrightarrow \{\exists t \mid Inform(p, q, m, t) \wedge (p, q, Parallel)\}$$

The **dynamic behavior** of supporting activities, say an activity q is supported by an activity p means

1. p.Output $\Longrightarrow$ q.Prec
2. p.Output $\Longrightarrow$ q.Invariant
3. p.Output $\Longrightarrow \neg$ q.Use

The specification of p and q activities should satisfy at least one of the above three rules to say that the activity p supports the activity q. The first rule says the output p implies pre-condition of q that is enabling of q. The second rule ensures the maintainability of invariants of q due to output of p. And the third rule tells the output of p makes the undesired side effects untenable. The above specification tells while two activities p and q are being executed concurrently and output due to execution of p satisfies invariants or pre-condition of q or disables q^c where q^c is an activity occurring in consequent to p.

3.4 Treatflow Verification

On composing a Treatflow it is required to verify it before making it operational as some mistakes might have crept in inadvertently while it being composed by a non-computer science professional. Mostly as we find in literature, workflow verification includes investigation on structural disorders [84]. They propose algorithms to detect errors prominently i,e lack of synchronization and deadlock. Whereas a graph-reduction based algorithm have been proposed [86]. As we have

already discussed , a graph G consists of nodes and edges. A Treatflow graph G has the following characteristics. A Treatflow graph has one start node and at least one stop node. For every And-split node there must be an And-merge node. Figure 3.9 presents a simple Treatflow graph for a patient with general fever either of kind viral or malaria fever.

A patient with fever on reaching a clinic goes through the registration- a necessity for hospital administration and then the patient is clinically examined. Based on this examination doctor intuitively decides a course of treatment either for viral fever or malaria. For viral fever the necessary pathological tests are done and then corresponding confirmatory treatment is initiated. Alternatively, if malaria treatment is decided then CBP (Complete Blood Picture) and MP (Malaria Parasite) tests are to be done and then only corresponding confirmatory treatment is to be initiated.

Because of the Or-node in the Treatflow in Figure. 3.9, the graph does have two different instances Figure. 3.10(a) and Figure.3.10(b) showing viral fever treatment and malaria fever treatment respectively.

In case of Treatflow verification we have expanded the scope of verification keeping the relevance of healthcare domain in view. The scope includes exploring two types of disorders. We will explain these disorders with the help of pictorial descriptions.

- **Structural Disorder**:
  - **Incompleteness**

    Treatflow is considered as an ordered graph G(V,E) with start and stop nodes. Stating initiation and termination of a treatment the incompleteness of a treatment exists only if the graph has a path starting from the start node and that does not reach the stop node. The presence

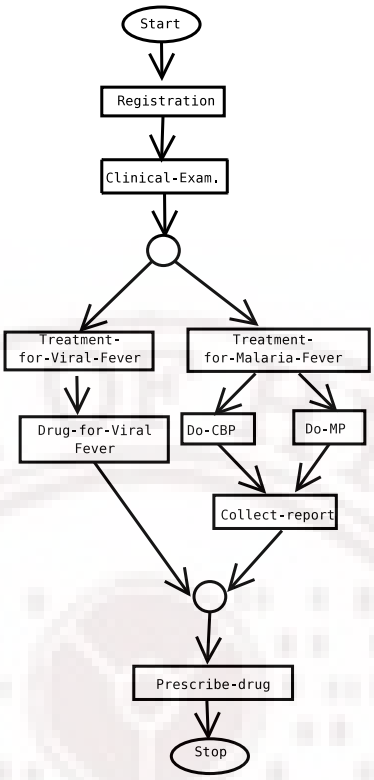
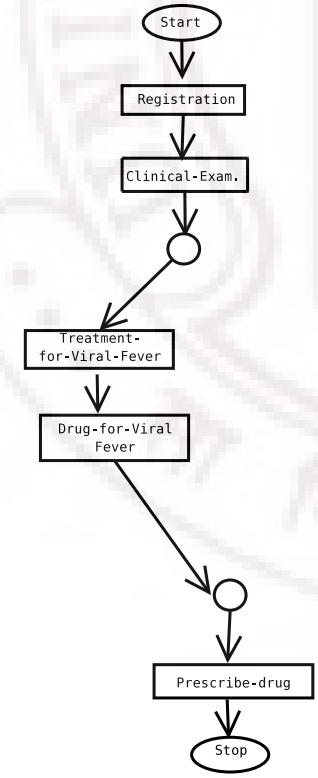
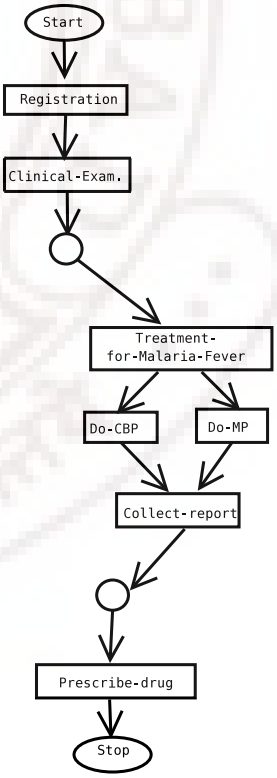


Figure 3.9: Treatflow Graph



(a) Treaflow Instance for Viral Fever



(b) Treaflow Instance for Malaria Fever

Figure 3.10: Treatflow Graph for Fever Treatment

of such an incomplete path in a Treatflow indicates that the treatment is incomplete and so not well defined as shown in Figure:3.11.

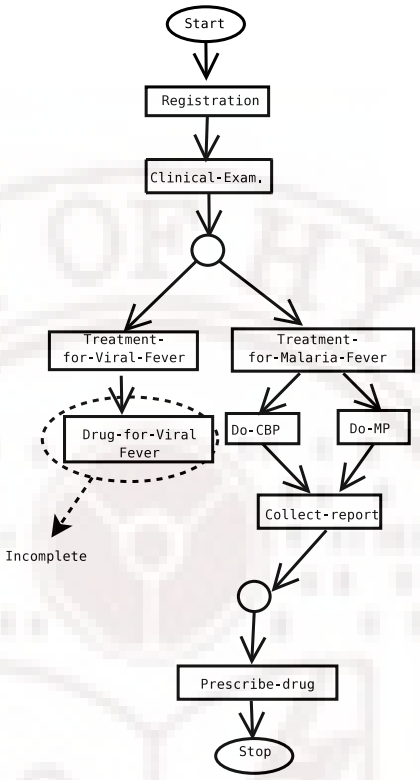


Figure 3.11: Incomplete Disorder in a Treatflow Graph

– **Lack of synchronization**

Lack of synchronization in a Treatflow occurs if an Or-merge node is traversed from more than one immediate parent node. Presence of such an error will cause repeat execution of a treatment leading from an Or-merge node. Of course such repetitions are not desirable and so a Treatflow must be corrected of such errors. In Figure. 3.12 a patient on preliminary diagnosis of malaria fever treatment is advised to carry out CBP and MP tests simultaneously. And then based on the test report prescribe-drug activity has to be started. Though concurrent activities i,e CBP and MP tests are executed simultaneously, but due to the absence of synchronization the next succeeding activity i,e prescribe-drug will be invoked twice which is undesirable. At the time of flow

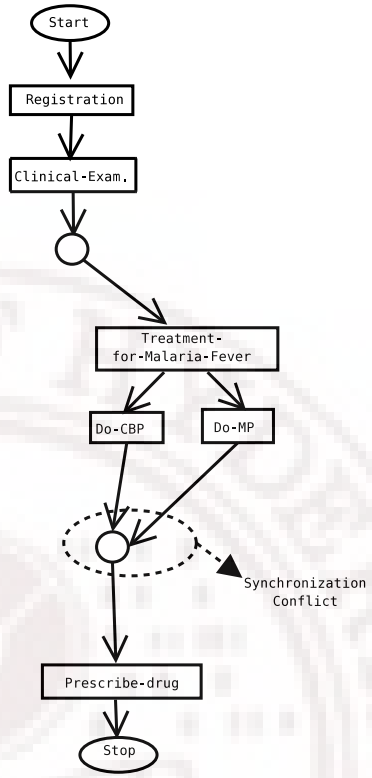


Figure 3.12: Synchronization Disorder in a Treatflow Graph

verification for this lapse must be identified.

– **Deadlock**

Deadlock in a Treatflow may occur when the progress of Treatflow execution depend on an event which is never happened before. Such a situation arises if an And-merge node is not traversed from all of its immediate parent nodes. In Figure. 3.13 based on clinical examination, doctor has to decide either to go for viral fever treatment or malaria fever treatment. In both the cases corresponding pathological tests are being done and based on test report, drug for the particular treatment is prescribed. In such case either of the treatment is followed by Prescribe-drug, but not both. But if the execution of Prescribe-drug activity is depend on both the malaria fever treatment and viral fever treatment which is never possible then in such cases error may occur in

a treatment. At the time of Treatflow verification, such type of error must be identified.

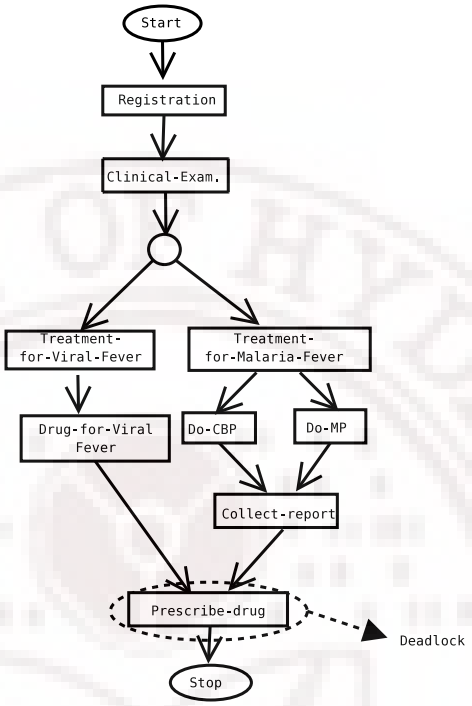


Figure 3.13: Deadlock Disorder in a Treatflow Graph

- **Behavioral**

  Other than structural disorder, a Treatflow may have behavioral and temporal incorrectness that need to be identified and corrected for safe and cost-effective treatment. We have classified these disordered into

  – **Retention conflict**

  Some of the health caring activities have retaining effects. For example once a tetanus injection is given, it has retaining effect for an year. In normal circumstances, say a diabetic patient is instructed to go for a blood test once in a month. So, for the patient such test has one month retention. Similar examples one could find for chemotherapy, x-ray or taking up some antibiotic drugs even. From this we are motivated to specify retention duration of an activity as a.rd that gives the duration

for which the effects due to the activity is valid i,e with the given period the activity need not be again executed to save time, money and even for the safety of a patient.

Let G' be an instance of a Treatflow graph G with activity 'a' represented by node v and the node has say two incoming edges. Suppose the activity has retaining duration T. If the activity has been invoked for the first time at time t then it should not be invoked again for execution before time t+T. For the second time say the node is in being visited at time t' and if $t' \leqslant t+T$ then the Treatflow had an error due to ignoring retention. Retention conflict in a Treatflow Graph G may occur if any of its instance graph has a node with more than one incoming edges. A healthcare activity like taking a drug or undergoing a pathological test has its retentive capability. For example a drug once taken should not be repeated within certain time period or a test once carried out on a body should not be repeated within a time period. This is so because the effects of an activity retains for certain period thus making repeat of such activity unnecessary. This aspect of a Treatflow, we name as Retention Conflict as shown in Figure:3.14.

– **Contextual conflict**

Two parallel or concurrent treatment activities are in contextual conflict if their respective pre-conditions are mutually exclusive. We consider the case as shown in Figure. 3.15. In case of treatment in healthcare different types of tests i,e Complete Blood Picture and Malaria Parasite can execute concurrently. All the pre-conditions for the execution of these test treatment activities are same. If Collect-report as an activity is included along with those tests, then that results to a contextual conflict because pre-condition of Collect-report activity and

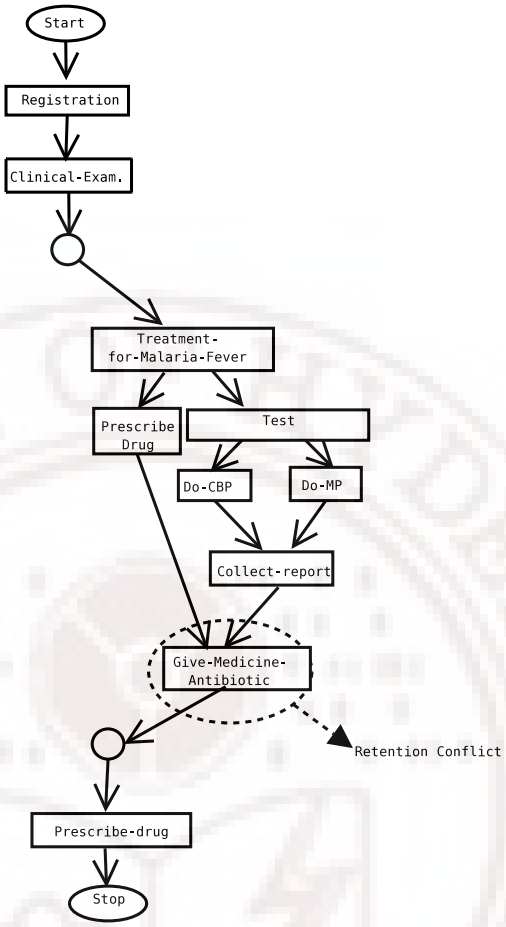


Figure 3.14: Retention Conflict in Treatflow

pre-condition of testing activities are mutually exclusive to each other.

– **Expectation conflict**

Two parallel treatment activities are in expectation conflict if their respective post-conditions are mutually exclusive. As an example types of tests namely Complete Blood Picture and Malaria Parasite can execute concurrently. All the post-conditions for the execution of test treatment activities must be same that means report should be ready after execution of test treatment activities. Once all the test report has been done, then collect-report activity can be started as a succeeding activity. But in case Prescribe-drug is considered as one among the

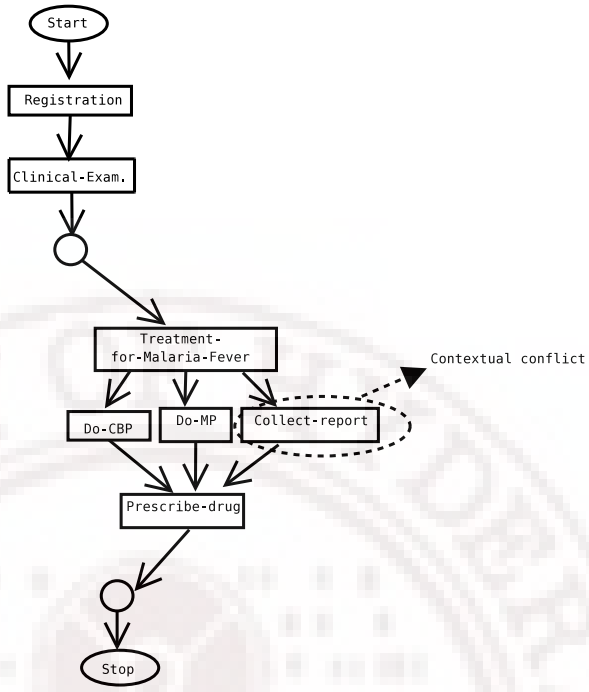


Figure 3.15: Contextual conflict in Treatflow

concurrent activities as shown in Figure. 3.16, then post-conditions of all these concurrent activities are not same which may result into an expectation conflict.

3.4.1 Verification algorithm

In order to verify a complete Treatflow graph, all the possible instances of Treatflow graph should be verified. After creating each instance verification should be done. The verification algorithm proposed in this section uses depth-first search and AO* algorithm in order to process Treatflow graph using AND-OR [86]. Definitions used in verification algorithm are as follows.

G: Original Treatflow graph

G': Part of graph G is traversed so far while executing the algorithm

visit_stack(vs) and or_split_stack(oss) and cycle_origin_stack(cos) Create_Instance starts with the start node "S" in the Treatflow graph G'. While traversing all the child nodes of the non-visited nodes are linked to the graph along with their edges

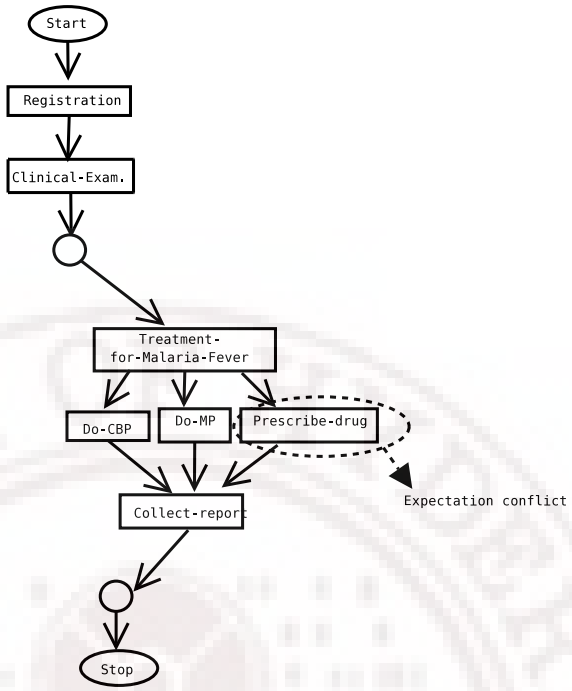


Figure 3.16: Expectation conflict in Treatflow

and also marked those nodes as visited nodes. In each iteration of the algorithm, an instance of the Treatflow graph is created with the help of Create_Instance. After creating an instance, flow of Treatflow graph can be verified using Verify_Instance. Finally next instance can be created with the help of Init_forNextInstance. According to algorithm procedure Create_Instance creates an instance graph G' from G with a start node S. If child node(s) of S is not visited in graph G' , then child nodes must be marked and linked to the graph. If the child node is an Or-split node then the leftmost unmarked chid must be first marked and linked to the graph. Create_Instance procedure uses depth first search mechanism for traversal. Otherwise in all other cases all the child nodes are traversed, marked and linked to the graph G'.

Procedure Verify_Instance verifies the instance graph starting from the start node S. While traversing any of the non visited node are taken into consideration. In case of non-visited And-split node, pre-conditions of all the child nodes are to be stored.. If pre-conditions are mutually exclusive, then the procedure

reports *Contextual Conflict*. Similarly for the case of an And-merge node, post-conditions of all child nodes are to be stored. If any one of the post-conditions are mutually exclusive to each other, then the procedure reports *Expectation Conflict*.

If the child node is already visited in case of an Or-merge node, then the procedure Verify_Instance reports *Lack of Synchronization* for that child node. In case of And-merge node if number of visits does not match with the number of And-join edges, then the procedure reports *Deadlock* in that child node.

In any iteration for the case of And-merge node, the retention duration for that node is stored as T and time for visiting that node must be recorded as t_i. In case that And-merge node is already visited once and time for next visit for that node is considered as t_{i+1}. If $t_{i+1} < t_i + T$ then the procedure reports *Retention Conflict* at that node.

After completing the iteration if the last child node does not exist in the set of terminating nodes TN, then the procedure reports *Incompleteness* for that particular instance of the graph. If the instance graph is correct then it calls for procedure Init_forNextInstance to create another instance. Init_forNextInstance chooses the leftmost deepest Or-split node created by Create_Instance. In this case one child has been taken into consideration. For the next instance, marking are moved to the next child of the Or-split node for the next iteration. Thus the proposed algorithm verifies the complete Treatflow graph by just verifying a subset of the instance subgraph.

Analysis of Algorithm

While expanding an instance graph, each node in the graph is considered and possible outgoing edges are linked at each step of the algorithm. The algorithm repeats at algorithm 2 line number 1 to 19 till all the nodes are considered. Thus in the process the algorithm always traversed the number of edges i,e less than the total number of edges present in a Treatflow graph. Thus complexity of algorithm

Algorithm 1 Verify_Treatflow(G)

```
1: initialize visit_stack(vs)
2: initialize or_split_stack(oss)
3: initialize TN a set of terminating nodes
4: Push_aNode(vs,s)
5: Install_InstanceAt(G,G')
6: repeat
7:     Create_Instance(G,G',vs,oss)
8:     Verify_Instance(G')
9:     Init_for_Next_Instance(G,G',vs,oss)
10: until empty_or_split_stack(oss)
```

Algorithm 2 Create_Instance(G,G',vs,oss)

```
1: while Not_empty(vs) do
2:     cn=pop_aNode(vs)
3:     if Not_Linked(cn,G') then
4:         Link_toGraph(cn,G')
5:         Mark(cn,G')
6:         if OrSplit(cn) then
7:             lcn=GetLeftmostUnmarkedNode(cn)
8:             Link_toGraph(lcn,G')
9:             Push_aNode(lcn,vs)
10:            Push_aNode(cn,oss)
11:        else
12:            if Not_Marked(cn,G') then
13:                Push_allchild(cn,G')
14:                Link_toGraph(cn,G')
15:                Push_allchildnode(cn,vs)
16:            end if
17:        end if
18:    end if
19: end while
```

Algorithm 3 Verify_Instance(G')

```
1: Initialize_InstanceVisitStack(ivs)
2: Initialize_pre-conditionStack(precs)
3: Initialize_post-conditionStack(postcs)
4: Initialize visitcount = 0 ∀ And Merge in G'
5: Initialize i=1
6: Retention_duration(rd)
7: initialize t_i for time_visit(cn)
8: initialize TN for set of terminating nodes
9: Label_Notvisit(G')
10: Push_aNode(s,ivs)
11: while Not_empty(ivs) do
12:     cn = Pop_aNode(ivs)
13:     if Not_Visited (cn) then
14:         LabelVisit(cn)
15:         Push_allChild(cn,ivs)
16:         if visited(cn) ∧ OrMerge(cn) then
17:             SynchMiss_at(cn)        ▷ Lack of Synchronization at child node cn
18:             if Not_visited(cn) ∧ AndMerge(cn) then
19:                 T=rd(cn) ∧t_i = t(cn)
20:                 increment i
21:             end if
22:             if visited_AndMerge(cn) ∧t_i = t(cn) then
23:                 if (t_i < t_{i-1} +T) then
24:                     Retention_Conflict_at(cn)          ▷ Retention Conflict
25:
26:                 end if
27:             end if
```

```
28:             if AndMerge(cn) then
29:                 IncrementVisitCount(cn)
30:                 Labelvisit(cn)
31:                 if AndSplit(cn) then
32:                     for all_childnodes do
33:                         Push_prec(childnode(cn),precs)
34:                     end for
35:                     if Not_equal_prec(childnode(cn),precs) then
36:                         Contextual_Conflict_at(cn)   ▷ Contextual Conflict at cn
37:                     end if
38:                 end if
39:             end if
40:         end if
41:     end if
42:     if AndMerge(cn) then
43:         for all_incomingchildnodes do
44:             Push_postc(incoming_childnode(cn),postcs)
45:         end for
46:         if Not_equal_postc(childnode(cn),postcs) then
47:             Expectation_Conflict_at(cn)          ▷ Expectation Conflict at cn
48:         end if
49:     end if
50: end while
51: if visited_AndMerge(cn) ∧ (VisitCount < CountAndMerge(cn)) then
52:     Deadlock_at(cn)                              ▷ Deadlock at child node cn
53: end if
54: if cn ∉ TN then
55:     Incomplete(G')                               ▷ Incompleteness in path
56: end if
```

2 Create_Instance remains O(E).

In case of algorithm 3 Verify_Instance there is a loop in between line number 11 and line number 50. In this while block, all varieties of verifications are performed while visiting each node of an instance graph. Because of the line number 15, a node may be visited more than once through incoming edges, thus in the algorithm each edge of an instance graph is visited. The number of edges of an instance graph is always less than equal to a Treatflow graph. Hence, the complexity due to Verify_Instance also remains O(E).

The algorithm 4 Init_forNextInstance only performs certain initializa-

Algorithm 4 Init_forNextInstance(G,G',vs,oss)

```
1: cn = pop_aNode(oss)
2: if NotEmpty(oss) then
3:     lcn = GetLeftmostUnmarkedNode(cn)
4:     Link(lcn,cn,G')
5:     Push_aNode(lcn,vs)
6: end if
```

tions of variables like *cn* and *lcn* for creating next instance. Thus the complexity of the algorithm is negligibly a constant unit.

Algorithm Verify_Treatflow has a loop in between line number 6 to 10 that contains a repeat loop executing the algorithms 2, 3 and 4. Based on the analysis made for these three algorithms the complexity for executing the repeat loop for single instance remains O(E). The number of times the repeat loop executed depends on the number of Or-split nodes and their branch factors. These branch factors at Or-split nodes exhibit a multiplying factor while creating instance graph. For example two Or-split nodes with branch factors say b_1 and b_2 will generate $b_1 \star b_2$ instance graphs. In practice in a Treatflow graph, a number of Or nodes would be very much less than N, the number of nodes in the graph and the average branch factor is usually a small number. These two factors contribute to worst case exponential complexity $o(E^{n^2})$. However the contribution due to these factors is much less than n^2. If there are o number of Or-split nodes with average branching b, then the contributing factor is o $\star$ b which is a constant say c. So, in practice the complexity of this verification algorithm is $O(E^c)$ where c $<< n^2$.

3.5 Modularization of Treatment

Healthcare processes are complex because they involve many participants dealing with complex information [48]. A large system needs to be divided into blocks so

that the system becomes manageable, clear, modifiable and reusable. These blocks known as modules are connected among themselves in such a way that module correctness can be verified from the aspects of horizontal and vertical compositions [50]. In order to manage a complex system, proposed an operational model viz. MCPN: Modular Colour PetriNet is proposed which is composed of colored petri net modules [72]. Though it is not suitable to model at one level only in order to represent a large and complex service process, therefore [56] propose to model web services using FSM modules. Refinement and modules of graph transformation system is introduced in [51]. Here syntax of refinement is given by rule expression which is based on rule names and rule composition operation symbol.

Now in practice, we observe that a complex health problem treatment is modularized not only for administering the treatment but also to monitor and to modify if necessary. This is also useful for administrative uses especially for estimating cost and arranging resources. Health services are also being offered as service packages. These trends motivate to plan a treatment in modular form. In this work we focus on problems in specifying treatment modules and composition of these modules to form a treatment plan termed as *Treatplan*. For composition purpose we have defined several operators e.g. linear, parallel, choice, supportive and cooperative and have shown that a prescription can be written as an expression of modules. We also show that using operator properties, a treatment expression can be rewritten to another treatment expression. Following the proposed method, a tool can be developed that can assist a medico in prescription writing. The safeguards specified in modules can alert medico, nursing staff as well as patients on prescription and administration of wrong medications. These factors emphasize the necessity of modularization of treatment process in healthcare domain.

3.5.1 Motivating example

Our previous work in last section defines primitives and proposes a method called *Treatflow* to model treatments as a flow of activities as found in workflows. It is observed that modeling a complex treatment i.e with multiple serious complaints could be complicated due to the large number of treatment activities and various order of executions. Hence we propose modularization of treatment plan. In order to motivate readers on modularization concept we would like to take the help of a case study shown below.

A male patient aged 40 yrs complains of increase in appetite, volume of urine and thirst. He also suffers from chest pain. After registration and assignment,, the doctor has recorded those symptoms as hyperphagia, polyurea and polydipsia.After clinical examination and assessment of chest pain, the doctor prioritize the presumptive diagnosis of Coronary Heart Disease (CHD) or Acid Peptic Disease(APD).The doctor has made a probable diagnosis of Diabetes Mellitus (DM).Therefore simultaneously both in the line of DM as well as chest pain investigation started. Controlling blood sugar, Diet & exercise are the mainstay of treatment for DM. For chest pain the doctor recommends ECG,UGI endoscopy which helps in making a definite diagnosis of chest pain in CHD the principle of ABC (Airway,Breathing,Circulation) with drug therapy is followed.

These activities are to be arranged in a schematic way using the modeling primitives. Such a schema shows the way a treatment can be carried out, thus said a Treatflow. Different aspects like composing Treatflow activities and verifying a Treatflow are discussed in last section. Here, we would like to introduce the concept of modularization of treatment. For this case, the patient has to undertake a number of treatment activities for treatment like CHD(Coronary Heart Disease),

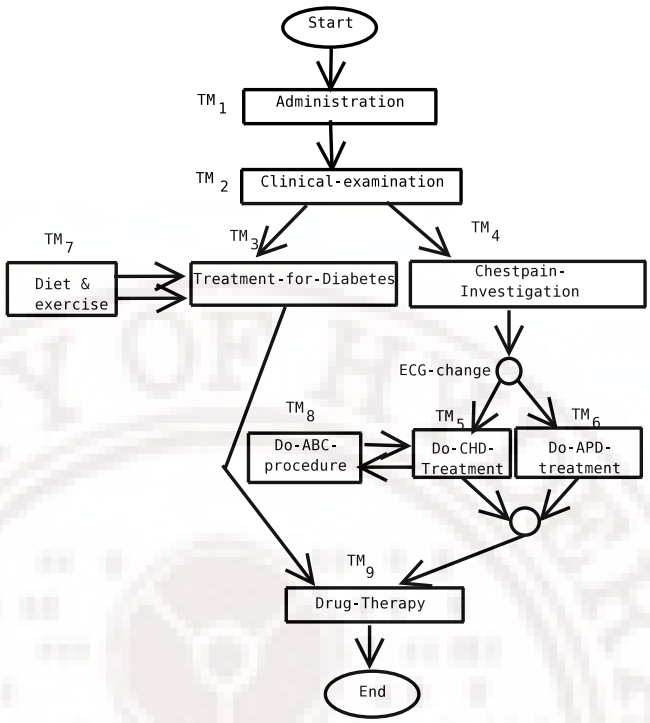


Figure 3.17: Treatflow Modules

APD(Acid Peptic Disease and Diabetes. Modeling Treatflow for all these treatments becomes complex to design, verify and follow. This necessitates modular design of healthcare activities. From the stated case study, modules are identified and treatment plan is presented in graph form as shown in Figure. 3.17. In next section we will specify the modules and operators for specifying a treatment plan in expression form.

3.5.2 Module Specification:

In order to motivate the reader with the help of hospital example given above, we discuss treatment module specification in case of treatment in a hospital. A treatment is a collection of treatment modules which can be represented as

Treatment $= \{TM_i\}_{i=1}^{n}$

whereas a Treatment Module is a set of Treatment Activities. That can be represented as

Treatment Module = $\{TA_i\}_{i=1}^{n}$

A specification of a treatment module needs to provide comprehensive view on a treatment module so that the specification will be useful for automation of Treatflow activities and also enabling a practitioner and a patient to have a quick view on a treatment. Ideally, a treatment module needs to specify conditions at which it can be applied to a patient. In general, conditions specify health status of a patient that can be specified by health parameters and their values. This we say *enabling-condition* of a module. Similarly, a module should also project *expected-condition* i.e. health conditions that would result on successful treatment of the module. A treatment may have some limitations at which it is not applicable. These warnings should be recorded in module specification as *limitation*. Similarly advises are also to be specified to meet the exigencies that may happen during a Treatment Module execution. These are specified as *caution* in module specification. A module in *treatflow-details* refers to the name of a *Treatflow* that records the sequence of treatment activities that are to be executed during execution of the module. In *treatment-duration*, it records the days for which a module should be executed or the treatment must take place. In case a treatment requires execution of another treatment modules(s) the same should be specified in *composed-of*, it enlists the names of the modules that are included in a parent module. On instantiation of a module for a patient, we need also to record information on observed health conditions and the period for which a module instantiation has been executed in *observed-conditions* and *treated-duration* respectively. The syntax of a treatment module specification is as follows:

Treatment-Module :: <name>

enabling-condition :: {<predicate>}

expected-condition :: {<predicate>}

composed-of :: [<treatment module >]

treatment-details :: [<treatflow >]

limitation :: {< warning-message > }

caution :: {<advises > }

treatment-duration :: {<days> }

observed-condition :: {< parameter > <value>}

treated-duration :: {<days> }

End Module_name

Based on the example cited in Figure.3.17 we specify a module as follows:

Treatment-Module :: <Diabetes-treatment>

enabling-condition:: { <is_hyperphagia(yes)> ∨ <is_polyurea(yes)> ∨ <is_polydipsia(yes)> ∨ <level(BS,⩾ 200mg %)> }

expected-condition :: {<level(90<FBS< 110mg %)> ∧ <is_hyperphagia(no)> ∧ <is_polyurea(no)> ∧ <is_polydipsia(no)> }

composed-of :: [< Diet_&_exercise >]

treatment-details :: [<Treatflow_Activity_Diabetes>]

limitation :: { < label(RBS)<150mg %>}

caution :: { <"fasting should be avoided" > }

treatment-duration :: { <forever> }

observed-condition :: {<FBS><nil>}{<weight><nil>}

treated-duration :: { <days> }

End Diabetes_treatment

3.5.3 Module Composition:

This section presents a technique to synthesize a treatment that is a combination of several treatment modules. A combination of modules is achieved using composition operators such as linear, parallel, choice, support and cooperate. The rules for composition using these operators are introduced in this section.

- **Linear**

  A Treatment Module TM_1 is linearly combined with another treatment module TM_2 when the following rule is satisfied.

  linear(TM_1, TM_2) ::
  $\{$ exec$((TM_1), exec(TM_2) \mid (TM_1.exc \Rightarrow TM_2.enc)\}$

  From the stated example, enabling-condition (Enc) and expected-condition (Exc) for Administration and Clinical-examination modules are specified as Administration.Exc :: $<$ available-patient-details(patient-id) $\wedge$ assigned-doctor(patient-id) $\wedge$ clinically-examined(patient-id) $>$ Clinical-examination.Enc :: $<$ available-patient-details(patient-id) $\wedge$ assigned-doctor(patient-id) $\wedge$ clinically-examined(patient-id) $>$
  In this case expected-condition of Administration module is equal to enabling-condition of Clinical-examination module. Therefore Administration and Clinical-examination modules can be composed linearly.

- **Parallel**

  Sometimes two or more independent treatment modules viz. (TM_1, TM_2) can be run in parallel when simultaneously their enabling-conditions are implied by expected-conditions of their previous module TM. Obviously

such treatments are recommended for speedy recovery. So, formally a parallel composition of treatment modules is specified as: enabling-conditions of (TM_1, TM_2) are simultaneously true.

Formally the rule for parallel composition is stated as:

parallel(TM,TM_1, TM_2)::

$\{ \exists (TM_1, TM_2) \mid (\text{TM.Exc} \Rightarrow TM_1.\text{Enc} \wedge \text{TM.Exc} \Rightarrow TM_2.\text{Enc}) \}$

For example according to the case study a patient having symptoms of hyperphagia, polyurea, polydipsia and chestpain undergoes both Diabetes-treatment and Chestpain-investigation simultaneously. Because in such case Enc and Exc of all the modules like Clinical-examination, Diabetes and Chestpain-Investigation are specified as follows.

Clinical-examination.Exc:: (level(BS,$\geqslant$ 200mg %)) $\wedge$ Is_hyperphagia(yes) $\wedge$ is_polyurea(yes) $\wedge$Is_polydipsia(yes) $\wedge$ Is_chestpain(yes)

Diabetes-treatment.Enc::(level(BS,$\geqslant$ 200mg %)) $\wedge$ Is_hyperphagia(yes) $\wedge$ is_polyurea(yes) $\wedge$Is_polydipsia(yes)

Chestpain-Investigation.Enc:: Is_chestpain(yes)

As enabling-conditions of both these treatment modules are equal, therefore both in the line of Diabetes-treatment as well as Chestpain-Investigation has been started for the patient simultaneously.

- **Choice**

  On administering a treatment module, the effect due to the module on patients may differ from person to person. And each type of expected-condition a separate treatment may be planned. This situation can be modeled by *choice* primitive and here we formally specify the same.

  Let there be a module TM with disjunctive expected-conditions say $Exc_1 \vee$

$Exc_2 \vee \ldots \vee Exc_n$.

If there exists a set of modules $TM_1, \ldots TM_n$ with enabling-conditions $Enc_1 \ldots Enc_n$ such that $Exc_1 \equiv Enc_1 \ldots Exc_n \equiv Enc_n$ then we say that the modules $TM_1 \ldots TM_n$ can be composed succeeding to the module TM with choice operator.

Graphically, the lines branching out from TM in $TM_1 \ldots TM_n$ can be labeled with their respective conditions for the sake of clear cut visual understanding.

Formally the choice composition is defined as

choice(TM,$TM_1 \ldots TM_n$) ::
$\{ \exists\, TM_1 \ldots TM_n \mid ((TM.Exc_1 \Rightarrow TM_1.Enc) \wedge$
$(TM.Exc_2 \Rightarrow TM_2.Enc) \ldots \wedge$
$(TM.Exc_n \Rightarrow TM_n.Enc)) \}$

Following the stated example it may be observed that Chestpain-investigation module may provide ECG changes suggestive of CHD, upper gastrointestinal endoscopy for APD. Based on each category respective modules viz. CHD treatment,APD treatment are to be invoked.

- **Cooperative/Supportive**

  On execution of a treatment module TM, say two modules viz. TM_1 and TM_2 are enabled for execution simultaneously in such a way that their executional environments are inter-dependent. That is one module receives a resource from another and similarly the later needs another resource from the former. Such modules are said to be cooperative. In reality such situation arises for example while treating a CHD (Coronary Heart Disease) patient a doctor may choose to run ABC (Airway Breathing Circulation)

procedure. and CHD treatment module as they support each other. Hence these two modules are said to be cooperative. A formal definition of two supportive modules is defined as.

Cooperative(TM,(TM_1, TM_2)::
$\{ \exists$ TM,$TM_1, TM_2 \mid TM.Exc \Rightarrow TM_1.Enc \wedge$
TM.Exc $\Rightarrow TM_2.Enc \wedge$
import$(TM_1, TM_2, v_1) \wedge$ export$(TM_2, TM_1, v_1) \wedge$
import$(TM_2, TM_1, v_2) \wedge$ export(TM_1, TM_2, v_2) $\}$

The above defines that the modules TM_1 and TM_2 are in parallel, health status v_1 due to the module TM_2 is used during execution of the module TM_1. Similarly, health status v_2 required by TM_2 is collected during execution of TM_1. This is modeled by two functions *export* and *import* as shown in the definition.
In case a module TM_1 during its execution imports a variable from another module say TM_2 but it does not import any variable from TM_1 then we say TM_2 supports TM_1 which can be formally represented as

Supportive(TM,(TM_2, TM_1)::
$\{ \exists$ TM,$TM_1, TM_2 \mid TM.Exc \Rightarrow TM_1.Enc \wedge$
TM.Exc $\Rightarrow TM_2.Enc \wedge$ $TM_2.Exc \Rightarrow TM_1.Enc \wedge$
import$(TM_1, TM_2, v_1) \wedge \neg import(TM_2, TM_1, v_2)$ $\}$

For example during Diabetes-treatment, Diet&exercise treatment needs to be carried out. Hence a module for Diet&exercise treatment supports a Diabetes-treatment module.

Table 3.1: Operators and Symbols

Operator	Symbol
Linear	$\rightarrow$
Parallel	$\parallel$
Choice	Ⓢ
Supportive	$\twoheadrightarrow$
Cooperative	$\leftrightarrows$

Table 3.2: Operators and Properties

Op↓/Prop →	Commu	Assoc	Trans
$\rightarrow$	X	$\surd$	X
$\parallel$	$\surd$	$\surd$	$\surd$
Ⓢ	-	-	-
$\twoheadrightarrow$	X	X	X
$\leftrightarrows$	$\surd$	$\surd$	$\surd$

3.5.4 Rewriting of a Treatment plan

In previous section we have introduced several operators that can be used to prescribe a treatment plan composed of several treatment modules. Compose operators correspond to their symbols as shown in Table 3.1. This indicates the sequence in which treatment is planned to proceed. But due to some problems like non-availability of resource like endoscope or certain drug, it may not be possible to run the module. In that case one needs to look for an alternative treatment plan. For the purpose, it is required to look for possible alternatives by exploring equivalent expressions to a given expression. A given expressions can be rewritten by using the properties of each composition operator. Some of the composition operators like linear, parallel, cooperative satisfy the associative property, whereas parallel and cooperative satisfy both commutative and transitivity property as shown in Table 3.2. Some of rewriting rules are listed below. With the help of which an equivalent treatment expression can be written.

Rewriting Rules:

- **parallel to linear**

  Suppose a treatment expression is represented as :

  $TM_1 \rightarrow (TM_2 \parallel TM_3)$

  If $TM_2.Exc \Rightarrow TM_3.Enc$ and

  Necessary condition: $TM_2.Exc \supseteq TM_1.Exc$

  Necessity: Necessity of re-writing is that

  if reqd-resource(TM_2)$\cap$ reqd-resource(TM_3) $\neq \phi$

  this may cause conflict for resource during execution. Hence if possible this parallel composition ($TM_1 \rightarrow (TM_2 \parallel TM_3)$) can be re-written as $TM_1 \rightarrow TM_2 \rightarrow TM_3$ if $TM_2.Exc \supseteq TM_1.Exc$

  Re-write rule

  $(TM_1 \rightarrow (TM_2 \| TM_3))::= TM_1 \rightarrow TM_2 \rightarrow TM_3$ if $TM_2.Exc \supseteq TM_1.Exc$

- **linear to parallel**

  Suppose a treatment expression is represented as

  $TM_1 \rightarrow TM_2 \rightarrow TM_3$

  If $TM_2.Enc = TM_3.Enc$ and $TM_2.Enc = TM_1.Exc$ and

  Necessary condition: $TM_3.Enc \subseteq TM_1.Exc$

  Necessity: The purpose of re-writing is that in case of emergency treatment modules like TM_2 and TM_3 should execute simultaneously, instead of one after another means $exec(TM_2) \wedge exec(TM_3)$

  Therefore linear composition of treatment modules can be re-written as $(TM_1 \rightarrow (TM_2 \parallel TM_3))$ if $TM_3.Enc \subseteq TM_1.Exc$

  Re-write rule

  $TM_1 \rightarrow TM_2 \rightarrow TM_3$::=$(TM_1 \rightarrow (TM_2 \parallel TM_3))$ if $TM_3.Enc \subseteq TM_1.Exc$

- **supportive to linear**

  A treatment expression comprises of treatment modules using support as one of the composition operator represented as follows

  $TM_1 \rightarrow (TM_2 \twoheadleftarrow TM_3)$

  If $TM_1.Exc \Rightarrow TM_2.Enc$ and $TM_3.Exc \Rightarrow TM_2.Enc$

  Necessary condition: $TM_3.Exc \subseteq TM_2.Enc$

  Necessity: In case of supporting treatment module is short-lived and terminate immediately after execution, then if possible this supportive composition can be re-written as linear composition.

  Re-write rule

  $TM_1 \rightarrow (TM_2 \twoheadleftarrow TM_3) ::= TM_1 \rightarrow TM_3 \rightarrow TM_2$ if $TM_3.Exc \subseteq TM_2.Enc$ and shprt-lived(TM_3)

- **supportive to parallel**

  $TM_1 \twoheadleftarrow TM_2 \equiv TM_1 \parallel TM_2$

- **cooperative to parallel**

  $TM_1 \leftrightarrows TM_2 \equiv TM_1 \parallel TM_2$

- **choice to linear**

  $TM_1 \circledS(\overrightarrow{c1}\ TM_2)\ (\overrightarrow{c2}\ TM_3) \equiv TM_1 \rightarrow TM_2 \vee TM_1 \rightarrow TM_3$

Using the properties listed in the table an expression can be rewritten to an equivalent expression. The given expression is

$(TM_1 \rightarrow TM_2 \rightarrow (TM_3 \parallel (TM_4\ \circledS(\ \overrightarrow{c1}\ TM_5)\ (\overrightarrow{c2}\ TM_6)) \rightarrow TM_9) \wedge TM_7 \twoheadrightarrow TM_3 \wedge TM_8 \leftrightarrows TM_5$

The above expression can be rewritten as

$(TM_1 \rightarrow TM_2 \rightarrow (TM_3 \parallel (TM_4 \rightarrow TM_5)) \rightarrow TM_9) \wedge TM_7 \parallel TM_3 \wedge TM_8 \parallel TM_5$

$\vee$

$(TM_1 \rightarrow TM_2 \rightarrow (TM_3 \parallel (TM_4 \rightarrow TM_6)) \rightarrow TM_9) \wedge TM_7 \parallel TM_3 \wedge TM_8 \parallel TM_5$

This shows that from the given case study a treatment plan stated in an expression can be rewritten to an equivalent expression. On making use of module specification, in the following section we investigate on change and anomaly managements in healthcare domain .

3.6 Exception Management in case of Treatment

Healthcare treatment often requires human participations of doctors, patients and collaborating staff ranging from paramedical to administrative personnel. Though each of them play well defined roles still human actions are liable to lapses. Patient treatment involves life risks; hence healthcare workflow management should have means to deal with these lapses. This feature of a system is widely known as *exception management*. Exceptions are to be differentiated from failures. Failure means that a system does not work say due to program error, device failure, communication failure etc. Exceptions are sometimes called as semantic failures which may arise when activities cannot be executed as planned or do not meet the desired result. Exceptions are categorized as *known* or *unknown* interactions. An exception that is identified during system analysis is termed as known exception otherwise unknown. A system is capable of processing known exceptions during

system analysis. The processing of these exceptions are explored and the system is designed accordingly. This kind of exception processing is widely found in workflow management systems in engineering domains or the domains where the roles and exceptions of the domain entities are well defined.

The management of exceptions in healthcare domain is strikingly different primarily due to variable behavior of patients even being affected by the same disease. This may result to discovery of new exceptions that are hither to not known. Hence a healthcare workflow management system must be in a position to deal with such undefined exceptions and to learn to handle with the same exceptions in future. In the following sub-section we have reviewed work related to exception management in healthcare domain. We focus on exception management in treatment process and present a comprehensive framework to deal with it. The proposed exception management process includes exception analysis, exception specification and processing of exceptions. These issues are elaborated in subsequent sub-sections.

Workflow describes the "normal behavior of a process where as an exception in case of workflow indicates "occasional behavior" [33] [60]. [28] proposed a methodology for modeling exception using activity graphs i.e WAMO (Workflow Activity Model) which enables the workflow designer in modeling not only current business process but also exception which may arise during execution. Thus a business process can be viewed as a composition of the core process and a collection of exception process. Guidelines are provided for specifying exceptional behavior in the three phases of exception handling that is detection, diagnosis and resolution. Behavior of an exception that represents deviations from a normal process, can be anticipated and handled accordingly [88]. Exceptions are specified using Chimera-Exc language which are specifically designed for expressing exception in WFMS [29][34]. In this formal properties of workflow application

with exceptions are discussed to indicate criteria for sound use of exceptions in workflow management.

There has not been enough work on exception management for healthcare workflow system. A work reported in [34] describes the usability of existing exception handling concepts in healthcare workflow systems. However, it suggests introduction of new techniques for exception handling. The suggested techniques include knowledge-based exception handling and dynamic exception handling. In general, it advocates that the existing exception handling techniques for workflow systems are mostly enough to manage exceptions in healthcare domain. As healthcare domain needs special attention to handle exceptions as life of patients are in stake, therefore the source of exceptions must include not only administrative entries but also patients, paramedical staff, doctors and resources. In nutshell, all entities that have a role in executing a treatment activity must be treated as a potential source of exceptions.

Though exception propagation in traditional workflow system is considered as hierarchical, we advocate a process that is not vulnerable to miss and delay. Again process based exception processing is traditionally considered. But, we are in opinion that this technique is computationally expensive with high latency time. Unknown exceptions are more expected in healthcare domain. And treatment to such an exception is exploratory. Hence, healthcare workflow management system should have capability to manage both unknown as well as exploratory treatment. In this section we make an attempt to evolve a holistic approach for handling exceptions in healthcare domain.

3.6.1 Exception in Treatflow

A Treatflow defines not only a sequence of treatment activities but also each treatment. For each treatment context i.e. pre-conditions and goal i.e. post-

conditions are defined. The scope of conditions for a treatment includes patient, staff as well as resources. Unsatisfiability of a condition raises an exception. In order to drive the point straight below we present an example. For example

A patient is admitted to the hospital with acute pain in abdomen. The patient is 40 yrs male having diabetes. He is having moderate fever and vomiting. The pain in abdomen is localized to right lower quadrant of abdomen. On admission necessary registration and investigation are being done. Then the patient receives injection for relief of pain in abdomen, vomiting and fever. His complete blood picture ,urine examination and random blood sugar is sent to lab for analysis. He is kept N.B.M (Nil By Mouth). IV (Intra Venous) fluids are started to hydrate the patient. IV antibiotics are started to control the infection (if any). Simultaneously the patient is examined by duty doctor to prevent any complications. As soon as the blood reports arrive the doctor is intimated. The doctor after going through the reports tentatively diagnoses the case to be acute appendicitis. The case is referred to general surgeon. After examining the patient he instructs the duty doctor to control blood sugar and advises for ultrasound scanning (U/S) of whole abdomen, X-Ray chest, ECG, diabetologist opinion and anesthetists opinion for the appendectomy operation. After the control of blood sugar and anesthetist PAC (Pre Anesthetic Checkup) the O.T.(Operation Theater) time slot is fixed .The nurse is advised to carry out the formalities such as information to patient and relatives, operation risk is to be taken in writing, preparation of the parts etc. The patient undergoes operation and post operative recovery is uneventful. The patient is observed in POW(Post Operative Ward) for 24 hours. Then the patient is shifted to the MSW(Male Surgical Ward).On seventh day the sutures (Stitches) are removed and the patient is discharged.

For analysis, we will make use of the above example. And a systematic analysis follows in the following sections.

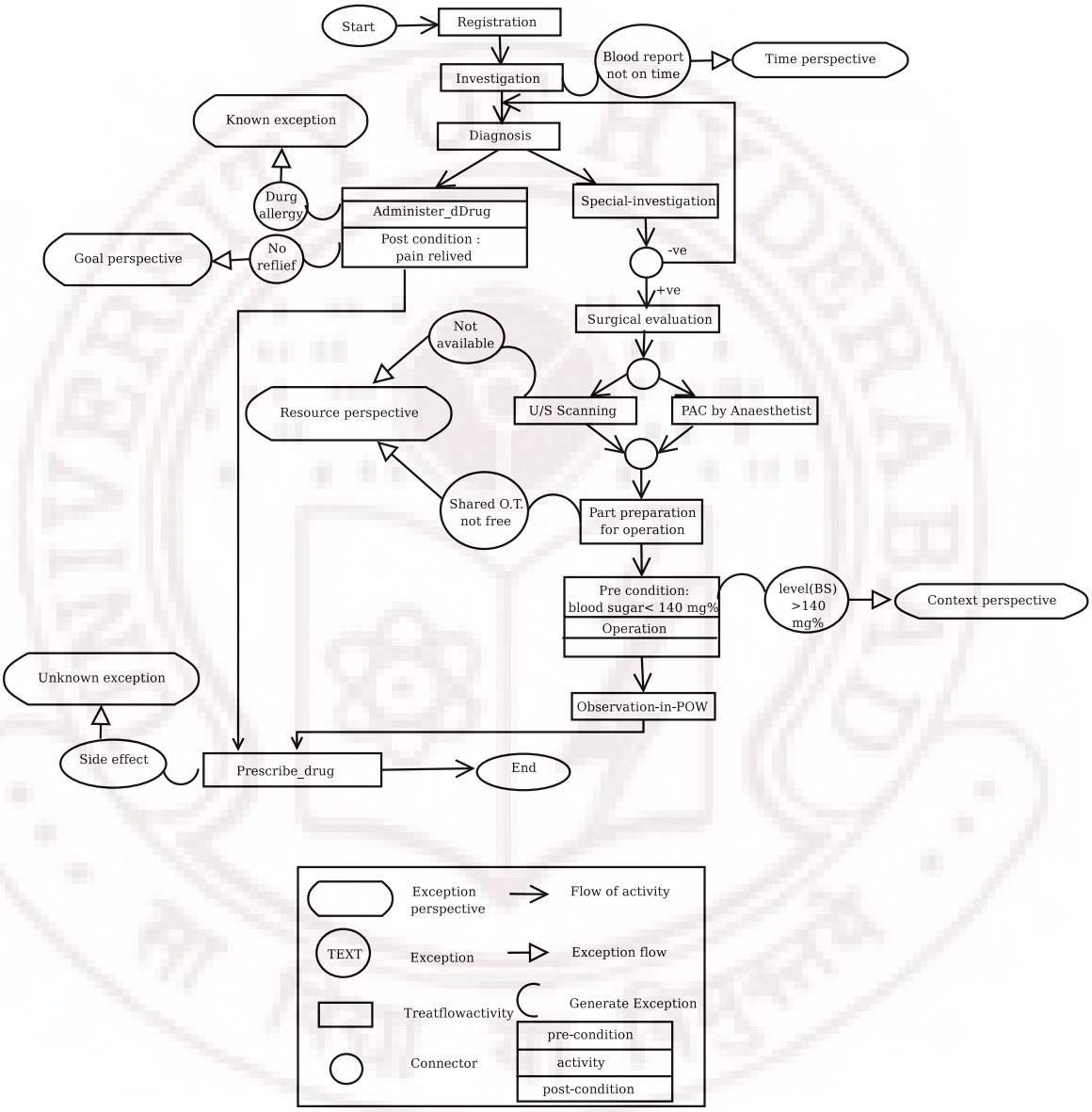


Figure 3.18: Exceptions in case of Treatflow

3.6.2 Exception Analysis

Usually there is a definite process defined to handle exceptions as seen in manufacturing domain and business domain. In case of healthcare, other than a generic way there could be specific method tailor made for a particular patient. Considering patient and process point of view analysis of exceptions has been done. In this section we analyze exception handling in healthcare domain with an aim to develop a process applicable in the domain.

A Treatflow means, for a particular ailment, same treatment can be applied to different patients suffering from the same ailment. Still, a treatment for a patient may proceed smoothly but for another patient with the same ailment may not go well through for patient specific reasons. Patient related exceptions can be either known or unknown type.

- **Known**

  Known exceptions are predictable deviation from the normal treatment such as patient suffers from allergy or sideeffect while undergoing a treatment. For example while administering drugs, doctor knows that patient may suffer from drug allergy which can be taken care of by mentioning these type of exception as known exception as shown in Figure. 3.18. For a system S, let there be a given treatment activity ta in a Treatflow with a set of treatment activities TA. There are specified exceptions that arise due to non-confirmation of any number of post-conditions or safety-conditions or confirmations of sideeffect during the execution of the treatment, while system is aware of an exception say *aware-of* (excep), if and only if that is present in the list of exception *exception-list*.

  This is formally represented as

aware-of(S,ta,excep) ≡ { ta ∈ TA ∧ throws(ta,excep)

| ((not-true (ta,post-condition) ∨ not-true(ta,safety-condition)

∨ has-sideeffect(ta,sideeffect)) ∧ is-in(excep,exception-list) }

where *aware-of* is a predicate with arguments system S, treatment activity ta and exception excep and *throws* a function that system S uses to signal occurrence of an exception and predicates viz not-true, has-sideeffect and is-in have their usual meanings.

- **Unknown**

  Unknown exception could be due to inconsistencies between a treatment process and biological process of a patient that is undergoing the treatment. A particular patient may get sideeffect while undergoing treatment regarding which doctor is not aware of from the beginning. For example while executing *Prescribe-drug* as a treatment activity, due to some patient specific reason he/she may get sideeffect which is not-known to doctor. Therefore it is considered as unknown exception and needs to be handled by contacting doctors in case of emergency as shown in Fig.3.18. During execution of a treatment activity *ta* within a Treatflow may throw an exception *excep* whcih is *not-aware-of* by the system S.This type of exception arises may be due to patient complaints or unsatisfied post-condition and which is not present in the list of exception *exception-list*.

  Which can be formally represented as :

not-aware-of(S,ta,excep) ≡ { ta ∈ TA ∧ throws(ta,excep)

| ((not-true (ta,post-condition)

∨ has-patient-complaint(ta,patient-complaint))

∧ is-not-in(excep,exception-list)}

where *not-aware-of* is a predicate with arguments system S, treatment activity ta and exception exce. *throws* a function that system S uses to signal occurrence of an exception and predicates viz not-true, has-patient-complaint and is-not-in have their usual meanings

A non-progressive state or a stalemate state arising during execution of a treatment triggers an exception. This situation is viewed as a failure of treatment process and such a failure of course, is to be attended. The action to deal with such situations is modeled as exceptions in treatment process. Considering the genesis of exceptions we categorize them into

- Resource
- Context & Goal
- Time
- Safety

perspectives. Exceptions of different perspectives are dealt in the following subsections.

Resource Perspective

Usually a treatment activity may require resources like a diagnostic report, doctor, nurse, operation theater etc. Non-availability of resources, on which treatment activities depend, may throw exceptions. For example the following treatment activity *U/S-scanning* is not able to execute and throws an exception due to the unavailability of scanning machine as shown in Figure. 3.18 In a system S, among the set of treatment activities TA, a treatment activity ta needs a resource r from a list of resources say *resource-list* for execution . Due to unavailability of resource

r, the treatment activity ta may throw an exception excep which can be formally represented as

$\exists$ ta $\in$ TA | to-execute(ta) $\wedge$ not-available(resource-required(ta))
$\Longrightarrow$ throws(ta,excep)

where *resource-required* a predicate with arguments treatment activity ta and attribute resource-required *not-available* and not-true are the predicates having usual meaning

In order to minimize nonavailability of resources and thus generation of exceptions, a healthcare workflow management system needs to have provision for resource management. In many areas of computer science like operating systems, distributed computing, the issue of resource management has been studied extensively. In this study we have not taken up this issue as our focus is on specification of treatment activities.

Context and Goal Perspective

Genesis of exceptions can be found on analysis of an activity in Treatflow. A treatment activity is taken up on a particular context with qualifying condition. For example, for blood sugar test a patient should not have breakfast on the test day. The qualifying condition no-breakfast-on-day(), in the context of blood sugar test can be thought of as a pre-condition. Obviously, a treatment activity is performed for a purpose and this is termed as the goal of the activity. On execution of the activity, it reached its goal state. The goal state of the activity is qualified with state conditions that are desired to be true at the goal state. These conditions are termed as post-conditions. Unlike pre-conditions that initiate an activity execution, there are cases where a set of conditions are required to be true during

execution of a treatment. For example, during an operation the pulse rate should be always within range 60-90 per minute. Any violation of pulse rate constraint in the context of operations generates exception. Here the constraint in the context of an operation can be thought as an invariant and any violation to an invariant causes exceptions. These cases are illustrated below by quoting activities shown Figure. 3.18.For example for *operation* as a treatment activity blood sugar level must be < 140 mg % which may not be true for all patients. In those cases this type of treatment activity may throw an exception. Generally a treatment activity ta in a Treatflow checks for the pre-condition before execution. An exception excep arises by the treatment activity ta in case context is not satisfied for that treatment activity. It can be formally represented as

$\exists$ ta $\in$ TA | not-true(ta,pre-condition)
$\Longrightarrow$ throws(ta,excep)

Similarly for example relief-of-pain may not be true after *Administer-drug* as a treatment activity as in Figure. 3.18. Usually goal of a treatment activity must be true after execution. In some cases exceptions may arise if the goal of the treatment activity does not satisfy which can be formally represented as

$\exists$ ta $\in$ TA | execute(ta) $\wedge$ not-true(ta,post-condition)
$\Longrightarrow$ throws(ta,excep)

where *execute* is a function where a system S uses to execute a treatment activity ta.

Temporal Perspectives

Some treatment in Treatflow may have temporal characteristics e.g physiotherapy is to be done 2 times in a week, a drug for blood pressure control is to be taken at a specified time or operation theater is to be disinfected in periodic interval. Thus, for a treatment temporal constraints can be associated and a violation of such a constraint usually generates an exception. These cases are illustrated below by quoting activities shown Figure. 3.18 According to the figure all reports should have arrived in time as a result of post-condition of treatment activity *Investigation*. This treatment activity generates an exception which violates time constraint due to the reports not received on time. In a system S, operation-time is given to each treatment activity ta for its execution. An exception can be thrown by the treatment activity ta if it is not able to complete its execution within the specified period of time. It can be formally represented as

$\exists$ ta $\in$ TA | under-exec(ta) $\wedge$
not-intime(ta,operation-time)
$\Longrightarrow$ throws(ta,excep)

where operation-time is the time for execution for treatment activity ta
and *not-intime* is a predicate with the arguments treatment activity ta and operation-time

Safety Perspectives:

Some treatment in healthcare may have safety-condition characteristics e.g while Pencillin is administered during *Administer-drug* activity, a test dose intradermally should be done to know sensitibity lest severe anaphylactic reaction may

arise causing death in some of the cases. Thus, for a treatment safetycondition can be associated and a violation of such a condition usually generates an exception. In a system S, safety-condition is given to each treatment activity ta for its execution. An exception can be thrown by the treatment activity ta if it is not able to satisfy the safety-condition. It can be formally represented as

$\exists$ ta $\in$ TA | under-exec(ta) $\wedge$
violate(ta,safety-condition) $\vee$
reported(ta,side-effect)
$\Longrightarrow$ throws(ta,excep)

where under-exec is for treatment activity ta under execution
and *violate* and *reported* are predicates with the arguments among treatment activity ta with safety-condition and sideeffect respectively.

Further, an omission of execution of a treatment in a Treatflow disturbs temporal sequence of treatment primitives. Such an omission is possible because a treatment primitive can be manually executed e.g. a patient skips a physiotherapy session. Again, failure of handling an exception in a defined time period should also throw an exception. Though this results to nesting of exceptions, but to manage complexity in exception processing we resort to procedural execution of exceptions. This aspect we deal in detail late during our description on exception processing. A treatment may generate several exceptions. Based on sources of exceptions we categorize them into different specializations. A given exception is could be one of the categories viz. Resource, Context & Goal, Safety and Temporal. In order to understand the types of exceptions in perspective of their resources, we have undertaken further analysis. For each perspective, we have defined several states and for each state we have specified a predicate that

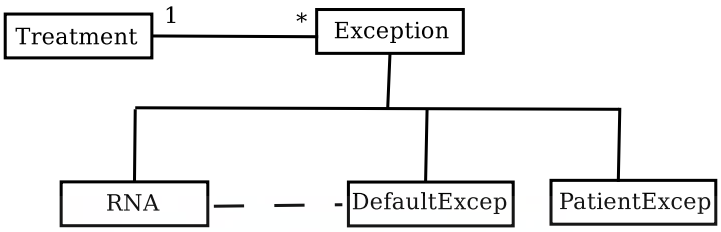


Figure 3.19: Exception Categorization

is instrumental in throwing exceptions as shown in Table 3.3. For example for resource perspective, the states defined are *rna*, *rnsrbl* and *srnf* and these state acronyms stand for ,'resource not available', 'resource not shareable' and 'shared resource not freed' respectively. For the rna, the predicate is Notavail (r) where r ∈ R means resource r not available.

Similarly for rnsrbl the predicate is Notsrbl(r) where r ∈ R means resource r not shareable

and for srnf the predicate is Srbl(r) $\wedge\neg free(r)$ where r ∈ R means shared resource not free.

For context and goal perspective, the states are *cns*, *agns*, *uwntg*, *udefg* and acronyms for these states are 'context not satisfied', 'anticipated goal not satisfied', 'unwanted goal' and 'undefined goal' respectively.

Predicate for cns is ¬ (C) | C ∈ context means context C not satisfies.

Similarly for agns ¬ (G) | G ∈ goal meansi not according to the anticipated goal.

For uwntg G ∈ UG where it is of type known exception and which is patient specific for example sideeffects and allergy etc.

But udefg is of type unknown exception which differs from patient to patient and is not pre-defined means not available. On presenting a comprehensive analysis on exception and their origins, in the next we would like concertize exception description with specification. And also discuss on exception handling issues. A treatment may generate several exception in context with the discussed perspectives. Treatment can throw exception of different types or may generate events as shown in Table 3.3 Apart from the origin of exceptional event, exceptional events

h

Table 3.3: Exception Analysis

Exception Type	**Type Description**
rna (Resource not available)	Notavail (r) r ∈ R
rnsrbl (Resource not shareable)	Notsrbl(r) r ∈ R
srnf (Shared resource not free)	Srbl(r) $\wedge\neg free(r)$ r ∈ R
cns (context not satisfied)	¬ (C) | C ∈ context
agns (Anticipated goal not satisfied)	¬ G | G ∈ Goal
uwntg (Unwanted goal)	G ∈ UG
udefg Undefined goal	G $\notin \{G\}$
nct (Not completed within time)	¬ complete(TA,t) | TA ∈ treatment activity $\bigwedge$ t ∈ time

and their corresponding actions is discussed in table 3.4. Actions for each exceptional event depend on the type of origin either known or unknown. Actions of each exceptional event specifically depends on event location i,e treatment activity(TA) level, multiple treatment activity in Treatflow(TF) or treatment module (TM) level. Again for each exception, an action is specified and each action may deal on a treatment or add few more treatments to the treatment plan.

3.6.3 Specification of Exception

Treatflow is a kind of workflow consisting of a sequence of Treatment activities where each activity is defined in 3.3 with pre-conditions and post-conditions. In the last section we have discussed on possible sources of exceptions. A Treatment activity may generate several exceptions and an exception could be of one of the types viz. rna, rnsrbl, srnf, cns, agns, uwntg, udefg and nct. We have discussed on each type in the last section. All these types of exceptions are of known types;

Table 3.4: Exceptional Events with Actions

Exception Events	Exception Type	Action
rna	Known	Allocate
rnsrbl	Known	Unlock/Getnew/Default
srnf	Known	Release/Pause
cns	Unknown	Pause/Default
cns	Unknown	Inform
agns	Unknown	Pause/Default
uwntg	Unknown	Default
uwntg	Unknown	Default
udefg	Unknown	Default
nct	Known	Pause
nct	Known	Inform

Table 3.5: Types of Actions

Action	Description
Allocate	Assign resource
Pause	wait in the current treatment activity
Getnew	Get new resource
Unlock	Unlock resource
Default	Call a doctor
Release	Release resource
Inform	Inform patient

hence can be specified even before the system development stage. In addition to those we will add two more types called DefaultExcp and PatientExcp as shown in Figure 3.19.These two types of exception events are designed to deal with unknown situations. As we have told earlier, in case of healthcare workflow handling of unknown exceptions is crucial as one of such exception could be life threatening. We have dealt with unknown world using default reasoning. For each patient, a doctor can define a closed world specifying permissible values to all concerned life protecting parameters. As and when a defined health parameter assumes a value that is out of its defined range, Treatflow management system throws an exception of type DefaultExcp . This provision really does not take care of all possible unknown types of exceptions. This is simply because, even sometimes

doctors may fail to define a complete closed world that is safe for a patient. Or there could be a miss for human error. In order to take care of such situations we propose to have another type of exception called PatientExcp to allow a patient to create an exception of the type. As and when a patient feels uneasy can express health conditions to systems for help and to intimate the concerned doctor for help. Typically, an exception is identified by a its unique identification number *id* and further described by its type, generator description, priority, condition and action. A formal description of exception is given as:

Exception :: {<Excepid>}
Egenerator :: {<TFid> <TAname>}
Priorty :: {<int>}
Condition :: {<predicate>}
Action :: {<action-name> <action-parameter><executer><time>}
End

Unique identification of an exception is considered for distinct accessibility as well as for clarity on dealing with exceptions. An exception is generated by an instance of a Treatflow being managed by Treatflow management system. At a given time, there could be several instances of Treatflows being operational. An instance while executing a treatment activity can throw an exception. In order to trace an exception we need to refer to the instance by *TfId* and corresponding activity *TAname* that has raised the exception. So, the field EGenerator::¡TfId¿.¡TAname¿ A type definition associated to an exception gives an idea on why the exception is generated. This explains who and why a given exception is generated. Before proceeding to take action for a given exception we have made a provision to doubly check on justification of action. And this has been formalized in field *Condition*.

The field can have predicates that are to be satisfied for taking an action with respect to the exception. Similarly, the field *action-name* describes what action is to be taken in connection with an exception and who is the concerned entity/enitities for doing that. It is formalized in subfield *executer* of Action. The subfield *action-parameter* further describes the resource an action may need. Latency for taking action is specified as a subfield to Action field. In nutshell, we follow the well known *event-condition-action* paradigm according to the exception specification discussed above. Here is an example of an exception stated as per the specification.

Exception :: {<ER1>}
Egenerator :: {<TF1> <Do-U/S-scanning>}
Priorty :: {<1>}
Condition:: {< to-exec(ta) $\wedge$ not-available(resource-required(ta)) >}
Action:: {<Allocate-resource(ta,r)>}
Priority :: {<1>}
End

On specifying exceptions, now we deal with the mechanism that is followed to catch exceptions and process them. We make use of *throw-catch* mechanism available with *Java* programming language.

3.6.4 Exception Processing

Processing a treatment activity includes testing the satisfiability of pre-conditions, executing actions pertaining to the activity and the satisfiability of post-conditions. A class diagram in Figure. 3.20 depicts the processing of exceptions of different types. These satisfiability tests throw exceptions. A class called SatisfyTreat-Condition with methods *TestPreCond* and *TestPostCond* respectively test pre-

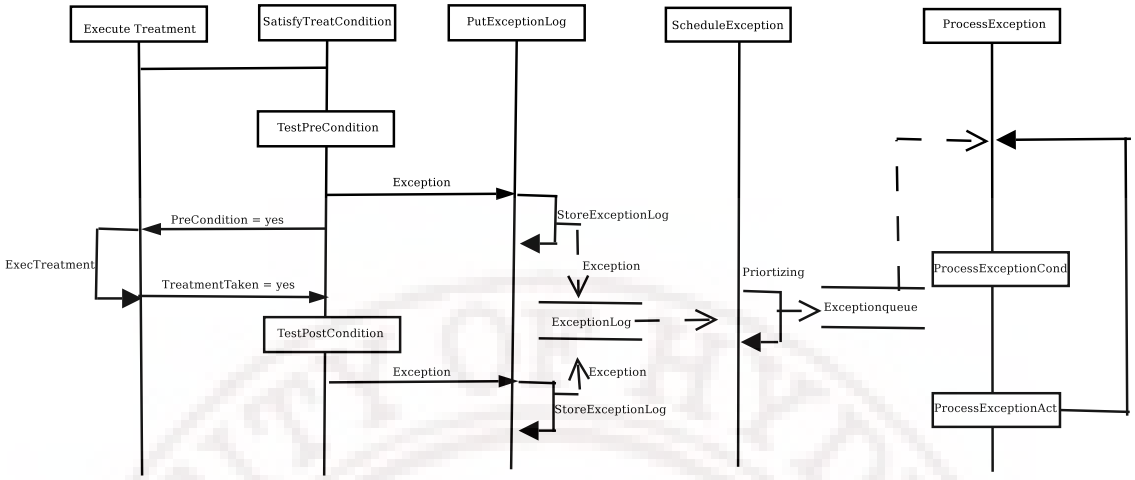


Figure 3.20: Processing of Exceptions

conditions and post-conditions associated to an activity. This class instance throws exception events when the predicates in pre-/post-conditions are not satisfied. An exception event could be an instance of types of exceptions listed in Table 3.3. As we understand generation of exception is always asynchronous activity. The exception handler of the class instance *SatisfyTreatCondition* catches an exception and instantiates it (by *PutExceptionLog* class calling method *StoreExceptionLog* as shown in Figure: 3.20. The instance of an exception event is stored in a buffer *ExceptionLog*. This buffer is organized as a queue data structure. At one end the generated exception events are stored and at the other end of the queue an exception event is deleted for processing. So, at any given time *ExceptionLog* will have exception events that are to be processed. The logging of exceptions by Treatflow management system is shown in 3.21 A monitor class *ScheduleExecption* comes

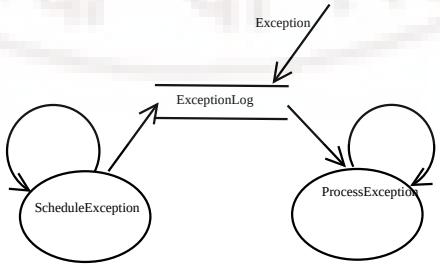


Figure 3.21: Logging of Exceptions

up in regular interval of time and generates a schedule of exception events. This

arranges exception events in descending order of priority. The time interval for activation of the monitor can be tuned so that the scheduling time and exception event processing together should be much less than the minimum of the time specified for execution of exception events. This is an ideal condition and sometimes hard to achieve for a system that mostly deals with very time sensitive treatments like open heart operations etc. The practical approach is to set the time less than the average of the time specified for all types of exceptions. The purposes of the scheduling is to provide maximum attention for high priority exception events. Of course, this will lead to stagnation of low priority exception events. This can be interpreted as a negligence of care to a patient which may further cause harm to the patient. In order to avoid such situations we propose to dynamically calculate priority that is proportional to the quantum of the time an exception is waiting on buffer for processing. A Java thread called *ProcessException* always haunts for exception events pending in queue *ExceptionLog*. It deletes an event from the queue and process it to take necessary action. As a nutshell processing of an exception includes evaluation of conditions and on succeeding to it, specfied actions are to be taken. The thread is mainly composed of two methods called *ProcessExceptCond* and *ProcessExceptAct* to evaluate conditions and to take actions respectively. The predicates are to be seen in different prospectives viz. resource, context & goal, temporal and safety. For an example a predicate could be on patient health status as *temperature ¿= 102c* . For evaluating this predicate, the method gets value fo temperature of the patient (related to the instance of Treatflow) and compares with 102c. For this, obviously it's required that healthcare management system is maintaining repository of all necessary information pertaining to each Treatflow instances under execution. So, the method has an interface to such a database for retrieving information required for the evaluation of predicates. On successful evaluation of all the relevant predicates associated to an exception; the *Proces-*

sExceptCond invokes the method ProcessExceptAct. The invoked method based on types of exceptions may take actions as listed in Table 3.5. The generic actions are *Allocate*, *Pause*, *Getnew*, *Unlock*, *Default*,*Release*,*Inform* etc. At a given time there could be several instances of Treatflow under execution. And any of them can throw an exception. All the exceptions generated by different instances of Treatflow are stored at ExceptionLog. So, the proposed exception handler of Treatflow management system treats all the exceptions globally instead of individually. The rationality of planning this way to provide equal opportunity for all the patients and to provide attentions to more needing patients.

3.7 Summary

Traditionally structured method is being used for system analysis and design. The reason for its popularity is due to the pictorial presentation of a system. However , the method lacks mathematical preciseness that can be used to prove the system mathematically. Formal method in software development helps in proving the system property mathematically.

The proposed work used some primitives viz, linear, split and merge , parallel , supportive , cooperative and choice that are basic to represent workflow. The uses of such primitives are shown in specifying workflow of a corporate hospital for treating patients. For each type of primitive other than pictorial representation, system behavior is also specified mathematically. Execution of each primitive is analyzed and shown that it is possible to make unambiguous inferences on execution of each workflow primitive. The analysis is implementable as it uses assertions as pre-condition and post-condition. A rule based system with assertions as rule can be used for verification and validation on workflow primitives pertaining to application domain. A business process can be synthesized as

a composition of workflow primitives. On reviewing works on modularization of workflows we have method to specify a treatment module and operators to compose a treatment. It is shown that a treatment prescription can be written as an expression of modules. As sometimes it may be necessary to explore all the possibilities in executing a treatment plan, we propose a method for rewriting a given expression by applying rules to rewrite.

Currently we are studying on correctness of a treatment expression written with modules. As modules at coarse level represent activities, we are interested in visualizing all possible control flows among defined activities. For the purpose, a modular expression is represented in a graph form and all possible paths are instantiated to study the Illegality especially incompleteness,deadlock and lack of synchronization. Here we focus on exceptional behavior of healthcare workflows and provide a comprehensive analysis on generation of such exceptions at the deviating behavior of a specified workflow. We also define actions that can be performed with respect to exceptions. The feasibility of this concept is known in a case study taken from healthcare domain.

Chapter 4

Software System Architecture on Treatment Plan in Healthcare System

[1]

This chapter identifies Treatflow Management System which aims at automating a treatment process. TFMS is an instantiation of a Treatflow. Here we propose a framework to implement Treatflow Management System. A requirement analysis for such a system considering users viz. (doctors, patients and staffs) and their roles in treatment process has been discussed. The analysis is presented with the help Usecase and Activity diagrams. These diagrams lead to a first-cut of a system architecture and the software modules that the system should have. Then we provide higher level design diagrams for each module. In order to be brief and at the same time to bring clarity on design decisions, we have made use of Class and Sequence diagrams.

4.1 Introduction

The proposed Treatflow Management System aims at automating treatment process and is different from other existing healthcare systems which focus on patient registration, bill collections, maintaining the history of medication undergone by patients especially EPR: electronic patient record management. We feel, such a system will be helpful for both doctors and patients in managing treatment process. On registration of a patient, a doctor can make use of the proposed system

[1]This work has been published published in proceeding of International Conference on Managing Next Generation Software Application, Coimbatore, pages 871-879, 2008

to synthesize a treatment process for the patient by choosing treatment modules which the doctor thinks useful for treating the patient. The patient views the treatment records and opdates his/her status. Using the system, the doctor can give health advice to the patient. The proposed system aims at addressing these requirements.

In previous chapters we have provided a theoretical basis to specify a treatment process and here we propose a framework to implement it. First, we present requirement analysis for such a system considering users viz. doctors, patients and staffs and their roles in the treatment process. The analysis is presented using Use case and Activity diagrams. These diagrams lead to first cut of a system architecture and the software modules that the system can have to meet the requirements. Then we provide a higher level design diagrams for each module. In order to be brief and at the same time to bring clarity on design decisions, we have made use of Class and Sequence diagrams.

In the following section we present a survey on state of the art of healthcare management systems. This section is followed by Requirement analysis of the Treatflow Management System(TFMS) is discussed in 4.3. The proposed system architecture is discussed in section 4.4. And then, a design overview of each major architectural component is presented in section 4.5.. The behavior of a prototype system is explained in section 4.6. Finally, the chapter ends with a summary.

4.2 Related Work

A software architecture for workflow management system model comprises of major components of workflow management system and their relationship amongst the components [38]. In some cases architectural description is basically divided

into two parts i.e. start up time architecture during which the workflow is constructed and build up time architecture during which the combined negotiation is conducted [10].

In case of healthcare, an architecture that integrates workflow management and context aware actions for personalized healthcare services has been proposed [3]. It supports interaction among hospital personnel and patients and stores patients' health status for future reference. The proposed architecture follows web service composition techniques to compose healthcare services for a patient and implements collaborative autonomous agent concept to deliver context-based personalized healthcare advisory to patients. The architecture deploys agents like guideline manager, supplier, context manager, monitoring devices, context record manager and activity execution agent. The agents like guideline manager, supplier, context manager, monitoring devices are treated as web services that can be triggered by invoking the other agents viz. context record manager and activity execution agent. These agents can be located on Internet thus making it a distributed architecture. Mainly, the architecture is meant for managing work schedules of healthcare staff and issuing health advisory to patients. It logically decouples workflow management and action management but is able to provide orchestrated services by deploying collaborative agents.

Flexible workflow management system in the area of patient care delivery was designed and implemented by using high level components. In this case OBJCHART was used as a coordination and communication manager. Similarly CLP(R) was used as a logic based deductive system for reasoning purposes and $W_a f^e$ wss used for graphical user interface and front end capabilities [58].

4.3 Requirement Analysis

As stated before,Treatflow Management System (TFMS) mainly focuses on treatment process management. A treatment process is initiated by a patient on the first visit to a doctor for consultation on some ailments. In the process of treatment actors (users) for example, administrative staff for patient registration play certain roles. Here, we have narrowed our scope by considering three types of users viz. doctor, patient and staff, and perform the analysis. Usually we find three types of users in a small or medium sized typical healthcare organization. A similar approach can be extended for other actors in correspondence to their roles in treatment process.

The purpose of this system is to specify and verify a Treatflow as well as to execute, monitor and handle exceptions while treating a patient. This system is to be used as an aid by different actors to participate in the treatment process and moreover the proposed system should have proper means for enabling interaction as well as making decisions. Next we analyze the requirements in perspective of each type of users.

- **Patient Perspective**

  Patient as a user can use the services provided by the system. A patient wishing to use system services is first to be authenticated. A patient can perform two types of services used for retrieval of treatment related information and providing feedback on effectiveness of treatment process. Treatment related information can be provided to patient on request basis (received from the patient) and by pushing the information to the patient. Thus the system provides both pull/push modes for information retrieval. Pushing of information to a patient is performed by a doctor or a paramedical staff in charge of

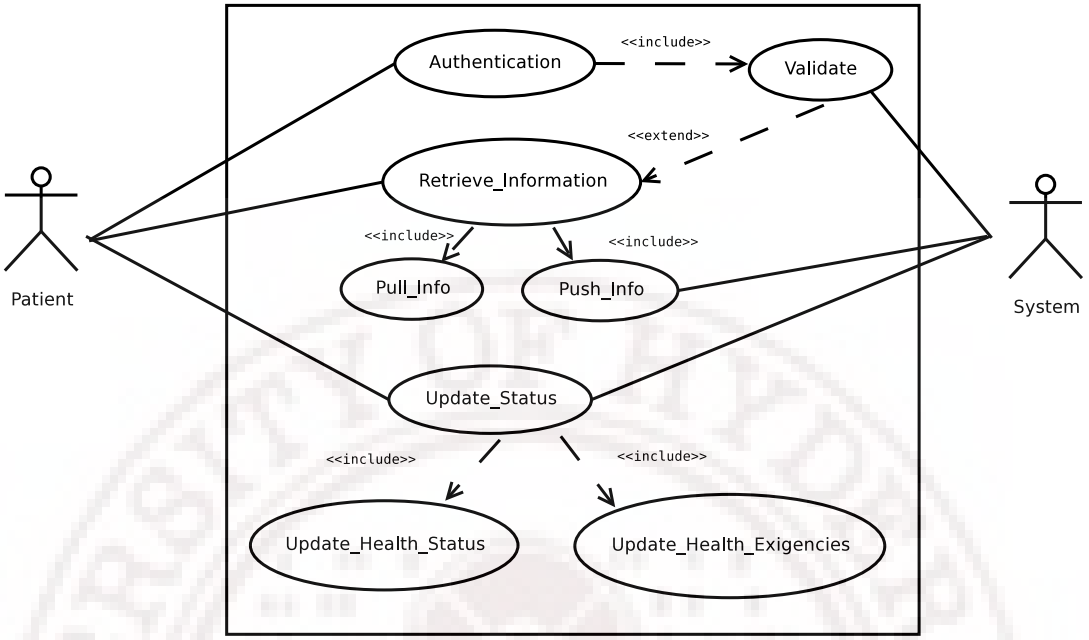
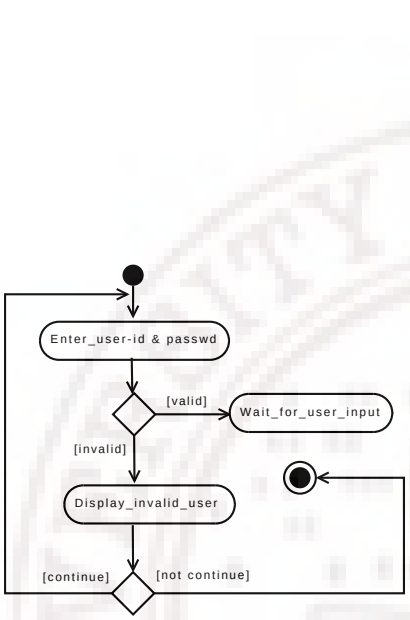


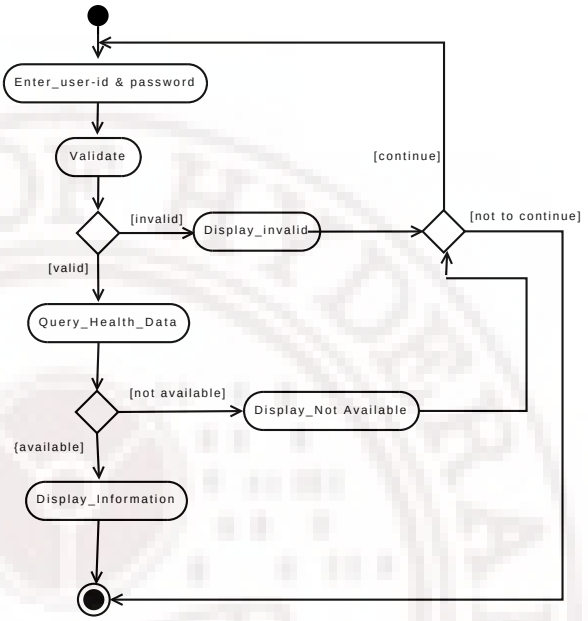
Figure 4.1: Use case Diagram for Patient

treating the patient. This can be implemented by making use of email/short message services. Pulling information is primarily performed by a patient for clarifications and to acquire necessary information on treatment process. For example, a patient may refer to nutrition advice during intervention of a particular drug. Treatment related information regarding current line of treatment, care, caution, prescription of drug etc must be available to patient on demand. As told earlier, the other objective for patient interaction is to provide feedback. For example, on intervention of a drug, the treating doctor may like to know its effectiveness and for that the patient is required to inform about his/her health status. Similarly, a drug may react and a patient may develop exigencies that requires doctor's intervention. In order to deal with situation, the patient should have means to feed the system on his/her health conditions and inform the concerned doctor and staff. The use case diagram given in Figure.4.1 has three major use cases viz. Authentication, Retrieve_Information and Update_Status. The first use case is

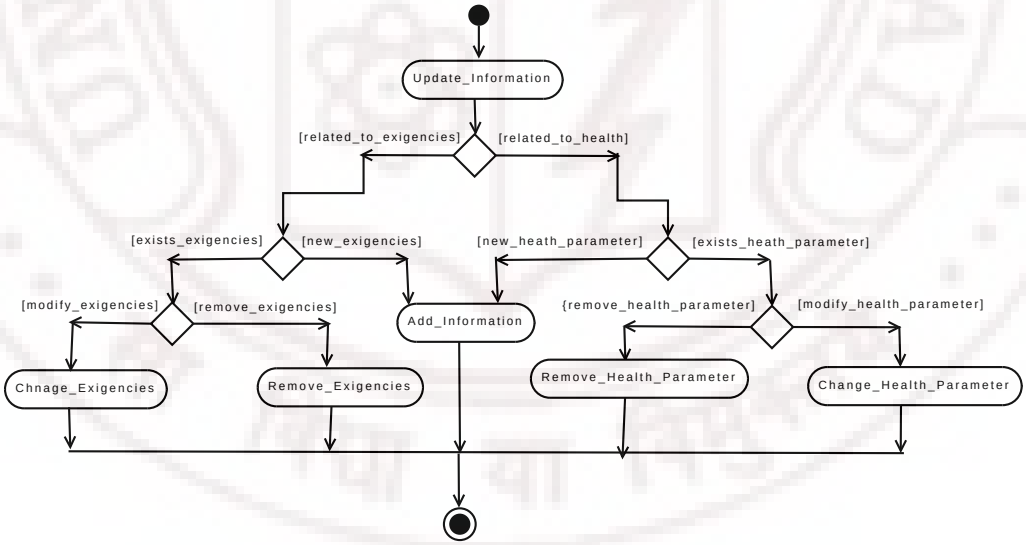
meant to ensure security and privacy for a patient so that only the patient is eligible to access/update his/her health information. The second use case Retrieve_Information uses two sub-use cases viz. Pull_Info and Push_Info. These sub-use cases receive information from database. This database contains information on patient's health status as well as treatment process. The database is instantiated at the beginning of a treatment process for a patient. And it is updated by the patient, the treating doctor and the staffs. While patient updates health status, the treating doctor update treatment process. The third major use case is Update_Status and it has two sub-use cases. These are Update_Health_Status and Update_Health_Exigencies. These sub-use cases are meant for a patient to update health status and to inform on health exigencies. Figure.4.2(a) presents flow of activities pertaining to Authentication use case as shown in Figure.4.1. In this case patient as a user has to enter his/her user-id and passwd which is to be validated by the system in order to maintain security. Lest an incorrect user should re-enter user-id and passwd if he/she has to continue further. Similarly, activities for patient Retrieve_Information use case are given in activity diagram as shown in Figure.4.2(b). After authentication, patient can query his/her health related information. The information is displayed if it is available, otherwise a message will be displayed informing about the unavailability of information. Activity diagram for patient Update_Status use case is given in Figure.4.2(c). In order to give feedback, patient has to update information regarding either health status or health exigencies. In both the cases, if information already exists and there is a need to change the information, then in those cases modification of information related to health status as well as health exigencies can be done. And if the information already exists and that information is not necessary, then that information can be removed.



(a) Activity Diagram for Patient Authentication Use case



(b) Activity Diagram for Patient Retrieve_Information Use case



(c) Activity Diagram for Patient Update_Status Use case

Figure 4.2: Activity Diagrams for Patient

Otherwise new information can be added.

- **Doctor Perspective**

  As an authenticated user, doctor can use the services viz. Authentication, Manage_Treatment and Handle_Exigencies provided by the system. Doctor has to make a treatment after going through patient's health status. Doctor can create a treatment with the help of Treatplan. Treatplan is available in database which can help not only in specifying but also in verifying a treatment. A treatment plan may have several treatments for patient having different ailments. During the treatment process, doctor can view the details about the treatment. Depending on the progress of treatment and based on doctor's past experience, he/she has to decide whether to continue with the same treatment plan or to make some change in the current line of treatment or altogether go for an alternative treatment. In certain cases exigencies may happen which have to be taken care of by doctor. Doctor can also update patient's health status in the database. The use case diagram given in Figure.4.3 has three major use cases namely Authentication, Manage_Treatment and Handle_Exigencies. Manage_Treatment use case which has two sub-use cases viz Make_Treatment and Update_Treatment. Make_Treatment has two sub-use cases viz Create_Treatment and Validate_Treatment. Doctor can make a treatment The Create_Treatment use case is meant for making a treatment. Doctor can make use of Treatflow which is already stored in the database, for creating a treatment. Therefore this Make_Treatment use case which can extend to Treatflow. After making a treatment, the treatment plan can be validated with the use of Validate_Treatment use case. Similarly Update_Treatment use case has three sub-use cases viz. View_Treatment, Modify_Treatment and Delete_Treatment. Doctor as an actor, can view all the details of treat-

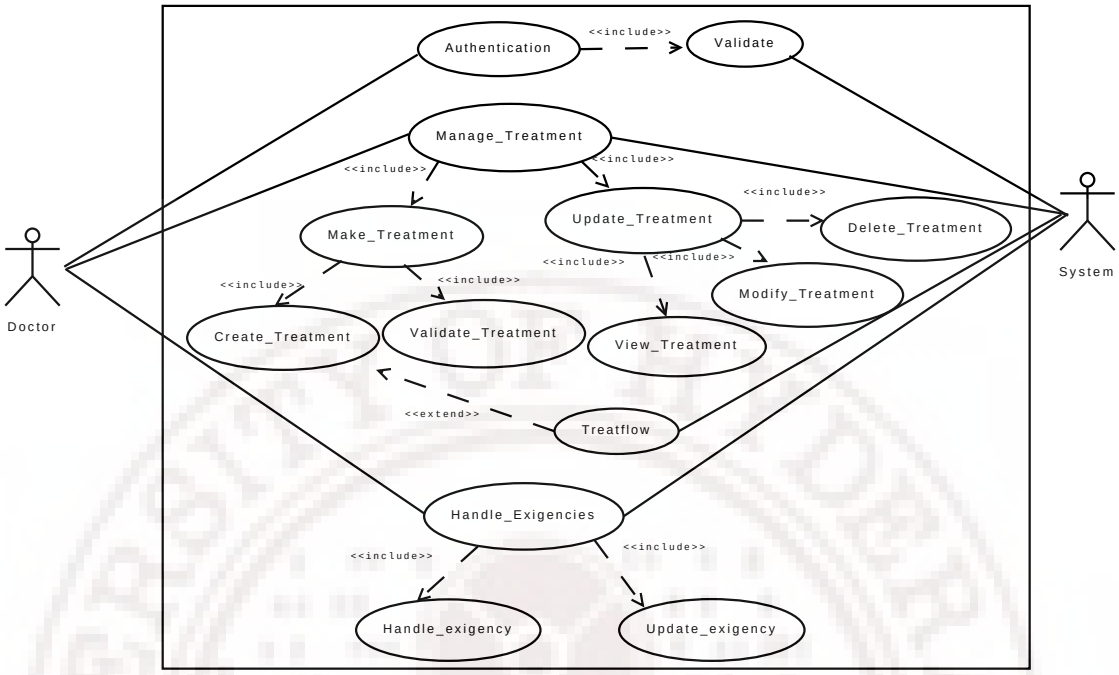
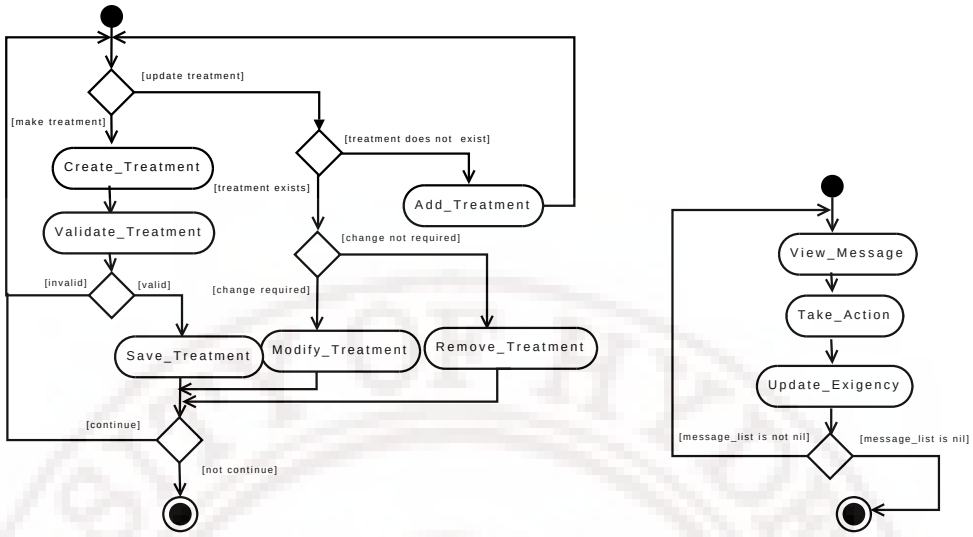


Figure 4.3: Use case Diagram for Doctor

ment with the help of View_Treatment use case. After going through patient's health status and treatment detail, doctor has to decide whether to continue in the current line of treatmet or to make changes in the treatment process. Apart from viewing, doctor can change a treatment plan with the help of Modify_Treatment use case or can remove treatment plan with Delete_Treatment use case. The last use case is the Manage_Exigencies which is used to view the exigency messages sent by patient or staff in case of emergency. This use case not only uses Handle_exigency use case for handling exceptions but also uses Update_Exigency use case for updating health related exigencies. Authentication use case is to provide security so that only authorized doctor and staff can access the system. Therefore in case of Authentication, activity diagrams for doctor and staff are same as patient activity diagram in Figure. 4.2(a). According to Figure.4.4(a) doctor has to first make a treatment which has to be validated. Doctor can save the treatment in case it is valid otherwise he/she has to go for an alternate



(a) Activity Diagram for Doctor Manage_Treatment Usecase (b) Activity Diagram for Doctor Handle_Exigencies Usecase

Figure 4.4: Activity Diagrams for Doctor

treatment. For an already existing treatment if a modification is needed, then doctor can modify the treatment. But if the ongoing treatment is not satisfactory, then the doctor can remove the treatment and can go for an alternate treatment. Similarly list of activities for handling exigencies are shown in activity diagram in Figure.4.4(b). Doctor has to first view the exigency messages sent by patients and take action for each and every message. Finally doctor has to update the health related exigencies in the database.

- **Staff Perspective**

  Services to be provided by the system to staff are as follows. System should first authenticate the staff as a user of the system. Staff has to do registration for patient by entering all the detailed information about the patient After registration, staff has to allocate user-id and passwd to the patient. Staff is responsible for recording symptoms as well as assigning doctor to patient during the first phase of Administration. He/she is also responsible

for updating patient's health status. Figure.4.5 has two use cases viz Au-

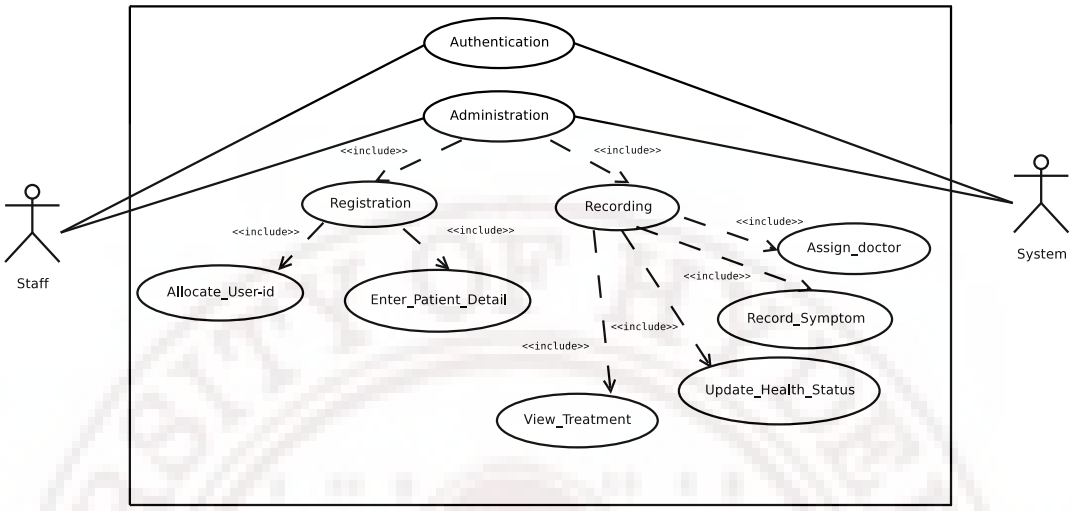
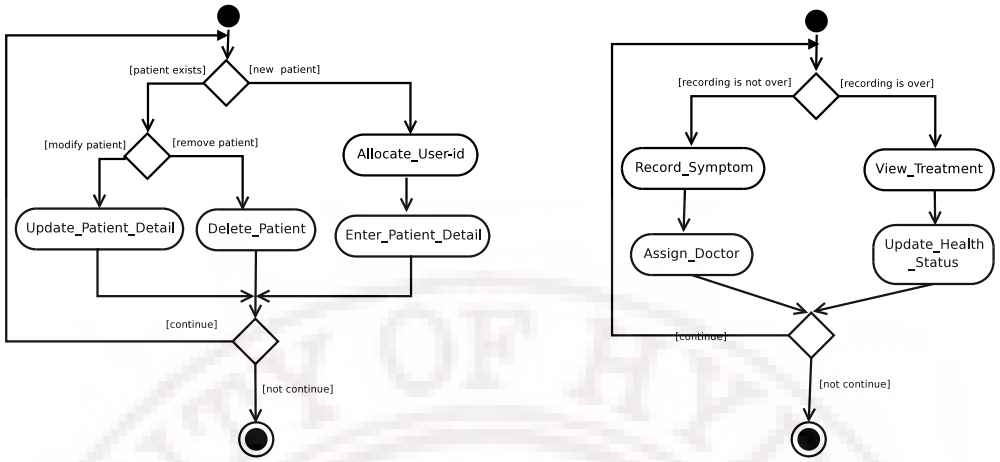


Figure 4.5: Use case Diagram for Staff

thentication, Administration. Authentication use case for staff is same as patient and doctor. Registration use case has two sub-use cases viz Allocate_User-id and Enter_Patient_Detail. When a patient arrives, staff has to allocate user-id for that patient. This is done with the use of Allocate_User-id use case. With Enter_Patient_Detail use case, staff can store all the detailed information about patient in database. If patient already exists, then if needed staff can modify patient information or can remove patient from the database. Recording use case has four sub-use cases viz Assign_Doctor, Record_Symptom, Update_Health_Status, View_Treatment. Therefore, in case of Recording use case, staff not only records symptoms but also assigns doctor to patient. With Update_Health_Status use case staff can update patient's health related status and with View_Treatment use case staff can view about patient's treatment. After entering as an authenticated user, staff has to start registration activities as given in activity diagram in Figure 4.6(a).On arrival of a new patient, staff has to create a user-id and enter all the detailed information about patient like name, address, phone



(a) Activity Diagram for Staff Registration Usecase (b) Activity Diagram for Staff Administration Usecase

Figure 4.6: Activity Diagrams for Staff

number, mail-id etc. But in case of existing patient, if there is a need to modify information about the patient, then staff can modify the information. Otherwise if wanted, staff can also remove the existing patient from the database. The list of activities for Recording use case are listed in activity diagram as shown in Figure. 4.6(b). If the patient is visited for the first time, then the staff records all symptoms of patients and assigns a doctor to the patient. But in case of already registered patient, if patient's feedback is to be included, then the staff can update patient's health status in the database.

4.4 System Architecture

System architecture and its components are discussed in this section. While considering different end users functionalities, system is basically divided into two parts:Treatflow_Libray_Make, which is prepared off-line and Treatflow_Consultation, which is considered to be on-line. System architecture for TFMS is shown in Figure 4.7. The system architecture comprises of modules which are represented as

follows.

A module is composed of sub-modules and these sub-modules are executed by

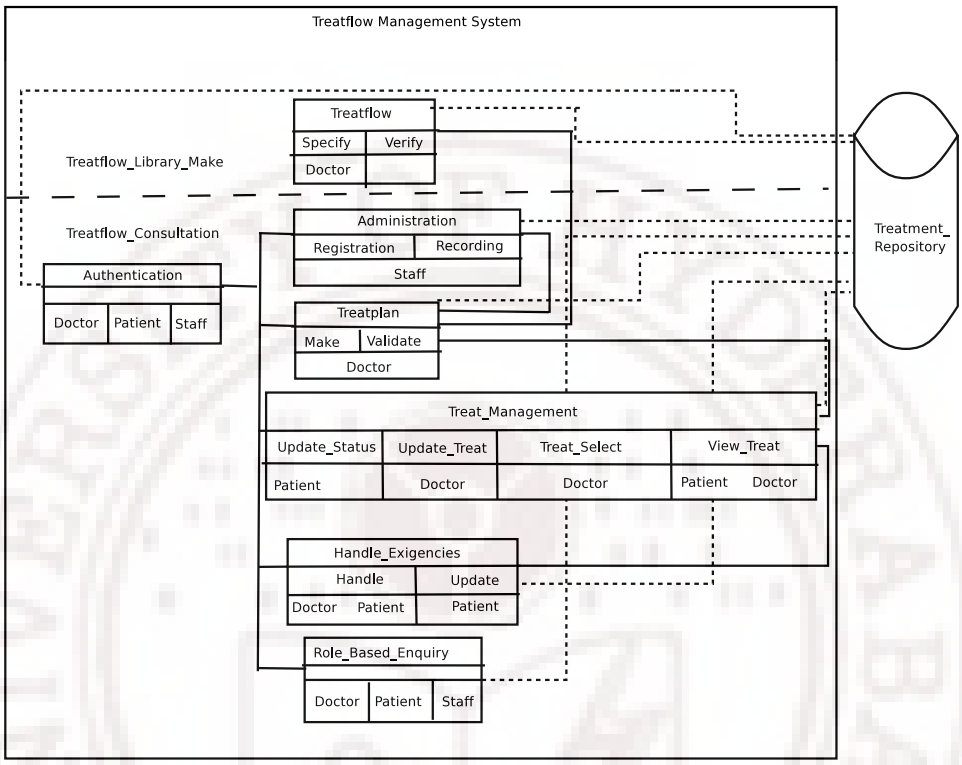


Figure 4.7: A Software System Architecture for Treatflow Management System

users. Each module of the system architecture is represented in a rectangular box. There are three layers in the box as shown in Figure.4.8. First layer shows module name, second layer represents names of sub-modules (if any) and whereas third layer shows the users of the module.

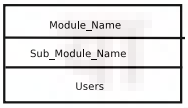


Figure 4.8: Representation of Module

- **Treatflow_Library_Make**

  As we have already discussed in the previous chapter, the order of execution of treatment activities is termed as Treatflow. Treatflow is specified

off-line by doctor and is stored in library for further use. Specification and verification of a Treatflow is implemented by the module Treatflow. Therefore, Treatflow_Library_Make a part of system architecture is responsible for specification and verification of Treatflow. It can be implemented by using Specify and Verify sub-modules.

- **Treaflow_Consultation**

  This part of system architecture is composed of modules namely Authentication, Administration, Treatplan, Treat-Management, Handle-Exigencies, Role-Based-Enquiry. They can be described as follows.

  - **Authentication**

    A user of any category namely doctor, patient or staff can access the services of the system by using his/her user-id and passwd. The functionality of Authentication module is to provide and maintain security and privacy for the Treatflow Management System (TFMS). A user can change his/her passwd using Authentication module. .

  - **Administration**

    The Administration module is used by staff for registration and for recording patient's health parameters. Therefore, this Administration module has two sub-modules namely Registration and Recording. With the help of Registration module staff can enter the detailed information of patient viz name,address, phone number etc. Similarly with the help of Recording sub-module, staff can record patient's symptoms and health parameters viz temperature, blood pressure etc. Staff is also responsible for assigning a doctor to the patient. The above administrative functionalities can be implemented by using the Administration module.

– **Treatplan**

The functionality of Treatplan module is to help a doctor in making a treatment plan for a patient depending on the symptoms. Treatment plan thus made can also be validated. This making up of Treatplan and validation of Treatplan can be implemented by Treatplan module.

– **Treat_Management**

With the help of Treat_Management module, a patient can view certain parts of treatment activities like drugs, dosage, effects, caution etc. In order to give feedback, patient has to update his/her current health status. Similarly doctor can also view the progress of treatment. Therefore as an end user, doctor can view the treatment details of patient and can modify / upgrade the treatment as well as patient health status. The Treat_Management provides doctor, staff and patient a facility to view treatment detail. Using Treat_Select sub-module, doctor can initiate a treatment process. And using Update_Treatment sub-module doctor can update existing treatment plan.

– **Handle_Exigencies**

During treatment, in cases of emergency like sideeffects or allergy which may erupt at any point of time, a patient has to inform the drug side-effects. Doctor has to take action for these type of exigencies and has to handle such exigencies which is possible with the help of Handle_Exigencies module. Also the flow of treatment may not proceed further in case Postc (post-condition) is not satisfied. For example, a patient has not taken medicine as per the treatment activity Take_Drug. In this type of exigency patient may get a message and he/she has to take action in order to handle exigencies.

– **Role_Based_Enquiry**

Authorized users like doctor,patient and staff who have special permission, can access allowable treatment related information for their purpose.

– **Treatment_Repository**

While looking for a repository in Treatflow Management System for storing treatment related activities and associated data well proved relational database technology RDBMS have chosen for this purpose. Information model is a fundamental concept to study in healthcare workflow repository. The information model determines how information is stored and retrieved from a repository and how well the integrity constraint can be maintained. Healthcare industry is very sensitive and health related information keeps changing and so does treatment procedures. Therefore there is a need to update information by extending the existing information model to accommodate the changes.

4.5 Module Design

We sketch a broad picture on design of each module with the help of class diagrams. While these class diagrams depict the structural view of the system architecture, the behavioral aspects of it are analyzed in the following section with the help of interaction diagrams. Design of each module of the system architecture is discussed below.

- **Treatflow**

  . Class diagram for Specifying a Treatflow has member functions namely Draw_tflow, Add_tflow, Modify_tflow and Delete_tflow as shown in Figure.4.9. A Treatflow (denoted as tflow) can be specified with the help of these operators. In order to start a Treatflow, Draw_tflow member function can

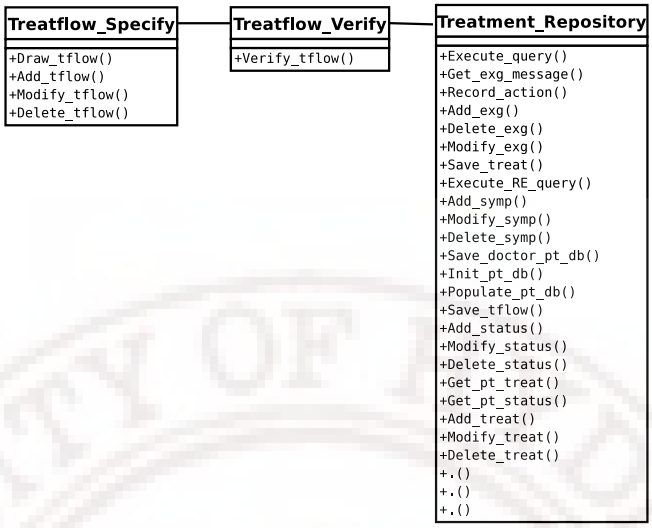


Figure 4.9: Class Diagram for Treatflow

be invoked. Similarly in order to connect treatment activities Add_tflow is used. After specifying a Treaflow it invokes another class Treatflow_Verify for verifying the Treatflow. With the help of Verify_tflow member function, the Treatflow can be verified. Once a Treatflow is correct, then by invoking Save_tflow as member function the Treatflow can be stored in Treatment_Repository.

- **Authentication**

  . As already shown in system architecture in Figure 4.7 In order to authenticate, users of any category namely doctor, patient and staff can access the system with their user-id and passwd. This is possible with the help of Ask_to_authenticate as the member function. This user-id and passwd must be validated in order to maintain the system secure which is done by the member function Execute_query available in Treatment_Repository. After validation a valid user can directly enter into the system and there is no need to inform to the user which can be done by the Hide function. Similarly in case of an invalid user, system has to display message which is possible by Show member function. Validate member function automatically invoked

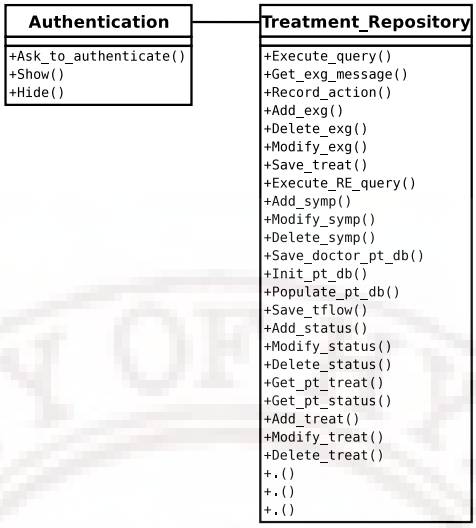


Figure 4.10: Class Diagram for Authentication

by the Ask_to_authenticate member function and therefore all these member functions are appeared in class Authentication as shown in Figure. 4.10.

- **Administration**

  . In Administration, staff can do registration and record the patient's health parameters with the help of registration and recording sub-module respectively as discussed in Figure.4.7. During registration, staff has to allocate user-id and enter detail information of patient. This can be implemented by invoking the member function Allocate_user-id and Enter_patient_detail respectively. This information has to be stored into Treatment_Repository. Staff has to record patient's symptoms, assign doctor to patient, for which member functions like Record_symptom and Assign_doctor can be implemented as shown in Figure.4.11.

- **Treatplan**

  . As we have already discussed, doctor has to prepare a treatment plan by invoking Create_treat member function. This treatment plan can be validated and stored in the treatment repository with the help of Validate_treat

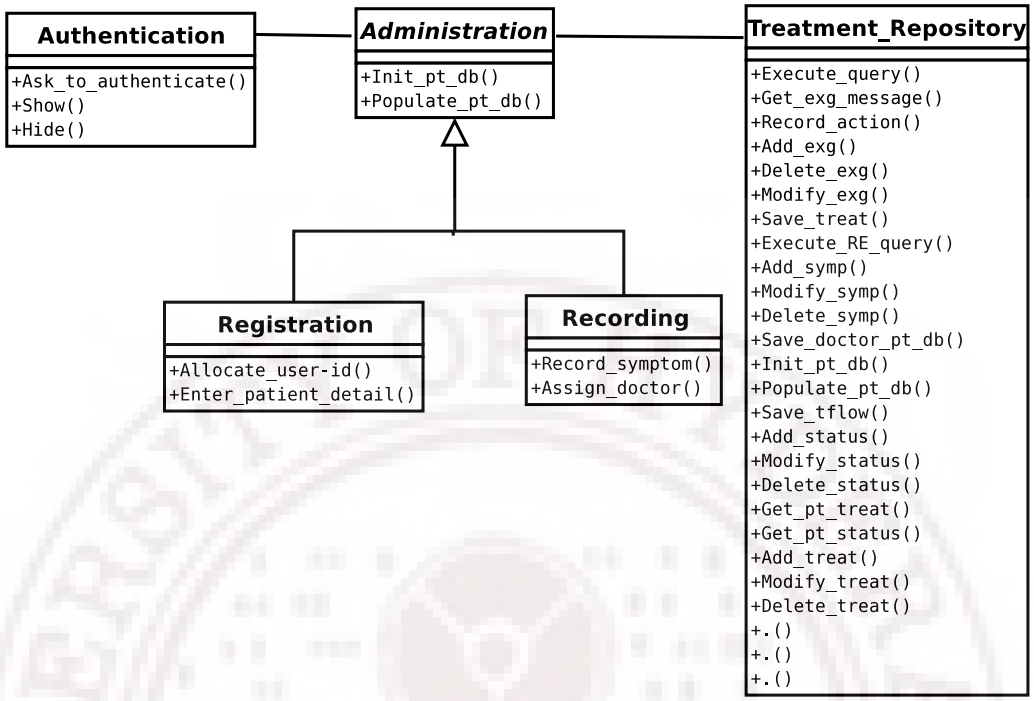


Figure 4.11: Class Diagram for Administration

and Save_treat respectively as shown in Figure.4.12.

- **Treat_Management**

  . Treat_Management module comprises of Update_Status, Update_Treat and View_Treat sub-modules as discussed in Figure.4.7. While managing treatment, doctor can update patient's health status by adding, changing or removing patient's health parameter by implementing Update_Status class. Before updating treatment, doctor has to view patient's treatment details and patient's health status. Update_Treatment class includes member functions like View_pt_treatment, View_pt_detail and Update_treatment as shown in Figure. 4.13. To support these functionalities we need Treatment_Repository.

- **Handle_Exigencies**

  . In case of emergency patient as well as doctor can update health related exigencies and take necessary actions (if required) to handle these exigencies.

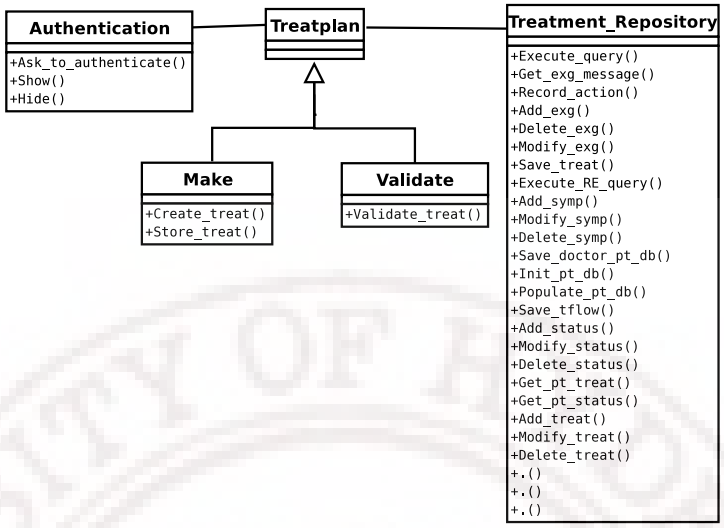


Figure 4.12: Class Diagram for Treatplan

The doctor and patient can view the messages related to exigencies and take action to handle those exigencies. So, in order to satisfy these functionalities, View_message and Take_action are the member functions that must be implemented. This is possible only with the help of Treatment_repository as shown in Figure.4.14..

- **Role_Based_Enquiry**

  . As an authenticated user, namely doctor,patient or staff can query about treatment related information and get the result based on their roles. For example, as discussed in previous chapter some treatment related attributes must be hidden from patient as well as staff. To achieve these functionalities, member functions like Take_query and Display_result must be implemented which is given in Figure. 4.15.

4.6 System Behavioral Analysis

Here we discuss the behavioral analysis of class diagram with respect to each requirement. In this case analysis is done with the help of interaction diagram.

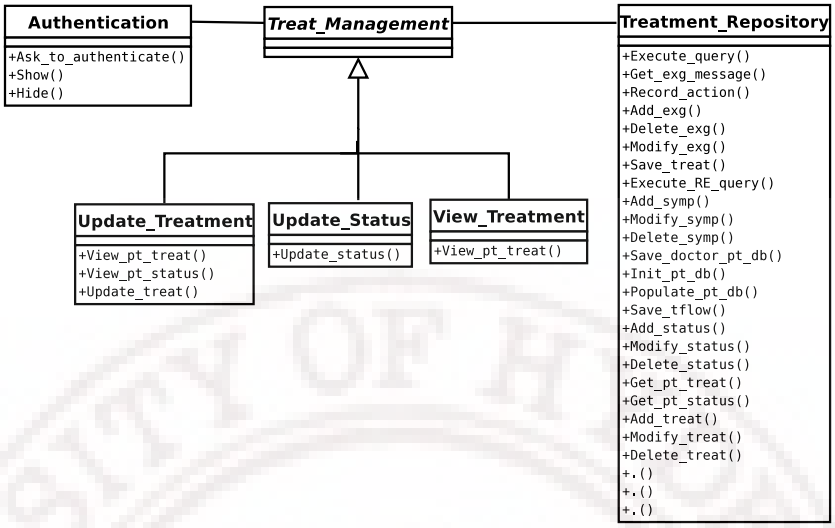


Figure 4.13: Class Diagram for Treat_Management

Based on each user requirements how the modules interact among themselves is discussed below.

- **Authentication** Considering patient,doctor and staff as users, authentication functionality provided by the system is shown in interaction diagram in Figure.4.16. As we have already discussed, to maintain the system secure, a user has to authenticate with his/her passwd. In this case, in order to authenticate, user has to enter user-id and passwd with the help of member function Ask_to_authenticate as shown in Figure 4.10. This user-id and passwd has to be validated by the member function Validate_treat which is declared in Validate class as shown in Figure 4.9. After executing query, it sends back the information regarding validity. In case of an invalid user a message is displayed with the help of Display function. But in case of a valid user it hides the message with the help of Hide function and directly enters into the system.

- **Handle_Exigencies:** In order to handle exigencies the way the doctor as well as patient interact with the system is given in Figure.4.17. During treat-

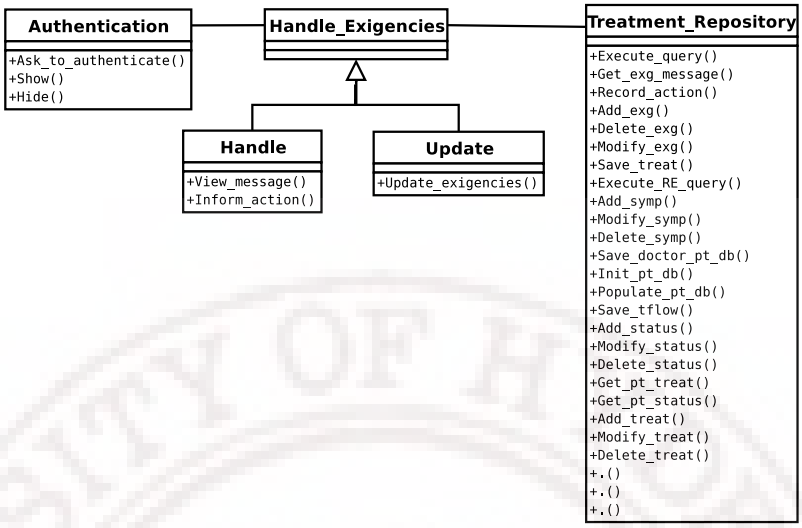


Figure 4.14: Class Diagram for Handle_Exigencies

ment, in case of exigency patient as well as doctor can update health related exigencies with the help of Update_exigencies member function which is given in Handle_Exigencies class in Figure. 4.14. While doing modification, user can select any of the option for example Add, Change or Remove which invoke Add_exg, Modify_exg or Delete_exg member functions declared in class Treatment_Repository as in Figure.4.14. In order to handle such exigencies doctor can view messages stored in repository. Once the message is accessed from the repository with the help of Get_message function, then doctor has to take action by invoking the member function Inform_action. This action has to be recorded in the repository with the help of Record_action. This process continues until there are no exigencies messages available in the repository.

- **Make_Treatment** . Based on symptoms, doctor has to prepare a treatment plan for patient. The treatment plan is prepared with the help of Create_treat member function which is to be validated by validate_treat function as declared in Figure. 4.12. After validation doctor can store the treatment plan in the Treatment_Repository using Store_treat function as

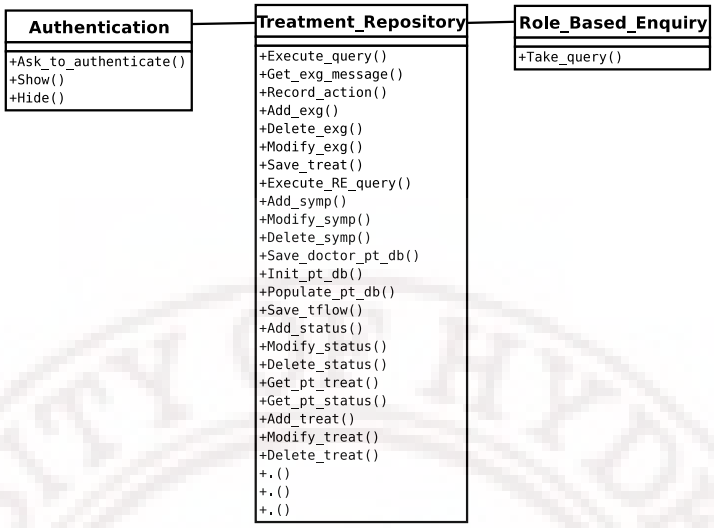


Figure 4.15: Class Diagram for Role_Based_Enquiry

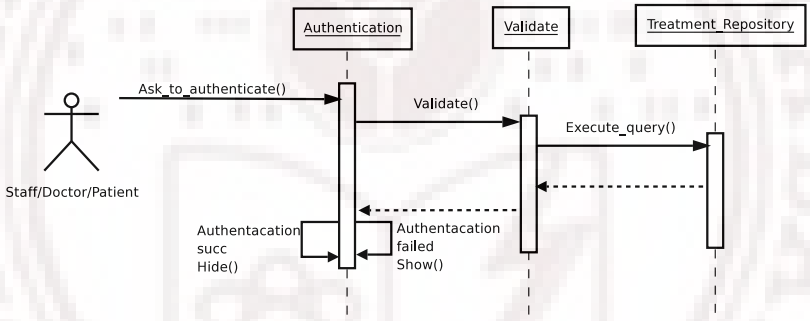


Figure 4.16: Interaction Diagram for Authentication requirement

given in Figure 4.18.

- **Role_Based_Enquiry** Staff, doctor and patient as users want to access some treatment related information stored in Treatment_Repository. Interaction diagram for role based enquiry is given in Figure. 4.19. Depending on their requirements, they can write a query which is implemented by Take_query member function declared in Role_Based _Enquiry class as in Figure 4.15. This query has to be executed by the member function Execute_RE_query available in Treatment_Repository class.

- **Recording** During patient's first visit staff has to record patient's symptoms into the repository. This can be implemented by using the Record_symp

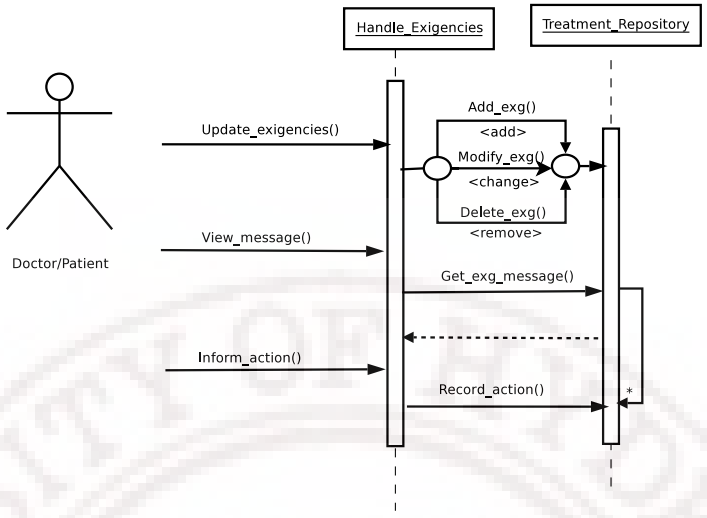


Figure 4.17: Interaction Diagram for Handle_Exigencies requirement

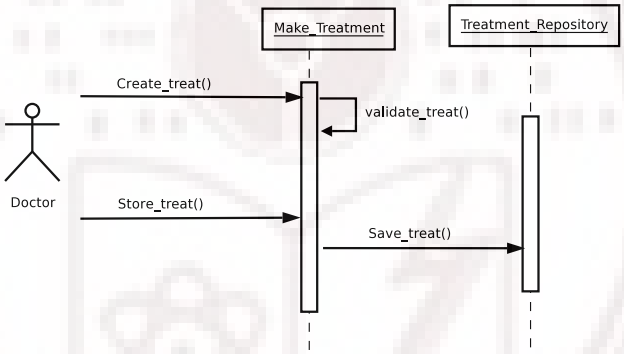


Figure 4.18: Interaction Diagram for Make_Treatment requirement

member function declared in Recording class given in Figure 4.20.While recording, in order to decide whether to add new symptoms or change or remove already existing symptoms, the functions like Add_symp, Modify_symp or Delete_symp must be implemented. After recording staff can assign doctor to patient based on his/her symptoms which are stored in repository with the help of member function Save_doctor_pt_db. This interaction of Recording_Symptom and Treatment_Repository is given in interaction diagram shown in Figure. 4.20.

- **Registration** When a patient enters into a hospital for treatment for the first time, staff must do registration for the patient. While doing regis-

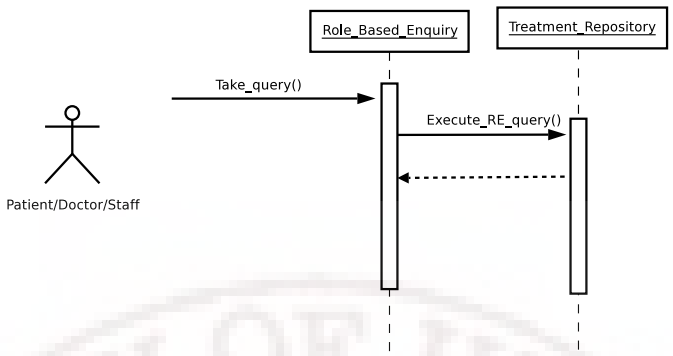


Figure 4.19: Interaction Diagram for Role Based Enquiry

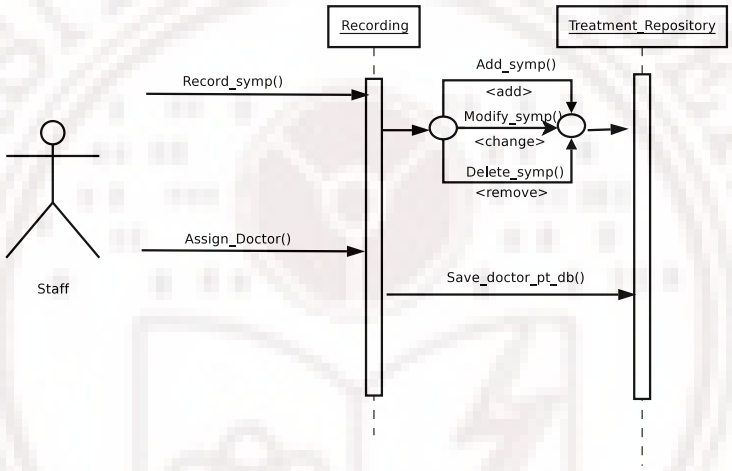


Figure 4.20: Interaction Diagram for Recording

tration he/she should allocate record for that patient by implementing the member function Allocate_user-id which is available in Registration class. This initiates a patient record in the database by implementing the function Init_pt_db in Treatment_Repository. The detail information of patient e.g name,address,phone no can be stored by implementing Enter_pt_detail member function. This helps in maintaining patient database with the help of Populate_pt_db function available in Treatment_Repository. During registration interaction of staff as a user is given in diagram Figure. 4.21.

- **Treatflow** For specifying a Treatflow, a doctor has to first draw a Treatflow using Treatflow primitives. This can be done by implementing Draw_tflow

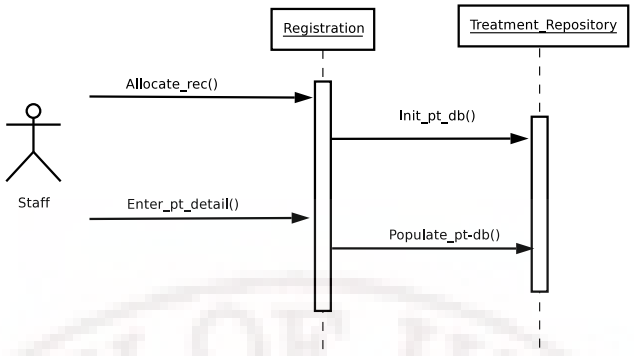


Figure 4.21: Interaction Diagram for Registration

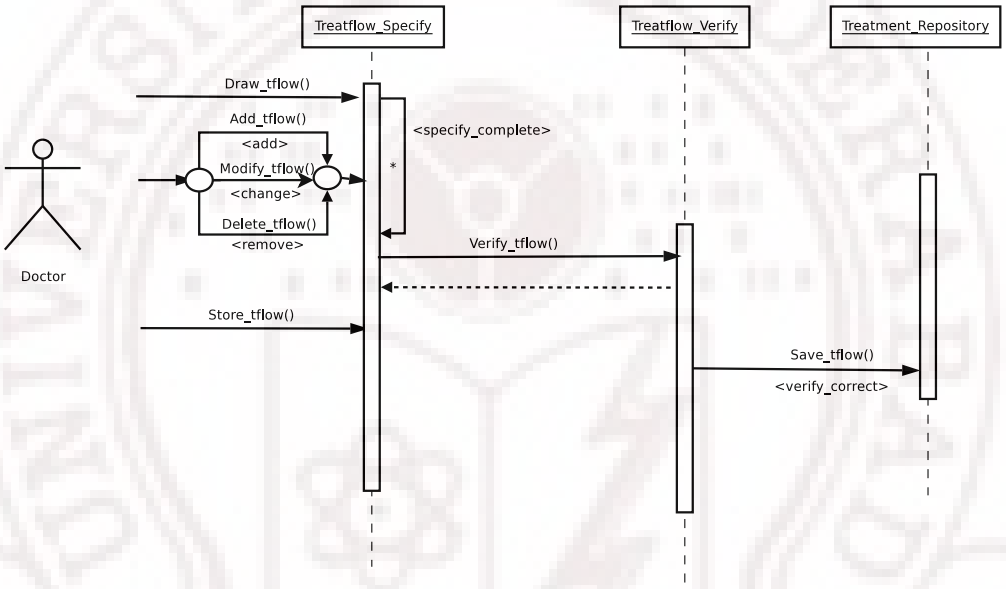


Figure 4.22: Interaction Diagram for Treatflow

member function available in Treatflow_Specify class as in Figure. 4.9. After drawing a Treatflow, doctor can add,modify or delete a Treatflow with the help of Add_tflow, Modify_tflow, Delete_tflow function respectively until the specification is over. Once a Treatflow is specified, that can be verified by implementing Verify_tflow member function available in Treatflow_Verify class. After confirmation about the correctness of Treatflow from Verify_Treatflow, doctor can send a message for storing it into the Treatment_Repository by implementing the function Store_tflow. Interaction between Treatflow_Specify, Treatflow_Verify and Treatment_Repository

is given in Figure. 4.22.

- **Update Status:** Users of any type can update the patient's health status

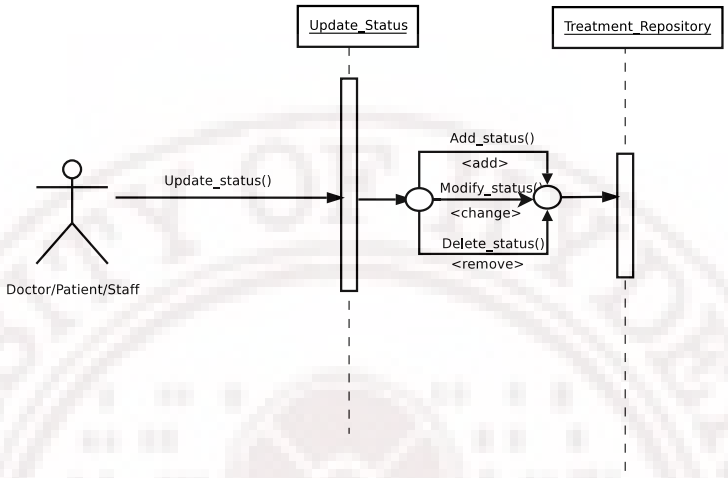


Figure 4.23: Interaction Diagram for Update Status

by invoking Update_status member function and for this type of updating status, the interaction diagram is given in Figure. 4.23. In this case patient,doctor or staff as a valid user can add, change or remove health status of patient by implementing the member function Add_status, Modify_status, Delete_status respectively which are available in Treatment_Repository class.

- **Update Treatment** Doctor as a user can update a treatment for a patient. Interaction diagram for updating treatment is given in Figure.4.24. Doctor has to first go through the details of patient's treatment. In order to view the treatment details it has to first invoke View_pt_treat function of Update_Treatment class Then to get information from the repository Get_pt_treat function must be invoked. After receiving information and before updating the treatment plan, doctor must view the current health status of the particular patient with the help of View_pt_status. After receiving information doctor can update the treatment with the function Update_treat. This helps in modification i,e either adding, changing or removing a treat-

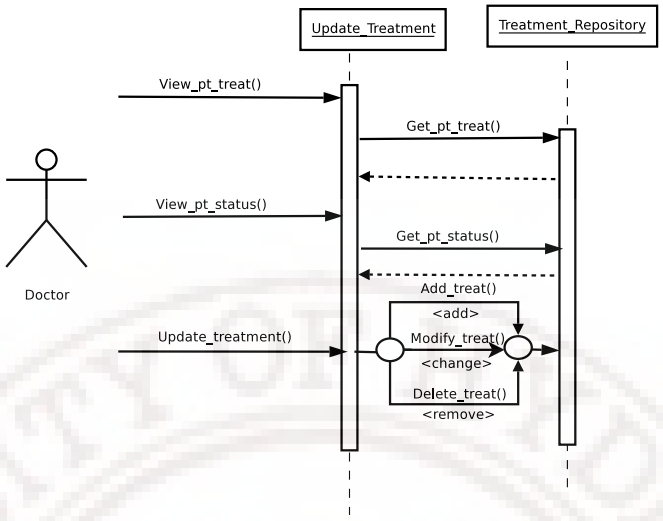


Figure 4.24: Interaction Diagram for Update Treatment

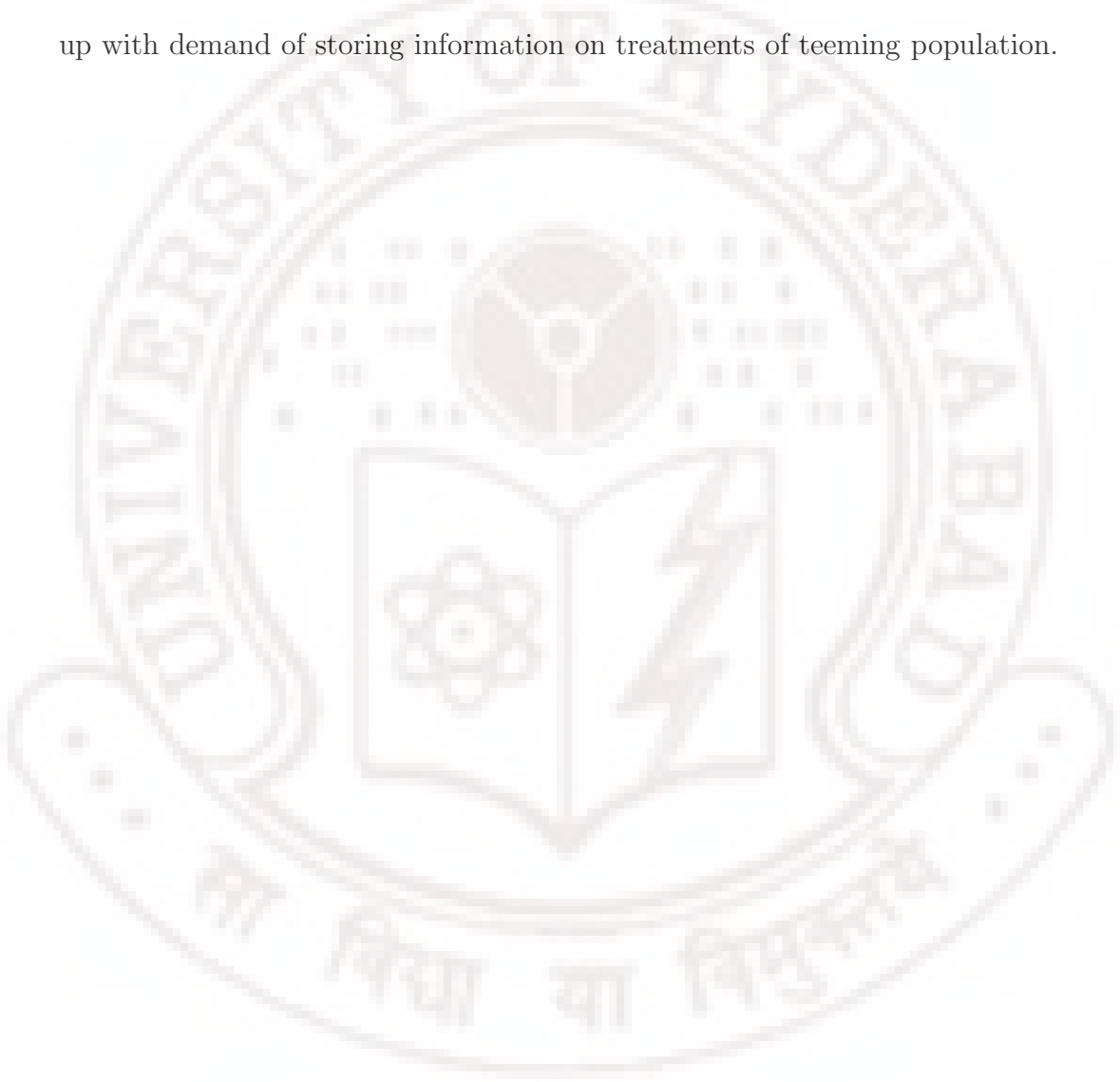
ment in the Treatment_Repository class using member functions namely Add_treat, Modify_treat or Delete_treat. In this way how different types of users interact among themselves as well as with the system is discussed in detail with the help of interaction diagrams.

4.7 Summary

Treatflow Management System aims at automating a treatment process which is an instantiation of a Treatflow. Here we proposed a software system architecture to implement Treatflow Management System. A requirement analysis for such a system considering users doctors, patients and staffs and their roles in treatment process has been discussed. The analysis is presented as Use case and Activity diagrams. These diagrams lead to a first-cut of a system architecture and the software modules that the system should have.Then we provide higher level design diagrams for each module. In order to be brief and at the same time to bring clarity on design decisions, we have made use of Class and Interaction diagrams.

Healthcare workflow is to be flexible for modification or even replace-

ments. Usually such a workflow has to have long life as a treatment happens in reality. Again, treatment for each patient is unique, so instances of ongoing treatments are to be properly stored so that during treatments these can be accessed readily. Treatment associated information about patient,doctor and healthcare staff are to be stored. As discussed, all these demand a scalable repository to cope up with demand of storing information on treatments of teeming population.



Chapter 5

Conclusions & Future Work

In this chapter we summarize the work presented in this thesis and then conclude with a discussion of related research issues that remain open for future work. Particularly for developing countries delivering healthcare services has been difficult because poorly defined treatment process is in practice. Firstly, the process does not take all the participants (patient, medicos, non-medicos, managerial staffs and resources) into consideration. Secondly, the process of treatment is never transparent to patient and so patient often develop distrust on treatment process. This is a general feeling in India where healthcare services are provided by both public and private agencies. Thirdly, management of resources for delivering healthcare services is often inadequate in developing countries because of under specification of practicing treatment process. Fourthly, the process is ill defined to handle exigency cases. Finally the management of treatment process instances becomes an arduous task particularly for populous countries with poor healthcare record. We believe, solutions to some of these problems can be met by following a well defined treatment process and making it accessible to all the participants with the help of information technologies.

Healthcare has been a challenge for each country with aging and growing population and to meet the challenge there have been endeavors by both government as well as private enterprises. The efforts have given rise to healthcare

industry with hospital of different categories ranging from small rural healthcare center to large corporate hospitals in city centers. Currently, information technology has come to an aid to manage mostly several administrative, personal and financial activities. But, there is also a need to make use of this technology to automate treatment process particularly for patients suffering from multiple ailment or chronic diseases. In this thesis there is an attempt to automate treatment considering it as a process with well defined set of activities.

In this work we coin a concept called Treatplan that models a modular treatment at higher level and in the next level this module is detailed. Based on our specification we have defined a framework on TFMS (Treatflow Management System) to manage several Treatplan instantiation.

5.1 Summary of the Work Done

Traditionally structured method is being used for system analysis and design. The reason for its popularity is due to the pictorial presentation of a system. However, the method lacks mathematical preciseness that can be used to prove the system mathematically. Formal method in software development helps in proving the system property mathematically. The proposed work used primitives viz, *linear*, *split and merge*, *parallel*, *supportive and cooperative* and *choice* that are basic to represent a workflow. The uses of such primitives are shown in specifying workflow of a corporate hospital for treating patients. We use *hybrid approach* in the sense for, each type of primitive other than pictorial representation, system behavior is also specified mathematically. Execution of each primitive is analyzed and shown that it is possible to make unambiguous inferences on execution of each workflow primitive. The analysis is implementable as it uses assertions like Pre and Post-conditions. These assertions are used for verification and validation on workflow

primitives pertaining to an application domain.

Once a Treatflow is specified, that has to be verified in order to avoid mistakes done by non-computer science professionals. We discuss various verification issues with the help of a Treatflow verification algorithm. We present Treatflow as a graph comprising of nodes and edges and the Treatflow verification algorithm with procedures *Create-Instance-Graph*, *Verify-Instance-Graph* and *Create-Next-Instance* traverses the graph for the purpose of verification. We categorize the verification issues into three different types namely *Structural*, *Behavioral* and *Temporal*. *Incompleteness*, *Lack of Synchronization*, *Deadlock* issues are considered within structural disorder whereas *Contextual Conflict*, *Retention Conflict* and *Expectation Conflict* are studied as behavioral disorder and *Delay Conflict* are discussed as verification issues in case of temporal disorder.

In order to handle large and complex treatment in healthcare, chapter 3 advocates the uses of the modularization concept in healthcare domain especially in planning of treatments. We have proposed a scheme to specify a treatment module and operators to compose a treatment. It is shown that a treatment prescription can be written as an expression of modules. Sometimes it may be necessary to explore all the possibilities in executing a treatment plan, therefore we propose a method for rewriting a given expression by applying rewriting rules. We demonstrate the applicability and suitability of re-writing rules with the help of an example taken from healthcare domain. A Treatflow meant for a particular ailment can be applied to different patients suffering for the same ailment. Still, a treatment while for a patient proceeds smoothly but for another patient with the same ailment may not go well for patient specific reasons. Patient related exceptions can be either *known* or *unknown* type. In chapter 3 we also focus on exceptional behavior of healthcare workflows. We provide a comprehensive analysis on generation of such exceptions at the deviating behavior of a specified

workflow. Considering the genesis of exceptions, we categorize them into *Resource*, *Context & Goal*, *Safety* and *Temporal* perspective. In addition to these we add two more types called *DefaultExcp* and *PatientExcp* which are designed to deal with unknown situations. We discuss about processing of exception as well as define actions that can be performed with respect to exceptions. The feasibility of this concept is shown in a case study taken from healthcare domain.

In order to automate a treatment process, we propose a system architecture in chapter 4. The system architecture consists of two parts namely *Treatflow-Library-Make* and *Treaflow-Consultation*. In Treatflow-Library-Make specification of Treatflow is implemented by using *Specify* sub-module and verification of Treatflow is implemented by using *Verify* sub-module. Whereas Treaflow-Consultation composed of modules namely *Authentication*, *Administration*, *Treatplan*, *Treat-Management*, *Handle-Exigencies* and *Role-Based-Inquiry*which we discuss in detail with the help of use case diagram, activity diagram, class diagram and interaction diagram.

5.2 Future Work

In this section we point out open issues and suggestion promising directions for future work. Much work remains to be done to improve what we have achieved so far and to explore new solutions. Treatflow Management System (TFMS) can be implemented with a variety of additional requirements. We present here some plans for future works that are well out of the scope of this thesis but can be followed up for further research.

- **Implementation of TFMS:** To study the impact of proposed system in a hospital, the performance of a healthcare workflow management system used to analyze in a real life situation to study the efficiency like adequate

response time and managing a large number of Treatflow instances which is in fact a stark reality. The increase in number of instantiation may badly degrade system response time. For improvement in the system, the design and implementation issues may require a re look.

- **3A(Adhoc,Adapt,Adopt) Treatment:** On imitating the way a treatment takes place in reality, we can divide a span of treatment into 3 phases. namely *Adhoc*, *Adapt* and *Adopt*. In the initial stage i,e Adhoc a patient is asked to take initial treatment and then the suitability of patient is studied and the treatment may possibly be refined to suit the patient i,e Adapt phase. And finally if a treatment is found progressive, then it can be Adopted for continuation. In order to implement 3A phase in treatment process, the execution behavior of Treatflow model needs to be properly specified so that phased transition can be monitored, analyzed and traced.

Bibliography

[1] Satyajit Acharya, Chris George, and Hrushikesha Mohanty. Domain consistency in requirements specification. In *QSIC 2005*, pages 231 – 240. IEEE Computer Society, 2005.

[2] J.Rajapakse A.H.Mer Hofstede, M.E. Orlowska. Verification problem in conceptual workflow specifications. *Data and Knowledge Engineering*, 362:239–256, 1998.

[3] Ardissono et al. Adaptive medical workflow management for a context-dependent home healthcare assistance service. *Electronic Notes in Theoretical Computer Science,Elsevier*, 146(1):59–68, 2006.

[4] P. Attie, M. Singh, A. Sheth, and M. Rusinkiewicz. Specifying and enforcing intertask dependencies. In *Proceedings of the 19th VLDB Conference*, pages 134 – 145. Morgan Kaufmann Publishers, 1993.

[5] Joon-Soo Bae and Seok-Chan Jeong. Integration of workflow management and simulation. In *Computers and Industrial Engineering*, volume 37, pages 203 – 206. Pergamon Press, 1999.

[6] K. Barkaoui, Ph. Dechambre, and R.Hachicha. Verification and optimization of operating room workflow. In *35th Hawaii International Conference on System Sciences IEEE*, pages 2581– 2590, 2002.

[7] P. Barthelmess and J. Wainer. Workflow systems- a few definitions and a few suggesstions. In *Organizational Computing Systems (COOCS'95)*, pages 138–147. ACM press, 1995.

[8] Michel Beaudouin-Lafon. Workflow technology. In Michel Beaudouin-Lafon, editor, *Computer Supported Cooperative Work*, volume 7 of *Trends in Software*, pages 29–54. John Wiley & Sons, 1999.

[9] Boualem Benatallah et al. Hiword: A petri net based hierarchical workflow designer. In *Proceedings of the third International Conference on Application of Concurrency to System Design (ACSD'03)*, pages 235– 236. IEEE Computer Society, 2003.

[10] M. Benyoucef and R. K. Keller. A conceptual architecture for a combined negotiation suppott system. In *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, pages 1015–1019. IEEE Computer Society, 2000.

[11] A. J. Bonner and M. Kifer. A logic for programming database transactions. In *In Logics for Databases and Information Systems*, chapter 5, pages 117–166. Kluwer Academic Publishers, 1998.

[12] M Broy. Toward a mathematical foundation of software engineering methods. *IEEE Transaction on Software Engineering*, 27(1):42–57, January 2001.

[13] Calinescu, R. Harris, S.Gibbons, J. Davies, J. Toujilov, I. Nagl, and S.B. Model-driven architecture for cancer research. In *SEFM 2007. Fifth IEEE International Conference on Software Engineering and Formal Methods*, pages 59–68, 2007.

[14] Chia-Chu Chiang. Structured design with mathematical proofs. *Information and Software Technology*, 39(10):669–677, 1997.

[15] Dickson K.W. Chiu et al. Alerts for healthcare process and data integration. In *Proceedings of 37th Hawaii International Conferences on System Sciences*, 2004.

[16] Carlos Renato Lisboa Francs. Edvar da Luz Oliveira et al. Performance evaluation based on system modeling using state charts extensions. *Simulation Modeling Practice and Theory*, 13:584–618, 2005.

[17] J. Dallien, W. MacCaull, and A. Tien. Dynamic workflow verification for health care. *Symposium on Formal Methods*, 2006.

[18] Eric D.Browne, Michael Schrefi, and James R. Warren. Goal-focused self-modifying workflow in the healthcare domain. In *Proceedings of 37th Hawaii International Conference on System Science*, page 60145b. IEEE Computer Society, 2004.

[19] W.M.P Van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management.

[20] W.M.P Van der Aalst. Formalization and verification of event -driven process chains. *Information and Software Technology*, 41:639–650, 1999.

[21] W.M.P.van der Aalst, S. Navathe, and T. Wakayama. Three good reasons for using a petri-net-based workflow management system. In *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201, Camebridge, Massachusetts, Nov 1996.

[22] F.Leymann D.Roller. Production workflow concepts and techniques, 2000.

[23] M. Dumas and A. ter Hofstede. Uml activity diagrams as a workflow specification language. In Springer, editor, *Proceedings of the 4th International Conference on the Unified Modeling Language (UML): Modeling Languages Concepts, and Tools*, volume 2185, pages 76–90, Toronto, Canada, October 2001.

[24] Marlon Dumas and Arthur HM ter Hofstede. Uml activity diagram :as a workflow specification language. In *Proceeding of the International Conference on the Unified Modeling Language (UML)*. Springer Verlag, Toronto, Canada,, October 2001.

[25] Steve Easterbook, Robyn Lutz, Richard Covington, John Kelly, Yoko Ampo, and David Hamilton. Experiences using lightweight formal methods for requirements modeling. *IEEE Transaction on.Software Engineering*, 24(1):1–11, 1998.

[26] R. Eshuis and R. Wieringa. Comparing petri net and activity diagram variants for workflow modelling– a quest for reactive petri nets. In *Petri Net Technology for Communication Based Systems, Lecture Notes in Computer Science*. Springer, 2002.

[27] Mohan Kamath et al. Correctness issues in workflow management. *Distributed. System Engg*, 3:213–221, 1996.

[28] Giuseppe Pozzi Fabio Casati. Modeling exception behaviour in commercial wokflow. In *Proceedings of the Fourth IECIS International Conference on Cooperative Information Systems*, page 127. IEEE Computer Society, 1999.

[29] Stefano Ceri Fabio casati. Specification and Implementation of Exception in Workflow Management System. *ACM Transaction on Database System*, 24(3):405–451, September 1999.

[30] C.W. George. The ndb database specified in the raise specification language. *In Formal Aspects of Computer Science*, 4(1), 1992.

[31] Elena Gospodarevskara, Leonid Churilor, and Lyn Wallace. Modelling the patient care process of a acute care ward in a public hospital : A methodological perspective. In *Proceeding of the 38th Hawaii International Conference on System Sciences IEEE*, page 145, 2005.

[32] Patrizia Grifoni et al. Aterus: A model for conceptual representation of a workflow. In *Proceedings of 8th International Workshop on Database and Expert System Application*, page 400. IEEE Computer Society, 1997.

[33] Claus Hagen. Exception hndling in workflow management system. *IEEE Transaction on Software Engineering*, 26(10), October 2000.

[34] Minmin Han, Thomas Thiery, and Xiping Song. Managing exceptions in the medical workflow systems. In *Proceeding of the 28th International Conference on Software Engineering*, pages 741 – 750. ACM Press, 2006.

[35] David hollingsworth. *Workflow Management Coalition.* Workflow Reference Model, 1995.

[36] Ling hong and Zhou iiangbo. Research on workflow process structure verification. In *International Conference on E-business Engineering*, 2005.

[37] Healthcare workflow from wikipedia, the free encyclopedia.

[38] S. Jablonski. A software architecture for workflow management systems. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, page 739. IEEE Computer Society, 1998.

[39] Cai Jian, Peking Universiv, Beijing Zhao Wen, Peking Universiv, Beijing Zhang Shikun, Peking Universiv, Beijing Wang Lifu, Peking Universiv, and Beijing. Correctness verification of synchronization based workflow model. In *IEEE International Conference on e-Business Engineering (ICEBE'05)*, pages 527–530, 2005.

[40] Mike Wright Khodakaram Salimfard. A petri net based modelling of worklfow systems:an overview. *European journal of operational Research*, 134:664–676, 2001.

[41] Eleanna Kofeza and Komalakar Karlapalem. A framework for speeding up workflow instances by exploiting alternate paths. In *Proceedings of 3rd international workshop Advanced issues of E-commerce and web based information system*, page 136. EEE Computer Society, 2001.

[42] Pinar Koksal, Ibrahim Cingil, and Asuman Dogac. A component-based workflow system with dynamic modifications. In *Next Generation Information Technologies and Systems*, pages 238–255, 1999.

[43] L.Ardissono et al. Adaptive medical workflow management for a context-dependent home healthcare assistance service. *Electronic Note in Theoretical Computer Science*, 146:59–68, 2006.

[44] R. Lederman and I. Morrison. Examining quality of carehow poor information flow can impact on hospital workflow and affect patient outcomes. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, volume 6, page 142. IEEE Computer Society, 2002'.

[45] Dr Hongchen Li. Discussions on workflow verification itee seminar. Seminar Discussion, Nov 2003.

[46] Dongsheng Liu, Jianmin Wang, Stephen CF Chan, Jiaguang Sun, and Li Zhang. Modelling workflow with colored petri nets. In *Computer in industry*, volume 49, pages 267–281. Elsevier Science Publishers, 2002.

[47] D. Luzi, F. I. Ricci, and L. D. Serbanati. Integrating clinical and managerial activities in healthcare units. In *Proceedings of the 10th IEEE Symposium on Computer-Based Medical Systems (CBMS '97)*, page 250. IEEE Computer Society, 1997.

[48] Pradhan M, Edmonds. M, and Runciman W. Sequence diagrams for visualising healthcare processes. *Quality in Healthcare: Process. Best Practice and Research Clinical Anaesthesiology*, 15(4):555 – 571, 2001.

[49] Maggiolo-Schettini, Peron A., and S Tini. A comparison of state chart step semantics. *Theoretical computer science*, 290(1):465–498, 2003.

[50] Ana Snchez Marisa Navarro, Fernando Orejas. On the correctness of modular systems. *Theoretical Computer Science*, 140(1):139–177, March 1995.

[51] Marta Simeoni Martin Groe-Rhode, Francesco Parisi Presicce. Formal software specification with refinements and modules of typed graph transformation systems. *Journal of Computer and System Sciences*, 64(2):171–218, 2002.

[52] Marco Masserolia. He@lthco-op: a web-based system to support distributed healthcare co-operative work. *Computer in Biology and Medicine*, 36:109–127, 2006.

[53] Hemant Kr. Meena et al. An approach to workflow modelling and analysis. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange*, pages 85 – 89. ACM, IIT Kanpur , India, 2005.

[54] G. De Michelis, C. Ellis, and G. Memmi. Modelling workflow management systems with high-level petri nets. In *Proceedings of the second Workshop on Computer-Supported Cooperative Work*, pages 31–50, 1994.

[55] Hrushikesha Mohanty, Ratan k. Ghosh, and Sumagna Patnaik. A hybrid approach to model workflow in business process. In *Proceedings of sixth International Conference on Information and Technology*, pages 91–96. Tata McGrawHill, 2003.

[56] Hrushikesha Mohanty, Jitesh Mulchandani, Deepak Chenthati, and R.K Shyamasundar. Modeling web services with fsm modules. In *Asia International Conference on Modelling and Simulation*, pages 100–105. IEEE Computer Society, 2007.

[57] Wai Yin Mok and David Paper. Using harel's statecharts to model business workflows. *Journal of Database Management*, 13:17–34, july-sept 2002.

[58] R. Mller and E. Rahm. Rule-based dynamic modification of workflows in a medical domain. In A.P. (ed.) Preprint of: Buchmann, editor, *Proceedings of BTW99*, pages 429–448. Spinger, Berlin, 1999.

[59] Edelweiss. N and Nicolao. M. Workflow modeling: Exception and failure handling representation. In *XVIII International Conference of the Chilean Society SCCC '98*, pages 58–67, Nov 1998.

[60] Wil van der Aalst Nick Russell and Arthur ter Hofstede. Workflow exception pattern. In *Lecture Notes on Computer Science*, pages 288–302. Springer-Verlag, 2006.

[61] Eyal Oren and Armin Haller. Formal frameworks for workflow modelling. Technical report, Digital Enterprise Research Institute (DERI). Galway, Ireland., 2005-04-07.

[62] Wolfgang Ortner and Chris Stary. Virtualization of organizations: Consequences for workflow modeling. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, volume 5, page 5049. IEEE Computer Society, 1999.

[63] Sumagna Patnaik. Primitives for structured workflow design: A mathematical specification and analysis. In *Proceeding of ninth International Conference on Information and Technology*, pages 299–300. IEEE Computer Society, 2006.

[64] Sumagna Patnaik. An information model for healthcare workflow management. In *Proceeding of International Conference on Managing Next Generation Software Application*, pages 871–879, December 2008.

[65] Sumagna Patnaik and Hrushikesha Mohanty. On specification of treatment modules. In *Proceeding of the 11th International Conference on Information and Technology*, pages 215–220. IEEE Computer Society, December 2008.

[66] Peter Fettke Peter Loos. Towards an integration of business process modelling and object-oriented software development. In *The Proceedings of the Fifth International Symposium on Economic Informatics*, Bucharest, Bucharest, 2001.

[67] David Lorge Prarnas. Predicate logic for software engineering. *IEEE Transaction on Software Engineering*, 19(9):556–862, 1993.

[68] Proceedings of the 16th international workshop on Database and Expert System application (DEXA 05). *Adaptive Workfloe Management in WorhSCo.* IEEE, August 2005.

[69] Martin K. Purvis, Maryam Purvis, and Selena Lemalu. A framework for distributed workflow systems. In *HICSS*, 2001.

[70] A. Ruiz R. Bastos, D. Dubugras. Extending uml activity diagram for workflow modelling in production system. In *Proceedings of 35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, volume 9, page 291, 2002.

[71] A Ruiz Ricardo M. Bastos, Duncan Dubugras. Towards an approach to model business process using workflow modeling technique in production system. In *Proceedings of 34th Hawaii International Conferences on System Sciences IEEE*, 2001.

[72] A. W. H. Ip Richard Y. K. Fung, Alan Y. M. Au. Petri net-based workflow management systems for in-process control in a plastic processing plant. *Journal of Materials Processing Technology*, 139(1):302–309, August 2003.

[73] P. Rittgen. Paving the road to business process automation. In Martin Bichler and Harald Mahrer, editors, *European Conference on Information Systems (ECIS)*, volume 1, pages 313–319. Vienna, 2000.

[74] Roberto and W.S Rodrigues. Formalising uml activity diagram using finite state processes. Technical Report 0006, Ludwig-Maximilians-Universitt, Mnchen, Inst. f. Informatik, 2000.

[75] E.E Roubtswa et al. Specification of real time system in uml. *Electronic Notes in Theoretical Computer Science*, 39(3), 2000.

[76] Sea Ling; Schmidt. Time petri nets for workflow modelling and analysis. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC'2000)*, volume 4, pages 3039 – 3044, Nashville, TN,, Oct 2000.

[77] R Siebert. An open architecture for adaptive workflow management systems. *Journal of Integrated Design and Process Science*, 3(3):29–41, 1999.

[78] Harald Storrle. Semantics and verification of data flow in uml 2.0 activities. *Electronic Notes in Theoretical Computer Science*, 127:35–52, 2005.

[79] Lynda A. Thomas. The implication of different learning styles on the modeling of object oriented system. In *Proceedings of the 22nd International Conference on Software Engineering*, page 774. ACM, 2000.

[80] W. van der Aalst. Loosely coupled inter organizational workflows : Modeling and analyzing workflow crossing organizational boundaries. *Information and Management*, 37:67–75, 2000.

[81] W.M.P. van der Aalst. Putting petri nets to work in industry. In *Computers in Industry*, volume 25, pages 45–54. Elsevier Science Publishers B. V., 1994.

[82] W.M.P. van der Aalst. A class of petri net for modeling and analyzing business processes. Technical report, Eindhoven University of Technology Eindhoven, 1995.

[83] W.M.P. van der Aalst. Verification of workflow nets. In P. Azma and G. Balbo, editors, *Application and Theory of Petri Nets*, volume 1248, pages 407–426. Springer-Verlag, Berlin, 1997.

[84] W.M.P. van der Aalst. Workflow verification: Finding control-flow errors using petri-net-based techniques. In *In Business Process Management: Models, Techniques, and Empirical Studies*, volume 1806, pages 161–183. Springer-Verlag, Berlin, 2000.

[85] W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of epcs: A vicious circle. In M. Nttgens and F.J. Rump, editors, *Proceedings of the EPK 2002 Business Process Management using EPCs*, pages 71–80, Trier, Germany, Nov 2002.

[86] W.M.P. van der Aalst, A. Hirnschall, and H.M.W. Verbeek. An alternative way to analyze workflow graphs. In A. Banks Pidduck, J. Mylopoulos, C.C. Woo, and M.T. Ozsu, editors, *Proceedings of the 14th International Conference on Advanced Information Systems Engineering (CAiSE'02)*, volume 2348, pages 535–552, Berlin, 2002. Springer-Verlag.

[87] W.M.P. van der Aalst and A.H.M. ter Hofstede. Verification of workflow task structures: A petri-net-based approach. In *Information Systems*, volume 25, pages 43–69. Elsevier Science Ltd., 2000.

[88] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. In *Distributed and Parallel Databases*, volume 14, pages 5–51(47). Springer, 2003.

[89] W.M.P. van der Aalst and K.M. van Hee. Framework for business process redesign. In J.R. Callahan, editor, *Proceedings of the Fourth Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WETICE 95)*, pages 36–45. Berkeley Springs, April 1995.

[90] W.M.P. van der Aalst and K.M. van Hee. Business process redesign - a petrinet based approach. *Computers in Industry*, 29:(1–2), 1996.

[91] B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek. Verification of epcs using reduction rules and petrinets. In O. Pastor and J. Falcao e Cunha, editors, *Proceedings of the 17th Conference on Advanced Information Systems Engineering (CAiSE'05)*, volume 3520, pages 372–386, Berlin, 2005. Springer-Verlag.

[92] Andreas Wombacher, Peter Fankhauser, and Bendick Mahleko. Match making for business processes based on choreographies. In *Proceedings of the 2004 IEEE international Conference on e-Technology, e-commerce and e-service (EEE'04)*, 2004.

[93] Peter Y. H. Wong and Jeremy Gibbons. A process-algebraic approach to workflow specification and refinement. In *Software Composition*, pages 51–65, 2007.

[94] Peng Xu and Balasubramaniam Ramesh. Supporting workflow management system with traceability. In *Proceeding of the 35th Hawaii International Conferences on System Sciences*, pages 91–101. IEEE Computer Society, 2002.

[95] A. Seth Y. Han and C. Bussler. A taxonomy of adaptive workflow management. In *Towards Adaptive Workflow Systems*. ACM CSCW 98 workshop proceedings, 1998.