Requirements modelling of real-time systems

Thesis

UNRESTRICTED

Requirements Modelling of Real-Time Systems

Tiptur K Sateesh B.Eng.

A thesis submitted

for the degree of

Doctor of Philosophy

The Open University

Department of Computing

September 1995

Author number: M7123518
Date of submission: 4 May 1995
Date of award: 5 September 1995

To my parents,

Krishnamurthy and Chayalakshmi

Abstract

Real-time systems are characterised by the critical nature of their missions, and the demanding environment with which they interact. Real-time systems are used for dedicated applications. Every application is the subject of special requirements enforced by the customer. Considering the vital role that these systems play, it is imperative that a systematic approach be adopted in modelling their unique requirements. In this thesis I propose such a treatment.

Real-time systems are time critical. Temporal requirements are the timing restrictions imposed by the application environment. Previous studies in requirements modelling of real-time systems have focused on adding the notion of time to modelling techniques of traditional systems without regard to the realities of requirements modelling. The information should be presented in the way the user handles it, and not the way which is convenient to the software engineer. I attempt to understand the needs of the users better by modelling the real world as close to the user's perspective as possible, and propose the Real World Model (RWM). RWM is assumed to be developed by users, and requirements engineers. An engineering approach to building the model is provided.

A real-time system has a well defined use to its community. A requirements model must rely on the user level activities, and aid the human understanding and communication. In the RWM, a real-time system is viewed as a set of concurrently acting automata, each representing a system entity. This model supports temporal reasoning in easily described ways, for all classes of timing properties. A generalised classification of timing constraints is provided.

A requirements modelling language facilitates the description of requirements, and serves as a medium of communication among developers and stakeholders. Jarke *et al* [Jarke 94] observe that there is a need for a requirements language that manages the relationship between the meta-level domain scheme, and the scenarios that actually instantiate the scheme under development. Here I propose Timed Requirements Language (TRL) to bridge this gulf between the world of stakeholders, and the world of specifiers. TRL has natural looking expressions for formulating the needs. TRL has a number of novel features including the treatment of causality, and the description of static, and dynamic constraints all integrated into one uniform framework. TRL has been used with a number of systems. The generality of the language is validated through its application to specific systems.

Contents

List of Figures

Acknowledgements

This work has resulted from the advice, encouragement, and attention of many people; I wish to thank them all.

My sincere thanks to my advisor Professor Patrick Hall for his continued support during the course of this work. Amongst many shortcomings of mine, Pat had to put up with my odd working schedules. He has taught me many things, and I am really grateful for having been under his tutelage.

I wish to thank Professor Darrel Ince, Dr. David Benyon, Dr. John May, and Dr. Hong Zhu for helpful comments. I also wish to thank Debbie Stone for her friendship, and help.

I am indebted to my brothers Vijay, Ramesh, and Uday for their encouragement. They have been a source of inspiration to me. It's a pleasure to acknowledge the support of my sister-in-laws Ramã, Mamatha, and Vani. As a child my interest in learning was instigated by my father Krishnamurthy who owned an excellent library, and my mother Chayalakshmi who encouraged me to spade through the library. Thank you Anna and Amma for everything. Finally my thanks to Indira (Anu) for her support and encouragement.

Chapter 1

Introduction

Computers are used extensively in industrial, medical, scientific, and military systems. Many of these systems operate under critical conditions. The critical nature of these systems, coupled with their inherent complexities, demand that a systematic approach be employed while modelling the requirements of these systems. This thesis proposes such a treatment.

1.1 Concept of Control

Control is the essence of technology. The word control is usually taken to mean *regulate*, *direct*, or *command*. The need to mechanise the process of achieving a result is ever increasing. The early processes were primitive. They were controlled and supervised manually. Considerable progress has been made since the development of computers. The evolution of process control has been astounding with the continued improvement in the capability of computer hardware[1]. A major application of computers has been in the control of physical processes such as controlling the traffic, regulating the power supply, and etc. In all such applications computers monitor and control the functions. In process controlled systems, an important aspect is the *process dynamics* i.e., the time behaviour of changes in operating conditions. In all these circumstances computer has to adapt to the changes. For such reasons scientists and engineers agree that these systems are difficult to model, specify and design. This dissertation is however concerned only with a subset of these activities. In specific, requirements modelling is the subject of this dissertation. The remainder of this chapter introduces the characteristics of these systems in more detail and shows how and why these systems introduce unique problems into the requirements modelling process. This study enables us to discover the objectives of a requirements model to be employed with these systems. This work proposes an approach based on the objectives identified.

[1] There is a great factor of improvement in the ratio of the cost of the microprocessor, to its capabilities. This has spiralled the ambitious growth of the process controlled systems.

1.2 Context and Motivation

A system which operates with a dynamic environment (variable environmental conditions) is forced to operate with temporal constraints. The temporal restriction depends upon the changes occurring in the environment. Such systems normally operate with a number of physical devices, to monitor and control the environment. These systems are termed differently depending on the area of application like process controlled systems, embedded systems, discrete event dynamic systems, and reactive systems. In general they can all be referred as real-time systems. The word real-time emphasises the fact that

- time criticality is crucial for correctness rather than convenience, and
- a number of semi-independent activities must be coordinated.

These systems range in size from very large like air traffic control systems to much smaller systems like patient monitoring systems. Real-time systems normally interact with physical devices that have to be monitored and controlled. Real-time systems perform complex functions like control of physical devices, communication between various devices, and coordination of user interaction with the system. Thus we consider a real-time system as a combination of interacting elements forming a collective entity[2]. These systems are used for dedicated applications. This means that every application is the subject of special requirements enforced by the customers depending on the application environment.

[2] Oxford dictionary definition for a system, "a set or assemblage of things connected, associated or independent, so as to form a complex unity".

1.3 The Nature of Modelling

A model is a representation of the problem usually on a smaller scale, and modelling is to create a model. A model represents the factors for the purpose being considered. For example a 'model of a shopping complex' (say, displayed in the city hall), does not provide any guidelines for the civil engineer to build the shopping complex. The purpose of such a model is to gain the public opinion on the proposed project. Model differs depending on its intended purpose. In fact a system development can be regarded as a series of model building activities.

1.3.1 Requirements and Specifications

In the computing literature the two words, requirements, and specification are used interchangeably, or mostly in conjunction. Abbott and Moorhead [Abbott 81] proposed that a distinction be made between requirements and specification. In their words, 'a requirements document defines the requirements of the system to be built, while a specification explains how a system that meets those requirements would look to the user'. In other words, requirements refer to the needs of the user, while specification gives a description of the system that meets those needs.

In this work we use the two words requirements, and specification as two distinct activities, and for such a reason we detail out, what is requirements?, and what is specification?

1.3.1.1 Requirements

Requirements reflect high level aims, or goals. The requirements are essentially conceptual[3]. The requirements reflect the needs of the user, and are descriptive. Requirements provides a description of the environmental oriented activities. This phase is primarily an activity of determining the requirements. At this stage it is only possible to validate the requirements model. The role of requirements model is to act as input to a specification model.

1.3.1.2 Specifications

Specification, specifies the properties of a system to be developed. In other words, specification is prescriptive. Gehani and McGettrick [Gehani 86] express very clearly the intention of specification. They state:

> 'There are important benefits from writing specifications, i.e., stating in precise terms the intended effect of a piece of software. For then it is possible to talk about such issues as the correctness of an implementation, a measure of the consistency between that specification and the effect of the program. The range of benefits are actually wider than this: they relate to the methods of programming, to possible approaches to verification and validation of programs, and even to the management and control of large software projects'.

[3] Conceptual -(Oxford dictionary meaning) - that is conceived or taken into the mind.

Thus the issues concerned during specifications closely relate to the implementation of a system. As mentioned earlier, in requirements explicit attention is given to the environment, while in specifications according to [Wing 90] it is often neglected. Sol [Sol 83] refers to the requirements model as 'conceptual model', and the specification model as 'empirical model'. Conceptual model is based on the belief that such a system is desirable from the human point of view. However whether a design actually meets the expectations, can only be determined when the 'conceptual model' has been refined into empirically determinable characteristics.

1.3.2 Requirements Modelling

Requirements reflect a certain subjective desire. In other words, understanding a system from the user perspective forms the requirements. As defined by a number of researchers, requirements describe the functions to be performed by the system from the viewpoint of user or external environment, without implying a particular implementation [Heninger 80, Davis 79, Boehm 76]. During requirements modelling the objectives of a system that characterise the user's needs are documented and agreed upon.

From the external (user's) point of view, a system can be characterised by the realistic descriptions of the service provided by it. Requirements document is the place to record that information. Requirements document serves the user, specifier, and acceptance tester. When a system is under acceptance testing, it is actually testing the system against the needs of the user. Any error made in identifying the requirements, may go undetected till the completion of the tests. Correction of such an error involves extensive reworking of the complete system. As noted by [Roman 85] and [Boehm 81] discrepancies discovered between the delivered system and the requirements are the most difficult and expensive to correct, and they may even make the entire system useless.

The basic activities in requirements phase are referred as requirements modelling. The languages used during requirements are referred as 'modelling languages', while the languages used during specification are termed as 'specification languages' [Greenspan 94].

1.4 The Problem

To model the system, we must understand the various problems the system poses. Real-time systems introduce unique problems while modelling, because of the nature of their application. In the following sections, we discuss modelling these features.

1.4.1 Real-Time Systems are Safety Critical

Real-time systems are used in such applications, where an error[4] could harm the plant and even the lives of the people[5]. These systems are safety critical. In the early days there was a reluctance to introduce the computers in safety critical systems. This reluctance was partly grounded in the fear of introducing an unknown (complex) factor. Safety-critical systems were largely controlled by mechanical or electronic devices, with the help of human. The human error was regarded as controllable and manageable before any damage could occur. As the

[4] [Leveson 86] discusses the vocabulary that has evolved to discuss safety.

[5] An extreme example is Bhopal.

microprocessors became cheap, and more powerful, the use of computers in safety critical systems could not be resisted, and are widely used. Some of the examples of these systems can be found in flight control, railway traffic control, aerospace, industrial plant control, and health care systems. The potential advantages of using the computers in safety critical systems are discussed in [Parnas 90][6]. Reviewing [Leveson 86, Leveson 91, and Parnas 90] we can conclude that system accidents are intimately intertwined with complexity. With the advent of more powerful microprocessors the potential for problems may also be on the increase. Despite such apprehensions, computers are used to control safety-critical systems. As Rouse [Rouse 81] suggests introducing computers can improve safety[7]. While Perrow [Perrow 84] argues that, though the increase in technological innovations can decrease the accidents, they (the technological innovations) also allow those making the decisions to run greater risks, in search of increased performance. This means, the safety factor may not get the consideration it deserves, before the demand for better performance. For example, 'feedback control makes it possible to design aircraft that are aerodynamically unstable (such as the X-29) so as to achieve high performance' [IEEE 87].

[6] The advantages are (1) possibility of building more logic into the system easily, (2) logic in software is easier to change (at least in theory), and (3) can provide more information to the operator.

[7] The techniques of improving the safety with computers are described in [Anderson 81], [Sennett 89], [Bowen 93], but are outside the scope of this thesis.

Safety criticality is associated with the consequences like loss of human lives, risk to the health of persons, environmental pollution, or damage to the property[8]. Safety is concerned with the causes, and consequences of accidents.

1.4.1.1 Modelling Safety

Safety is a system wide property [Leveson 86]. Safety relates mainly to the environment surrounding the target system. Safety requirements of a system depends on the application environment. For example, a temperature controller used in a home heating system, and in a nuclear reactor have totally different safety requirements. Thus safety requirements can be stated concentrating on the application environment.

1.4.2 Real-Time Systems are Time Critical

A crucial aspect of real-time system is its dynamicity. This aspect, makes the system time-critical. This is clear in the definition of real-time system in the Oxford Dictionary of Computing

> 'Real-time system is any system, in which the time at which the output is produced is significant. This is usually because the input corresponds to some movement in the physical world, and the output has to relate to that same movement. The lag from the input time to output time must be sufficiently small for acceptable timeliness' [Oxford 90].

[8] [MIL-STD 84] defines safety as "freedom from those conditions that can cause death, injury, occupational illness, or damage to or loss of equipment or property".

Robert Glass [Glass 83] stresses the importance of timeliness as follows:

> 'The computer is controlling something that interacts with reality on a timely basis. In fact timing is the essence of interaction. · · · An unresponsive real-time system may be worse than no system at all' [Glass 83].

It is evident that the timeliness requirement, is defined by the application environment, and not by the computer.

1.4.2.1 Modelling Timeliness

Time as a property of the universe has intrigued people since centuries. Real-time systems are time critical. Temporal constraints are resulted from the characteristics of the environment. There does exist some difference of opinion among researchers, on the explicit use of time, while modelling the real-time systems. In an interesting article Turski [Turski 88] warns against the over reliance of timing factor. However Turski agrees that sometimes timing is the only viable way to express the interactions of a real-time system with a physical process. Mok [Mok 91] discusses at length the necessity of temporal considerations in real-time systems, and argues for the use of timing constraints as a control mechanism in a systematic way. Jaffe *et al* [Jaffe 91] discuss the importance of timing in requirements. These requirements are constraints on the real-time behaviour of the system. The criticality of functions arise due to the timeliness requirement. Timeliness has to be observed even under extreme load conditions. As Harel [Harel 92] pointed out behaviour over time is much less tangible than either functionality or physical structure, and more than anything else, this is the aspect that renders these systems so slippery and error-prone.

Real-time systems are time critical, and the consequences of this on the requirements description language are:

- the description language must consider explicitly not only what happens, but also when it happens;

- the description language must include syntactic mechanisms suitable for the definition of timing constraints.

Timing constraints are dependent upon the application environment. For example the applications like, spray painting a car by robot have stringent timing restrictions. The job done either too early or too late can be dangerous, or ineffective. Timing constraints are determined by the environment. The dynamics of the environment imposes the timing constraints.

1.4.3 Real-Time Systems are Reactive

Harel and Pnueli [Harel 85] introduce two different views on computing system.

The first view regards the behaviour of a computing system as a function from an initial state to a final state in a deterministic case, and as a relation between initial and final states in a non-deterministic case. This view is appropriate for the systems, where all inputs are available before the beginning of the computation, and outputs are produced at the termination of computation. Such systems are referred as 'transformational systems'.

On the other hand, there are systems, that cannot be covered by the transformational view. These systems are those that, ideally never terminate, since their purpose is not to attain a final result, but rather to maintain some interaction with their environment. These systems are called as 'reactive systems'.

Real-time systems fall into the latter category of systems. As such, the purpose of a real-time system, is to maintain an ongoing relationship with the environment [Stankovic 88a]. A real-time system controls a physical system, by taking into account all interactions with the environment where the physical system works. The real-time system must be aware of each change in the environment, and the action by the control system may change the environment in some manner.

1.4.3.1 Modelling Reactivity

A real-time system is driven by the events happening in the environment. These events occur irregularly, and a control system cannot control these events. A well known example is a telephone switch, where the telephone switch has no control over the subscribers initiating a call. Thus a reactive system cannot block the occurrence of events not under its control. A sufficient condition for reactivity is the enabling property proposed in [Lynch 88]. A reactivity can be modelled with the explicit notion of 'trigger' - where the system events are the result of an earlier trigger. The triggering mechanism plays a very important role in analysing the behaviour of real-time systems. The changes that take place in the system are resulted by some other change. For example, a telephone exchange is idle, if no subscriber initiates a call. When a subscriber picks up his/her handset, it causes a series of actions and reactions. The term reactivity gives the implication of a strong sense of cause and effect. The notion of causality plays a very important role in modelling the real-time systems. Reactivity can be modelled by focusing on the environment.

1.5 Scope of Requirements Model

Gray and Thayer [Gray 91] identify two key components of any software requirements methodology: (1) to aid in determining the requirements and (2) to represent the software requirements. Requirements modelling is regarded as the core activity of Requirements Engineering. Davis [Davis 90] suggests that Requirements Engineering is the analysis, documentation, and ongoing evolution of both user needs, and the external behaviour of the system to be built. Greenspan *et al* [Greenspan 94] stress the importance of research in requirements modelling, 'it is our contention that such representation and reasoning issues must continue to be addressed and that their resolution is a prerequisite to progress in all aspects of Requirements Engineering research and practice'. The glaring limitation of the research in requirements modelling can be noted in the words of [Potts 91], "requirements engineering research seems to me to have been conducted because the people involved wished to apply techniques already developed for 'downstream' software development phases further 'upstream'; for example, the application of plan-based program skeleton recognition and reuse techniques to domain model schemas, or the application of program transformations to requirements volatility". As such the research emphasis in software engineering has been a 'bottom-up' approach. In the 1960s emphasis was on 'coding', in 1970s emphasis was on 'design', in 1980s emphasis was on 'specification', and in 1990s the emphasis is focused on 'requirements'. Because of such an approach, the practice of using the same upstream activities for downstream activities arises. Each activity has its own unique problems to be addressed, and requires recognising those before addressing them. In contrast to the upstream phases, requirements modelling is firmly based in the problem world, rather than in the solution world.

Due to the advances in the processor chips, the realm of real-time system is expanding rapidly, involving most computer products. The requirements description of real-time systems must capture the real-time aspects of the system discussed above. It is essential that the requirements model is validated by the stakeholders, as the end product should meet those needs. Thus along with the components identified by Gray and Thayer [Gray 91] the approach must also support validation of the requirements.

1.6 Objectives

So far we discussed the significance of the requirements model. In this section we briefly recall the arguments to extract the objectives of requirements model for a real-time system. These objectives determined the course of the work reported here.

Argument 1

As discussed in earlier sections, requirements modelling is a need oriented approach rather than a strategy oriented one. A basic purpose of a requirements model is to serve as a reference frame for communication among developers and stakeholders. As Potts [Potts 91] puts it Requirements Engineering is about the communication of human intent.

Objective 1: Requirements description must be understandable by naive users.

Argument 2

Requirements model characterises the users' needs. In other words, requirements model concentrates on the application domain, rather than on the characteristics of the system to be delivered. As reasoned out earlier, stakeholders must be able to comment and validate the requirements model. This validation helps to reveal the errors in the model. An error in this stage is the error in perceiving the features of the system, as perceived by the users.

Objective 2: Users' participation in the validation of requirements model is essential.

Argument 3

Requirements descriptions can be large. It must be possible to uncover static errors (e.g., syntax errors, range violations) in the requirements descriptions. Typically such improvements (though small) can be quite significant.

Objective 3: Requirements description must be amenable to machine assisted reasoning.

Argument 4

Real-time systems are normally complex. A research challenge identified in the control system conference [IEEE 87][9], looks for an approach for the description of the system, and states:

[9] A joint report by the leading researchers in control system.

> Solving almost any significant engineering problem requires finding a framework for identifying subsystems which interact with each other in easily described ways [IEEE 87].

Objective 4: A framework used for the problem description, must identify the subsystems.

Objective 5: Description of the interaction of the subsystems must be simple for the users to understand.

We recall the characteristics of real-time systems, and the modelling concepts of these aspects discussed earlier.

Argument 5

Real-time systems are time critical. A realistic description of the system must include not only the functional description, but also the evolution of such descriptions over time.

Objective 6: Requirements description, must include both the functional and temporal aspects in the same framework.

Objective 7: Requirements description must handle all classes of quantitative timing requirements.

Argument 6

Real-time systems are safety critical. Safety considerations involve real-time constraints. The timing constraints are derived from the safety of the objects in the control system. The limiting factors (like temporal constraints) are derived from the operational conditions, like the loss of data as time passes, or maintaining a safe distance between two vehicles in vehicle control system, and so on. If timing constraints cannot be met, then a timing error will occur. In such occasions it is necessary to describe the reaction to timing errors.

Objective 8: Requirements description, must provide a framework to describe reaction to timing errors.

Argument 7

Real-time systems are reactive. In earlier sections we discussed reactivity, and its modelling respectively.

Objective 9: Requirements description, must explicitly handle causality.

The work described in this thesis addresses these objectives. Many issues discussed in this thesis have been reported in articles [Sateesh 95a, Sateesh 95b, Sateesh 95c, Sateesh 95d, Sateesh 94a, Sateesh 94b, Sateesh 94c, Sateesh 94d].

1.7 Structure of the Thesis

In chapter 2, we review the research efforts in requirements modelling that have been addressed in the past few years. We provide a classification of the research efforts by means of their underlying mechanism. We evaluate some of the representative techniques based on the characteristics of real-time systems. Here we set the background and the criteria for thinking about modelling the real-time systems. The criteria concentrates on the characteristics of real-time systems. This review helps us to discover the need to address the problems identified in modelling the requirements of real-time systems.

In chapter 3, the notion of the requirements is examined in detail, and we propose the modelling approach, namely the real world model formalism. Following a brief overview of the guiding principles that motivated our approach, we present the basic components of our model. Here we present an approach for thinking and reasoning about a perceived application domain. An engineering approach to building the conceptual model of a system is provided.

In chapter 4, we provide an automata-theoretic approach for the real-world model discussed earlier. We discuss the various formalisms of time, and provide justification for the choice of our model - dense time. We introduce time constrained automata to model the dynamic nature of real-time systems. The model discussed provides a single formalism to describe both the functional, and temporal aspects of the system. A system is viewed as a set of concurrently acting automata, each representing a system entity.

Chapter 5, presents TRL (Timed Requirements Language). We present an overview of TRL (Timed Requirements Language) followed by its syntax, and semantics. Here we model the system by user oriented concepts and the constructs are easily readable. Elements of the language are discrete events and this applies to a wide class of systems. We provide a generalised classification of timing properties that may arise in a real-time system. We demonstrate that TRL conveniently handles all classes of timing constraints. TRL projects operational behaviour through time.

Chapter 6, provides a practical demonstration of the use of the concepts developed in the previous chapters, by means of its application to the problems, for which the requirements are derived and described using the criteria developed earlier.

Chapter 7 provides evaluation of our approach with the representative techniques discussed earlier in chapter 2.

Chapter 8 summarises the conclusions of this dissertation. It also identifies the possible paths for future research.

Chapter 2

Requirements for Real-Time Systems

The problem associated with requirements become amplified for real-time systems [Stankovic 88a]. A number of techniques have evolved over recent years to support this difficult task. The goal is to represent the high level objectives of a system. Here we examine the characteristics associated with real-time systems, and critically review the techniques suggested by various researchers.

2.1 Introduction

The "software crisis" is dead! [Freeman 89]. Yes software is no-more regarded as an unmanageable beast as it was considered to be. As Harel argued [Harel 92] the engineers in software community have fairly understood an insight into building the software. A million lines of code is the norm, and not an exception. Requirements has been identified as the main teething problem during system development. Requirements engineering emphasises the activities during requirements stage. In this phase of the software development life cycle, the external behaviour of the system is described [Davis 82]. Requirements description is now widely recognised as a critical step in the development of large software systems. Both Brooks [Brooks 87] and Turski [Turski 86] highlight this activity as the essence of software engineering.

McMenamin and Palmer [McMenamin 84] divide the system development activities into the essence of a system and its incarnation. The essence of a system constitutes understanding the system level activities, while the incarnation includes the side effects (like the availability of the technology/tool to implement the system, the social conflicts, and the cognitive limitations). The first step which describes the activities of the system provides the essential model of the system. This suggests that at the front end is the needs of the user, and at the other end is the (control) system to be designed to meet these needs.

The features of a system differ depending on the environment, and the needs of the users. For example the requirements of office systems (example: database systems, decision support systems), public information systems (example: home tele-shopping, transport information system), knowledge-based systems (example:

advice-giving systems), and real-time systems differ. Each class of systems has a set of distinguishable characteristics. Accordingly the needs and the objectives of a system differ as the properties differ from one class to another. The requirements method employed must be suitable to these systems. For such a reason we will briefly discuss the important features of real-time systems. Distinguishing features of real-time systems is dealt in detail in [Stankovic 88a, Foster 81, Burns 90, and Mellichamp 83].

2.1.1 Features of Real-Time Systems

A typical real-time system consists of a controller (computer) and an environment as a controlled object. Environment may comprise of physical processes and humans. The environment and the controller have a mutual influence upon one another. Koymans *et al* [Koymans 88] define real-time system as a particular kind of interactional system: one that maintains an on-going relationship with the dynamic environment. Consequently, a real-time system is fully responsible for the proper operation with respect to its environment. In a dynamic environment, the situations are characteristically complex and immediate. The control system must deal with the immediate situation. Such requirements poses restrictions on the real-time behaviour of the system. Wirth [Wirth 77] singled out this time dependency as the one aspect that differentiates real-time systems from other systems.

Burns [Burns 91] provides a classification of the systems, defining utility[10] as a function of time. Utility is the contribution of the execution of a task towards the

[10] Utility as a time varying function is defined in [Jenson 85].

system's objectives. The key idea is that the completion of a task has a value to the system that can be expressed as a function of time. Depending upon such a classification the various activities carried out by a computer are either real-time tasks or non real-time tasks.

Real-time tasks are time critical tasks and can be sub classified as [Stankovic 88a]

- periodic tasks,
- aperiodic tasks, and
- alarm tasks

While non real-time tasks may be performed as background tasks.

Periodic tasks are started at regular intervals specified by their period. Aperiodic tasks are activated randomly as a result of the environmental action. These are asynchronous. System has no control over such incidents. Aperiodic tasks can have stringent timing constraints. Alarm tasks are aperiodic tasks but they run with absolute priority over all real-time tasks. Alarm tasks are intended to handle exceptional conditions. Background tasks are tasks with no real-time properties.

Real-time systems are normally required to respond within a specified time. For example consider the firing operation of spark plug in an engine control system. Here it is the time at which the service is provided is important, not merely providing it, in which case the engine may never start at all. In some systems a right result produced late may contribute to a wrong result or may cause a catastrophe. We refer to these requirements as 'timing constraints'. Timing constraints depend on the physical characteristics of the plant. For example, advanced variable cycle jet engines can blow up if correct control inputs are not applied every 20 to 50 milliseconds [Lala 91]. Depending on the prominence of time criticality the systems are classified as soft real-time systems and hard real-time systems [Shin 87]. Faulk and Parnas [Faulk 83] provide a concise definition of hard real-time systems: 'we use the term 'hard real-time' to describe systems that must supply their information within specified real-time limits. If the information supplied is too early or too late it is not useful'.

2.2 Requirements Document

Computer controlled systems are complex entities [Dasgupta 91]. A process of abstraction is essential in understanding the user's expectations. The requirements are considered as an abstract representation of the system [Verrijn-Stuart 87]. Requirements engineer makes use of a description language to represent the needs of the user. These languages are requirements languages. Requirements languages provide frames with which the user's needs are defined. User needs are recorded in the requirements document. The requirements document is written using the terminology of the task environment, reflecting the user's view of the problem [Wasserman 79]. The purpose of the requirements document [Parnas 86] is

- to serve as a common reference frame for communication among customers, users, and developers;

- to serve as a model of reality, offering insight into the application domain;

- to provide documentation in order to facilitate the modifications or enhancements;

- to serve as a basis for test plan development.

At the heart of the requirements engineering process are the users and the customers. The primary requirement for the language employed for the requirements elicitation and representation is that it be understandable by naive users [Fraser 91]. The purpose of the requirements is to provide a model of the system. Thus requirements languages are referred as modelling languages [Greenspan 94]. These languages employ a variety of approaches. These approaches vary from employing a natural language to formal language. The language is formal in the sense that it has a well defined syntax and semantics [Davis 82]. Much research has been done in the design of languages. To get a more detailed view of the ongoing research, we provide a rough genealogy of these languages.

2.3 An Overview of the Approaches

Each system is unique in its own way. The needs of a system depends upon its application environment. Thus the expected features of the language differs from one class of system to another. Also the difference of opinion among researchers on such basic questions like: what should requirements be? how should requirements be stated? has led to the use of different approaches. The approaches suggested by various researchers are based on different flavours. Some of these flavours are appropriate during specification and design phase. For the sake of completeness we briefly refer to the various approaches.

2.3.1 Data-based Approaches

Early attempts at expressing the requirements shared a common view of the systems as data manipulators. Traditionally all the activities performed by a computer can be regarded as the manipulation of data. Entity relationship model [Chen 76, Hall 76] emphasises the structure and the relationship between the data items. These have been extensively used to model the static properties of the data. Entity relationship diagrams are the basis of high level data models. Diagrams emphasise data and the associations among data elements.

Data flow models extend this concept by incorporating the flow of information [Yourdon 79, DeMarco 78, Gane 79]. DFD (data flow diagrams) shows the flow of data. It shows how data entities are progressively transformed as they are processed by the system. Popularity of DFD is attributed to its simplicity, it requires no formal training to read the diagrams: bubbles are used to represent functions resulting from system decomposition, arcs connecting them represent functional dependencies among their input and output data, and suitable representations are provided to represent data stores and data exchange with the external environment [Fuggetta 93]. DFD has several weakness for real world modelling. DFDs are inherently ambiguous and incomplete for any procedural interpretation. The flow includes data flow, information flow, control flow and material flow. DFD has been criticised for its failure to model the dynamics in a proper way and for the lack of formal basis [Richter 86]. For example it is not clear when a process is activated, and how the complex combination of input can influence the activation. Many extensions have been suggested to address some of these issues like [Ward 86, France 92].

An early attempt to improve the practice during system development was the CRIS-effort. Unfortunately the CRIS-effort [Olle 82] limited its focus to the design phase. In this way it sought better ways to improve the design method. The efforts of Ward and Mellor [Ward 85, 86], Hatley and Pirbhai [Hatley 87], and Gomaa [Gomaa 84, 86] were directed towards providing a design method for real-time systems. Database modelling was the popular choice to represent solutions. The basic components in data-oriented model are entities, and data types. Data oriented perspective places emphasis on a complete analysis of data and its relationships. Data oriented models are solution centred. GIST [Goldman 80] is based on operational modelling over relational databases. Operational base of GIST allows executable specifications. Kung [Kung 89] proposes a graphical approach for conceptual modelling. An ER-like language is used along with the traditional DFD technique.

DFDs stress on logical decomposition of system into modules and on data dependencies. Heitmeyer [Heitmeyer 83] has shown that functional decomposition of system is implementation dependent and always results in an inferior system owing to the boundaries imposed by the decomposition. Yourdon [Yourdon 90] proclaimed the limitations of DFD and suggested to knock away the old technique.

2.3.2 State-based Approaches

Several attempts have been proposed to use finite state machine for modelling the system. An early suggestion can be traced to [Parnas 69]. The notable works include [Alford 77] and [Heninger 80]. SREM (Software Requirements Engineering Methodology) [Alford 85, Alford 77] was developed by a consortium of contractors for the specification and analysis of systems. It comprises of a set of tools and is based on stimulus-response paths and finite-state machine representation. SREM identifies events subject to timing constraints. SREM has very little support for abstractions and modularity [Berzins 85]. Heninger [Heninger 80] describes the external behaviour of systems in terms of events defined by transitions. However this approach has not explicitly modelled the timing constraints associated with the system. Recently Leveson *et al* [Leveson 94] propose a modified Statechart [Harel 87] notation. It may be noted that Davis [Davis 88] compares ten specification languages, and rates statecharts 3rd from the bottom in understandability to the naive users.

Dasarathy [Dasarathy 85] added timer alarms to finite state machine to model the temporal constraints. A state based language RTRL is reported in [Dasarathy 85], [Taylor 80]. SREM's RSL (Requirement Statement Language) and RTRL share a common view of the system, in which a response at any instance is determined by the system's present state and the stimulus that has arrived. State based languages have been found to be unsuitable for describing complex systems [Davis 88]. Descriptions in a state based language tend to be monolithic.

Recent works have addressed the issue of providing a temporal framework for the finite state machine. Lewis [Lewis 90] extends finite state graphs to incorporate timing constraints which is expressed as lower and upper bounds. Alur, Courcoubetis, and Dill [Alur 90] proposed the use of Timed Büchi Automata (TBA) to model the behaviour of finite-state real-time systems. TBA is a Büchi Automata augmented with a mechanism to express the timing constraints. Timing constraints are expressed using a finite set of clocks for each automaton. The clocks are set instantaneously with each transition. Nancy Lynch [Lynch 88] proposed the use of Input-Output automata as a model of computation, and this model is extended to include timing [Lynch 90]. The timed model allows the specification of lower and upper bounds on the transition.

A pure graphic formalism called Statecharts is proposed by Harel [Harel 87]. Statechart decreases the number of states by introducing the multiple active state notion. Jahanian and Mok [Jahanian 86, 88, 94] proposed modechart as a structured way of representing real-time systems. Modecharts is similar to Statecharts. In Modechart, a transition can be a time-bound pair which defines the smallest time (the delay), and the largest time (the deadline) for making a transition. For the purpose of reasoning about the specifications, Modecharts are translated into RTL (Real-Time Logic).

2.3.3 Petri-net-based Approaches

Petri-nets [Peterson 81, Reisig 85] consist of two basic components: a set of places and a set of transitions. In addition the movement of tokens represent the control flow. Tokens are passed from place to place through transitions by simple rules. Several researchers have proposed extension of Petri-nets to include the notion of time. The two earlier extensions are of Ramchandani [Ramchandani 74] and of Merlin [Merlin 76a, Merlin 76b]. Ramchandani associates computational delays with transitions. Here each transition is associated with a (finite) firing duration (a delay) of time 't'. The transition is prevented from occurring for the period 't', and is fired immediately after the elapse of time 't'. Ramchandani proposed this scheme mainly for performance evaluation. Merlin introduced the extension to specify and evaluate the communication protocols. Here each transition is associated with two values of time (l, u) lower bound and upper bound where l < u. If a transition is enabled then it remains enabled for a minimum time of 'l' before it fires, and 'u' is the maximum time during which a transition can remain enabled without being fired. The latter extension is more general and can incorporate the former. Associating delays on transitions violates the instantaneous firing feature of basic Petri-nets. This was remedied by associating delays on places rather than transitions [Coolahan 83]. Timed Petri-nets have been used for performance evaluation [Holliday 87] and safety analysis [Leveson 87].

A high level Petri-net formalism called ER nets [Ghezzi 91] is proposed to specify control, function, and timing issues. ER nets is similar to other high level Petri-nets [Agarvala 79] and integrates the timing extension mentioned above.

A certain amount of practice is needed in understanding Petri-nets and relating them to the real world. Petri-net lacks the ability to model the plant (environment) and controller separately. Petri-net handles plant and controller as one system. Thus Petri-net does not accommodate a systematic exploration of the system.

2.3.4 Process-Algebra-based Approaches

Algebras provide an abstract approach for the analysis of systems. A popular formalism is Hoare's theory of Communicating Sequential Processes (CSP) [Hoare 78, Hoare 85]. CSP provides a set of constructs for writing concurrent programs and laws for reasoning about them. The work of Davies and Schneider [Davies 89] has extended this model to include the timing. Timing is included with the addition of process *wait d*, where *d* is the non negative unit of time. The wait process terminates after *d* units of time. A conceptual global clock is used for delaying the process.

Another notable approach is Milner's CCS (Calculus of Communicating Systems). CCS views the system computation as a finitely-branching tree. CCS is based on two central ideas: firstly the notion of observationally equivalent processes, i.e., processes that are indistinguishable to an observer. Equivalence classes of processes are the basic objects of CCS; and secondly the definition and manipulation of these basic objects using algebraic operators. Various notions of observational equivalences have been proposed and studied. Untimed CCS is also extended with time [Wang 91].

LOTOS [Faci 91] [Bolognesi 87] (Language of Temporal Ordering Specification) is developed within ISO for specifying communication protocols. LOTOS makes use of a combination of methods like Act One (for the description of data structures) and CCS with some CSP influence (for the description of process behaviours and interactions). LOTOS notations have been criticised for the difficulty in reading it. It is remarked in [Ruggles 90] that LOTOS really stands for 'Lots Of Terribly Obscure Symbols'.

2.3.5 Logic-based Approaches

Pnueli in his seminal work [Pnueli 77] suggested the use of temporal logic for the specification of non terminating programs. From then on several researchers have greatly contributed to this field. Temporal logic makes use of the modal operators to describe the order in which the events happen rather than the actual times at which they happen. The structure of state is an important concept in temporal logic. A formula containing temporal operators is interpreted over this structure of states (sequence or a tree). Lamport [Lamport 83] suggested that time can be modelled by introducing a clock as a global variable. Then the assertions involving real time will be temporal logic formulae involving the clock variable. Here clock has to be incremented by the time required to execute that action at the end of every action. Ostroff and Wonham [Ostroff 87, 89] instead suggested the use of an infinite loop process (a clock process) to increment the clock variable ad infinitum. Ostroff proposed suitable structures to specify real-time constraints. In this formalism it is difficult to state some quantitative timing constraints [Ostroff 92]. Temporal logic has been found to be more suitable to state the global properties of the system like safety and liveness. Temporal logic notations tend to be terse, and as noted by some researchers (for example [Wing 90]) temporal logic specification is simply an unstructured set of predicates.

Unity [Chandy 89] gives specifications as formulas in logic which is similar to temporal logic. It also provides a collection of inference rules to deduce additional formulas that are satisfiable by a system. Shankar and Lam [Shankar 93, Lam 90] make use of a combination of styles. Safety properties are specified using automaton, and liveness properties by temporal logic formulae.

Allen [Allen 81] proposed a method for maintaining a network of relationships between temporal intervals. Seven types of relationships are defined that can hold between the two intervals. These relationships between the intervals has been a fundamental tool to think about the intervals, and has been used by a number of researchers. However Allen's relations face difficulties in handling the metric constraints.

2.4 Requirements Languages

A representation language is used to describe the essence of a system. Here the word 'language' is used in a very general sense, it includes natural languages, diagrammatic notations, or artificial languages based on different representation formalisms. An often used formalism is data oriented like, SA-diagrams [Yourdon 89], ER-diagrams [Chen 76], SADT [Ross 77a, 77b] etc. These semi-formal languages use a combination of graphics to describe system requirements. Other languages are the variants of Petri-nets or state-transition diagrams. Some of the other approaches are influenced by the concepts of the programming/simulation languages like Simula/Smalltalk. It includes knowledge representation languages like RML [Borgida 85]. Another notable approach is the executable language PAISLey [Zave 82].

A basic requirement of the language employed for the description of requirements is that it be suitable for the task, and must aid the communication among the various parties involved in the process. The requirements language is employed for reasoning and communication.

2.4.1 Comments on Specification Languages

As we pointed out in earlier chapter requirements is different from specification. Here we focus our attention on requirements languages rather than on the specification languages[11]. In an excellent introductory work on specification languages Wing [Wing 90] points out that specification languages neglect the environment. Fraser *et al* [Fraser 94] state that the specification languages are inappropriate to use during the early stages of lifecycle.

[11] Specification languages are discussed briefly in Section 2.7

2.4.2 Discussion of Features for Requirements Languages

It is widely recognised that well defined requirements is vital to the success of the project. The language employed to describe the requirements, must be suitable for the application. Depending on the type of the system, the tool to be employed for requirements description also varies. The effectiveness of the technique can be discussed with respect to some goal. The intended goal is the requirements model for real-time systems, and the technique must address all the aspects of requirements modelling. As we are interested in real-time systems the feature relies on the characteristics of real-time systems. We extend the dimensions suggested by Kung [Kung 83] with real-time requirements. A requirements model must support the following features:

- understandability
- expressiveness
- processing independence
- checkability
- changeability
- capability to handle quantitative timing requirements
- causality, and
- capability to handle timing errors

The first feature deals with the style of the content. User's involvement in the requirements development process is regarded as a crucial factor for the success of a system [Sølveberg 80]. This suggests that the model must include user-oriented concepts, and constructs should be easily readable. However the use of a natural language increases the ambiguity of the expressions at the same time. Understandability includes unambiguity, clarity and intuitivity. Intuitivity and clarity includes more than the representational formalism (i.e., graphs or tables so on). It essentially involves the aspects of enhancing the understanding of the application-oriented features.

The second feature deals with the description of the human perception of the reality. This refers to the concepts and constructs that are used - is this powerful enough to describe the features that need to be described without much effort. Model must include the time domain. Time perspective is required not only by the application domain but also improves the expressiveness [Bubenko 80].

The third feature refers to avoiding the premature design decisions. Designers must have an unrestricted choice of design alternatives. The requirements model must not cut into the space of the design alternatives. This implies that the model must be kept free of data processing considerations.

The checkability feature concerns the validation of the model. The model must not contain inconsistencies. It should be possible to determine whether the model represents the user intended goals.

The changeability feature deals with the nature of reality. The only truth about requirements is that it changes [Scharer 81]. To achieve a high degree of changeability a model must be localised and loosely structured. It should be possible to add and remove the system components while readjusting the schema.

The importance of the other three features has been discussed in detail in Chapter 1.

These features in general deal with the relative merits of the modelling formalism. In addition we concentrate on the generic characteristics of the time model. This deals with temporal functionality issues like primitive temporal notions and temporal reasoning formalisms. Characterisation of a technique according to our chosen dimensions has two advantages (1) it provides valuable information on the intended goals, and (2) it provides a basis for relating our observation on the actual use of the technique.

In the following sections we review the techniques. A fuller description of the approaches with example is further discussed in Chapter 7.

2.5 Specific Languages

2.5.1 Structured Analysis and Design Technique (SADT)

The development of SADT[12] was pioneered by Ross [Ross 77a, 77b]. SADT is a network of diagrams consisting of boxes representing activities. The arrows on the four sides of the box represent input, output, control and mechanism for the activity involved. The activities can be decomposed in a top down fashion. A natural language or an artificial language can be embedded into this graphical framework.

[12] SADT is a trademark of SofTech Inc.

An indexing scheme is used to state the relationship between boxes and arrows. SADT is often used during requirements phase.

Although SADT has a visual formalism, the large number of primitive constructs (around forty) can hinder the understanding. The mechanism concept may force an analyst to deal with premature implementation issues. SADT has no underlying formalism and any language can be used with it. SADT is a manual method.

Davis and Vick [Davis 77] characterise SADT as primarily an MIS technique. Zave and Yeh [Zave 81] note that SADT is grossly inadequate for real-time systems.

2.5.2 Requirements Statement Language (RSL)

RSL is a part of SREM [Alford 77, Alford 80, Alford 85]. RSL makes uses of a stimulus - response mechanism, and views requirements in terms of processing paths. Each processing step represents the arrival of a stimulus and the generation of a response. Each processing step is known as Alpha. Each Alpha can be replaced by a number of lower level of Alphas. The processing paths and step are represented in a graphical form known as R-nets. R-net is a data flow - like description of the processing steps to be performed.

R-Nets are used to input all the necessary constraints like maximum and minimum values, allowed ranges of the data and the timing constraints. It is difficult to keep track of the timing requirements, as they may span several R-nets. Timing constraints can be represented on stimulus - response paths. This allows timing constraints to be associated from a stimulus to a response. RSL is limited to describe the requirements only along the control flow path in an R-net.

Requirements in RSL is very difficult to express even for the experienced persons [Scheffer 85]. Also it has very little support for abstraction. It is more appropriate during specification, rather than requirements [Scheffer 85].

2.5.3 Real Time Requirements Language (RTRL)

RTRL [Taylor 80, Casey 82, Dasarathy 85] is based upon finite-state-machine and stimulus-response sequences. RTRL essentially consists of states and transitions. RTRL is nothing more than the textual representation to record the state-transition diagrams. Description in RTRL tends to be cryptic and the finite-state machine model shows through the syntax of the language. RTRL provides timer extensions to the finite-state-machine to describe temporal constraints.

2.5.4 PAISLey

PAISLey (Process oriented Applicative Interpretable Specification Language) is aimed at specification of embedded systems [Zave 82, Zave 84, Zave 86]. Both the environment and the system are modelled as a set of co-operating sequential processes. The language is based on APL, and is interpretable. The main thrust of PAISLey is on the output. The input to the system is modelled as an output from the environmental processes. The specifications can be executed.

Zave [Zave 82] emphasises on timing constraints and is implemented in PAISLey as comment. (a part of BNF of PAISLey is given below)

<timing attribute> :: = ! → <comment>

<comment> :: = any string of ASCII characters

PAISLey provides a mechanism for denoting the timing constraints, but does not enforce the same. A timing constraint always refers to the evaluation time of a particular function. When the specification is executed, the printer attached to the simulator prints the timing of each event. Thus it can be known, whether the timing requirements are satisfiable.

With PAISLey to state what a system must do, it is required to state how the system should do it. Such a mechanism severely compromises the basic tenet of requirements engineering - the separation of concerns.

2.5.5 Requirements Modelling Language (RML)

RML is a sibling of the TAXIS [Mylopoulos 80] programming language. RML [Borgida 85, Greenspan 86] expresses the requirements in terms of objects organised in classes. In RML everything that is described is an object. RML distinguishes entity, activity, and assertion objects in order to model different kinds of things. An object can only be described by describing its relation to other objects. Similar approach was also suggested by [Bubenko 80].

The classes in RML can be built into generalisations or is-a hierarchies. The is-a relation allows sub-classes to be defined, providing a notion of inheritance. The idea is that general classes can be defined first and then sub-classes can be defined while developing the details at a later stage. Subclass hierarchies are well known by Simula/Smalltalk.

In RML temporal information can be expressed by defining interval relations suggested by Allen [Allen 83]. Predicates like during, before, and overlaps can all be defined as classes in RML. The temporal description in this form is verbose [Greenspan 94].

Requirements are the top level objectives of a system. An object model developed in the requirements phase can be an actual base for construction of the system. Such an approach may lead to a structure that is not stable and maintainable [Jacobson 92]. Similar opinion is expressed by McDermid [McDermid 93]:

> 'In the author's experience, the greatest problem with requirements is that they typically start at too a low level - indeed they are presented in implementation terms. A stress on object orientation may well exacerbate this problem'.

2.5.6 ERAE (Entity-Relation-Attribute-Event)

ERAE [Dubois 85, Dubois 87] is based upon ER analysis [Chen 76]. It involves the definition of entities and relationships between them. It is an extension of the E-R model. The basic component of the model are objects and associations. In this sense both ERAE and RML share a similar view regarding the development of requirements. An object can be an entity or an event. Time is introduced as a distinguished value type. These concepts are handled in the framework of multi-sorted first order temporal logic.

In ERAE time is considered to consist of a linear sequence of states, with a set of events labelling the transitions between states. Each state is associated with a time value which increases along the sequence. A set of temporal operators [Dubois 87] is employed to refer to the past or future.

It may be difficult to translate customer's requirements in to the first-order temporal logic. Also customers cannot read and comment on the description.

2.5.7 FOREST

FOREST [Finkelstein 87, Goldsack 91] project makes use of SCS (Structured Common Sense) and MAL (Modal Action Logic). SCS is based upon the known methods like JSD [Jackson 83], CORE [Mullery 79] and ER [Chen 76]. SCS provides the method to write the specifications. Specifications are written in MAL. MAL is based upon a many-sorted first order logic. The logic includes the definition of variables, predicates, constant symbols, logical symbols, function symbols and a number of axioms and inference rules. The logic is extended with two sorts, actions and agents and a branching line temporal interval logic. Interval logic is used to describe time related objects. Agents identify the entities in a system, as is the case in CORE with viewpoints. Action describes the processes that the agents can carry out. Steps between SCS and MAL are not very clear. Also, it is difficult to express quantitative temporal requirements using intervals.

2.6 Discussion

A widely recognised problem with requirements is as follows. Firstly the complexity of the systems renders the description of the functionalities and constraints very difficult, and secondly a complete and correct set of requirements is seldom known in advance. These problems are increased with real-time systems because they are time critical and reactive. Reactive systems differ from the traditional information systems in being environmentally driven [Harel 85]. A sufficient condition for reactivity is the input enabling property proposed in [Lynch 88]. This admits the causal nature of physical processes. It requires that locally controlled actions be produced only as a result of an earlier trigger. Thus causal relationships are necessary to capture the environmental oriented activities. Although PAISLey is designed for embedded systems, it fails in many respects. Time is added as an afterthought and the notion of causality is non-existent.

Coombes and McDermid [Coombes 93] describe temporal logic as conceptually unsuited to the specification of distributed systems. They conclude that temporal logic can be used to represent certain issues, but at the expense of clarity. Similarly Bowen *et al* [Bowen 95] remark that trying to specify a concurrent system in a model-based specification language, such as Z or VDM, is like using a hammer to insert a screw. The languages based on notations adopted from mathematical logic are inappropriate for communicating with the end user during the requirements elicitation and confirmation stages [Fraser 94]. Fraser *et al* [Fraser 94] discuss at length the problems associated with such representational notations and state:

> Preliminary empirical evidence from cognitive science suggests that in the stages of problem solving, when the problem area is relatively ill structured, the use of formal representations inhibits the exploration of alternatives and is detrimental to the quality of the outcome. Thus · · · formal specification languages may not be an ideal tool for exploring and discovering the problem structure during the problem refinement process.

Some of the approaches [for example Kung 89, Fraser 91] have tried to redress this situation. These approaches have tried to bridge the gap of providing user understandability while providing the rigour of languages based on mathematical logic. Fraser *et al* [Fraser 91] propose the use of data flow diagram and decision tables to develop a complete set of requirements. While [Kung 89] proposes an ER like language to be used with traditional DFD technique. These approaches are commendable but they are not suitable for real-time systems. Though DFD provides intuitivity and understandability it fails to provide the processing independence, and the temporal informations are an afterthought and ad hoc. This study makes us to understand the deficiency of a language which

- provides a common reference frame for communication among developers and customers;
- provides a model offering insight into the application domain;
- provides processing independence;
- deals with the features of real-time systems;
- allows the expression of stringent timing constraints for time critical activities;
- deals with tasks of different nature, to integrate real-time and non real-time activities.

2.7 Quest for a Requirements Language

Most researchers in requirements engineering (for example Greenspan 94, Jarke 94) believe that research on requirements language will remain central to further development in the field. I believe this faith is rooted in two propositions:

1. Languages are the primary notational vehicle of our field. As concepts are explored, and become woven into the fabric of the field, they invariably find expression in languages.

2. There is an implicit hypothesis that the nature of the language shapes the ways in which we think about the problems. Although it is difficult to substantiate this directly, it is believed that the person equipped with a language suitable for the purpose is better equipped to deal with complex problems.

A requirements language has at least three goals:

1. It is an analysis tool.
2. It is a vehicle for human communication.
3. It is a vehicle towards automation.

A fuller description of these goals with examples is provided in Chapter 7, here we provide an outlook of these goals.

2.7.1 Analysis Tool

The requirements engineer faced with a task, has to choose a model that will accomplish the task. The model must be amenable to inevitable modifications, as the requirements do change. The initial stages of this process are generally best conducted at an abstract level.

The perspective of requirements engineer, specifier, and designer is different. Each perspective is different, in that it is dealing with a different set of constraints relevant to that perspective. For example:

- Requirements Engineer: Deals with utility or usability constraints in the conceptual view of the end product. It provides a conceptual model of the system.

- Specifier: Deals with the logical view of the product, and considers the operational constraints. It provides an empirical model of the system.

- Designer: Deals with the physical view of the product, and considers the design (constructional) constraints. It provides a solution model of the system.

The basic focus is the identification and recording of the requirements essential to the system. The figure 2.1 describes the factors that influence analysis. The basis for analysis is the belief that the document can be improved.

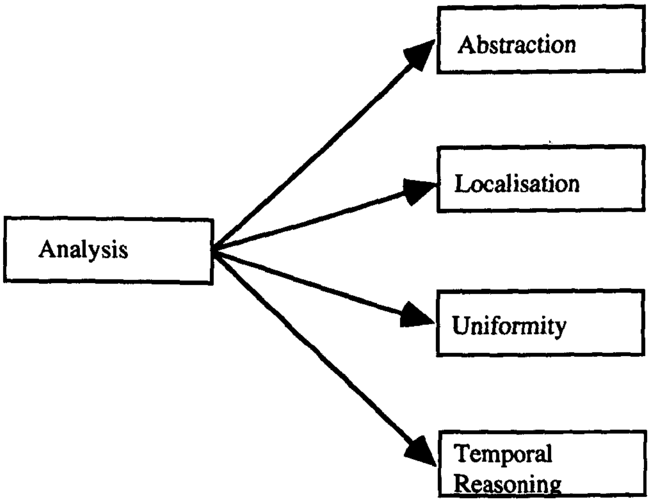


Figure 2.1 Characteristics of analysis tool

Abstraction: The concept of abstraction is to extract the essential properties while omitting the inessential details. The use of abstraction permits one to work with concepts and terms that are familiar in the problem environment without the need to transform them into unfamiliar structure. The concept of abstraction is perhaps among the oldest in computing (see Parnas 72). However the concept of abstraction at the requirements level is still a matter of controversy[13].

Localisation: Localisation builds on the notion of abstraction. Localisation is the idea of grouping the requirements. The requirements can be grouped depending on the environment, and the proposed system. The localised requirements provide a framework to understand the needs of the system better. This improves the reviewability of the document. There is no argument that the document be more reviewable as it could be improved to cater to the needs, while discovering the mistakes.

Uniformity: The concept of uniformity is applied to notational matters. The notation must provide a uniform way of describing all types of requirements. For example, with the notation the functional requirements, and the temporal requirements must be describable at the same level. The concept of uniformity provides a notation free of confusing terminologies.

Temporal Reasoning: This is an important concept that concerns real-time systems in particular. The notation must provide a uniform way of defining all types of temporal requirements that may arise in a system. The description of timing

[13] Davis [Davis 90] provides a detailed discussion of what versus how controversy.

constraints must also obey the concept of abstraction. The temporal requirements must emphasise the needs, not the way of implementing temporal requirements.

A language facilitates analysis by allowing the persons to use simple representations. If the representation of the requirements is closer to the problem space, then its applicability can be clarified through interaction with the users. Usually we understand a system by its expected features. Similarly the requirements document is validated with respect to user needs. Essentially the requirements must reflect the needs of the user. The users are concerned with the way they use the system. The requirements language must emphasise the way the users interact with the system. It is necessary that the model be expressed in a non-computing presentation mode. The representational factors influence human communication, and is discussed below.

2.7.2 Human Communication

A requirements language serves as a communication medium in two contexts:

1. After a requirements document is created, it is required to be used by a number of persons like specifiers, acceptance testers and users.

2. In large multiperson projects conveying the expression of thought, or concepts is important.

In both contexts one's ability to read and understand a fragment (of requirements) is more important than the ability to write the same fragment. A language's direct inclusion of central concepts that are characteristic of those class of systems is a major factor in making the concepts comprehensible. Also the document is to be used by various persons, and the computing concepts must be made invisible and unobtrusive as possible. The underlying concepts of computing system should be hidden from the user to the greatest extent possible. The way in which the requirements are integrated into the environment is significant in conveying the concepts. The requirements should be easily adaptable to conform to the changing user requirements. Most real-time systems are complex. Thus the language reflecting the features of real-time systems embedded with readable constructs increases the effective communication among persons involved in the project.

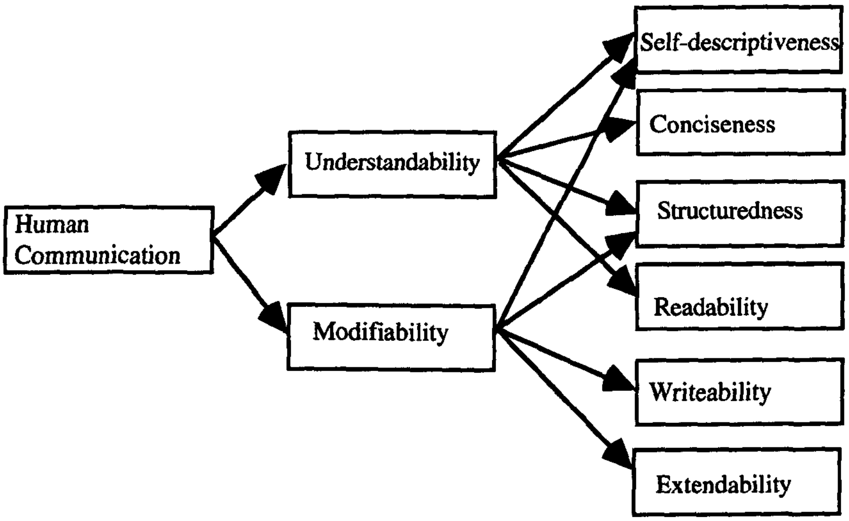


Figure 2.2 Characteristics of human communication

As implied in Figure 2.2, we are of the opinion that human communication improves with understandability, and modifiability. In the realm of requirements development, objectives are stated in terms of desired properties of the resultant document.

The influence of understandability depends upon the intended audience: users, management, or technical. Understandability involves the entire conceptual model.

Real-time systems are inherently complex. The notation must provide a structure to ease this complexity. Understandability involves many factors such as self-descriptiveness, conciseness, structuredness, and readability. These factors also improve the analysis. It is obvious that an understandable description can be analysed easily. The notion of self-descriptiveness implies a clear statement of requirements. Self-descriptiveness helps to ascertain the correspondence between the requirements document, and user needs. With conciseness the problems become intellectually manageable by highlighting the important features. Conciseness makes the description of the goal easier. The notion of structuredness denotes the ability to organise the requirements as a number of small units. Structuredness makes it easier to describe large systems. The notion of readability combined with structuredness, and conciseness makes the description of the requirements lucid. It is important that the stakeholders must be able to read the document before they can agree to it. Also the requirements development team consists of a number of persons, and readability helps in conveying the concepts.

Modifiability requires the ability to have an adaptable and evolutionary structure. The factors such as structuredness, conciseness also affect modifiability. The two other factors that interest us here are extendability, and writeability. Extendability implies controlled change, in which some parts of the document are altered while retaining some of the aspects. Extendibility is important as requirements change for different reasons. Writeability is a much less rigorous factor compared to other factors discussed above. Writeability imposes that the notation employed to document the requirements must be easily expressible. Writeability depends much upon the syntactic aspects of the notation. Writeability describes how easy it is to document the requirements in the chosen notation.

2.7.3 Vehicle towards Automation

Requirements document has different roles within the software life cycle. It serves as an input to the specification, and acts as a checkpoint in the design phase. Acceptance testing ascertains the correspondence between the deliverable system and the requirements document. Unfortunately, the requirements are never perfect, and requirements engineers are forced to reconceive their description of the system. Modifications and enhancement to a system requirement are common. To a certain extent the language must help in propagating the changes. Also with the notation employed the errors like syntax errors, or timing range violations must be easily checkable.

2.8 Related Issues

A number of issues concern the initial phase of a development of a system. In this section we briefly visit those issues.

2.8.1 Requirements Engineering

Prototyping has been suggested by many researchers to come to grips with problems associated during early stages [for example, Balzer 82]. Prototyping is successfully used in other disciplines like automobile industry. This is a very successful approach for massively produced systems. Prototyping is a solution oriented activity. It may become difficult to isolate customer requirements and implementor's responsibilities. Despite these difficulties, Luqi *et al* [Luqi 1988] have developed a prototyping tool that helps with the construction of prototypes.

Knowledge-based tools like KATE [Fickas 87], Requirements-Apprentice [Reubenstein 91], Analyst Assist [Adhame 89], have been suggested to help the analyst. KATE makes use of the domain knowledge to identify potentially missing components in requirements. Requirements-Apprentice uses 'frame' as the underlying concept, and can also use the domain knowledge in the same way as KATE. Analyst Assist, makes use of conceptual graphs, as the underlying mechanism, and involves - method knowledge, and domain knowledge. The motivation for the tool TARA (Tool Assisted Requirements Analysis) [Finkelstein 88] was based on the concepts of validation through animation, and reuse. The concepts were investigated in the context of CORE [Mullery 79]. Finkelstein concluded that reusability can be added, although not in a clean way.

Requirements modelling involves a number of persons. This involvement with a number of people may lead to conflicts in requirement. In recent years, many researchers have felt the need to address this issue. Nuseibeh and Finkelstein [Nuseibeh 94] propose the notion of a Viewpoint model, where one person can have several viewpoints, and also one viewpoint can represent several people. The tools are provided to support the environment. While Feather [Feather 89] uses a basic specification as a source which can then depart along different lines depending on the concern. These different parallel specifications are later merged. Another question that appears in this context is how to manage the conflicts. Anderson and Fickas [Anderson 89] suggested to look for the help of experts in the field to manage the conflicts. While Easterbrook [Easterbrook 93] proposed a tool (Synoptic) which allowed the participants to compare their viewpoints.

Another important aspect in requirements engineering is traceability. Ramesh and Dhar [Ramesh 92] propose a model to support this aspect. [Ramesh 93] also discusses the importance to assign accountability to identifiable team members. This helps to determine the criticality of the requirements.

The social issues that surround the requirements modelling was identified by [DeMarco 78, Checkland 81]. The problem articulation produces a picture of the stakeholders involved, and the goals people have [Checkland 81]. This places a solution in the socio-technical context. Recently ethnography a social process has been suggested to investigate the requirements [Gougen 93, Sommerville 93]. [Dobson 93] discuss the issues of safety in a system with human components. They argue that safety be modelled as a part of a process in the human activity system. The philosophical issues concerning the articulation of problems is discussed in [Hirschheim 89].

2.8.2 Specification Languages

SDL (Specification Description Language[14]) [CCITT 88] is a very popular language among communication engineers. An extension of SDL, (an object version of SDL - OOSDL) is its underway. LOTOS (Language of Temporal Order Specification) [Bolognesi 87] has been proposed by ISO for protocol specification. The present LOTOS (ISO accepted) however does not provide the facilities to represent quantitative timing constraints[15]. State based specification languages like Z [Spivey 89], and VDM [Jones 90] have been popular in the literature. Mahony *et al* [Mahony 92] discuss an approach to specify timing information with

[14] Also see [Rockström 83], and the whole issue of COM-30.

[15] Research efforts have been reported suggesting the ways to specify quantitative timing constraints in LOTOS.

Z. Ledru [Ledru 93] discusses a method to specify temporal information with VDM. Schobbens [Schobbens 93] propose a decomposition method for algebraic specification. Schobbens decompose the specifications into defaults (those that follow the rules), and exceptions. Dardenne *et al* [Dardenne 93] discuss a general approach to requirements acquisition in the context of KAOS (Knowledge Acquisition in autOmated Specification) an AI project. A set of rules is provided for transforming KAOS objects and actions into Z data and operation schemas. Kurki-Suonio [Kurki-Suonio 92, 93] discuss DisCo language. They discuss stepwise design with DisCo specification. Ghezzi *et al* [Ghezzi 91] discuss TRIO a temporal logic language. The specification language ASTRAL (a derivative of RT-ASLAN [Auernheimer 86]) can be translated into TRIO. Ciapessoni *et al* [Ciapessoni 93] discuss a revised version of TRIO to allow the reasoning on metric time. This extension is similar to the extension of temporal logic - metric temporal logic (MTL) discussed by Koymans [Koymans 90]. Specification language based on Petri-nets is also suggested [Ghezzi 91]. Fickas *et al* [Fickas 92] combine Petri-nets and temporal logic for the design description.

While Shaw [Shaw 92] discusses the use of CRSM (Communicating Real-Time State Machines) in the specification of real-time systems. Raju *et al* [Raju 94] discuss a prototyping environment for CRSM with the programming language C++. The other specification formalisms are based on Statecharts [Harel 87]. Timed Statecharts is proposed in [Kesten 91]. Gabrielian [Gabrielian 91] propose a method based on Petri nets, Statecharts, and temporal logic called HMS (Hierarchical State Machines). ENCOMPASS environment supports incremental construction of Ada programs [Terwilliger 87]. In ENCOMPASS, software is specified using PLEASE, an Ada based executable specification language.

2.9 Summary

As noted by Pohl [Pohl 94] the three phases of requirements engineering are representation, agreement, and specification. It is evident from the current literature that the majority of the work done is to support the specification, and incremental design. This is not necessarily surprising as the research work in specification, and design has matured (the upstream activities, as suggested in Chapter 1). While there is very little work done in bridging the gap between requirements, and specifications. This gap is also noticed by Jarke *et al* [Jarke 94], they state 'we do see a need for a formal requirements language that manages the relationships between meta-level domain scheme, actual specification, and instance scenarios of this specification'. Here we perceived such a gap, and in further chapters we discuss our approach to bridge this gap.

Chapter 3

Real World Model of Real-Time Systems

We attempt to understand the needs of the users better by modelling the real world as close to the user's perspective as possible. This model is assumed to be developed by users, and requirements engineers during the requirements acquisition process. Here we introduce an approach for thinking and reasoning about a perceived application domain. Our approach is non-data processing friendly more than the traditional approaches.

3.1 Introduction

Ramamoorthy and So [Ramamoorthy 78] state 'system requirements, needs, and objectives are generally vague and ambiguous, chiefly because they are at the top level and arise directly from the application area problems'. Since this statement, much work has been done in the field. We studied in the earlier chapter some of the suggested approaches and noted that real-time systems need some special attention. As Brooks [Brooks 87] noted 'the difficulty is not in saying but to know what to say'. For such a reason we need an abstract representation of the system to determine its requirements. A model of a system provides such a representation. In the following sections we discuss the modelling approach.

3.2 Modelling the Real-Time System

Stankovic [Stankovic 88b], and Ward and Mellor [Ward 85] characterise real-time systems by the existence of non-trivial interfaces between computers and their environment. The environment includes various technical components (devices) and people interacting with the controller (computer). In general a real-time system is an arrangement of physical components connected or related in such a way as to command, direct, or regulate itself or another system. With real-time systems everything that happens alters the environment in some manner. The system dynamics is understood by measuring the changes in the operating environment. The changes occurring in the environment is monitored by the sensors. Sensors provide the information on environment variables like temperature, pressure, velocity, position, level, and flow. The controller processes this information and determines the desired control actions. These actions are sent to actuators. An operator often supervises the system functions. The operator has a greater control on the system. As in flight operation, the operator can replace the control system and run the operation manually. A system can be described from external user's point of view as shown in Figure 3.1. Each of the components shown in Figure 3.1 have some kind of associated behaviour. A system can be thought of as a parallel composition[16] (||) of these components. Thus a real-time system can be modelled as controller || sensor || actuator || operator.

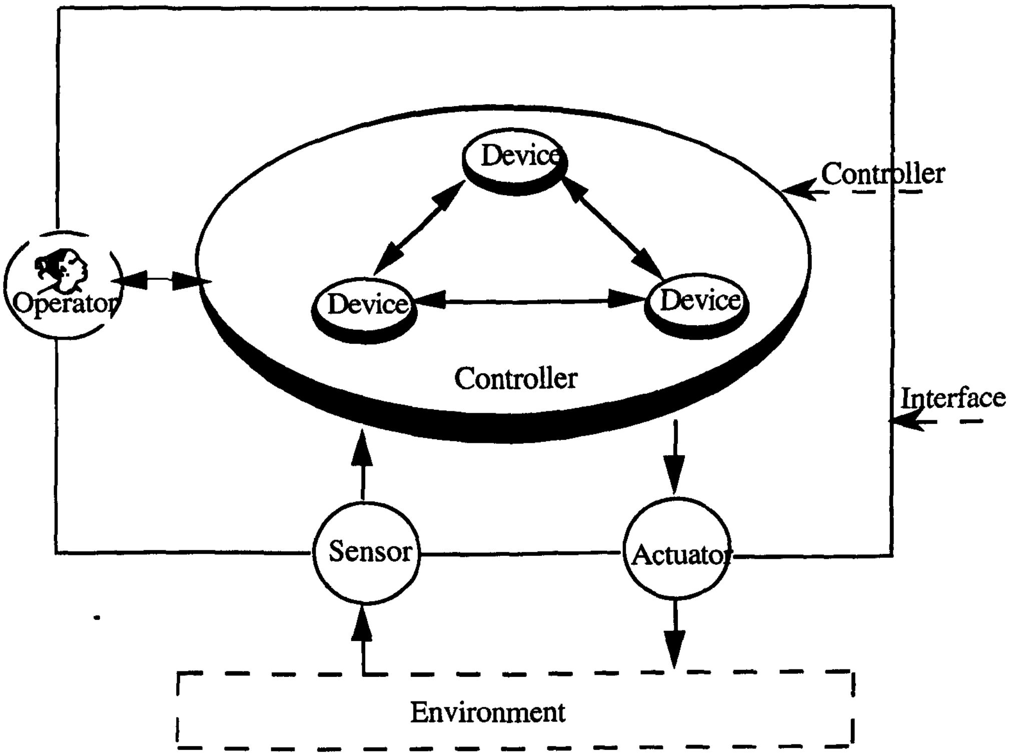


Figure 3.1 Abstract model of a real-time system

A model of a system is a simplified representation of a system (postulated or real) [Stavely 83]. A system can be modelled from the views of an observer. As Zeigler [Zeigler 76] notes:

[16] Formalisation of the operator (||) is provided in Chapter 4.

> The real system refers to nothing more or less than a source of observable information. The system may be a natural one, such as biological or ecological system, an artificial one, such as a computer operating system, or a mixed one involving both natural and artificial elements such as transportation, urban or world systems. The important characteristic is the identification of a segment of reality and the distinguishing of it from the rest, permitting measurements and other observations to be made on it.

Similar observation is also made by Hoare [Hoare 90], a model of a computational paradigm is a set of direct or indirect observations that can be made of a computational process. The observer view of the system is a very high level and domain-specific view of the system. Requirements describe only externally visible behaviour of a computer system [Heitmeyer 83]. It is easier and natural to modularise the requirements by means of features perceived by the user. Such a model of the system is called as a conceptual model.

A conceptual model provides a user understanding of the system behaviour. A conceptual model is not an actual construction model, it only provides a synoptic view of what is going on in the system. The phrase conceptual model was popularised in 1970's, and was used as a synonym for data modelling. It was often used in discussion with the design of database (see for e.g. Brodie *et al.* 84). In the literature, conceptual model is used at various levels. As Bennett [Bennett 91] comments there are different conceptual models dependent upon the observer: the designer's model, the user's model, and the assessor's model. Similarly Deutsch [Deutsch 88] proposes three different models (viewpoints) that are related to the major parties involved in system development: the customer, the user, and the implementor.

In the literature conceptual model is also referred as user model. Reviewing the obligation of a conceptual model, we can notice two different perspectives of a conceptual model, viz. the use perspective and the user perspective. Conceptual model discussed here emphasises on the use, rather than on the 'user'. Use perspective emphasises on the use of the model like:

- to provide a conceptual framework for precise thinking;
- to provide a framework to initiate communication among people;
- to check that the model reflects the intentions of stakeholders;
- to provide a framework for the stakeholders, on which they can test the end product.

While the user perspective emphasises the roles of different persons involved in the project. A number of persons are involved in a project, and their requirements of a product can vary, like the requirements of

- end users,
- specifiers,
- designers,
- quality engineers,
- maintenance engineers, and
- project managers.

The user perspective identifies that differences exist in the view of the system depending on the role of the person. This perspective models the views and their relationship [Finkelstein 92].

Our view of conceptual model refers to a highly abstract level of system description, and emphasises the use perspective. At a conceptual level the characteristics of the system are important. It provides an integral aspect of the system's definition. Conceptual modelling is closer to the human conceptualisation of the problem domain [Gorski 89]. Description at this level is aimed to enhance the communication between persons involved in the project including the customers. For the purpose of determining the requirements the conceptual models are abstracted at the highest level. In conceptual modelling the conceptual process is essential. Then what is the conceptual modelling process?

3.2.1 Conceptual Modelling Process

The conceptual modelling process deals with understanding the purpose of the system. As Ross [Ross 85] expresses 'at first, you don't actually know what the problem is. You have to get into the details to find out how it shapes up'. To get into the details, we need an orderly procedure. An orderly procedure (method) helps to determine the requirements, i.e., to build the conceptual model of the system. A model represents understanding of the system without having to deal with every detail of it. The modelling process detailed in Figure 3.2 provides a systematised way of reflecting the inherent structure of the model. In Figure 3.2, the left column represents the activities in the application world, and the right column represents the activities of the modelling process. The figure provides a description of the interrelated phases that occur during modelling. A layered approach is suggested for the development of a conceptual model. The first layer emphasises understanding the application domain, in the second layer we identify the various components that make up the application, in the third layer we develop an engineering understanding of the various components that enter the system, in the fourth layer we develop the specific use of the system, and in the last layer we revisit the model by developing the safety critical aspects of the system.

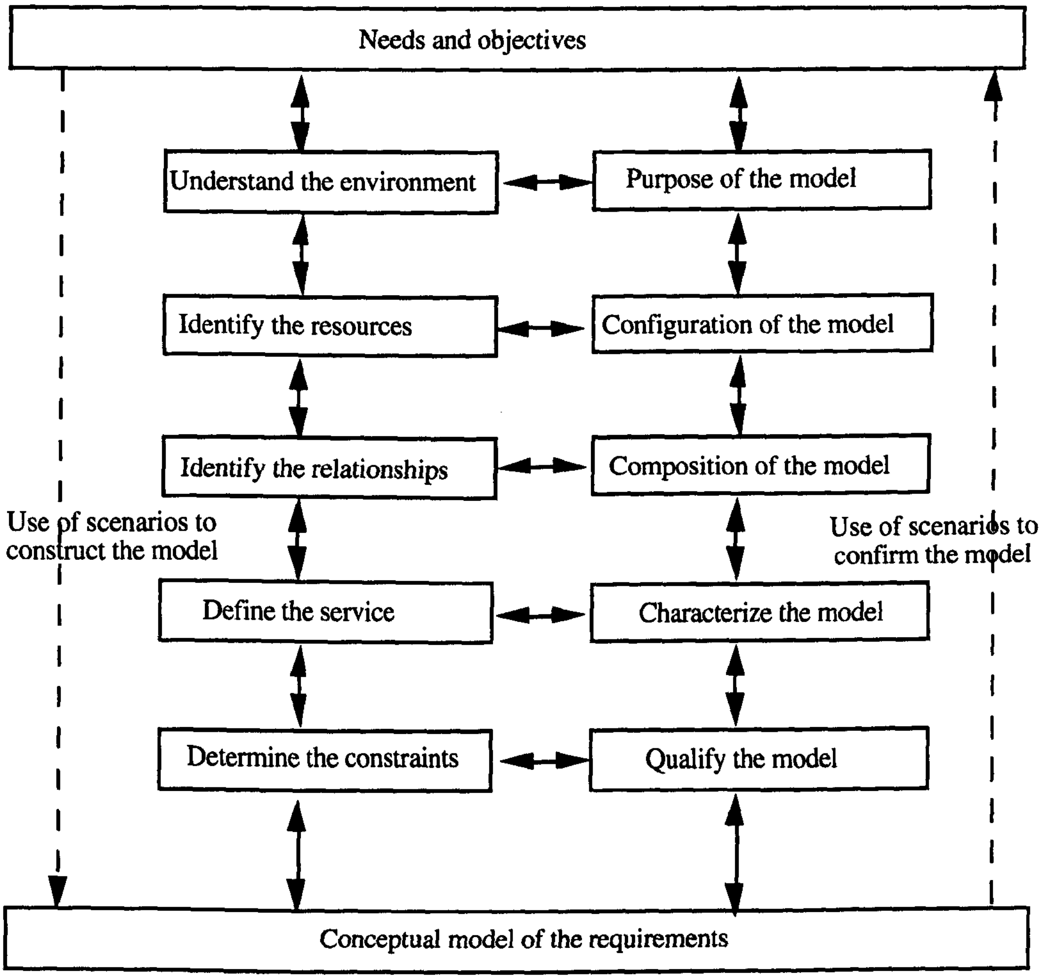

Needs and objectives
Understand the environment
Purpose of the model
Identify the resources
Configuration of the model
Identify the relationships
Composition of the model
Use of scenarios to construct the model
Use of scenarios to confirm the model
Define the service
Characterize the model
Determine the constraints
Qualify the model
Conceptual model of the requirements

Figure 3.2 Different phases of conceptual model

In Chapter 1, we discussed the importance of modelling the environment. The important characteristics of the real-time systems concerns the environment. In Section 3.4, we discuss the concepts of identifying the components that take part in a system. This details out what we mean by components, and what is its use to the definition of the system. In Section 3.5 we detail out how a component interacts with another. Section 3.6 discusses an approach to define the use of a system. A definition of a system is provided by defining its use. Section 3.7 discusses the constraints that introduce the restrictions to the behaviour of the system. The use of the system is refined with the constraints identified. A requirements model is not constructed by the requirements engineer alone, the model building activity is a shared task involving stakeholders. In Section 3.8 we discuss the validation of the model involving stakeholders. Section 3.9 discusses the significance of the approach, and summarises the approach.

We consider an example to motivate our discussion.

3.3 A Railroad Crossing Example

Consider the rail road crossing system shown in Figure 3.3. This problem was introduced by Leveson [Leveson 85]. This system involves operating a gate at a railroad crossing. The requirement is whenever a train is in the crossing, the crossing gate must be down. We make use of this example for discussing the various concepts.

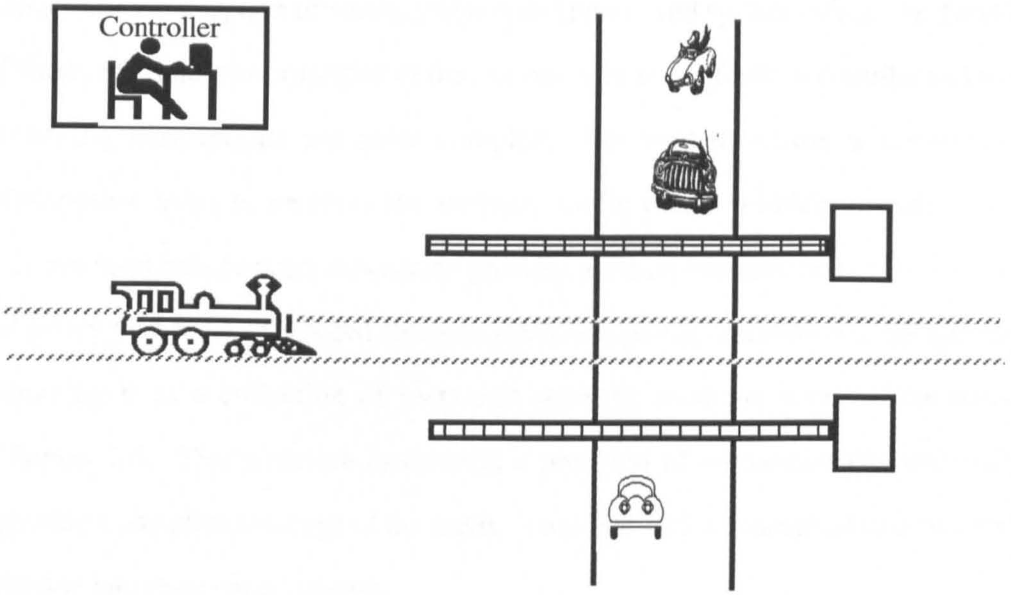


Figure 3.3 Railroad crossing system

3.4 Real World Model

Software systems are typically large and complex, and reasoning about such systems is a difficult task. An approach that has been suggested by many people to deal with this complexity, is to build a model that focuses only on those properties that are of interest, while ignoring the masses of irrelevant detail. This abstraction focuses on the identification of what an application does. An application takes place in the real world, and this calls for modelling the real world.

Determination of requirements is based on understanding the problem environment [Davis 82a]. Understanding the problem environment becomes essential to perceive what is important, and what is needed. The need for understanding the environment for the effective development of a system has been widely recognised [Jackson 83, Zave 83]. The requirements of a system is always in its relationship with the environment. Environment influences the requirements in three key dimensions: perception of needs, problem definition, and system safety. As Turski [Turski 86] notes the properties of the environment are difficult to describe and the resulting descriptions are quite complex. For such a reason a systematic description helps to perceive the intrinsic nature of the problem domain. An environment based system description provides a conceptual structure of the system at a very high level. The word structure refers to a partial description of the system showing it as a collection of parts and showing relations between the parts [Parnas 74]. This structure establishes a portfolio of responsibilities that will provide a complete coverage of the needs. To create such a conceptual structure we need to introduce some concepts.

I am of the opinion that for the reason of simplicity and comprehensibility (which is vital in the initial phases) only few basic concepts have to be introduced. In other approaches, often a large number of artefacts (for example, see Alford 85) are used. Our approach does not promote countless artefacts, and several steps. We present concepts that are suitable for understanding the system, and describe an approach to use these concepts. For consideration about the conceptual model two basic concepts are sufficient, namely `agent` and `role`. The concept of agent is well known [Feather 87, Finkelstein 87]. The notion of viewpoints introduced in CORE [Mullery 79] characterised as something that does things, is similar to an agent. As in [Feather 87, Finkelstein 87] we name agents those that contribute to the behaviour exhibited by the controller and its environment. In fact I define an agent as an artificial device that serves a representational function. In this sense an agent still refers largely to 'components', we mentioned in the earlier Sections. `Role` relates to a specific set of characteristics to be exhibited by an agent.

3.4.1 Concept of an Agent

A model of a system is identified in terms of the devices (parts), and its properties. Devices have specific capabilities. The capabilities are tailored depending on the customer needs. We name these devices as agents. An agent is an abstraction of a problem domain which models the characteristics of an entity. An agent is described by its external operations and usage restrictions. Agents are identified during problem analysis. These are characterised by what they do rather than what they are. Agent characterises the resources and the operations assigned to it. Agents can be either concrete or abstract. A concrete agent may have a representation in a system like a switch, a printer and so on. An abstract agent may have no direct representation in a system, instead it models a behaviour which is a set of operations that it can be requested to carry out. An agent has a particular responsibility to the system.

3.4.2 Concept of Role

Role describes an agent that has been selected for modelling. In essence, it is the role, that clarifies the intended purpose of an agent in the context of the problem domain. The description of the role of an agent forms a part of the requirements document. The role of an agent is provided by the customers. Role is a way of categorising agents on the basis of what 'it' does. For example in an organisation we can identify two agents 'programmer' and 'manager'. Any agent can play the role of a programmer or a manager. The difference between the two agents is attributed to the roles they play, rather than to the agents. Thus the agents are characterised by the roles they play. Each role has a specific goal associated with it. The two questions that arise in this context are:

- Does the role of an agent consistent with the objectives of the system?, and

- What steps are necessary for an agent to achieve a goal?

Here we are making a subtle distinction between objective and goal. An objective refers to the overall system expectations. While a goal refers to the expectations of an agent in a role ascribed. For example, the objective of a nuclear reactor is to produce electricity, while the goal of a 'plant protection subsystem' is to shut off the system in abnormal situation.

3.4.3 Agent Identification

In the literature many approaches have been suggested that merits discussion. Abbott [Abbott 83] suggests, writing an English description of the problem (or a part of the problem) and then underlining the nouns and verbs. Nouns represent the 'candidates' and the verbs represent operations on them. Similar idea was also suggested by Booch [Booch 83]. Ward and Mellor [Ward 86] suggest that 'candidates' may be derived from external entities, data stores, control stores, and control transformations. Coad and Yourdon [Coad 90] suggest another source of information like, structure, locations, organisational units, events remembered, the different roles of users, devices, and other systems.

In our case, the identification of agents is highly domain-specific. The agents are identified on the grounds of their utility rather than their approximation of the system behaviour.

For example as suggested by Booch [Booch 83], or Abbott [Abbott 83] we cannot rely on the descriptions provided by the customers. In the rail-road crossing example, a description may run like this, 'the cars, and vans move on the road'. This description may influence one to consider a van or a car as a 'candidate'. Similarly, it is too early to get trapped into the realm of DFDs. It is necessary to step back from the description of the system provided by the customer, and to think on the objectives of the proposed system.

The identification of the agents begins from recognising the objective of the system. At this stage we are concerned with the objective of the system, and not the implementation issues like functional decomposition. The objective is firmly grounded in the environment.

We need to know:

(1) What is the environment?

(2) In a real-time system, the environment acts as a source and a recipient. All the environment oriented activities are either the monitored activities, or the controlled activities. This raises an interesting question, what to monitor, and what to control.

(3) This analysis makes us to understand, what the system is intended for?

(4) What a system should do?, and

(5) What a system should not do?

Considering the example again, we have:

(1) The environment comprises of trains moving on the rail track, the cars and other vehicles moving on the road, and a gate in the crossing region to control the traffic. The gate stops the cars, and vans crossing the rail track.

(2) Monitor the train entering and exiting the region of interest.

(3) Control the operation of the gate.

(4) The system is intended to allow for the smooth flow of traffic in both the directions, on the road, and on the rail track.

(5) The system must close the gate while a train is in the crossing region.

(6) The system must not open the gate, while a train is in the crossing region.

(7) The system must not keep the gate closed unnecessarily (i.e., when a train is not in the crossing region)

Thus in this example, we need a sensor to detect the arrival and exit of a train, a gate to stop the traffic on the road, and a controller to manage the system.

We name the three agents as 'Train Monitor', 'Gate', and 'Controller'. The role of the 'Train Monitor' is (a) to monitor the arrival of a train, and (b) to monitor the exit of a train from the crossing region. The role of Controller is to co-ordinate the 'Gate'. The role of a Gate is (a) to make the gate to go 'Up', or (b) to make the gate to go 'Down'. The vehicles that pass across the road, have no roles to play, i.e., no role can be assigned to the vehicles which pass across the road. Thus the agents of the system are as shown in Figure 3.4.

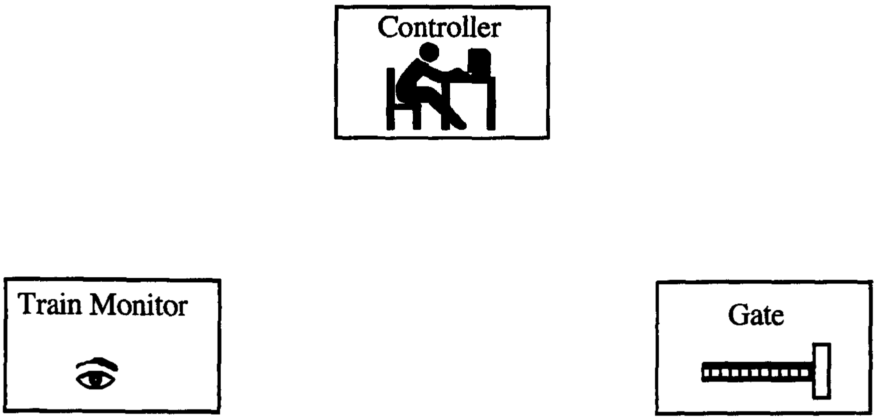


Figure 3.4 Agents of railroad crossing system

3.5 System as a Web of Agents

A system cannot be modelled as a single agent that does everything. The focus here is to express the requirements of a system as a set of agents which interact with each other. Relationship pertains between agents. Agents communicate with other agents in a system in order to achieve its responsibilities. The two questions that interests are:

- how the relationship evolves over time?, and
- how an agent interacts with another agent?

We model a system as an organisation of agents. Agents interact through shared information. We do not model the interaction of agents through requests transmitted by other agents. This approach is different from many object-oriented approaches like [Wirfs-Brock 90, Coad 90, Shaler 88] where services are requested through messages. The models that use such features become more solution-oriented, as the message that needs to be modelled is always a feature, that is outside the realm of conceptual model [ISO 87].

An agent keeps track of its user's focus of attention. Agent identification step involves two units of knowledge concerning the system: the purpose, and the function. An examination of these two units recognises the issues like:

- what is the role of an agent?
- what activities to be performed?
- what causes these activities?
- how these activities influence other activities?

These questions reflect the pragmatic issues like what are the things we are talking about, and how do we provide explanations of these activities. To deal with such pragmatic issues we need a general approach, which bounds the problem space and aids in the efficient search of requirements. Such an approach is discussed below.

3.6 Building the Real-World Model

Brooks [Brooks 87] feels that: 'the most important function that the software builder performs for the client is the iterative extraction and refinement of the product requirements'. Simply asking users to state the requirements is not sufficient. Davis [Davis 82b] identifies four broad strategies for determining the requirements as: asking; deriving from an existing system; synthesis from characteristics of the current system; and, discovering from experimentation with an evolving system. In practice, all these approaches are used. A true understanding of the system can emerge from understanding the needs of the individuals. Thus an approach used must have the following features.

Simplicity. It must enable an efficient interaction with the stakeholders. It must attempt to involve the stakeholders.

Informative. The approach used must encourage the user to reason on the requirements like, what s/he wants to do, why s/he wants to do, and when s/he wants to do.

Flexibility. An objective is to provide a tool of thought for the user to navigate with the problems. The user must be able to experiment with what-if situations. Dealing with such situations must be easy and straightforward.

Familiarity with the user's world. The vocabulary of the requirements document should be that of the application environment, not of the software engineer. The facts about the environment should follow the working rules of the user, and not the logic of the system. The information should be presented in the way the user handles it and not the way which is convenient to the software engineer.

A real-time system evolves by reacting to the requests it receives from the environment. A system description through the observable effects on the domain - as what happens to the environment makes the objectives clearer. Here we propose a scenario based approach to elicit the requirements.

3.6.1 Modular Scenario Based Approach (MSBA)

A scenario is a sequence of situations a user would experience when operating the proposed system. A scenario is a frame for the description of a particular sub-problem, which needs to be tackled by the system. Hooper *et al.* [Hooper 82] suggest that scenarios have the advantages of rapid prototyping without the overhead of actually building implemented prototypes. Scenarios provide natural ways of describing, how things behave in a system. Scenario based approach increases the communication between users and analysts.

A real-time system has an ongoing relationship with the environment. In a real-time system we can identify several patterns of reaction of behaviour. System evolution can be characterised by identifying several patterns of reaction as time progresses. These patterns are best understood by examining the change that occurs in the environment. A pattern of reaction can be referred as a scenario.

Our approach (MSBA) is different from the approach suggested by [Holbrook 90, Jacobson 92, Carroll 92, Hsia 94,]. [Holbrook 90] suggests to create a task hierarchy, and then to create the scenarios. [Jacobson 92] suggests the descriptions of use-cases from users to identify requirements. [Carroll 92] suggests a scenario based approach in understanding the activities directed at design. While in [Hsia 94] scenarios are generated for the system from the point of view of different users. In general task decomposition may lead to a rigid structure of the system [Heitmeyer 83]. Also decomposing goals in a top-down way is possible only for toy problems[17]. Generating the scenarios for the whole system is a very difficult

[17] Ross [Ross 85] complains that there is a magic in such an approach.

task. The number of scenarios in any system grows out of hand, making it tedious, and difficult to analyse. Our approach views the system as a network of agents. Here the scenarios fall into groups. These groups are natural for the users to analyse and comment. A set of scenarios define the requirements for an agent. It is possible to capture the responsibility of an agent with a reasonable number of scenarios.

An agent has a responsibility to the system. This responsibility sheds light on the expectations of an agent. This expectation symbolises a particular 'use' of the agent as conceived by the user. This 'use' provides a scenario. The concept of scenario generation is explained further in Section 3.6.3. Now consider an example.

For example consider the 'Train Monitor' discussed in the Section 3.3. The role of the 'train monitor' is to monitor the train in the crossing region. Monitoring involves, monitoring the arrival of a train (a train approaching the crossing region), and monitoring the exit of a train from the crossing region. This provides two scenarios:

(1) if a train is approaching the crossing region, then report 'train is entering'.

(2) if train has left the crossing region, then report 'exit'.

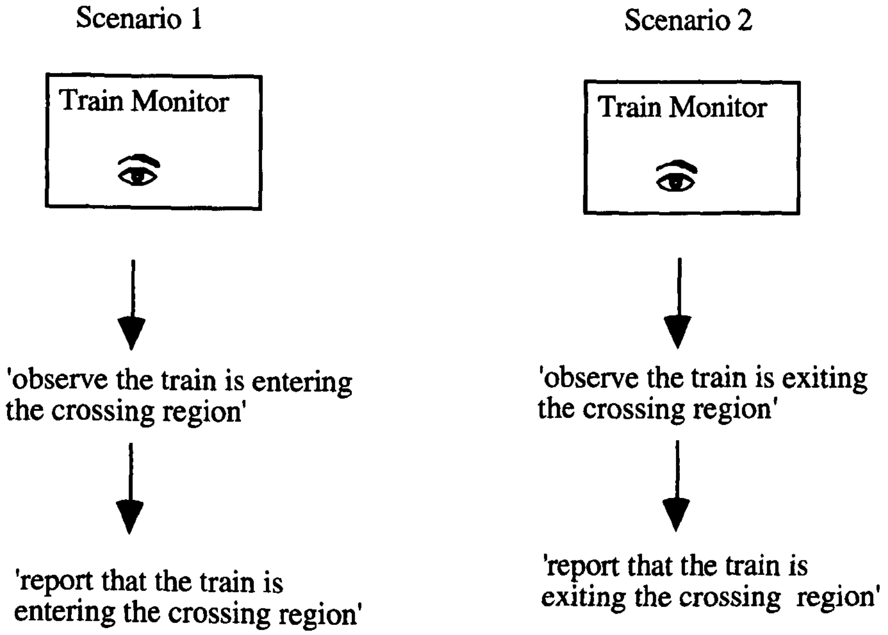

Scenario 1
Train Monitor
'observe the train is entering the crossing region'
'report that the train is entering the crossing region'
Scenario 2
Train Monitor
'observe the train is exiting the crossing region'
'report that the train is exiting the crossing region'

Figure 3.5 Elaborating the role of train monitor

3.6.2 Philosophy of MSBA

The approach - MSBA conveys a sense of the purpose of an agent by elaborating its role. This approach emphasises the utility point of view as conceived by a user. An agent has a perceived utility to the system. Such a responsibility driven approach is also suggested by [Hsia 88]. In general the stakeholder's interest is in what gets done, not how it gets done. This suggests that we consider important non-data issues such as context and role. The primary focus of conceptual model is concepts. The approach does not depend on a model of data. This view is in line with the conceptualisation principle advocated by the ISO document [ISO 87]. The conceptualisation principle states:

> A conceptual schema should only include conceptually relevant aspects, both static and dynamic, of the universe of discourse, thus excluding all

aspects of (external or internal) data representation, physical data organisation and access as well as all aspects of particular external user representation such as message formats, data structures, etc.

This approach is different from the traditional approach to problem solving that stems from the top-down approach, where the system functions are sub-divided into smaller and smaller problems. Such an approach tries to fit a problem into one mould at a very early stage. With complex systems the requirements modelling is rather an outside-in[18] approach, which allows to add more detail to the model as we gain further insights to the system. Requirements modelling as indicated by Feather [Feather 91] consists of a series of incremental steps that converge in a model with the appropriate content.

3.6.3 Characteristics of MSBA

The identification of an agent recognises the responsibility it has for the system. An elaboration of the role (as we discussed above) makes one to recognise the use of an agent. A comprehensive description of the use provides a scenario. A scenario accomplishes a goal. Malhotra [Malhotra 80] in studying the dialogue between people involved in problem solving, noted that the dialogues were composed of cycles like (1) goal statement, (2) goal elaboration, (3) solution outline, (4) solution elaboration, (5) solution explication, and (6) agreement on solution. Conveniently we can summarise this structure (as shown in Figure 3.6) by the following stages of user activity:

[18] We are using the terminology of [Ross 85].

- Determine the use,

- Conceive the purpose, and

- Specify the sequence of activities.

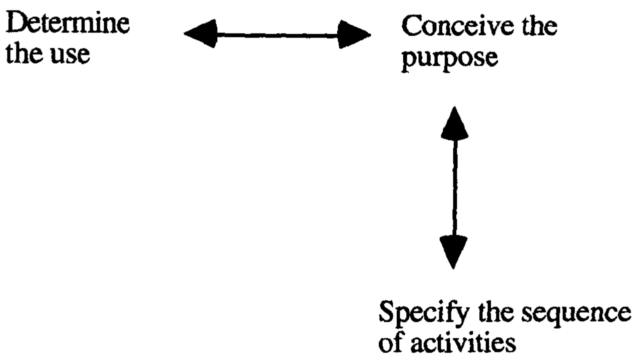


Figure 3.6 Notion of a scenario

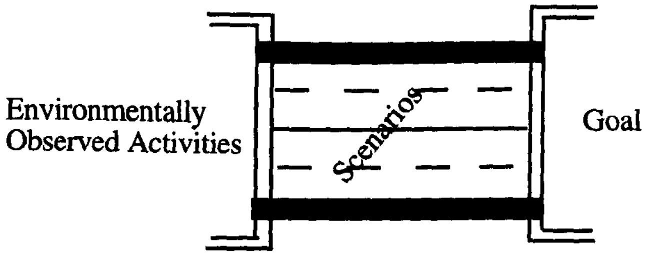


Figure 3.7 Visualisation of a scenario

A scenario bridges the gulf between environmentally observed activities and the intended purpose as shown in Figure 3.7. This brings out the relationship between the two. An activity can contribute to a requirement in three ways (as shown in Figure 3.8):

- An activity can `cause` a requirement. For example a person pressing a button, causes a requirement to be satisfied;

- An activity can `form part of` a requirement. When a person presses a button, acknowledging this action forms a part of a requirement;

- An activity can `fulfil` a requirement. When a person presses a button, displaying the required information fulfils the requirement.

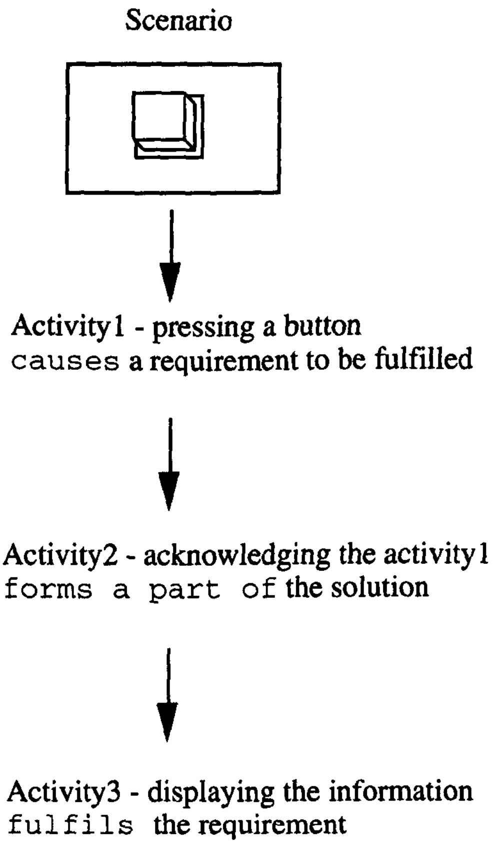

Scenario
Activity1 - pressing a button
causes a requirement to be fulfilled
Activity2 - acknowledging the activity1
forms a part of the solution
Activity3 - displaying the information
fulfils the requirement

Figure 3.8 Association among the activities

Thus the scenarios provide a suitable formalism in establishing the connections among the user perceived activities. Scenarios essentially involves something that the agent wants to accomplish. This accomplishment is described by activities. A scenario is an encapsulated description of achieving a specific outcome under specified circumstances. In real-time systems the agents have to accomplish a particular purpose under specific restrictions. The restrictions are influenced by the environment as described in Chapter 1. Analysing the restrictions with the described scenarios is essential. The next Section discusses such an analysis.

3.7 Modelling the Constraints

Constraint is a restrictive condition [Oxford Dictionary]. In general while working in the real world some set of constraints can be observed. Real-time systems have some special kinds of constraints. Some of these constraints arise from the technical capabilities of the system itself, and others from the nature of the activity, that is appropriate to the application. Constraints are essentially conditions imposed on the goals. We classify the constraints as static and dynamic constraints as shown in Figure 3.9. These are explained below.

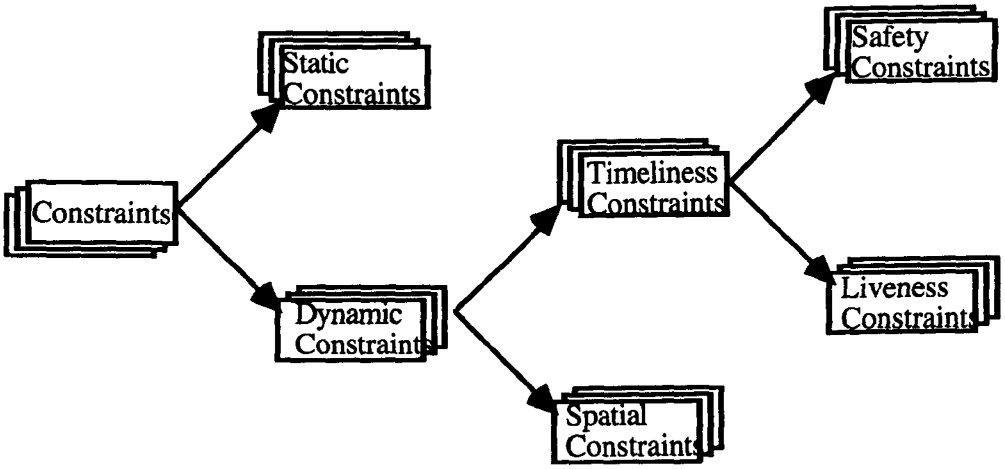


Figure 3.9 Classification of constraints

3.7.1 Static Constraints

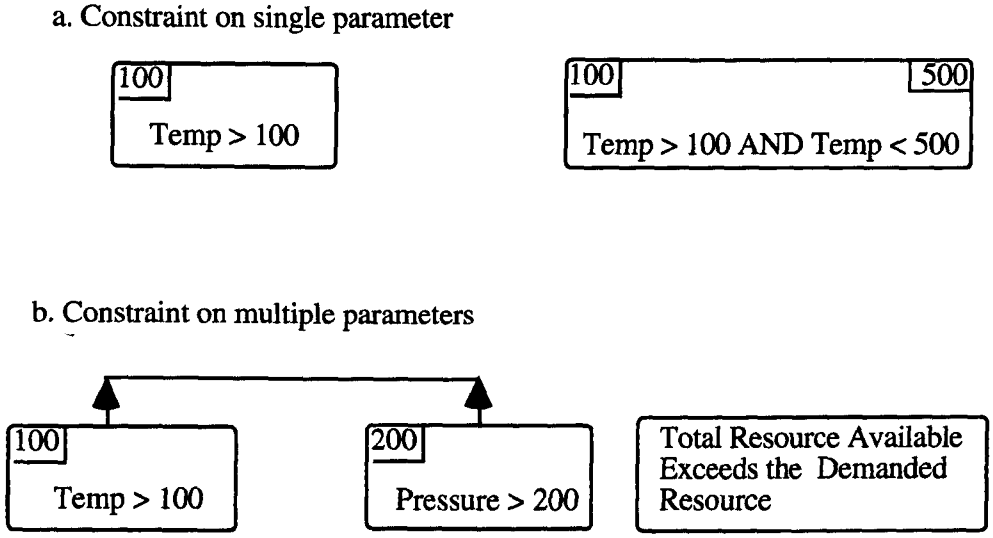
Static constraints are constraints that are independent of time. They specify the static aspects of the application domain. Static constraints stem from two sources. Firstly, a system cannot be assumed to have infinite resources. Every system has a limited resource like memory, number of channels, and so on. For example, when a car arrives at the parking centre, car can be allowed inside only if a space is available. Secondly many of the system's action is conditional depending upon the circumstances. For example while monitoring the temperature, a requirement can be, to raise an alarm if the monitored temperature exceeds 100 degrees. Here the temperature read by the sensor causes an alarm to be raised, only if its value exceeds 100 degrees. Such conditional requirements reflect static constraints. Static constraints can be sub-divided into two types as shown in Figure 3.10; static constraints as constraints over a single parameter, or constraints over multiple parameters.

a. Constraints on a single parameter:

1. Temperature > 100 degree
2. Temperature > 100 degree AND Temperature < 500 degree

b. Constraints over multiple parameters:

1. Temperature > 100 degree AND Pressure > 200 psi
2. Total resource available exceeds the demanded resource. Here the resource may consist of more than one parameter.



a. Constraint on single parameter

100
Temp > 100
100
500
Temp > 100 AND Temp < 500

b. Constraint on multiple parameters

100
Temp > 100
200
Pressure > 200
Total Resource Available Exceeds the Demanded Resource

Figure 3.10 Static constraints

3.7.2 Dynamic Constraints

Dynamic constraints are performance requirements. I agree with Zave [Zave 82] that performance requirements is what really characterises real-time systems. All performance constraints are constraints concerning time or space [Smoliar 81]. Here we will be referring to constraints on time. Timing constraints are an essential part of real-time systems. We refer to timing constraints as timeliness constraints, as they dictate the response time of the system. Timing requirements in real-time systems arise because of the importance of the activities of the controller upon its environment. For example, in a manufacturing plant, if the computer controlling a robot does not command it to stop or turn on time, the effect can be disastrous. Timing constraint imposes a temporal restriction on the environment and on the controller. Timing constraints provide a temporal relationship between the activities. Two types of temporal relationships can be distinguished depending upon the causal relationship between the activities [Koymans 88].

Qualitative: temporal relationships are concerned with only the order in time; and,

Quantitative: temporal relationships are concerned with the order and the distance in time. Quantitative temporal relationship refers to the time of occurrence of an action. Here we refer to such relationships as timeliness constraints. Timeliness constraints refer to the moment of occurrence of an action.

Requirements model for real-time systems must incorporate real-time features. An important real-time feature is the ability to measure time. For example, the data available from the sensor is time sensitive. After some time elapses, the data obtained from the sensor is of no value, as it may not reflect the true state of the environment. For such a reason a quantitative temporal reasoning is required. For example, in a rail-road crossing system it is required to state that the gate closes in a certain duration of time rather than to state that it eventually closes, in communication protocol, if message acknowledgement is not received within a certain time, then action is to be taken to re-transmit the same within a fixed time, and in a manufacturing system, a particular job like painting a car by robot may have to be started and completed at a particular time. These systems are time dependent, and require explicit quantitative temporal reasoning.

In literature there is interesting discussion on how best to represent the quantitative timing requirements [Jahanian 86, Alur 92, Lamport 83]. Global clock paradigm is a well known paradigm used to represent quantitative timing requirements. Global clock paradigm is not suitable for real-time systems [Jahanian 86]. Real-time systems are often distributed. The clocks drift, and it is difficult to synchronise the clocks. [Lamport 78] has argued that to avoid inaccuracies in timing only observable events should be used for timing other events in the system.

Timing constraints are bound to the environment and to the controller. A generalisation of the timing constraints is needed to discuss the temporal requirements. Such a generalisation was provided by Dasarathy [Dasarathy 85][19]. Dasarathy [Dasarathy 85] categorises the timing constraints by three types of temporal restriction on the events in a system.

maximum - no more than *t* time units must elapse between the occurrence of two events,

minimum - no less than *t* time units must elapse between the occurrence of two events, and

durational - event must last for *t* units of time.

A slightly different approach to temporal requirements is employed in [Mok 84]. This classification derives its origin from the scheduling problems. It categorises timing constraints as sporadic (quasi-periodic) and periodic [Mok 83]. Dasarathy views temporal restrictions from the point of view of user. While Mok views the temporal requirements from the point of view of controller. In a controller view the realm of temporal restriction, falls with the scheduler. Sporadic timing constraint requires some action to be executed before a specified time. For example a sporadic requirement can be, to open the valve within 10 time units of pressing a

[19] We discuss the limitation of this classification in Chapter 5, and provide a very general classification.

button. On the other hand, a periodic timing constraint requires some action to be executed at fixed intervals. For some applications a periodic timing constraint may exist from system initialisation, like monitoring the temperature in a nuclear reactor control, and for others it may come into existence dynamically, like radar tracking an aircraft, this comes into existence when the aircraft enters the traffic control region and ceases to exist after aircraft leaves the region.

In general, a model must reflect both types of timing requirements. In this Chapter we shall not reflect on the expressiveness of this classification of temporal requirements. We visit this aspect in Chapter 5. This study has presented the intricacies of timing requirements.

3.7.3 Timeliness Requirements

The fact that temporal properties naturally partition into two disjoint classes was first observed by Lamport [Lamport 77]. Thus timeliness requirements arise from two sources:

(1) Safety requirement, and

(2) Liveness requirement

3.7.3.1 Safety Requirement

Safety requirement depends upon the operational context. Safety requirement stipulates that 'bad things' do not happen during the operation of the system [Lamport 83]. For example in a railroad crossing system, the safety requirement may state that, accident should not happen. An analysis of what is an accident makes us to investigate the possibilities of avoiding it. This examination introduces timing constraint. The timing constraint depends upon the environment, the maximum speed with which a train can travel, and so on. This analysis introduces a timing constraint to be incorporated into the requirement.

Safety: The gate must be closed within 100 time units of detecting the arrival of a train.

3.7.3.2 Liveness Requirement

Liveness requirement, stipulates that 'good things' happen eventually [Lamport 83]. A system to be of use to its community must be live. For example a railroad crossing system, can achieve safety by closing the gate always. The system, to be of use must open the gate eventually, i.e., the gate must not remain closed for long. This requirement ensures that the system is live. The requirement that the gate eventually be raised, is only a qualitative requirement. If the system is to be of much service, then a quantitative requirement is needed, like that the gate be raised within 500 time units of the exit of the train. Thus real-time liveness criteria suggests temporal constraint.

Real-Time Liveness: The gate is never closed for more than 500 time units, after the exit of the train

It may be noted that Alpern *et al.* [Alpern 89] have proved that every property[20] is the conjunction of a safety property and a liveness property. The safety and

[20] Examples of the properties are: *partial correctness* (if precondition is satisfied then eventually postcondition holds good); *abortion freedom* (if precondition holds good, then eventually system

liveness properties belong to the environment (to the problem model), while other properties (mentioned in the footnote) belong to the solution model. Thus the properties that are of interest to the solution model are a derivative of the properties of the problem model.

Restrictions imposed by the safety and liveness requirement, refine the scenarios. Now we shall refer to the evaluation of our model.

3.8 Validation of the Requirements

So far the discussion surrounded the technique which provided a process of articulating the objectives and the needs of the system. After formulating the needs, the evaluation phase begins. The evaluation phase provides feedback on the user requirements. The scenario based technique discussed above, can be used both for generating the scenarios, and for validating it. The scenario-based technique is used to provide feedback on what the user thinks the focus is. This allows the user to change the focus if necessary.

does not enter a state where the program aborts); *total correctness* (in a finite program, if a precondition holds good then it satisfies some postcondition, and the final value of the program counter denotes the end of program); *normal termination* (in a finite program, if precondition is satisfied then eventually the state where it ends the program, is not the one where program aborts); *mutual exclusion* (a condition such that, two processes are not inside a critical region); *deadlock freedom* (a condition, where a process has entered such a state, where it has no enabled action, and no other process can alter that); *guaranteed service* (a request is serviced eventually); *first-come first-serve* (receive the request in the order of arrival); *starvation freedom* (if a process is enabled frequently enough, it will progress eventually).

The evaluation phase involves the review of the conceptual model (which symbolises the provisional understanding of the system i.e., the user solution of the system). This review focuses on the goal set, and the user solution to achieve the goal. The user may explain his/her needs by giving a solution. Such a review along with the user can uncover the unstated requirements which are known as 'mistakes' [Malhotra 80, Boehm 76]. Review of Malhotra's dialogue [Malhotra 80] suggests that a good portion concerns obtaining feedback from the customer - that the requirements engineer has understood some specific aspect of the problem. This study suggests that the problem definition and user solution are not independent activities, they are interrelated. This relationship between problem definition and the user solution is made clear with scenarios.

The scenarios provide a list of actions for a specific situation. The scenarios are fragmentary in nature. The fragmentary nature of the scenarios suggest that they play a significant role in stimulating the acquisition process, rather than relying on the predetermined information. Scenario description is more concrete, and this helps to understand and resolve the conflicts more quickly. This approach of problem definition/user solution seems to match the prototype development strategy. This approach is radically different from the serialised life-cycle approach, where it is unrealistically assumed that all the requirements are captured at the very beginning. From Malhotra's study it is apparent that during the requirements definition activity, unless the provisions are made to capture such solution elements, important information may be lost.

3.9 Discussion

The central activity during system development is requirements determination, whereby requirements are established. An analysis of the approaches suggested for requirements determination by Davis [Davis 82b] provides a central theme. The approaches are:

(1) Asking the user

(2) Derive them from the utilising system, that is, from an analysis of the needs of those who will use the system. This involves studying the work that users actually perform using interviews, observations, sample documents, etc.

(3) Derive them from an existing system - one that was previously installed as developed. An understanding of system requirements is obtained by reverse engineering.

(4) Evolve them through the process of prototyping. That is, by iterating through building -> use -> feedback -> modification of requirements -> building cycles of system development. Here the system itself is the requirement. This still evolves in the minds of the user and system developer as iteration progresses.

All these approaches rely on asking the user for information, although steps 1 and 2 are heavily dependent upon this. Rapid prototyping methods like PAISLey [Zave 82], or Gist [Balzer 82], allow the analyst to understand the system behaviour. Prototyping involves experimenting with problem solving. This means that the person responsible for prototyping must have a solution in mind, before s/he starts prototyping. Prototyping postulates a solution. The prototyping language does not portray the intended or expected behaviour by the user. While the scenario based technique describes the external system behaviour from the user's point of view. Scenarios describe the proposed use of the system. Scenario description involves environment and the controller.

Our approach is to analyse the objective and assign the role responsibilities to the objective. This helps in the identification of agents. An agent can be represented with twin views as shown in Figure 3.11. Responsibility view is the extrinsic view, it provides a description that stems from the use of an agent. While the behavioural view is the intrinsic view, it furnishes the behaviour that the agent is capable of producing. An agent has both an external representation, and an internal representation. Scenarios provide an interface between the two representations. This approach provides a tool of thought for both the requirements engineers, and specifiers. For stakeholders, an agent has a person view that accommodates a particular responsibility. While a specifier has a system view of an agent, that provides some functionality. For a customer the agents distribute responsibilities, while for a specifier the agents distribute functions. Thus agents support both responsibility, and functionality. This dual role helps to reveal any mismatch between customer's expectations, and requirements engineer's understanding.

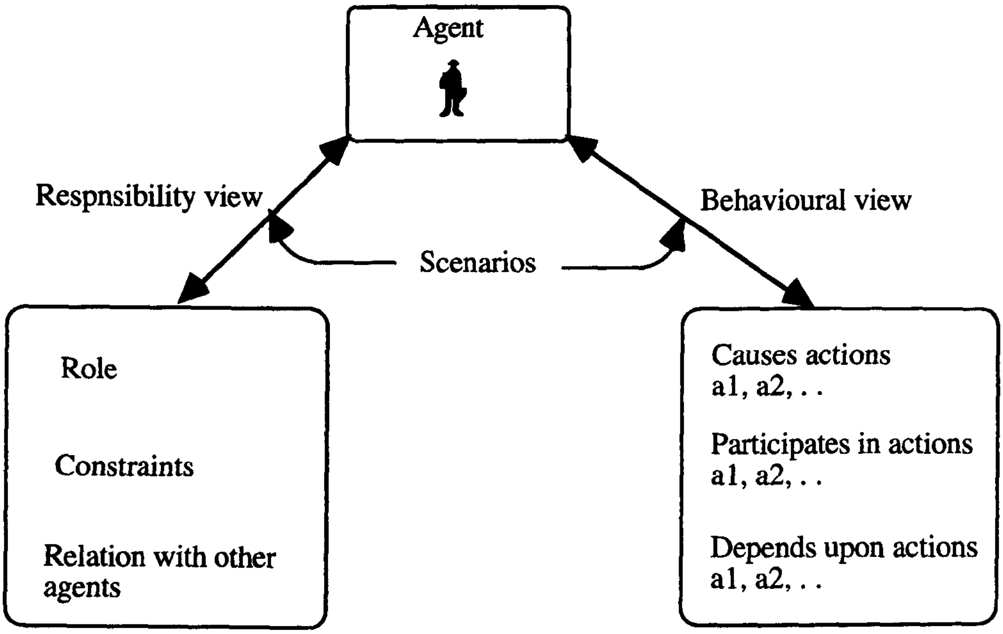


Figure 3.11 Twin views of an agent

In responsibility view the role is dominating. The role represents the mission of the agent. The mission of the agent is explained as scenarios by the users. The scenario characterises the responsibility as deemed by the user. While the behavioural view provides the functional view to achieve that mission. The functional view represents the behaviour to be portrayed by an agent. This distinction is outlined in Figure 3.11.

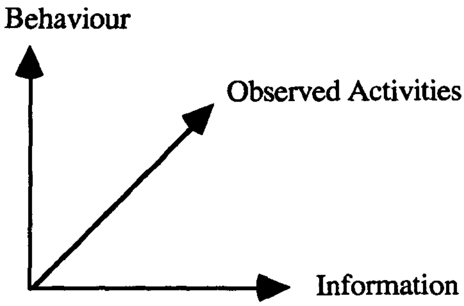


Figure 3.12 Dimensions of a scenario

In Figure 3.12, information represents the needs, objectives, and desires as regarded by the user. The behaviour is what the system adopts. Behaviour arises out of time ordered observed activities. The observed activities provide a qualitative description of how the agent behaves. This description is grounded in the real world. Requirements description based on the real world features are transparent and easy to understand.

3.91 Summary

Here we suggested a five layered approach during the requirements stage (seen in Figure 3.2), as its first purpose to develop an understanding of the problem domain, as its second purpose to develop a user understanding of the objectives that enter the overall system - which guides the identification of agents, as its third purpose to develop an understanding of the agents co-operation to provide the required objective, as its fourth purpose to develop an understanding of the specific use of the system, and as its last purpose to develop an understanding of the safety critical aspects of the system. The model is refined later with stakeholders. A formal view of the model is necessary to aid the analysis. A formalism specifies a class of objects under discussion in an unambiguous and general manner [Zeigler 84]. A formal view of the model will be discussed in the next chapter.

Chapter 4

Time-Constrained Automata Model

A system often consists of several agents, and these agents are time-constrained. We introduce time-constrained automata to model the dynamic nature of an agent, which needs to evolve over time. This is achieved by enriching the elements in the domain, with an explicit time component. This model describes both functional, and temporal restrictions using the same framework. A real-time system is viewed as a set of interacting automata, each automaton representing an agent in the system.

4.1 Introduction

The objective of a model is to represent an abstract knowledge about a universe of discourse [ISO 87]. In the preceding chapter we discussed an approach to derive the conceptual model with dynamic properties. Real-time systems are time-sensitive, and necessitates dynamic properties to be modelled. A real-time system often consists of several agents. An agent is characterised by the important incidents that occur. An incident is an abstract representation of the chunk of information handled by an agent. An incident provides a dynamic instance of the description and is called as an event. Thus an event is an assertion about some behaviour parameter of an agent. Now we can visualise a scenario as a sequence of events that accomplishes a mission. This style of description is oriented towards activities occurring in the user's world. Thus an event model provides a context in which the requirements are abstracted as observable effects on the domain. In the following sections, an event-based model is discussed.

4.2 Characteristics of an Event

Traditionally the behaviour of a system is captured by continuous variables modelled by differential equations [Kuo 67, Ogata 90]. A system can be modelled by symbolic changes of a system rather than as changes in the numerical values (as modelled by continuous variables). Such a model identifies the important events that occur in a system. We think of a system in terms of events. The notion of observation is crucial to the philosophy of event-based models. An event refers to observable information. A system is modelled by such discrete-events. The events are discrete in the sense that it is assumed to occur instantaneously. A discrete event system, is a dynamic system that evolves with discrete events which occur at unknown irregular points of time. For such reason the model is also called as discrete event dynamic model.

Events of a system are identified in principle by the independent observation of the system. An observer recognises the interactions that take place in the system. An observer cannot influence the system in any way. The dynamics of a system is understood by the events it is associated with. Event conveys different information, for example an event like 'temperature exceeds 50 degrees', defines a dynamic change in an ongoing process. While an event like 'temperature set point modified to 25 degrees' characterises an operation. An event may characterise an environmental operation, or controller operation. Events are firmly associated with the evolution of the system. Events are instantaneous and mark a point in time. Continuous events which have a duration are represented by two atomic events like start of the event and the end of that event. By atomicity we mean that the events are indivisible. Each event has a unique name. Event model allows a system to be described without referring to its internal operations. As noted earlier, an event may refer to an operation from the environment or from the controller. To an observer ongoing activities are the flow of events from and to the environment. Event-based models are advocated by [Hoare 85], or in control systems by [Inan 88], and in hardware by [Snepsheut 85]. These models do not explicitly deal with functions of time. In our model, we explicitly deal with the function of time. We also consider non-terminating interaction of reactive systems.

4.3 Event-Based Model

An event based model sets up the basic abstraction of a system. Event based model is a tuple < E , T > where E is the event set and T is the time base set. In an event based model the concept of event is central.

4.3.1 Event set

Event set represents the incidents that can take place in a system. These events are observed over a time base T. Time base T provides a chronological pattern to the events occurring in a system. Event based model consists of events and their relations [Lamport 78].

Definition: An event is an instantaneous atomic instance of the description in a system.

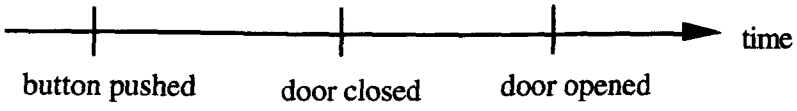


Figure 4.1 Ordering the events

For example the incidents like button pushed, door closed, door opened are the events. These events can be ordered depending on the time of their occurrence as shown in Figure 4.1, where 'button pushed' happens before 'door closed' and so on, or in other words 'button pushed' precedes 'door closed' and so on. This precedence relation is a relation between two events and denoted by symbol <. Thus an event structure is represented by (E, <).

The distinctive property causality (that there is no effect without a cause) can be represented as

$$\forall e \exists e' \ e' < e$$

The binary relation < is irreflexive, transitive, and anti symmetric[21].

Events with duration are modelled using two atomic events marking the beginning and the end of the event (with duration). □

4.3.2 The Perspective of Time

Time is thought in terms of points or intervals [Benthem 91]. We refer to time as a non-empty set T, consisting of objects called time-elements. With a point perspective, time consists of a series of time points, like bullets triggered continuously from a gun. These time points are duration-less. Traditionally time-points are regarded as the basic elements, and time-interval a derived concept [Koymans 92]. An interval can be regarded as a series of time points. Relations between time intervals proposed by Allen [Allen 83] provide a useful mechanism to think in intervals. With intervals, it is very difficult to state complicated timing constraints that arise in an application [Alur 90, Stokes 91]. Also it is very difficult for the user to interpret the relations between the intervals. Time points provide a viable mechanism to represent any complicated timing constraints. The relation between the time points is straightforward. Both the environmental and controller dependent timing constraints can be clearly stated. For such a reason we make use of the point structure of time.

[21] These properties are explained below, with time set.

4.3.3 Point Structure of Time

A point structure P is an ordered couple (T, <) where T is a non-empty set of ordered points, with a binary relation < on T. This ordering notion has the following properties.

Irreflexivity (i.e., a time point cannot precede itself)

IRREF $\forall x \; \neg x < x$ and

Transitivity

TRANS $\forall xyz \; (x < y \wedge y < z \rightarrow x < z)$. From these two conditions the condition of asymmetry follows.

ASYM $\forall xy \; (x < y \rightarrow \neg y < x)$

For any two time points, either one precedes the other or they are the same point. Thus linearity is

LIN $\forall xy \; (x < y \vee y < x \vee x = y)$

Each time point has a neighbouring point in past and future, and this implies a succession property

SUCC $\forall x \exists y, \; y < x$ (past), $\forall x \exists y, \; x < y$ (future)

The point time structure gets classified into dense time structure or discrete time structure, depending on whether we assume an infinite divisibility between two points or not.

Thus with infinite divisibility we have

DENS $\forall xy\ (x < y \rightarrow \exists z,\ x < z < y\ )$

and using a stepwise succession we have

DISC $\forall x\ (\exists z(x < z \wedge \neg\exists i\ x < i < z)),$

$\forall y\ (\exists z(z < y \wedge \neg\exists i\ z < i < y))$

A dense time structure observes IRREF, TRANS, LIN, SUCC, DENS and a discrete time structure observes IRREF, TRANS, LIN, SUCC, and DISC.

The two types of models that originate from the point structure of time are discrete and dense time. In discrete time model, time increases in steps. This is familiar to the number system N (natural numbers). Dense time model is familiar to the number system R (non negative real numbers)[22].

[22] It may be noted that, the number system Q (the rational numbers) is dense, but not continuous, while R (the non-negative real numbers) is both dense and continuous.

4.3.4 Need for Dense Time

Real time systems operate in intimate coordination with its associated physical systems. These operating domains progress at widely separated time scales. Independent events may appear arbitrarily close together in time. Such events cannot be faithfully modelled with discrete time. With discrete time, time increases in steps. When time increases in a stepwise succession, a prior commitment to a quantum of time is needed. If time quantum chosen is τ then the time points (x) that can be studied are

$$\forall x \ \exists n \in N, x = n \tau.$$

After choosing a time quantum intermediate points cannot be studied. If time quantum is chosen as 1, then event sequence consisting of a, b, c, d can be represented as

(a , 1) , (b , 2) , (c , 4) , (d , 5)

In the above case if b occurs at 1.1 then it will be denoted as time(b) = 2, this limits the expressiveness. Also in modelling realm, explicit reference to discrete time can be made redundant, by adding null events (Ø) to mark the passage of time. The above timed sequence can be represented as { a , b , Ø , c , d }, where timing is implicit[23].

The argument for dense time can be summarised as below [Joseph 92]

[23] Such an approach is used in linear time temporal logic, with the repetitive use of next operator.

1. The independent events in a distributed computation may appear arbitrarily close together in time, and so time must be represented in dense domain.

2. Physical processes are modelled with time in a continuous (real) domain, so programs that interact with physical processes must represent time in a similar way.

4.3.5 Timing Axioms

A time sequence T consists of infinite sequence of time values, and satisfies the following constraints:

Progress: time value strictly increases.

This states that time never decreases.

Non-Zeno Property: Between two time values, there is never an infinite number of time values. □

This rules out the possibility of representing an infinite number of computations in arbitrarily small time. Such machines which perform infinitely many computations in finite time are called as Zeno machines [Witrow 80]. Zeno machines are hyper arithmetical [Joseph 92, Kurki-Suonio 94]. As, such systems are non existent the above axiom rules out such behaviour.

4.3.6 Timed Event

A timed event associates a time parameter with an event. It is expressed as (button_pressed , t1) where t1 is the timing parameter associated with the event button_pressed. Time parameter t1 marks the occurrence of event button_pressed.

Definition: Given a set of events E and a totally ordered time set T, then a timed event is a pair of an event and a time point $(e_i , t_i) \in E \times T$ where $e_i \in E$ and $t_i \in T$. □

4.4 Abstract Model of a Real-Time System

As noted earlier real-time systems are reactive systems, and in this respect differ from transformational systems. A transformational system accepts input, performs transformations on them to produce output as shown in Figure 4.2(a). Transformational systems prompt the environment for additional required inputs, while reactive systems are prompted by the outside world. Reactive systems are interactional systems, as shown in Figure 4.2(b). As Pnueli [Pnueli 86] expresses, "reactivity characterises the nature of interaction between the system and its environment. It states that this interaction is not restricted to accepting inputs on initiation and producing outputs on termination. In particular, it allows some of the inputs to depend on intermediate outputs". In this respect we can notice a subtle link between reactivity, and distributivity or concurrency. Concurrency or distributivity refers to an internal organisation of the system, and a component in a concurrent system should always be viewed as a reactive component. This is because typically a component in a concurrent system maintains a reactive interaction with other components in a system. Thus a component is studied in terms of the interaction it maintains with the other components. In Chapter 3 we modelled a real-time system as a combination of a number of concurrently acting agents (components). In the following section we discuss an abstract view of this model.

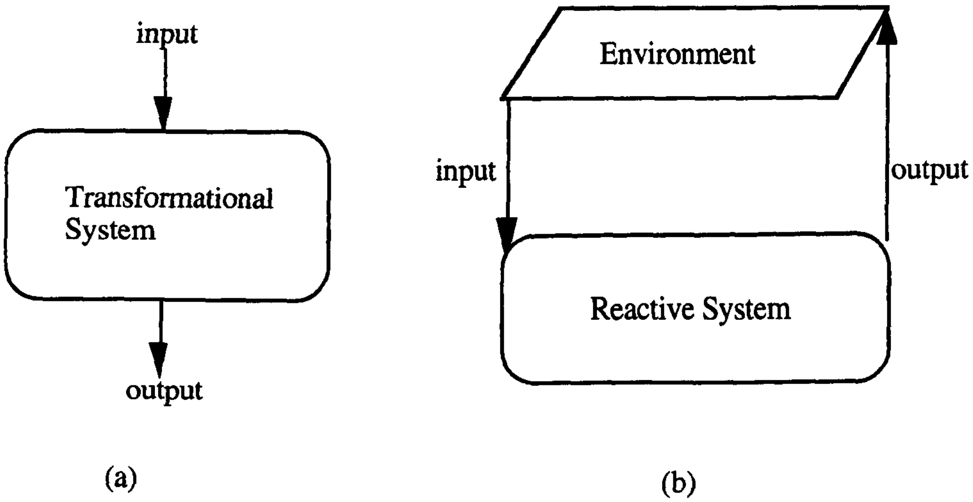


Figure 4.2 (a) Transformational system (b) Reactive system

One of the often suggested approaches is the use of finite-state automata. Since the unique feature of real-time system, is its ability to deal with infinite computations, it is appropriate to consider automata over infinite sequences (ω - automata).

4.4.1 ω - Automata

The theory of automata is a foundation stone for computer science. We recall some of the well known concepts of classical automata [Hopcroft 79]. The concepts of ω-automata are not so well known, and are found in articles[24] [Büchi 72, Choueka 74, Hoogeboom 86, Thomas 81]. The theory of ω-automata are based on

[24] The concepts of ω - automata can also be found in [Eilenburg 74].

the theory of finite automata. We develop the theory of ω-automata with the classical automata.

An alphabet Σ is a finite nonempty set. The elements of an alphabet Σ are called as letters or symbols. We refer to a finite sequence of letters as a *word*, and to an infinite sequence of letters as an *infinite word*, or ω-word. As usual, ε denotes the empty word, Σ^* represents the set of all finite words over Σ, Σ^+ denotes the set of all nonempty finite words ($\Sigma^+ = \Sigma^* - \{\varepsilon\}$), and Σ^ω denotes the set of all infinite words.

An ω-word u over Σ, is an infinite sequence over Σ, written in the form ($u = u_1, u_2, u_3$ ....). The set of all ω-words over Σ is denoted by Σ^ω.

Definition: If Σ is an alphabet then an ω- word over Σ is a mapping from N into Σ, where N denotes the set of nonnegative integers.

Consider a word w. We use $|w|$ to define the length of w. For a word $w \in \Sigma^*$, we let w_i be its component at the (i)th position, if $1 \leq i \leq |w|$. The *concatenation* of a word $w \in \Sigma^*$, with the symbol $\beta \in \Sigma$ is represented by $w \cdot \beta \in \Sigma^*$. A word $v \in \Sigma^*$, is a *prefix* of w, if $|v| \leq |w|$, and $v_i = w_i$ for $1 \leq i \leq |v|$.

For a finite word $\sigma \in \Sigma^+$ and an infinite word $\sigma' \in \Sigma^\omega$, we denote by σ (σ' the fact that σ is a proper finite prefix of σ', i.e., a prefix that differs from σ'. We can note the relation σ (σ', requires σ to be finite. The word $\sigma \cdot \sigma'$ is obtained by concatenating σ' to the end of σ. The concatenation ($\sigma \cdot \sigma'$) is defined only if σ is finite. □

For example, consider a sequence of even numbers. The property of even numbers defines a characteristic set, such that the elements of the sequence, s

$(x = 2), (x = 4), (x = 6), \ldots$

belongs to even numbers.

While the sequence, r

$(x = 2), (x = 4), (x = 5), \ldots$

does not.

Recalling the discussion in Chapter 3, a scenario defines a sequence of actions (events) for a particular situation. If events are modelled as symbols of the alphabet, then a scenario is a word. A system comprises of a set of scenarios, or a set of words. Another name for such a set is a *language*. A *language* is a set of words from an alphabet. An ω - language consists of ω-words. Thus an ω - language over an alphabet Σ is a subset of Σ^{ω}.

In the abstract model considered, a property judges some sequences to be acceptable (follow the property), and other sequences to be unacceptable (those that do not have the property).

A *finite automaton* is a five tuple $A = (\Sigma, Q, Q_0, \delta, F)$ where Σ is a finite alphabet, Q is a finite nonempty set of *states*, $\delta \subseteq Q \times \Sigma \times Q$ is a set of transitions, $Q_0 \subseteq Q$ is the set of *initial states*, and $F \subseteq Q$ is the set of *final states*. A is said to be *deterministic* iff $|Q_0| = 1$, and for all $q \in Q$ and $a \in \Sigma$, $|\{q\} \times \{a\} \times Q| \leq 1$.

If automaton is in state q, then it can move to q', while reading a symbol. Thus an automaton moves to various states while reading a word from its initial state. A word x is accepted by A iff there exists $q_0 \in Q_0$, and $q_f \in F$, and a path $q_0 \xrightarrow{x} q_f$.

A *run* is the sequence of states that an automaton occupies while reading a word. A *run* r of automaton A on a word x, is such that

(a) $r[1] \in Q_0$, and

(b) for all i, $1 \leq i \leq |x|$, $(r[i],x[i],r[i+1]) \in \delta$

A run r on the word x is *accepting* iff

(c) x is finite, and $r[|x|] \in F$

The automaton A that satisfies the condition (c) is the classical FSM (finite state machine).

4.4.1.1 Acceptance of Infinite Words

In the automata A considered above if x is infinite, then a run of A over x, consists of some state from the set F repeating infinitely often along r (Büchi acceptance[25]). In other words, a run r of A over a word $x \in \Sigma^\omega$ is an accepting run iff $Inf(r) \cap F \neq 0$ (Büchi acceptance).

Thus a run r on the word $x \in \Sigma^\omega$ is *accepting* iff

[25] In the literature various types of ω-automata are studied. Here we consider only Büchi automaton.

(d) x is infinite, and *Inf* (r) $\cap$ F $\neq$ 0.

The automaton A that satisfies the condition (d) is the Büchi automaton.

4.4.2 Büchi Automata

Recalling the above discussion, in Büchi automata a run is accepted if the intersection of the infinite set of run with that of accepting states is not empty i.e., Inf(r) $\cap$ F $\neq \emptyset$. Inf(r) is the set of automaton states that appear infinitely often in a run of the automaton over a given word, and F $\subseteq$ Q, is the set of accepting states.

An ω-language acceptable by Büchi automaton, can be constructed from the language acceptable by finite state machine. This is explained below.

Infinite *behaviour* of the automaton A, denoted as Beh_ω (A) is the set of all the labels of the run starting in q_0 and going infinitely often through the set F. The family of all acceptable subsets of Σ^ω is denoted as $R(\Sigma^\omega)$. Similarly finite behaviour of automaton A is denoted by Beh_* (A), and the family of recognisable subsets of Σ^* by $R(\Sigma^*)$.

Theorem: An ω-language $L \subseteq \Sigma^\omega$ is Büchi recognisable iff L is a finite union of sets of $U.V^\omega$ where U and V $\subseteq \Sigma^*$.

Given U and V $\in R(\Sigma^*)$ and $W = U.V^\omega = L(A)$

The word W is a concatenation of two words U and V^ω

or W = Beh_ω (A)

The labels on the run of the automaton A consists of the word U followed infinitely by V. This exhibits the path of the automaton, which starts in initial state $q_0 \in Q$, and then moves to a final state $f \in F$, and then keeps looping back to the same state.

The word U takes the automaton from the state q_0 to f, and this can be represented as $U_{q,f} = (Q, q_0, f)$. After the automaton moves to the state f, the automaton is made to revisit the same state infinitely often by V, and this can be represented as $V_f = (Q, f, f)$.

$$W = \bigcup_{q \in Q_0} \bigcup_{f \in F} U_{q,f} V_f^{\omega}$$

Conversely U and V $\in$ R(Σ*) we can build an automaton A such that $U.V^{\omega} = Beh_{\omega}(A)$.

4.4.3 Timed Scenarios

In the above discussion, we considered untimed language i.e., set of untimed words. The above formalism is sufficient to consider the untimed scenarios. In the earlier chapter, we argued the need for a temporal reasoning. For example, in a cat and mouse problem, the cat after observing the mouse, must catch it within a couple of seconds, if not the mouse will vanish. This is a real-time scenario. To incorporate the temporal reasoning, we introduced the notion of a timed event (an event associated with a time element). In essence a real-time scenario is a sequence of timed events, i.e., a timed word.

Definition: A finite timed word is a finite sequence of timed events.

In the earlier section, we discussed the need for a dense-time domain. In this section, we shall see how temporal reasoning can be incorporated.

4.4.4 Technique to Represent the Timing Constraint

Most of the temporal representations suggested in the literature fall into one of the two forms:

> (1) delaying a transition for a finite time (same as the one suggested by Ramchandani 74); or
>
> (2) constraining a transition for a lower bound time, and an upper bound time. Here a transition is delayed for a lower bound time l, and constrained to occur within an upper bound time u. (This is the same as the one suggested by Merlin 76).

Both these formalisms consider a temporal constraint as a restriction over one symbol. This concept arises from the inherent stimulus-response mechanism, where response is considered as a symbol. In practice, as we argued earlier[26], timing constraint may involve temporal restriction over several events. For example, in the Figure 4.3, the time of occurrence of symbol d, is constrained not only by the time of occurrence of symbol a, but also by the time of occurrence of symbol c. Such temporal constraints cannot be stated in the formalisms mentioned earlier. For such a reason we make use of the concept of multiple clocks mentioned in [Alur 92].

[26] We argue this further in Chapter 5, and present an alternative tool for stimulus-response mechanism.

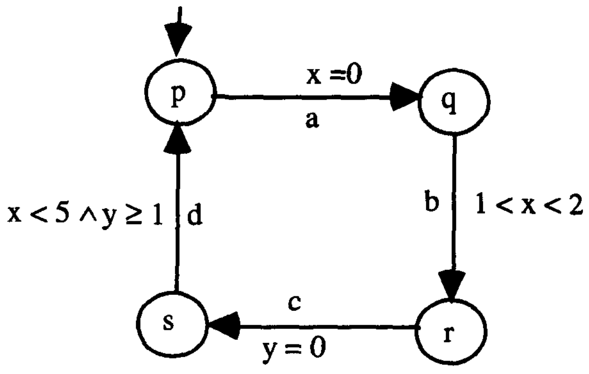


Figure 4.3 Timing constraint over many events

4.4.5 Multiple Clock Paradigm

Here a finite number of clocks are used to represent the temporal requirements. Each clock is initialised (set to zero) before it is used. These clocks are fictitious clocks which are used like stop watches.

Timing constraints are stated, as a constraint over the clock. The reading of a clock at any instant equals the time elapsed since the clock was initialised. A clock can be initialised over a transition, and timing constraint is stated over transition. For example in Figure 4.3, automaton A starts in state p, and moves to state q with the occurrence of symbol a. The clock x gets initialised along with this transition. The value of a clock always reads the time elapsed since it was reset. When the automaton is in state q, the clock x reads the time elapsed since the symbol a occurred. The transition from state q to state r occurs if symbol b occurs while the value of this clock x is within (an upper bound of) two units of time and a lower bound of one unit of time. The transition from state r to s occurs with the arrival of symbol c. The clock y is initialised to zero along with the transition from state r to state s. Similarly the transition from state s to state p occurs, if symbol d occurs while the value of clock x is less than five units of time, and the value of clock y is greater than or equal to one unit of time.

The automaton can make a transition if the values of associated clocks satisfy the enabling condition. The transitions are instantaneous i.e., transitions between states take zero time. A state in a timed automaton represents the state of the automaton and the values of all its associated clocks. Thus state is a pair $(q, \bar{x})$ where $q \in Q$ is a location of the automaton and $\bar{x} \in \mathbb{R}^n$ is the value of its associated clocks. For time values $d_i \in \mathbb{R}$ the transition can be represented as $(q,\bar{x})\xrightarrow[(a,d)]{}(q',\bar{x}')$. A run r for a timed automaton is an infinite sequence of states $q_i \in Q$, clock vectors $\bar{x} \in \mathbb{R}^n$ and time values $d_i \in \mathbb{R}$.

The runs of timed automaton is in correspondence with the runs of the normal automaton. For example, say an observer is watching the transitions that take place in Figure 4.3. According to the observer, if timed word is (a, 4), (b, 5.5), (c, 6), (d, 8) then with the values of clock it can be denoted as

$$(p,[0,0]) \xrightarrow[(a,4)]{} (q,[0,4]) \xrightarrow[(b,5.5)]{} (r,[1.5, 5.5]) \xrightarrow[(c,6)]{}$$

$$(s, [2,0]) \xrightarrow[(d,8)]{} (p, [4,2])$$

The automaton A makes use of two clocks x, and y. All the states associated with A (i.e., state p, q, r, and s) is associated with two clocks. Initially at p, the value of both the clocks is zero, and is represented as p[0,0]. The first transition is noticed as (a,4). This increases both the clocks by 4 units of time (i.e., x = 4, y = 4), but the clock x is initialised to zero over the transition, so the value of clocks at location q, is (x = 0, y = 4) and represented as q[0,4]. Similarly all the other transitions modify the clock values as shown above.

Formalism of Clock Constraints

As shown above, the clock constraint is allowed on the transition. A simple clock constraint compares a clock value with a time constant. A clock constraint is such Boolean combinations of simple clock constraints.

Definition: If X is a set of clocks, the set $\Phi(X)$ of clock constraints γ is defined inductively as follows (c is a time constant, and x is a clock in X) :

$$\gamma = x \leq c \mid x \geq c \mid x < c \mid x > c \mid x = c \mid \neg\gamma \mid \gamma_1 \wedge \gamma_2 \quad \square$$

4.4.6 Timed Büchi Automata

The formalism of timed Büchi automata is as given below [Alur 92]. Timed Büchi automata (TBA), provides a temporal reasoning with dense-time domain. The formalism can incorporate all the temporal representations. A timed Büchi automaton (TBA) is a 6-tuple $A = (\Sigma, Q, Q_0, C, \delta, F)$, where Σ is an alphabet, Q is a finite set of states, C is a set of clocks (for example $\{x_0 \ldots x_n\}$), Q_0 is a set of start states and $Q_0 \subseteq Q$, δ gives the set of transitions, denoting $\delta \subseteq Q \times \Sigma \times 2^C \times \Phi(C) \times Q$, and F is a set of acceptance states $F \subseteq Q$. Each transition might reset a clock and has an enabling condition expressed as a constraint on the values of associated clocks.

4.4.7 Related Information

Büchi automata are popular in temporal logic also. Given a formula in linear temporal logic, it is possible to construct a Büchi automaton that accepts those infinite sequences that are models of that formula [Clarke 86]. This relationship has been exploited in temporal logic for verification purposes, to show that an implementation meets the specification. This is termed as temporal logic model checking. Model checking is an effective method to prove that a concurrent program satisfies a temporal logic formula [Vardi 86]. In this approach the specification of a system is defined in temporal logic formula. Then the Büchi automaton extracted from this formula is checked for containment with Büchi automaton obtained from the implemented system. The automata are tested for containment by checking their languages [Kurshan 87]. If automaton A accepts the language L(A) and B the language L(B) then to test $L(A) \subset L(B)$, the complement of automaton B i.e., B', is constructed, and the language produced by the product automata is tested for emptiness i.e. $L(A * B') = \emptyset$. The complexity of this approach is atleast as complicated as finding the complement of the automaton and is commonly known as emptiness of the complement problem. The complementation construction is presented in [Sistla 87].

4.5 Modelling an Agent

In the above sections we discussed the formalism of timed scenarios, and the timed automata that is capable of representing such properties. As discussed earlier, an agent is characterised by scenarios. Thus each agent is represented by a timed automaton. In terms of the event model discussed, each symbol refers to an event. As discussed above the behaviour of an automaton is set of timed event sequence. We shall consider an example.

Example: Consider an agent performing a communication by sending and receiving the messages in an environment. We assume that the agent sends a next message, only if the previous message had been received. The sending and receiving of the message is abstracted by events 'send', and 'receive' respectively. Then the possible behaviour is

(send, t1), (receive, t2), (send, t3), (receive, t4) · · ·

To discuss the event model at the right granularity, we consider the concept of process[27]. We can characterise the behaviour of an agent as a process. A process is defined in terms of a set of timed events, and a set of traces. A trace is a finite sequence of timed events. In the concrete model a trace refers to a scenario.

Definition: A trace is a finite sequence of timed events. □

For example, If events a and b are in A, then the trace (a,t1), (b, t2) is a sequence of two timed events. An empty trace, that is a sequence of no events is denoted by <>.

In the following section we formalise the definition of a process.

Definition: A process essentially has two meanings (1) to define all possible events, and (2) to define the behaviour of a process. Thus,

a process P is a pair (αP, traces(P))

where αP is the set of events that the process P is characterised by, and traces(P) is the set of all traces that P can engage. □

[27] Further discussed in Chapter 5.

In chapter 3, we regarded a real-time system as a web of concurrently acting agents. Here we formalise the composition of agents.

4.5.1 Composition of Agents

A real-time system is an arrangement of agents. Processes are used to describe the behaviour of the agents. The behaviour of an agent is asserted by defining a process. A process consists of sequence of timed events that the real system may engage. As we studied in the previous chapter these agents are reactive by themselves. Thus system behaviour is described as a parallel composition of agents. Here we define the parallel composition of two processes.

4.5.1.1 Modelling the Composition

The parallel composition (||) of a set of processes describes the joint behaviour of all the processes running concurrently. The rendezvous between the two processes can be modelled either by means of shared action, or by means of communicating action (as in the case of CSP). In CSP two processes are connected by a channel. If a process say, M1 wants to communicate with M2, then M1 performs a 'send action'. This 'send action' will not be executed, until M2 performs a 'receive action'. The concept of channel, message transmission, and reception mechanism, can influence the implementation process. A conceptual model must represent abstractions. We do not make use of the concept of communicating action. The synchronisation between the two processes is achieved by means of shared event - i.e., the two processes share the same event label. Such a rendezvous is called shared action.

Here processes synchronise via common events, and concurrency is modelled by all possible interleaving of the events. For example in a manufacturing plant, consider a situation where a robot bends a pipe, and places it on the conveyor belt. The robot, and the conveyor belt are two independent processes. The robot can place the 'bent pipe' on the conveyor belt, only if it is ready to do so, and the belt is ready to receive the 'bent pipe'. Thus the event 'bent pipe' requires simultaneous involvement of both the processes. This suggests that the event 'bent pipe' is in the event set of both the processes. In other words, the event 'bent pipe' is a possible event in the independent behaviour of both the processes.

4.5.1.2 Formalising the Composition

The parallel composition of two processes P and Q is denoted as P || Q. To help us in the definition of parallel composition, we define an operator $\uparrow$ such that, an expression $(t \uparrow B)$ denotes the restriction of traces t, to the set of events B, and is equal to the trace t with all events outside B omitted.

Given the process P and Q, the parallel composition of the two processes, denoted as P || Q is defined by

$$\alpha (P \parallel Q) = \alpha P \cup \alpha Q$$

$$\text{traces}(P \parallel Q) = \{t \mid (t \uparrow \alpha P) \in \text{traces}(P) \wedge (t \uparrow \alpha Q) \in \text{traces}(Q) \wedge \alpha\{t\} \subseteq (\alpha P \cup \alpha Q)^{\omega}\}$$

Processes P and Q execute in parallel and synchronise on common events. For example, P || Q can execute an event a, if process P and Q simultaneously execute a, or if one of these processes, say P executes a, and a is not in the event set of Q.

An Example of Parallel Composition

Consider a process P with its behaviour as alternating the events a, and b respectively, such that symbol b arrives after one time unit of the occurrence of symbol a, and the alternating a arrives at a fixed length of 3 time units. The trace representing this behaviour is

$$(a, t) \quad (b, (t+1)) \quad (a, (t+3)) \quad (b, (t+4)) \cdot \cdot \cdot \cdot$$

Now consider another process Q connected to the above process P. Process Q sends symbol c after receiving the symbol b. The time delay between the symbol b and c, is one time unit. The trace representing this behaviour is

$$(b, (t+1)) \quad (c, (t+2)) \quad \cdot \cdot \cdot \cdot$$

Then the parallel composition of two process P and Q has a unique timed trace:

$$(a, t) \quad (b, (t+1)) \quad (c, (t+2)) \quad (a, (t+3)) \quad (b, (t+4)) \quad (c, (t+5)) \cdot \cdot \cdot \cdot \blacksquare$$

We think of two processes, as two automata M_1 and M_2. Then the parallel composition of M_1 and M_2, denoted as $M_1 \| M_2$ is given as follows.

As we noted in the earlier section, here a symbol is a timed symbol, i.e., a symbol associated with its time of occurrence. (The notation P ~ Q is used to represent the set of elements in P, but not in Q)

$$M_1 \| M_2 = (Q_1 \times Q_2, A1 \cup A2, C1 \cup C2, f, (q_{01}, q_{02}))$$

where $f((q_1, q_2), (a, t)) = (f_1(q_1, (a, t)), f_2(q_2, (a, t)))$ if $a \in A_1 \cap A_2$

$= (f_1(q_1, (a, t)), q_2)$ if $a \in A_1 \sim A_2$

$= (q_1, f_2(q_2, (a, t)))$ if $a \in A_2 \sim A_1$

= undefined otherwise.

Here $A_1 \sim A_2$ is the set of elements in A_1 that are not in A_2. Thus $M_1 \parallel M_2$ can execute an event $a \in A_1 \cap A_2$, if both M_1 and M_2 execute a at the same time, or if only one of the machines executes a, and a is not in the event set of the other machine, or the time of occurrence of a is different.

The parallel composition ($\parallel$) of a set of processes describes the joint behaviour of all the processes running concurrently. Here processes synchronise via common events. The serial product of automata is used for a long time to formalise the parallel composition [Lamport 89, Merlin 83, Arnold 94, Lustman 94]. It can be noticed that the binary operator $\parallel$ is associative and commutative, i.e., $M_1 \parallel M_2 = M_2 \parallel M_1$, and $(M_1 \parallel M_2) \parallel M_3 = M_1 \parallel (M_2 \parallel M_3)$

4.7 Summary

The purpose of our model is to capture the user model of the ongoing activities of a system. User model of a system is narrated in principle by the independent observation of the system. Event model provides a succinct approach to model the dynamic nature of the systems. The notion of event covers all the incidents that are of interest. As our interest is in discrete systems, we assumed that the event occurrences have no duration, i.e., it marks a point in time.

Real-time systems often consist of several agents. A system is regarded as a composition of concurrently acting agents. We discussed the formalism for the parallel composition. The model accommodates both the functional and temporal aspects in the same framework. In the next chapter, we discuss the language to allow for the easy expression of the requirements.

Chapter 5

Timed Requirements Language - TRL

During requirements, active participation by the users is essential. Active participation by the users is possible only if the requirements descriptions are understandable. TRL has simple constructs, and promotes a descriptive method. TRL has a number of novel features including the treatment of causality, and the description of static, and dynamic constraints all integrated into one uniform framework. An approach to model the controller, and environmental actions is discussed. A generalised classification of the timing constraints is provided.

5.1 Introduction

According to Freeman [Freeman 87, Chapter 5] the main teething problems that arise during the development of complex system are: gathering of the information about problem domain, and its representation. These problems are interdependent, in the sense that the way in which we try to represent the requirements, influences our ability to gather the requirements. As Guinan and Bostrom [Guinan 86] express:

> The process of information requirement determination requires effective communication between system analysts and users of the system to be developed. The analysts ability to discover user requirements is partially determined by the analyst's familiarity with and ability to communicate in the user's domain of knowledge and discourse.

In this statement, the first part concerns acquiring the information, and the latter part on its representation. Both these aspects stress the involvement of users. The approach developed in Chapter 3 focused on the acquisition of requirements, and involved the users. In this chapter, we discuss the representational aspects, and introduce the language - TRL to represent the requirements.

In TRL, requirements are represented in the terminology of the user. As we discussed in Chapter 3, requirements of a system evolve over time. We learn more about the requirements, as our understanding of the environment improves. This understanding is further refined or put to test in discussion with the stakeholders. The requirements undergo refinement, before it approaches towards agreement. For such a reason, the stated requirements must involve the active participation by the users. Active participation by the users is possible only if the descriptions are understandable by the users. Understandable descriptions also help in the modification of the requirements. TRL promotes a descriptive method. TRL emphasises understanding what takes place and when it takes place in the system. Here a system is modelled in the terminology of the user. Such a model provides a description of the operational behaviour of the system. An operational explication is a problem oriented system description. Requirements description based on real world models are transparent and easy to understand. TRL is designed to facilitate the easy description of operational behaviour of systems. TRL is event driven, and provides constructs for the determination of timing constraints.

Leveson [Leveson 86] observes that, the greatest problems associated in software engineering, are due to the computer system being treated merely as stimulus-response system. We describe an approach to describe the real-time systems by their intended goals (missions). The approach discussed here, describes the mission of the system as conceived by the user. The missions are the features that the customer envisages. We also notice the limitations of the temporal classification provided in [Dasarathy 85], and provide a general classification of timing constraints of real-time systems. It is very well known that the temporal requirements cannot always be guaranteed. We emphasise the need for timing exception handlers in the representational languages, and provide suitable constructs in TRL.

5.2 Basic Premises

The descriptions of the requirements of a system is defined in terms of the observable events. The observable events include, electro-mechanical signals to control the apparatus, the actions taken by humans, and the environmental actions. The system requirement is expressed through such observable events. These observable events are ideally described through some enumeration of a list of events that achieve some mission. The temporal requirement of the mission can be provided over this flow of events, like at what time a particular event has to occur, and so on. This flow of events furnishes the system behaviour to achieve a desired mission in-time. Such a flow is depicted in Figure 5.1. An event is significant for describing the required behaviour of the system. As remarked earlier an event may refer to the controller, or the embedding environment, or to the interactions among them.

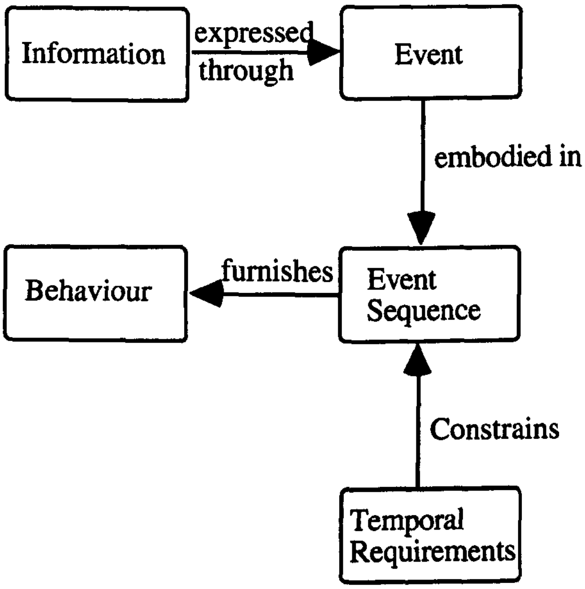


Figure 5.1 Behaviour in TRL

However a list of events alone could not provide a comprehensive description of the system. A system description is naturally done at the right granularity. The granularity of events is too fine. In order to make the requirements description comprehensive, the overall description, corresponds to the description of the processes. A process provides the behaviour of an agent that constitute the system. As shown in Figure 5.2, a process simply consists of sequence of related events. A process is composed of activities. An activity incorporates a set of tasks to be completed. Tasks are the smallest units of work, and consists of events.

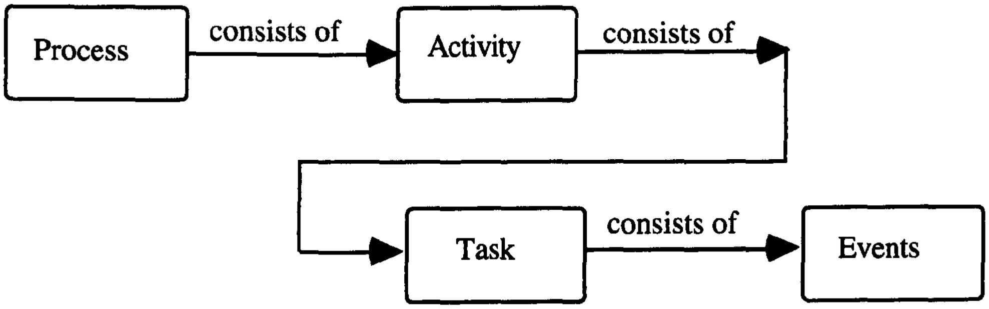


Figure 5.2 Process in TRL

5.2.1 Conception of Requirements

Functional requirements originates from a sense of causation. For example, a message cannot be received unless it is sent, or in a restaurant a customer gets the food after s/he orders, similarly in a tank controller, opening a valve causes the liquid level to be raised. The requirement is a chain that mirrors this causal relationship among several events occurring in a system. Real-time systems interact with physical devices which are monitored and controlled. A complex system is a combination of interacting components. In all these systems one device triggers another. The behaviour of a system is this causal relationship among real world events. Requirements evolve from this simple set of reasoning. Requirement of a system involves the order of occurrence of events and the constraints on the time of occurrence. As Bubenko [Bubenko 80] observes a conceptual model represents abstractions, and constraints about an application domain. An event model provides such features for the narration of a conceptual model.

We make use of event based model to capture the behaviour of the system. In event model, one is interested in the ongoing process involving real world entities (i.e., how an event is caused, how an event affects other events, and which event is dependent on other events). Description of behaviour produces a chronological relationships between corresponding events. With real-time systems we are interested in the precise sequencing of operations and the detailed timing and control characteristics of devices. Event model provides such details.

5.2.2 Timed Requirements

Event based model provides the basis to express the real time requirements of a system. A real-time system requires temporal reasoning. For such a reason in TRL every event is a timed event. An event associates a time parameter with an event name. It is expressed as (button_pressed , t1) where t1 is timing parameter associated with event 'button pressed'. The syntax diagram of event is shown in Figure 5.3. The syntax diagrams make use of standard notation, non-terminals are shown in rectangular boxes, and reserved words in bold letters.

```
<event> ::= "(" <event parameter> "," <time parameter> ")"

<event parameter> ::= <identifier>

<time parameter> ::= <identifier>
```

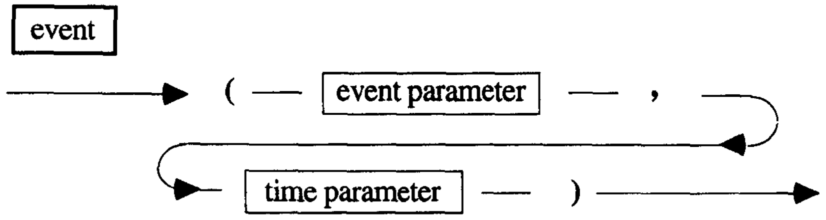


Figure 5.3(a) Syntax diagram of event

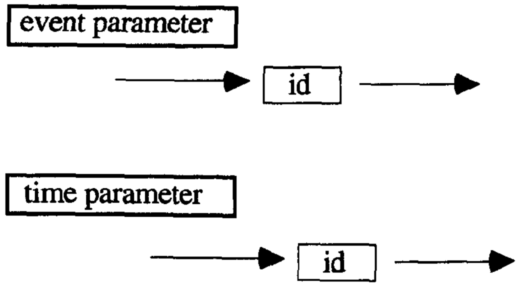


Figure 5.3 (b) Syntax diagram of event and time parameter

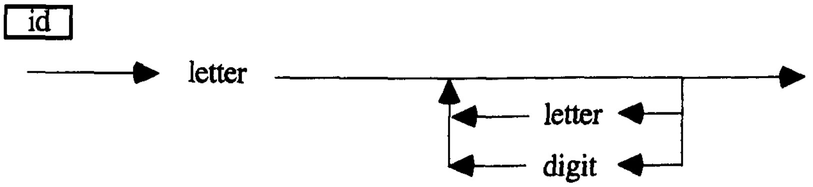


Figure 5.3 (c) Identifier

5.2.3 Description of Requirements

As discussed in the earlier chapters (Chapter 3, and Chapter 4) a system can be deemed to be made of a number of concurrently acting processes[28]. Thus, a system can be modelled as a set of processes (Figure 5.4), i.e.,

<system> ::= "requirements" <id> {<processes>}

<processes> ::= <process> { " || " <process> }

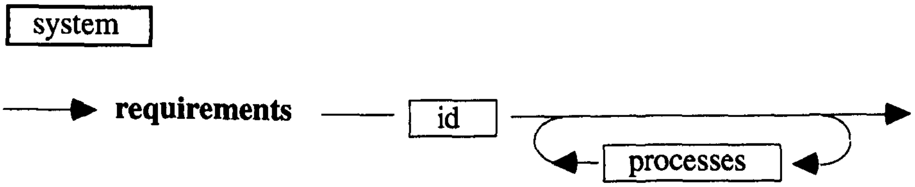


Figure 5.4(a) Syntax diagram of system

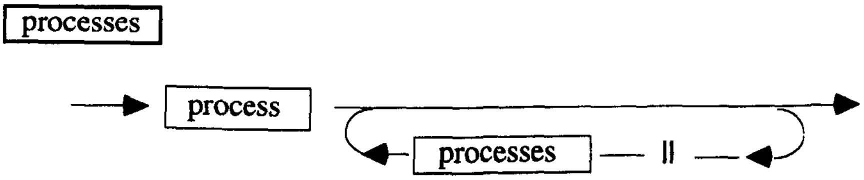


Figure 5.4(b) Syntax diagram of processes

[28] In Chapter 3, we referred it as 'agents', and in Chapter 2, before introducing the concept of agents, we referred it by the generic name 'components of the system'.

The formalisation of the parallel composition ($\|$) has been discussed in Chapter 4. The parallel composition of a set of processes describes the joint behaviour of all the processes running concurrently.

A process consists of events that must be executed in a prescribed order. As we discussed in Chapter 2, a real-time system is characterised by mainly two types of processes: periodic and aperiodic. A periodic process consists of events that is executed repeatedly, once in a fixed period of time. The common example of periodic process is to read the sensor information, or update the calendar time. Aperiodic processes[29] (or also called as asynchronous processes) consists of events that correspond to internal or externally motivated events. A common example of aperiodic process is to respond to operator requests.

System activity is asserted by defining bodies of processes. Processes are used to describe the dynamics of the system. Process consists of a set of behaviour definitions, where each behaviour definition is justified by the behaviour definition previous in the sequence.

In TRL a behaviour definition is of the form

(beh_id $(e_1, e_2, \ldots e_n)$)

[29] For the purpose of scheduling analysis, Mok [Mok 83, Mok 84] suggests to translate an aperiodic process into a quasiperiodic process (or sporadic process), by providing a minimum separation time between the motivating events. Polling is an example of this. In this scheme, a polling task checks to see if an aperiodic event has occurred, if it has occurred then processing begins, if not then nothing is done till the beginning of the next polling period.

where beh_id is the language construct that relates events (e_1 , e_2 , . . e_n) .

A process P is a set of behaviour definitions of the form

(beh_id (e_1 , e_2 , .. e_n))

. . . .

(beh_id (e_1 , e_2 , .. e_n))

<process> ::= "process" <identifier> "begin" <named behaviour>
{ <named behaviour> } "end"

Each behaviour definition is regarded as a behaviour expression, which is named with an unique identifier.

<named behaviour> ::= <behaviour name> ":" <behaviour> <endstmt>

<endstmt> is the statement separator. We call such an expression as a named behaviour. A named behaviour can be abstracted as:

s1: $e_1 \rightarrow e_2 \rightarrow e_3$ <endstmt>

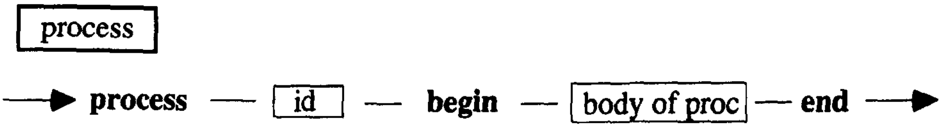


Figure 5.5(a) Process definition

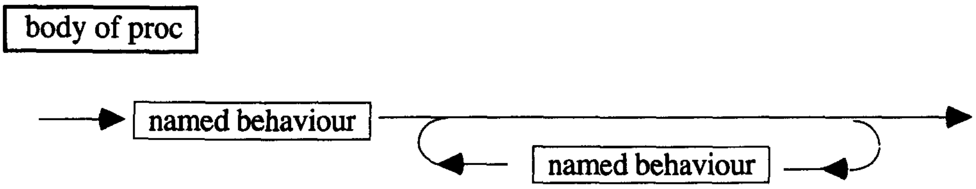


Figure 5.5(b) The body of a process

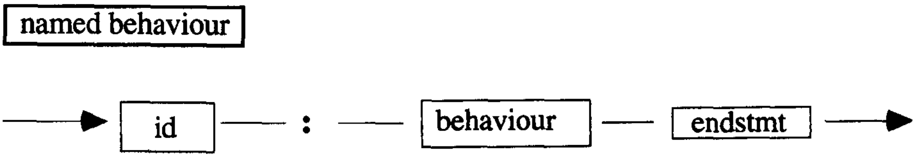


Figure 5.5(c) Behaviour definition

In an event based model, a behaviour is regarded as the manipulation of events, within the specified timing constraints.

<behaviour> ::= "do" <event sequence> ["where" <timing constraint>]
[next behaviour name] | <special behaviour>

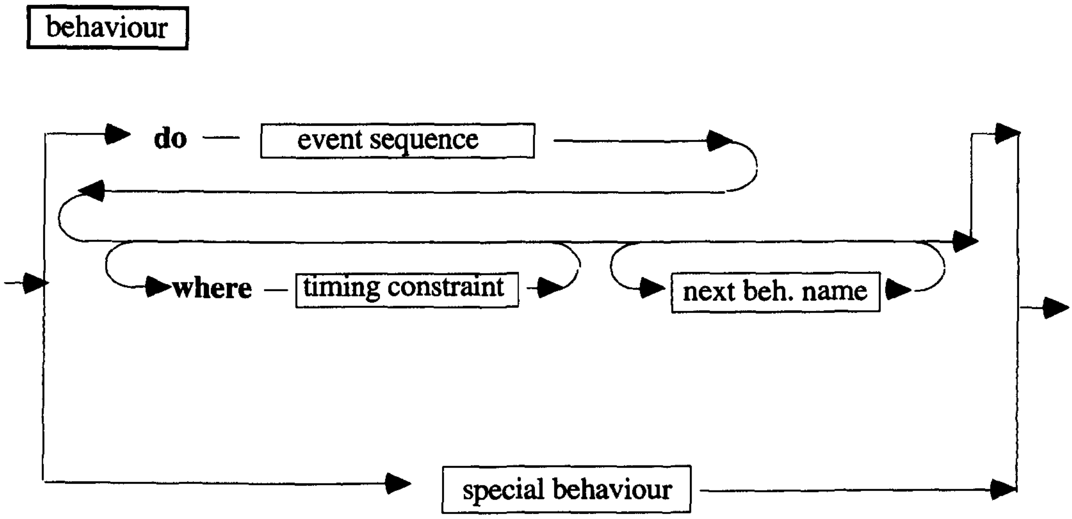


Figure 5.6(a) Behaviour expression

At present we shall ignore the non-terminal 'timing constraint'. We deal with timing constraints exclusively in later sections. The non-terminal 'next behaviour name' provides an approach to relate various scenarios[30].

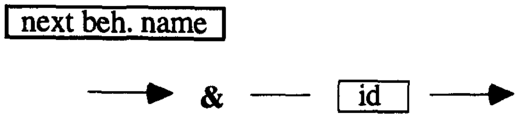


Figure 5.6(b) Next behaviour definition

Let's consider a simple example. When a person visits a restaurant, he is seated, and then if he orders for the food, then he is served with food. This can be abstracted by behaviour expressions as described earlier.

[30] Scenarios are fragmentary in nature. Thus there is a need to relate the scenarios to get the whole story about a particular agent.

s1: visits → seated & s2

s2: orders → food served

The statements (s1, and s2) in the language is terminated with a statement separator as mentioned earlier.

The behaviour that is of much interest are periodic and aperiodic behaviour. Aperiodic behaviour occurs at irregular points of time. Aperiodic behaviour is normally the result of an environmentally triggered event. On the other hand, a periodic behaviour is characterised by an event that has to occur at regular intervals of time. These two types of behaviour is further discussed in the following sections. Now it is sufficient for us to mention of their importance in the study of real-time systems.

<special behaviour> ::= <periodic behaviour> | <aperiodic behaviour>

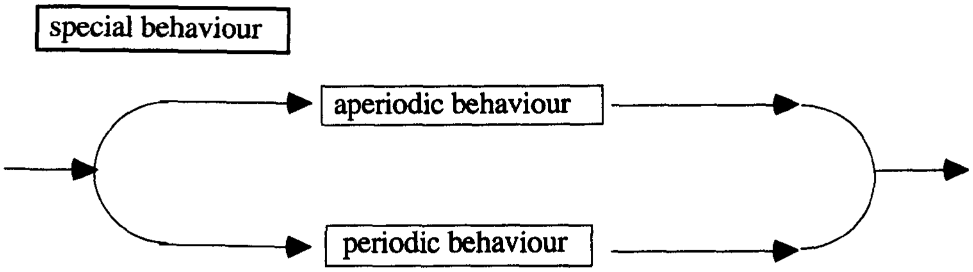


Figure 5.7 Syntax diagram of special behaviour

5.3 Conceptual Analysis

The objective of conceptual analysis is to produce statements on the aim and purpose of the system. This high level activity identifies the needs, i.e., what is to be accomplished, and what is to be avoided. This structure of thinking of a system is based on the purpose-driven framework. This purpose-driven framework emphasises goal specifications. Taylor [Taylor 82] observes that goal specifications have advantages for error and safety analysis. In event model, the purpose driven framework is postulated in terms of what happens, and how the things that happen can interact. In a reactive system, environment regularly invokes the controller. This behaviour is essentially asynchronous. These events are not controlled by the software system, and depends only on the environment. This behaviour can be analysed with cause-effect analysis.

5.3.1 Cause - Effect Analysis

Cause - effect study describes the external behaviour of a system [Elmendorf 74]. Cause is an event, and the effect is a sequence of events directly triggered by the causal event. An inherent property of this reaction is it being driven by some event happening in the system. This is what happens with reactive systems. It captures the causality in the system. Causality asserts that one event triggers another event. This triggering notion is fundamental to reactive systems. For example, consider a simple system a water tank controller. In a water tank controller, say a requirement is, when a switch is pressed (FILL) the tank is to be filled with water. This behaviour involves activating event which triggers an effect. The effect specifies the goal to be achieved. Effect can consist of more than one event, in essence it consists of an event sequence. Thus an effect may be primitive or composite. In the above example, the effect, fill the tank (FT) is a composite one. The primitive events that constitute (FT) are open the valve (OV), and turn the motor on (TM) and is given by OV ; TM. In general if event e_1 causes event e_2, then e_1 = initiator(e_2), or e_2 = effect(e_1). Where effect(e) is a set of (possibly empty) events created by the event e, and effect(e) defines an event sequence.

Thus an effect characterises an event sequence, and can be represented as (Figure 5.8):

<event sequence> ::= <event> { ";" <event> }

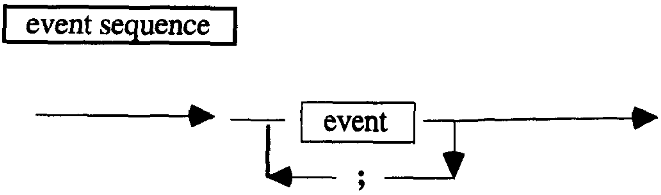


Figure 5.8 Event sequence

In some situations, an intended effect may be to ignore the activating event i.e., to 'do nothing'. Such an effect may be considered as 'defunct effect'. A defunct effect does not engage in any events, and is built into TRL. A defunct effect is denoted by 'nil'.

Let's reconsider the water tank controller discussed above. In this example, it is necessary to check whether water is available to pump in to the tank, before starting pumping the water. This requirement associates a condition, which can be either true or false. The condition is associated with the causal event pressing the switch. This event triggers a required effect only if the condition is true i.e., water is available. Thus the cause-effect analysis is essentially a cause - condition - effect analysis. The condition, models the static constraints discussed in Chapter 3.

5.3.1.1 Condition

Condition models the physical status of the system. The physical status of a system varies. For example, a person can be booked in a flight only if a seat is available. We model the conditions by their names. A primitive condition name $c \in C$, where C is the condition name set. Condition name c is a variable which is characterised by a pair (value(c), assignment(c)), where:

value(c) $\in$ {true, false}, the value true or false is assigned to c;

assignment(c) i.e., assignment to c is an event so as c takes the value, value(c) at the instant time (assignment (c)).

Examples of such conditions are "water is hot", "seat is available" and, etc. These conditions describe the dynamics of the system.

5.3.1.2 Effect

Requirements, as discussed in Chapter 3, are described as a set of scenarios, describing the changes in the system operation. Example of such scenarios are, when you press this switch, then the system resets. The words like 'press', 'cause', 'affect', 'pull', 'turn', and so on, provide a narration of cause and effect. The notion of effect plays a vital role in the analysis of requirements. This description starts with a description of the causal event, followed by the sequence of events representing the effect. This description provides credibility to the observed or postulated behaviour. This description provides information on how the things actually happen. In real-time systems the effect is time dependent. This time dependency is discussed a little later. Let's study some illustrative examples to reflect on the causal analysis in requirements definition.

Example 5.1: If an aircraft is approaching, and not identified as a friend then activate threat analysis with a deadline less than 2 sec.

In this example, approaching aircraft triggers 'threat analysis' only if it is not identified as a friend. This condition is modelled as a constraint on the causal event 'aircraft approaching'. The timing constraint is associated with the event 'initiate threat analysis'.

Example 5.2: If letters are keyed without selecting a window, then display an error message 'nobody is hearing'.

The event 'keying the letter' causes an error message 'error report' only if a window is not selected.

Example 5.3: If the temperature read by the sensor is less than 273 degrees or greater than 500 degrees, then initiate alarm of type 2.

Here the condition can be expressed as follows,

InvalidTemp = (temperature < 273) or (temperature > 500)

The event 'temperature' causes an alarm of type 2, only if it is of 'InvalidTemp'.

In these examples, the activating event triggers an effect, if a condition holds during that moment. Triggering event can be guarded by conditions. Requirements can be elicited by stepping through scenario in which triggering an event initiates a particular behaviour pattern. A triggering mechanism provides the basis for describing these events and appropriate reactions.

Following the above analysis we can define the types in TRL.

5.3.2 Types in TRL

Following Martin-Löf's constructive type theory [Nordström 84] we define types as predicates that state the properties of system or its components. For example it may be an expression that a certain variable has a positive value, or that a certain resource is available. This mechanism provides a natural way of representing the dynamics of the system. It may be expressed as

valid_temperature = $15 < \text{temp} < 25$

register_available = a register is available for processing

We use the predicate names such as "valid temperature", or "register_available" to represent system properties. We assume that these have been suitably defined.

5.4 Aperiodic Behaviour

Let's recall that aperiodic behaviour deals with events which occur at irregular time. Aperiodic behaviour arises due to dynamically triggered events. For example in telephony, a subscriber going off-hook (lifting the handset) causes an aperiodic behaviour. Aperiodic behaviour generally has complex timing constraints associated with it. Aperiodic behaviour can be analysed with cause-effect analysis discussed above. Aperiodic behaviour deals with complex situations. We discuss these situations, and then generalise the syntax of aperiodic behaviour in Section 5.4.3 (Figure 5.18).

As discussed above, a simple scenario of aperiodic behaviour is as shown in Figure 5.9, and its syntax in Figure 5.10.

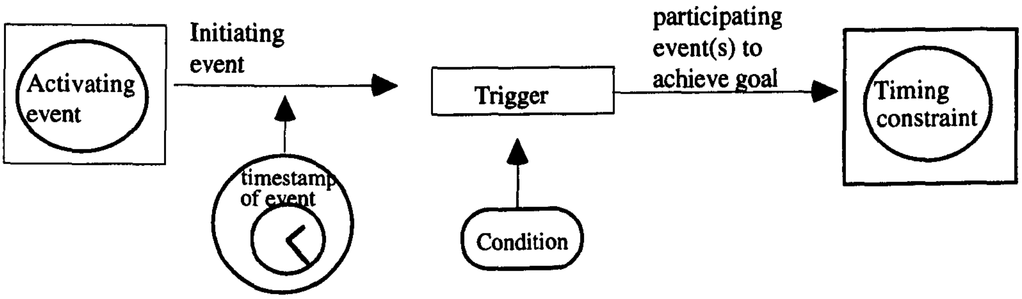


Figure 5.9 Scenario of aperiodic operation

The syntax of aperiodic behaviour can be expressed as:

<aperiodic behaviour> ::= "if" <initiator> "then" <participator>

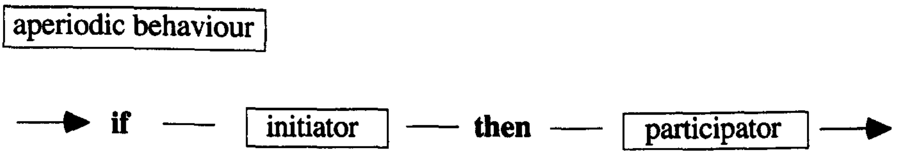


Figure 5.10(a) Syntax of aperiodic expression

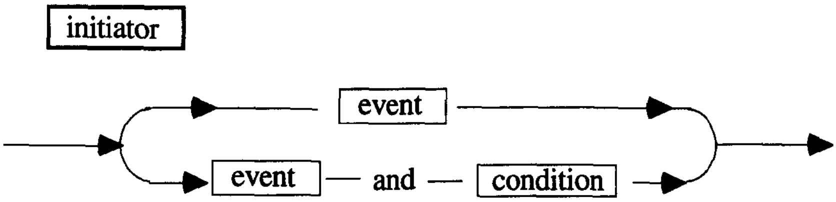


Figure 5.10(b) Syntax of 'initiator'

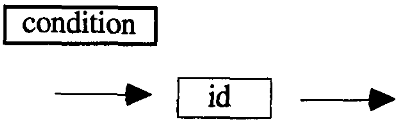


Figure 5.10(c) Syntax of 'condition'

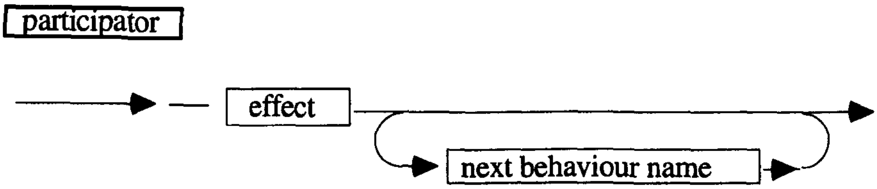


Figure 5.10(d) Syntax of 'participator'

At present (as discussed in the above section) we shall assume that the effect consists of an event sequence (including an empty sequence). The syntax of effect is summarised in section 5.4.3.

Consider a simple aperiodic behaviour of the form

if event 'e' occurs, execute 'f'

In this example event 'e' is the motivating event, while the event 'f' represents the effect. The above behaviour can be expressed in TRL as follows:

```
if (e, t1) then (f, t2)
```

As discussed earlier, the effect depends upon the causal event. This means that a system can have different effects, at a given moment depending on the causal event. This is discussed below.

5.4.1 Situation Dependent Effects

A system at a given moment may be expected to behave differently, depending on the input. For example consider a simple help system, in which if Hotel is pressed the information regarding the nearby hotels is displayed, if Bus is pressed then information regarding bus transportation is displayed, and if Taxi is pressed then information regarding taxi service is displayed.

```
if  initiator$_1$  then  participator$_1$
  elsif  initiator$_2$  then  participator$_2$
     elsif  initiator$_3$  then  participator$_3$


          elsif  initiator$_n$  then  participator$_n$
```

Figure 5.11 Syntax of modelling the situation dependent effects

In this system the resultant effect depends upon the type of motivating event. Alternative effects depending on the triggering event can naturally be expressed by elsif clause. Such a situation is as shown in Figure 5.11. The syntax of 'effect' is provided in Figure 5.19 after introducing some concepts involving time.

The above mentioned syntax of aperiodic behaviour is extended to include the selection as:

<aperiodic behaviour> ::= "if" <initiator> "then" <participator>
{ <alternative event sequence> }

<alternative event sequence> ::= "elsif" <initiator> "then" <participator>

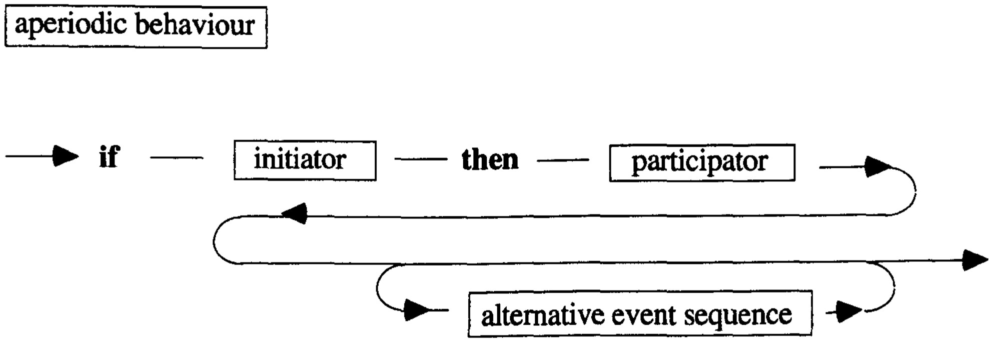


Figure 5.12(a) Syntax diagram of aperiodic behaviour

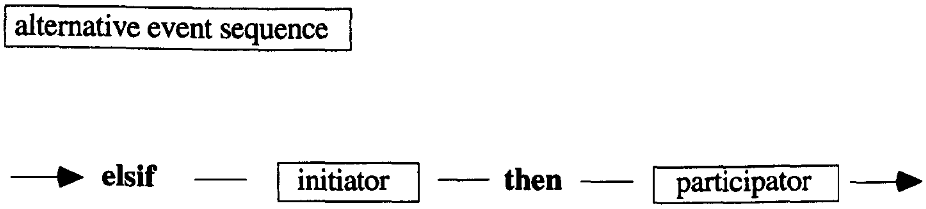


Figure 5.12(b) Syntax diagram of alternative event sequence

Aperiodic behaviour may be associated with timing constraints such as deadline. Here we informally used the word 'deadline'. By the way what is deadline? Is this the only type of timing constraint that arises in a system? Can we generalise the timing constraints at a conceptual level, and provide suitable mechanisms to discuss such timing requirements? These issues will be discussed in detail below.

5.4.2 Timing Constraints in a Conceptual Model

A model is conceptual in the sense that the requirements manifest at an application level. Timeliness requirements are expressed at a higher level of abstraction. At the highest level of abstraction, an event cuts the timeline at the point of occurrence. The timing constraints are expressed as a restriction on the moment of occurrence of event(s). These timing constraints may be expressed through the timing relationships involving the time points denoting the occurrence of events. The temporal requirements are an important aspect of real-time systems. We discuss the temporal requirements at the user level. Recalling the classification of temporal requirements provided by [Dasarathy 85] we have:

minimum - no less than t units of time must elapse between the occurrence of events;

maximum - no more than t units of time must elapse between the occurrence of events;

durational - exactly t units of time must elapse between two events.

In [Dasarathy 85] the end points of the intervals, between the pairs of events are classified as one of the following types: (1) stimulus - response, (2) stimulus - stimulus, (3) response - stimulus, and (4) response - response. This framework though provides a general classification of timing constraints from the user point of view, it suffers from an implicit assumption, that every timing constraint involves just a single pair of events. Let's consider an example, to describe a timing constraint.

Example 5.4: Consider the Figure 5.13, where the event a, causes further events b, c, and d. If the timing constraint on event d is such that, event d must occur within six time units of event a and three time units of event c.

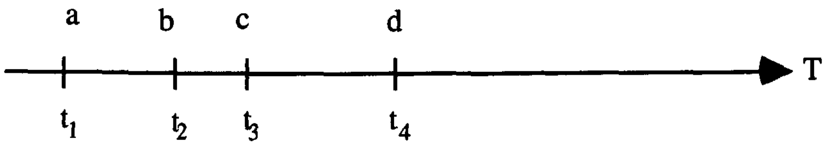


Figure 5.13 Time constrained events

A timing constraint of this sort falls outside the framework of [Dasarathy 85].

Also this framework does not consider timing constraint on periodic processes. In real-time systems periodic processes are predominant. In the following sections, we generalise the framework to describe the various types of timing constraints that may arise in a system. Our framework does not treat the timing constraint as a temporal restriction between two events. We recognise that a temporal constraint can involve many events. All the timing constraints are discussed in a single formalism.

5.4.2.1 Timeliness Requirements

At the lower level of requirements, timeliness requirements can be expressed, by considering the usefulness of an action in a time period. Jensen *et al* [Jensen 85] define value function as a way to express the timing constraints of real-time systems. The value function also provides a natural means to classify the real-time systems viz. hard, and soft [Burns 91, Abbott 88]. For example consider an event "close the door (CD)" as shown in Figure 5.14 (a). This event has a duration, and as explained in Chapter 4 we represent it by two instantaneous events $start_{CD}$, and end_{CD}.

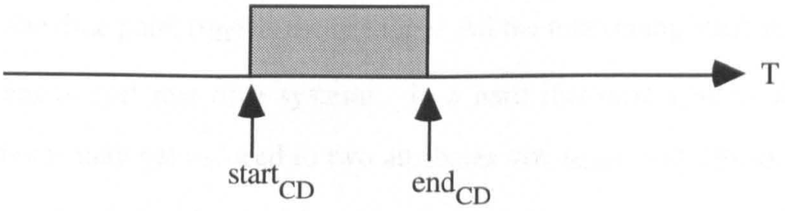


Figure 5.14 (a) Representation of a continuous event

The utility of the event 'close the door', can be explained with four attributes as shown in Figure 5.14(b). The four attributes are:

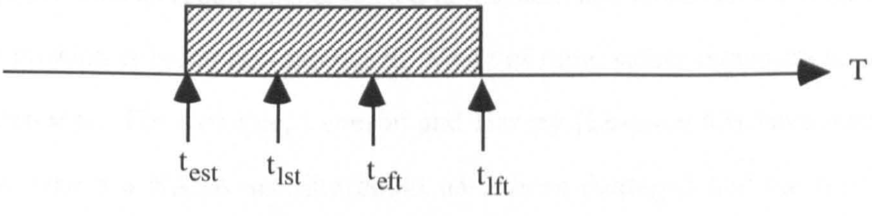


Figure 5.14 (b) Attributes of timing constraint considering value function

Earliest starting time (t_{est}) - the earliest time point during which the event $start_{CD}$ can occur;

Latest starting time (t_{lst}) - the latest time point during which the event $start_{CD}$ can occur;

Earliest finishing time (t_{eft}) - the earliest time point during which the event end_{CD} can occur;

Latest finishing time (t_{lft}) - the latest time point during which the event end_{CD} can occur.

The time period (t_{lst} - t_{est}) is the latency, the time period (t_{lft} - t_{eft}) is the delay, and the time point (t_{lft}) is the deadline. All the four timing attributes are naturally present in soft real time systems. In a hard real-time system, the above four attributes may get reduced to two attributes viz. t_{start} and t_{finish}, where t_{start} denotes the time point at which the start event (say $start_{CD}$) can occur, and t_{finish} denotes the time point at which the end event (say end_{CD}) can occur.

As discussed earlier in Chapter 3, timing constraints in a system may arise as a result of the safety requirement. The safety requirement may arise as a result of the physical laws and rules of operation. Leveson [Leveson 86] classifies system requirements as requirements related to mission, and those related to safety while the mission is being accomplished. Many of these safety requirements are time-dependent. For example, Leveson and Harvey [Leveson 83] have mentioned a case where a NASA satellite could have been damaged had the time interval between the occurrence of two events been short. Real-time systems have different timing constraints associated with them, and such timing constraints are discussed below.

5.4.2.2 Representation of Timing Constraints

A timing constraint restricts the moment of occurrence of an event. Lamport [Lamport 78] has argued that to avoid any inaccuracies in timing only observable events should be used for timing other events[31]. In our model, we use observable events for timing other events.

The syntax of timing constraint in TRL is as given below, and in diagrammatic form in Figure 5.15(a)

<timing constraint> ::= <timing factor> {"and" <timing factor> }

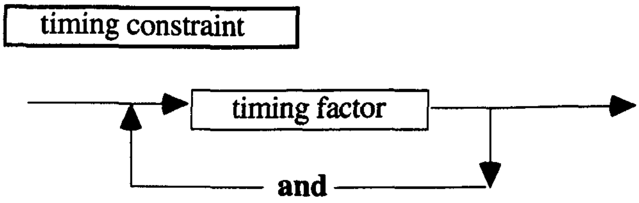


Figure 5.15(a), Syntax of 'timing constraint'

<timing factor> ::= "(" <time parameter > <relation operator> <time parameter > " + " <timing duration> ")"

[31] This advice is in line with the philosophical observation made by Leibniz, "time and space are not the things, they are the order of the things".

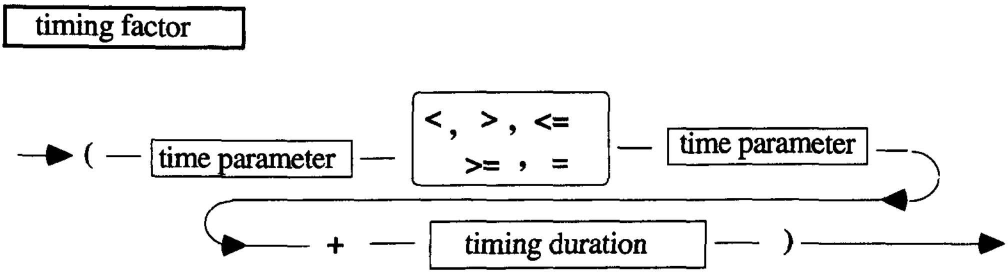


Figure 5.15 (b), Syntax of 'timing factor'

<timing duration> ::= <integer> | <real>

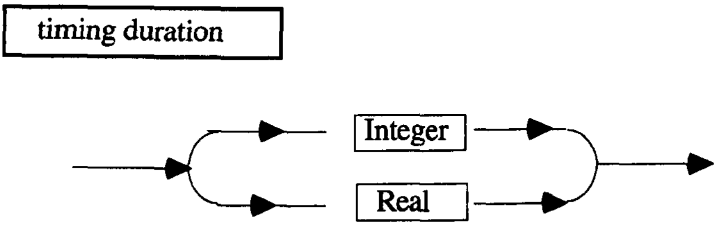


Figure 5.15 (c), Syntax of 'timing duration'

<integer> ::= <digit> { <digit> }

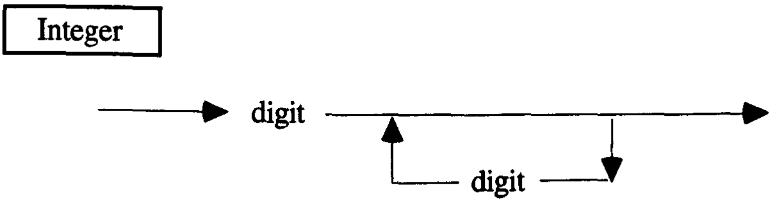


Figure 5.15 (d), Syntax of 'integer'

<real> ::= <integer> "." <integer>

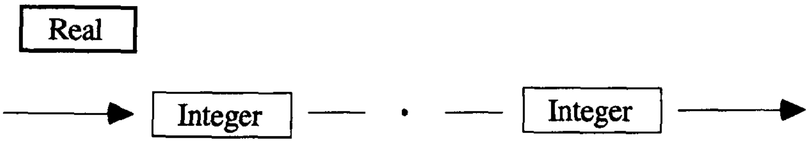


Figure 5.15 (e), Syntax of 'real'

Let's describe the various timing constraints with an illustrating example.

Example 5.5: Consider a requirement such that when a switch is pressed, the controller must start the job of closing the door within 10 time units, and must complete the job within 6 time units of having started the job.

The events of interest are switch pressed, start closing the door (start_{CD}), and door closed (end_{CD}). The above requirement can be expressed as follows:

if (switchpressed, i) then (start_{CD}, j) ; (end_{CD}, k) where ($j < i + 10$) and
($k < j + 6$) endstmt

With this example, we shall explain all the types of timing constraints that can arise in a system. These timing constraints are described in TRL. Following Jensen [Jensen 85] the timing constraint, in essence describes the utility of a task with respect to time. These systems may be hard or soft. The language employed must be capable of expressing all types of timing constraints.

A timing constraint can constrain, earliest starting time (t_{est}), latest starting time (t_{lst}), earliest finishing time (t_{eft}), latest finishing time (t_{lft}), or any combinations of these as shown in Figure 5.16. The example given below describes in TRL the timing restriction on all these parameters.

Example 5.6: Timing Restriction on earliest starting time (t_{est}), latest starting time (t_{lst}), earliest finishing time (t_{eft}), and latest finishing time (t_{lft})

if (switchpressed, i) then (start$_{CD}$, j) ; (end$_{CD}$, k) where (j >= i + 5) and
(j <= i + 10) and (k >= j + 4) and (k <= j + 6) endstmt

The timing constraints discussed in Figure 5.16 includes the types of timing constraints discussed by [Dasarathy 85], and are more general. All class of timing constraints are expressed in a single formalism. The types of timing constraints expressed by [Dasarathy 85] can be expressed as below:

Minimum: if (e1, t1) then (e2, t2) where (t2 > t1+ 5) endstmt

Maximum: if (e1, t1) then (e2, t2) where (t2 < t1+5) endstmt

Durational: if (e1, t1) then (e2, t2) where (t2 = t1+5) endstmt

As noted above this classification does not deal with timing constraint over several events.

Possible Types of Timing Constraints		
Constraints on		
t_{est}	t_{est} — t_{lst}	t_{est} — t_{lst} — t_{eft}
		t_{est} — t_{lst} — t_{lft}
		t_{est} — t_{lst} — t_{eft} — t_{lft}
	t_{est} — t_{eft}	t_{est} — t_{eft} — t_{lft}
	t_{est} — t_{lft}	
t_{lst}	t_{lst} — t_{eft}	t_{lst} — t_{eft} — t_{lft}
	t_{lst} — t_{lft}	
t_{eft}	t_{eft} — t_{lft}	
t_{lft}		

Figure 5.16 Classification of timing constraints

5.4.3 Addressing What if Situations

For a real-time system to be robust, it must use a mechanism that can cope with system failures. Exception handling deals with such failures. Exception handling are of two types, general exceptions, and time-related exceptions. The former deals with functional errors. For example, a functional exception handler deals with situations such as, division by zero, or finding the square root of a negative number. On the other hand time-related exception, endeavours to take evasive action when a particular timing constraint cannot be guaranteed. This section looks at exception handling, to deal with timing constraint violations.

The representation language must have provisions, to state what actions to take, when a timing constraint cannot be guaranteed. These exceptions enable a real-time system to fail gracefully. In this way a real-time system is consistent, as it is aware of the timing constraints that are not satisfied. If the syntax of the representational language provides an exception handler with any time constrained construct, then the analyst is forced to consider alternative actions at every possible situation, where a timing failure could occur. In the author's opinion such a provision is essential. It is difficult to deal with timing constraint failures at later stages. Suitable actions in these situations can only be determined, in concurrence with the users.

Real time systems are required to behave properly under all circumstances. Real time requirements involve constraints related to time in the real world. Complete and correct action within the timing constraint specified, could never be guaranteed. In managing the real world environments, an action simply cannot be ensured even by increasing the speed of processors [Stankovic 88b]. This reflects the reality of real time system that we must be able to accept the deviations from the desired goals and settle for the weaker goals. This involves making trade-offs between different goals in a reasonable manner. Goal abandonment and substitution are important means by which graceful degradation of the behaviour can be achieved [Chandrasekaran 91]. In practice the notion of goal abandonment and substitution is important. It also provides a mechanism to denote the safe behaviour of the system. Whether the control system offers the desired goal or the weaker one depends upon the real-time behaviour of the system. The time at which the system responds to the request determines whether the goal will be abandoned in lieu of the weaker one.

This means that when a task is not guaranteed within the required timing constraint, then a *timing fault* can occur. In this situation, an alternative task which has a shorter computation time can be invoked. If the latter is done, then timing fault is masked. As shown in Figure 5.17 the temporal switch determines the choice between the two goals.

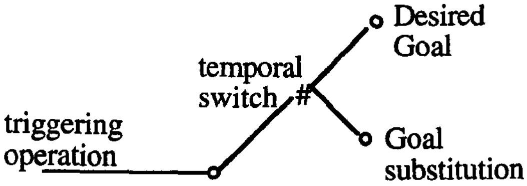


Figure 5.17 Modelling the temporal behaviour

Incorporating this temporal behaviour mechanism, the triggering mechanism of Figure 5.9 gets modified to as shown in Figure 5.18.

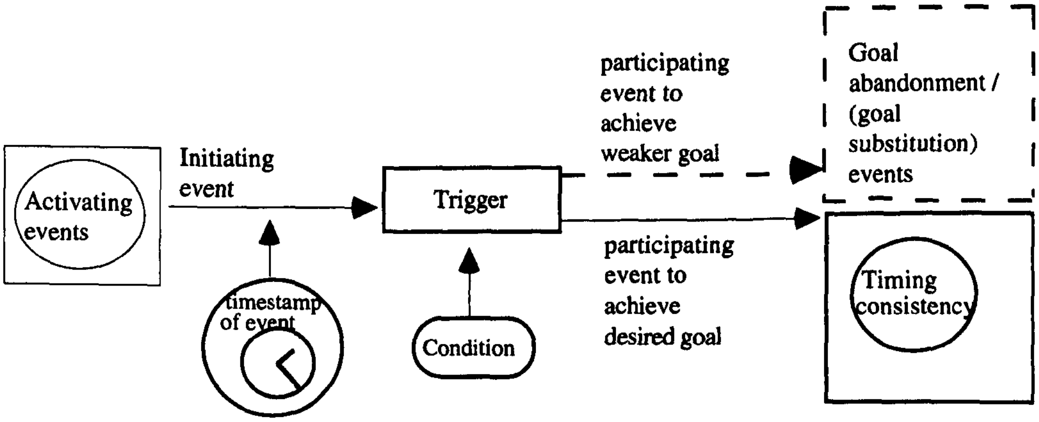

Activating events
Initiating event
timestamp of event
Trigger
Condition
participating event to achieve weaker goal
participating event to achieve desired goal
Goal abandonment / (goal substitution) events
Timing consistency

Figure 5.18 Aperiodic behaviour with time-related exceptions

Figure 5.18 describes tasks with fault tolerance requirements. In a system, a higher level activity, can decide which activity should be performed. This notion acknowledges that the tasks have different levels of temporal needs, and importance.

We shall illustrate a situation through an example. In this example, a timing constraint has a corresponding time-related exception handler. The example shows that, if the temperature measured is greater than 50, then switch on the cooling system within 30 milliseconds. If this timing constraint cannot be adhered, then the time-related exception handler is activated. This example is of a hard real-time, and takes a drastic action of shutting down the controller.

```
                Temperature Controller

    if  temperature measured > 50
       then  switch on cooling system within 30 milliseconds
    endif

Time-related exception handler
          Issue immediate shut-down
```

Consider a situation where,

Initiating event: Temperature measured (temperature)

Condition: over_the_limit = (temperature > 50)

Responding event: Switch on cooling system (switch_cool)

Timing constraint: Respond within 30 milliseconds

This can be expressed as

```
if (temperature, i) and (over_the_limit) then (switch_cool, j) where
                                        (j < i + 30)  else  (shut_off , k) endstmt
```

Now having worked out the various features required in an aperiodic behaviour, we can provide the generalised syntax diagram (as shown in Figure 5.18):

```
<aperiodic behaviour> ::=  "if" < initiator > "then"  < participator >
                                        { <alternative event sequence> }

<alternative  event sequence> ::=  "elsif"  <initiator>  "then"  <participator>

<initiator > ::=  <event>  |  <event>  "and"  <condition>

<participator > ::= <effect> [ <next behaviour name> ]

<effect> ::=  "nil"  |  <event sequence >   | <event sequence>  "where"
                    <timing constraint >  [ <timed exception> ]

<timed exception>::=  "else"  <event sequence> |  "else"  < event sequence>
                    "where "  <timing constraint>
```

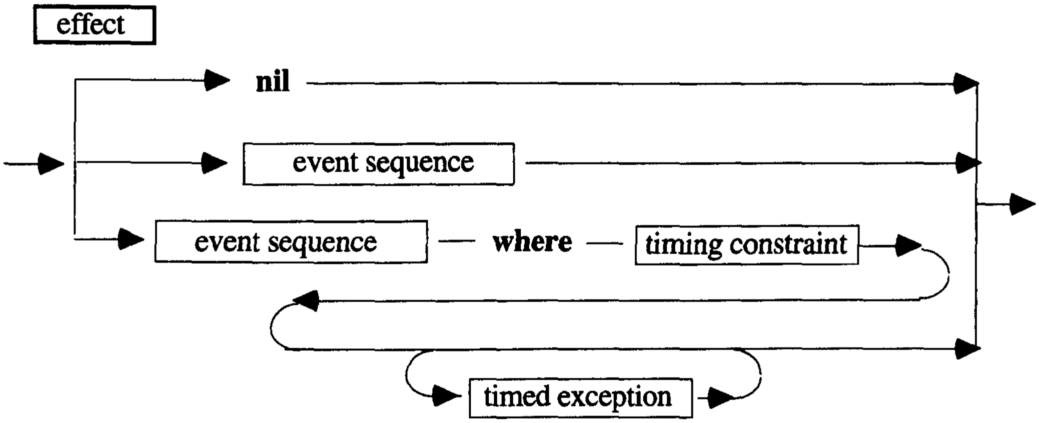


Figure 5.19(a) Syntax diagram of 'effect'

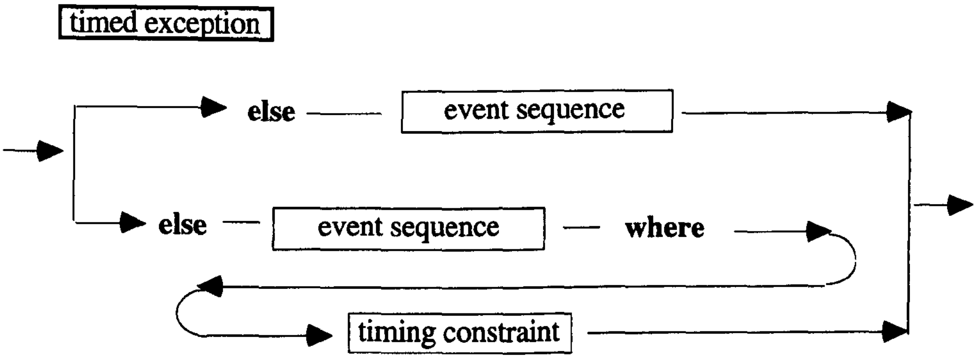


Figure 5.19(b) Syntax diagram of 'timed exception'

5.5 Periodic Behaviour

Contrary to aperiodic requirements, periodic requirements need to be repeated over an interval of time. Some typical examples of periodic behaviour are, monitoring the sensors in a process controlled application, or monitoring an aircraft in a radar application. A periodic behaviour may come into existence dynamically, or be present from the time the system is put into service. A task like 'monitoring the sensor' comes into existence, from the time the system is put into service, and ceases to exist when the system is put off. A task like 'monitoring an aircraft' is an example of a dynamically created task, the task comes into existence when the aircraft enters the control region of the radar, and ceases to exist when the aircraft leaves the region. Similarly, in a telephone exchange, a periodic task is dynamically created, once a subscriber goes off-hook, and this task ceases to exist after the subscriber completes the 'dialling of the digits'. The responsibility of this periodic task is to collect the digits dialled by the subscriber. Periodic tasks exist for reasonably long intervals of time.

A periodic timing constraint requires some task to be executed at fixed intervals, in the time-region of interest. This time region is delimited by two events, the one which initiates the task, and another event which terminates the task. The timing constraint on periodic behaviour is simple. A periodic behaviour is one in which the timing constraint has the form $\forall i \in \{3..n\}, t_i - t_{i-1} = t_2 - t_1$.

The syntax of a periodic behaviour is (Figure 5.20):

<periodic behaviour> ::= "from" <event> "repeat" <event sequence> "every"
<timing duration> "until" <event>.

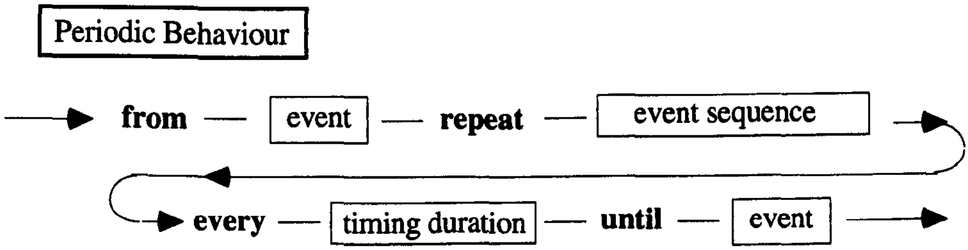


Figure 5.20 Syntax of periodic behaviour

An example of the above syntax is:

from $e_{initiate}$ repeat e_{body} every δ until $e_{terminate}$.

5.6 Summarising the BNF

Items enclosed in [square brackets] may appear zero or one time, and items enclosed in { braces } may appear zero or more times. Terminal symbols appear in " double quotes ".

<system> ::= "requirements" <head> {<processes> }

<processes> ::= <process> { " || " <process> }

<process> ::= "process" <identifier> "begin" <named behaviour>
{ <named behaviour> } "end"

<named behaviour> ::= <identifier> ":" <behaviour> <endstmt>

<behaviour> ::= "do" < event sequence> ["where" <timing constraint>]
[<next behaviour name>] | <special behaviour>

<next behaviour name> ::= "&" <identifier>

<special behaviour> ::= <periodic behaviour> | <aperiodic behaviour>

<periodic behaviour> ::= "from" <event> "repeat" <event sequence> "every"
<timing duration> "until" < event>

```
<aperiodic behaviour> ::=  "if" <initiator> "then"  <participator >
                           { <alternative event sequence> }

<alternative  event sequence> ::=  "elsif"  <initiator>  "then"  <participator>

<initiator > ::=  <event>  |  <event>  "and"  <condition>

<participator > ::= <effect> [ <next behaviour name> ]

<effect> ::=  "nil"  |  < event sequence >   | <event sequence>  "where"
                           <timing constraint >  [ <timed exception> ]

<timed exception> ::=  "else"  <event sequence>  |  "else" <event sequence>
                           "where " <timing constraint>

<event> ::=    "(" <event parameter>   ","  <time  parameter>  ")"

<event sequence> ::=  <event>   { ";"  <event> }

<time parameter> ::=  <time parameter name>   |  <don't care>

<timing constraint> ::=  <timing factor>  { "and"  <timing factor> }

<timing factor> ::=  "("  <time parameter>   <relation operator>   <time parameter>
                                      "+"  <timing duration> ")"

<time constant> ::=  <integer>  |  <real>

<event name> ::=  <identifier>
```

```
<time parameter> ::= <identifier>

<condition> ::= <identifier>

<head> ::= <identifier> <endstmt>

<identifier> ::= <letter> { <letter> | <digit> }

<relation operator> ::= "<" | ">" | "<=" | ">=" | "="

<integer> ::= <digit> { <digit> }

<real> ::= <integer> "." <integer>

<endstmt> ::= "$ "
```

5.7 Summary

Real-time systems include electronic gadgets, power plants, aircraft and railroad control. These systems are highly interactive, and usually require complex temporal behaviour. Real-time systems are often constructed from many concurrent components. As Leveson [Leveson 86] observes, the greatest problems associated in software engineering, are due to the computer system being treated merely as stimulus-response system (for example see [Alford 85], [Davis 82]). Real-time systems are described by their intended goals (missions). The approach discussed here, described the mission of the system as conceived by the user. These descriptions encapsulated the static and dynamic constraints.

The development of requirements for real time systems is a difficult task. The process of requirements development is incremental in nature. For such a reason we observe a system as a collection of components, which co-operate with each other to achieve a desired result. A TRL description can be checked to reveal its lexical, syntactic, and semantic errors. A TRL description undergoes three phases of analysis like: Phase 0: Lexical analysis; Phase 1: Syntax analysis; and Phase 2: Semantic analysis. The general treatment of the techniques employed in Phase 0, Phase 1, and Phase 2 can be found in the standard compiler literature (e.g. Aho 86), and is not discussed here.

Description of a system requires the identification of events contained in the system. As this method is parametrized with respect to events in a system, it allows to treat different systems in a uniform way. TRL is primarily intended for representing the conceptual model of a system. Conceptual model of a system controls the complexity of large systems by identifying the various components of the system. System behaviour is then the composition of the behaviour of the various components. The language has a simple underlying model. It proposes a system at a simple abstract level.

Chapter 6

Case Study

The various aspects of the technique discussed so far is illustrated with two examples. The examples reflect the essential features of real-time systems.

6.1 Introduction

In the previous chapters, we discussed the modelling approach, and the language - TRL to represent the system. To illustrate the use of the techniques derived in the previous chapters, we demonstrate two real world examples.

We chose these example for the reason that:

(1) The applications are realistic, and significant. The applications demonstrate the essential feature of real-time system, and provide effective means of demonstrating the problems and deficiencies in the definition of requirements. In many circumstances these problems are revealed only when carrying out the task in a timed language.

(2) These systems involve timing constraints, which are intrinsic, i.e., timing constraints arise while understanding the intended operation of the system. This is typical of many real-time systems. Timing constraints arise because of the nature of work, not because of the need to do the job fast[32].

6.2 The Railroad Crossing Example

The railroad crossing problem has been proposed as a benchmark for the study of real-time system by the Naval Research Laboratory [Heitmeyer 93]. We briefly introduced this example in Chapter 3, to discuss the modelling approach. We

[32] This means that, Real-Time System is not same as 'Be Quick as a Bunny'.

reiterate the basic needs of this system. The basic requirement is whenever the train is in the crossing region, the gate must be down.

The system has two basic properties, the safety property - whenever the train is in the crossing, the gate must be down, and the utility property - the gate must be up, when no train is in the crossing region. The utility property avoids a lazy solution to the problem. In a lazy solution, once the gate is lowered, the system can keep it lowered.

6.2.1 Requirements - First Level

This system operates a gate at a railroad crossing. The crossing region (say X) lies in the region of interest (say R), where X < R. The region of interest is greater than that of X, so that the gate is lowered before the train enters the region X.

6.2.1.1 Environment Analysis and Modelling

The objective of this phase is to describe the existing world for the application. This analysis is an abstract description of the agents that are useful for the problem. As explained in earlier chapters, the agents are initially identified by recognising the influence they bear on the system. This brings out the factors such as, purpose, and function. The 'purpose' involves the determination of what the objective should be. This basically answers the question, is this of use to the system? Similarly, the function involves the determination of accomplishing this purpose. This is elaborated with scenarios as discussed in Chapter 3.

6.2.1.2 Modelling Agents

In Chapter 3, we discussed modelling the requirements of a real-time system. In the initial steps we identify the agents, and rewrite the requirements as a set of scenarios.

By considering each agent, we can list all the functional elements of this agent. The functional elements are abstracted by the events it is associated with.

6.2.1.3 Train Monitor

We need a train monitor to detect the train approaching the region of interest, and the absence of train in the region of interest. The train monitor reports the same to the controller (another agent). The controller, in turn takes a decision depending on the report by the train monitor, and informs the gate (another agent) either to raise the gate, or close the gate. Thus effectively, we have three agents, the train monitor, the gate, and the controller.

The train monitor essentially detects whether the trains are in the region of interest, or not. The function, and the purpose can be analysed with scenarios.

It is difficult to generate the scenarios for the whole system. The number of scenarios not only grows out of hand, but it becomes tedious and difficult to analyse the situation. For such a reason we consider the scenarios of each agent. When we consider the scenarios of each agent, the scenarios fall into groups, making it easier to analyse.

The purpose of train monitor is to watch the region of interest. The train monitor reports the same to the controller. If we abstract this information as events, then

The signature of the train monitor is:

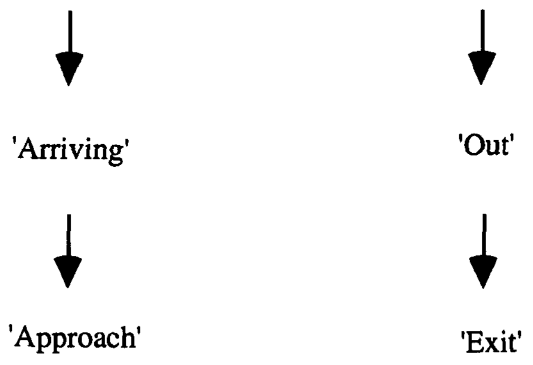
Train monitor detects that the train is approaching the region of interest - denoted by event 'Arriving'

Train monitor reports that the train is approaching the region - denoted by event 'Approach'

Train monitor detects that no train is in the region of interest - denoted by event 'Out'

Train monitor reports the absence of train - denoted by event 'Exit'

This is summarised in Figure 6.1, by means of scenarios.



Scenario 1, when crossing region is idle

Scenario 2, when crossing region is busy

Figure 6.1 Scenarios with train monitor as an agent

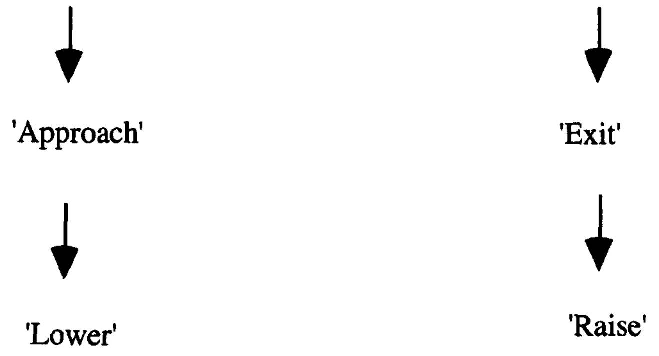
6.2.1.4 Controller

The controller, essentially manages the operation of the gate, in co-operation with the train monitor, and the gate. When the train monitor informs about the arrival of a train, the controller actuates the gate to be closed, and similarly when no train is in the region of interest, the controller actuates the gate to be raised.

Thus the signature of the controller is:

> Controller is informed by the train monitor that the train is entering the region of interest - denoted by the event 'Approach'
>
> Controller actuates the gate to be lowered - denoted by the event 'Lower'
>
> Controller is informed by the train monitor that no train is in the region of interest - denoted by the event 'Exit'
>
> Controller actuates the gate to be raised - denoted by the event 'Raise'

This can be summarised as in Figure 6.2.



Scenario 1, when the gate is up

Scenario 2, when the gate is down

Figure 6.2 Scenarios with controller as an agent

6.2.1.5 Gate

The Gate accomplishes the task of closing and opening the gate.

The signature of the gate is:

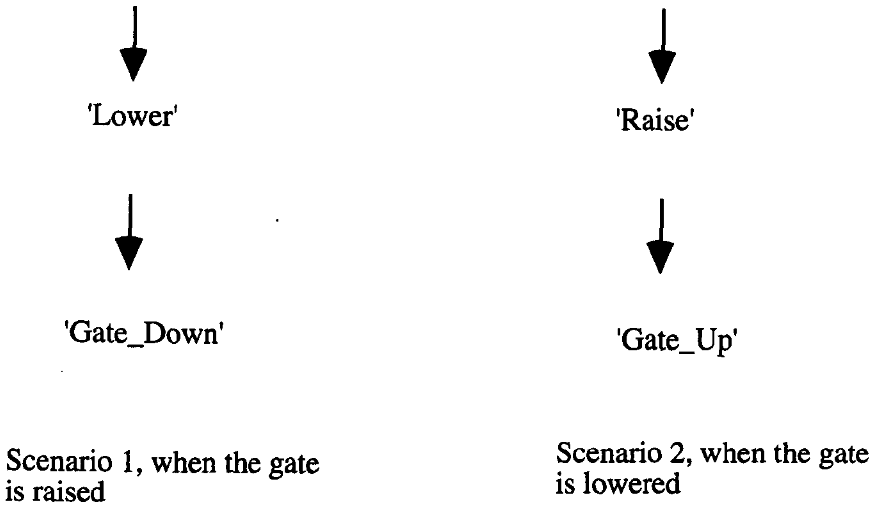
Gate is being requested by the controller to lower the gate - denoted by the event 'Lower'

Action taken to move the gate down - denoted by the event 'Gate_Down'

Gate is being requested by the controller to raise the gate - denoted by the event 'Raise'

Action taken to move the gate up - denoted by the event 'Gate_Up'

This can be summarised as in Figure 6.3.



Scenario 1, when the gate is raised

Scenario 2, when the gate is lowered

Figure 6.3 Scenarios with gate as an agent

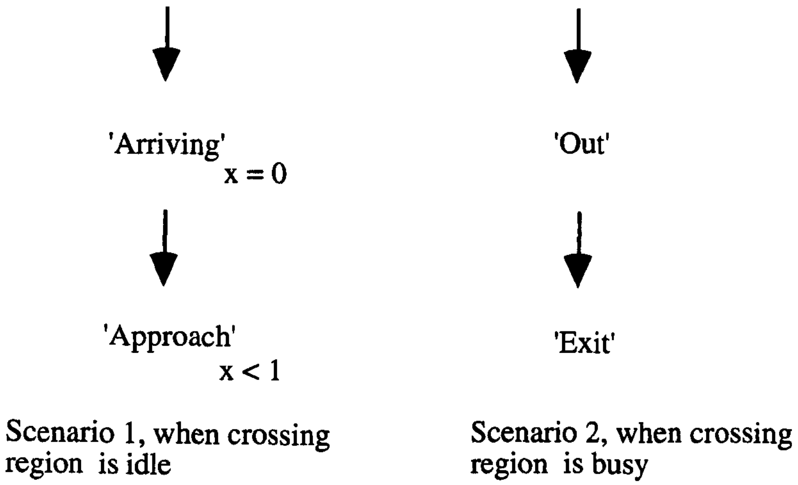
Thus the railroad crossing system consists of three agents, the train monitor, the controller, and the gate.

6.2.2 Higher Level Requirements

The higher level requirements involve obtaining additional information from the customers. Additional information is needed to describe the constraints in the operation of the agents.

6.2.2.1 Train Monitor

In the scenario of Figure 6.1, the train monitor must report the controller about the arrival of a train at the earliest. This restriction is a temporal constraint on the train monitor. The scenario of Figure 6.1 is modified in the Figure 6.4 to describe the timing restriction.



Scenario 1, when crossing region is idle

Scenario 2, when crossing region is busy

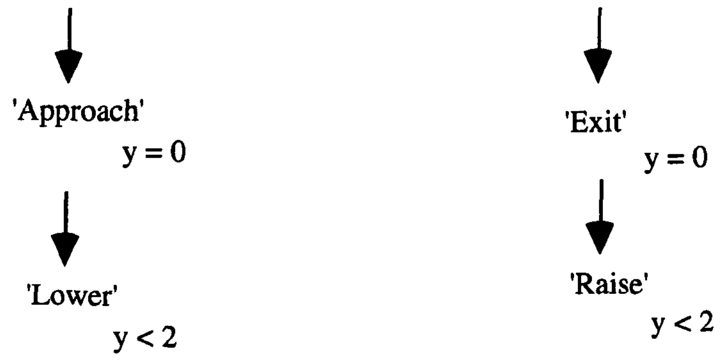
Figure 6.4 Scenario of train monitor with timing constraint

We can translate the behaviour of train monitor, as a TRL process as shown below.

```
Process Sensor
   begin

      s1 : if (Arriving, i) then  (Approach, j) where (j < i+1)  & s2   $

      s2 : if (Out, k)  then (Exit, l)  $

   end
```

6.2.2.2 Controller

Recall the scenarios described in Figure 6.2. The controller must operate in-time for the safe operation. This places temporal restriction on the controller. For example, when the controller receives the signal 'Approach' from the train monitor, it responds with the signal 'Lower' say within two time units. This is a safety requirement. Similarly, the controller must open the gate at the earliest possible time. This is a liveness requirement. This requirement restricts the operation of controller, such that, the controller responds with the signal 'Raise' the gate say within two time units of having received the signal 'Exit' from the train monitor. This is shown in Figure 6.5.



Scenario 1, when the gate is up

Scenario 2, when the gate is down

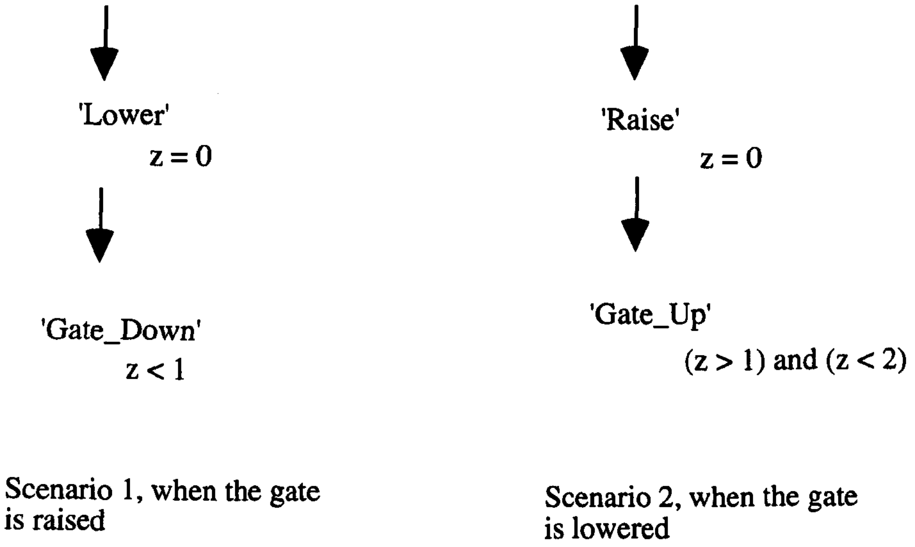
Figure 6.5 Scenario of controller with constraints

Translating this behaviour in TRL we have,

```
Process Controller
   begin

      s1 : if (Approach, i) then  (Lower, j) where (j < i +2)   &  s2   $

      s2 : if (Exit, k)  then (Raise, l) where (l < k+2)  $

   end
```

6.2.2.3 Gate

For the safe operation of the system, the gate must accomplish the job in-time. The gate must lower the gate, say within one time unit of receiving the request from the controller. Similarly the gate must be up say within 2 time units, but after one time unit of receiving the request 'raise' from the controller. This scenario is shown in Figure 6.6.



Scenario 1, when the gate is raised

Scenario 2, when the gate is lowered

Figure 6.6 Scenario of gate with constraints

Translating this behaviour in TRL we have,

```
Process Gate
   begin

     s1 : if (Lower, i) then  (Gate_Down, j) where (j < i +1) & s2   $

     s2 : if (Raise, k)  then (Gate_Up, l) where  (l > k+1) and (l < k+2) $

   end
```

As shown in Figure 6.7, the entire system is then the composition of the agents,

Train Monitor || Controller || Gate

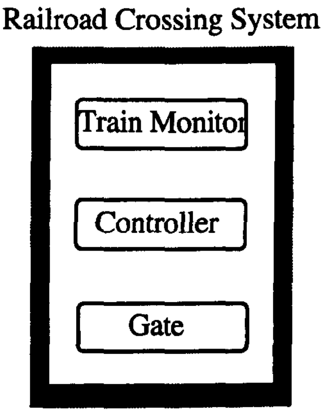


Figure 6.7 Railway crossing system as a composition of agents

The event set of the system is the union of the event set of all three agents. As we discussed in Chapter 3, real-time system, are characterised by real-time liveness, and safety. The liveness property only states that the gate once closed must eventually open. This is not sufficient to provide any information either for the customers, or for designers. Where as real-time liveness, constraints the system temporally. Thus,

Safety Property: The gate must be closed, before the train arrives at the crossing region.

Real-Time Liveness Property: The gate is never closed at a stretch for more than 10 time units.

The safety property states that the gate must be closed, before the train arrives at the crossing region. This ensures that the train can be inside the crossing region, only when the gate is down.

if (Arriving, w) then (Gate_Down, x) endstmt

Similarly, the real-time liveness property states that, once the gate is closed, it should be followed by a gate up within ten time units.

if (Gate_Down, y) then (Gate_Up, z) where (z < y + 10) endstmt

With this example we can observe an interesting property of the 'safety requirement'. Safety is a global requirement of the system. Safety requirement is normally a pure qualitative property, like robot must not crash a person, and so on. It may be noted that the safety requirement cannot be achieved without temporal restriction in a real-time system, as we observed in this example.

6.3 Another Example

Truck Loading System

This case study is adapted from [David and Alla 92]. The problem statement is expressed as below, and described with Figure 6.8.

> A truck may move between points **A** and **B**. At **A** the operator may ask for the truck to be loaded. The truck proceeds up to point **B**. Upon arrival , it is loaded by opening a hopper. When loading is complete, the hopper is closed and the truck returns to **A** where its load is made use of. It will set off again when the operator asks for a fresh loading. In the initial state, the truck is in stand-by position at point **A** .

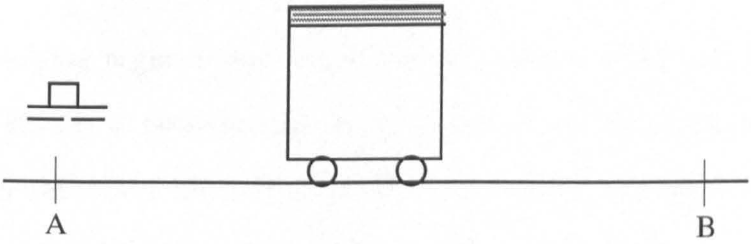


Figure 6.8 A truck loading System

As we show, the problem description is far from complete. This illustrates specific lapses with the system description and the need to employ a timed description language to comprehend many of the requirements which are lurking behind. Now let's consider the basic operations of the system.

6.3.1 Basic operations of system

System operation is initiated by the operator. An operation cycle consists of following moves:

- move from platform A to B;
- wait for the truck to be loaded, at platform B;
- after loading is over, start moving back to platform A;
- at platform A the load is to be utilised

6.3.2 Resource Structures

Requirements analysis begins by considering the environment. In the environment we can readily identify an operator, and a truck. To control the movement of truck, we should know about the position of truck. For such a reason we need to monitor the position of truck. Thus the system consists of four agents, an operator, truck, monitor, and controller.

6.3.3 Modelling Agents

Requirements is elicited by classifying the features perceived with each agent individually. System behaviour is then the composition of the behaviour of the agents.

6.3.3.1 Operator

We consider that an operator presses a switch to initiate the system operation. We assume that it is a snap-action switch. A snap action switch is normally open and makes a non-maintained contact when pressed. When a switch is operated, a request is sent to the controller, to operate the system. Thus the scenario of operator is: (shown in Figure 6.9)

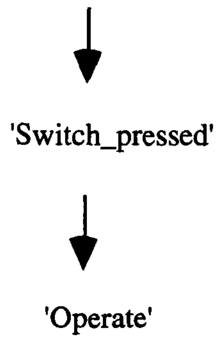


Figure 6.9 Scenario of operator

The signature of Operator is:

The switch is pressed - denoted by the event 'Switch_pressed'

Request sent to controller - 'Operate'

6.3.3.2 Truck

Truck moves in both the directions, this means the objective of a truck is to move towards platform B (forward) or towards A (reverse). Thus at a given time the truck is either stationary, or moving forward, or reverse.

The truck movement is managed by the controller. The three scenarios that arise with the truck are: to move forward, to move reverse, or to stop. Let's consider each individually.

To move towards platform B

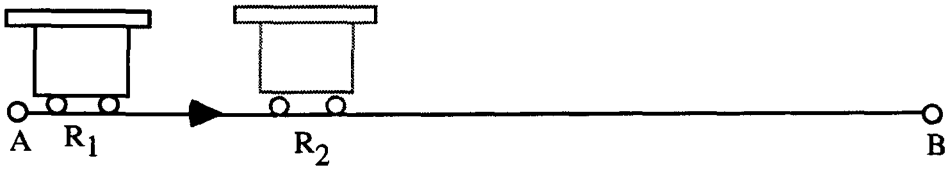


Figure 6.10(a) Truck moving in forward direction

The truck can start moving forward, only with the request from the controller. Let's say initially the truck is at position R_1, and starts moving towards B with the request, 'Move forward'. Initially the truck moves at a slow pace, and then increases the speed at R_2. The scenario of this is shown in Figure 6.10(a) and Figure 6.10(b).

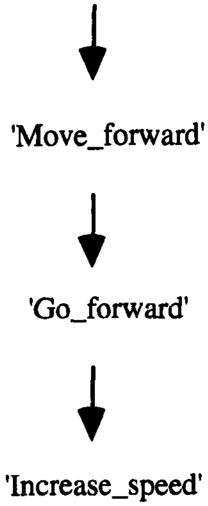


Figure 6.10(b) Scenario representing the truck moving towards platform B

To move towards platform A

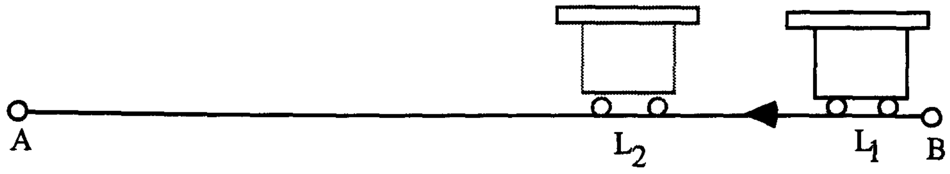


Figure 6.11(a) Truck moving in reverse direction

The truck can start moving towards platform A, only with the request from the controller. Let's say initially the truck is at position L_1, and starts moving towards A with the request, 'Move reverse'. Initially the truck moves at a slow pace, and then increases the speed at L_2. Such a scenario is shown in Figure 6.11(a) and Figure 6.11(b).

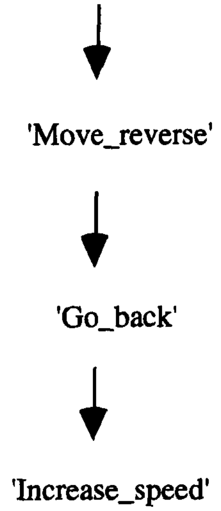
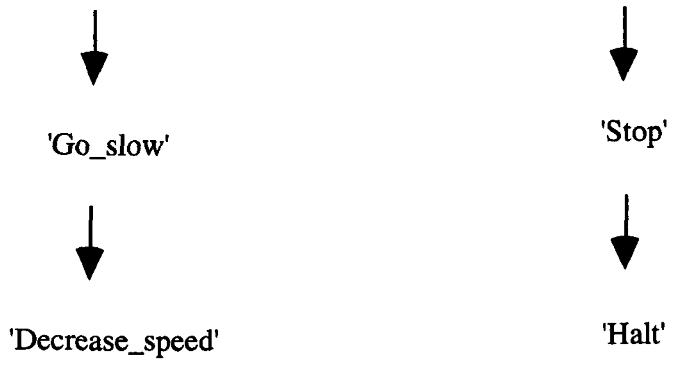


Figure 6.11(b) Scenario representing the truck moving towards platform A

To stop the truck

The moving truck requires to be stopped at Platform B, and A. The moving truck cannot be brought to halt suddenly. This activity involves two sub activities viz. making the truck to decrease the speed, and then to halt. The two scenarios are as shown in Figure 6.12



Step 1 Decrease the speed

Step 2 Halt the moving truck

Figure 6.12 Scenario while stopping the truck

Thus the signature of the truck is:

The controller requests the truck to move towards platform B - 'Move_forward'

The truck starts to move towards platform B - 'Go_forward'

The truck increases the speed - 'Increase_speed'

The controller requests the truck to move towards platform A - 'Move_reverse'

The truck starts to move towards platform A - 'Go_back'

The controller requests the truck to move slow - 'Go_slow'

The truck decreases the speed - 'Decrease_speed'

The controller requests the truck to stop - 'Stop'

The truck stops - 'Halt'

6.3.3.3 Monitor

The truck must stop at Platform B, during loading operation, and at Platform A during unloading. To stop the truck at a platform, the position of truck relative to the platform must be known. The monitor reports the position of truck with respect to the platform. We assume that the monitor also watches the loading of truck at Platform B, and reports the same to the controller.

As remarked above truck is halted in two steps, first by decreasing the speed, and then after a while the vehicle is halted. For such a reason, the monitor first reports, that the truck is approaching towards the platform, and then the truck's arrival at a platform.

Thus the monitoring operation is as shown in Figure 6.13.

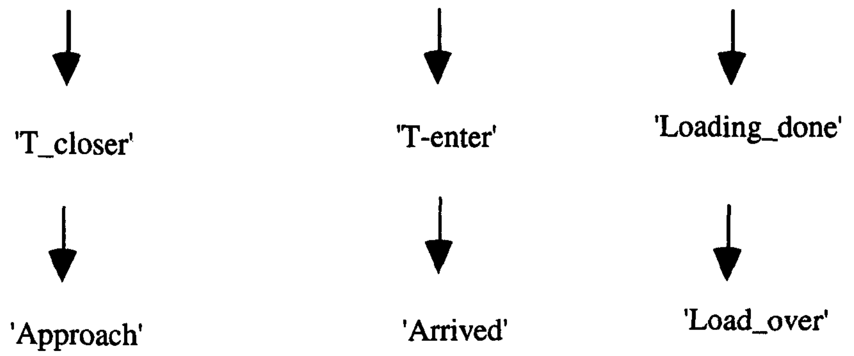


Figure 6.13 Scenarios representing the purpose of 'monitor'

The signature of Monitor is:

The monitor observes the truck approaching a platform - 'T_closer'

The monitor reports to the controller that the truck is arriving at a platform - 'Approach'

The monitor observes the truck is entering a platform - 'T_enter'

The monitor reports to the controller that the truck has arrived at a platform - 'Arrived'

The monitor observes that the loading in to the truck is completed - 'Loading_done'

The monitor reports to the controller that the loading is completed - 'Load_over'

6.3.3.4 Controller

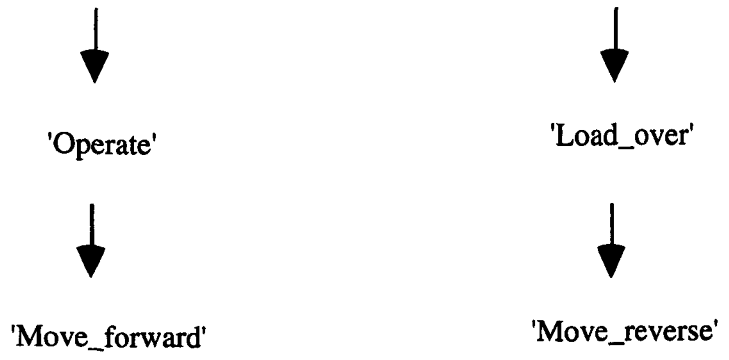
Controller manages the movement of truck in co-operation with the monitor, and the operator. We assume that the loading, and unloading operations are not dependent on the controller. The scenarios with the controller are:

(1) to move the truck from A to B (with operator request);

(2) to move the truck from B to A (when loading is completed); and

(3) to stop the truck at a platform

The controller initiates the loading operation, with the request from the operator. The controller similarly starts the unloading operation (i.e., moving the truck from B to A) after the loading is done at platform B. Thus the scenario of controller pertaining to this operation is as shown in Figure 6.14(a).

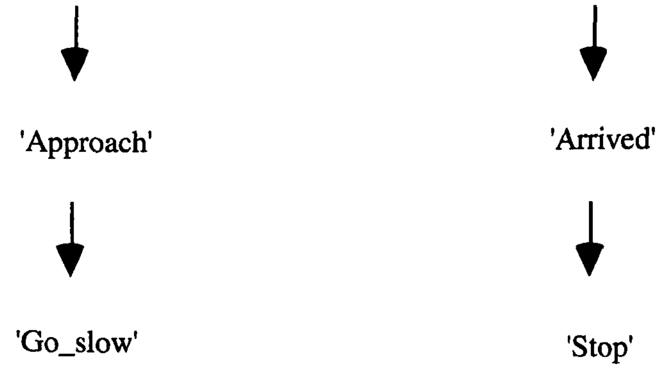


Scenario 1 Initiate the loading operation

Scenario 2 Initiate the unloading operation

Figure 6.14(a) Scenarios of 'controller' for moving the truck

A truck is stopped by knowing its position with respect to the platform. The position of truck is reported by the monitor. The controller commands the truck to decrease the speed, and then to stop as shown in Figure 6.14(b). This operation is done in two steps for the reason of safety.



Scenario 3 Reduce the speed of the Truck

Scenario 4 Stop the Truck

Figure 6.14(b) Scenarios of 'controller' for stopping the truck

Thus the signature of the controller is:

The operator signals the controller to start the operation - 'Operate'

The controller requests the truck to move towards platform B - 'Move_forward'

The monitor reports to the controller that the loading is completed - 'Load_over'

The controller requests the truck to move towards platform A - 'Move_reverse'

The monitor reports to the controller that the truck is approaching the platform - 'Approach'

The controller requests the truck to go slow - 'Go_slow'

The monitor informs the controller that the truck has arrived at a platform - 'Arrived'

The controller requests the truck to stop - 'Stop'

6.3.4 Higher Level Requirements

At this stage the requirements described above are refined in consultation with the users. We may not need any refinement at the function of operator, as it is very simple.

6.3.4.1 Operator

Translating the behaviour of operator in TRL we have:

```
Process Operator
   begin
      s1 : if (Switch_pressed, i) then  (Operate, j)    $
   end
```

Let's consider the operation of the agent 'truck'.

6.3.4.2 Truck

As indicated in Figure 6.10(b), and Figure 6.11(b), the speed of the vehicle has to be increased after some time since it started to move. A timing constraint of this sort has both a minimum, and a maximum timing constraint[33] associated with it.

In Figure 6.10(b), and Figure 6.11(b), we indicated that the speed of the truck can be increased after some time elapses, since starting the vehicle. The refined scenarios with the temporal constraint is shown in Figure 6.15(a).

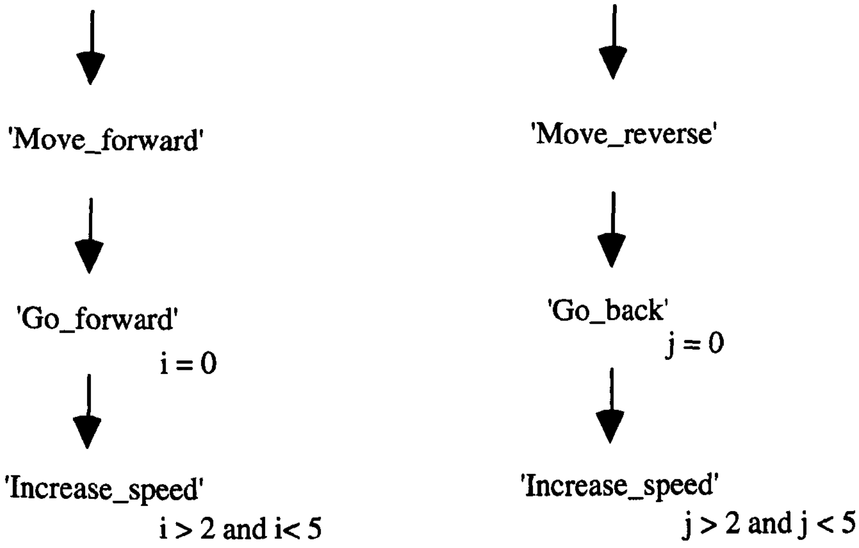


Figure 6.15(a) Scenarios of 'truck' with constraints, while in motion

[33] A minimum timing constraint, restricts an event to occur after a stipulated delay, and a maximum timing constraint enforces an event to occur within a maximum time.

Recall the scenario considered in Figure 6.12. In this scenario the truck is required to stop at a platform. The truck has to be stopped within some time. This is shown in Figure 6.15(b)

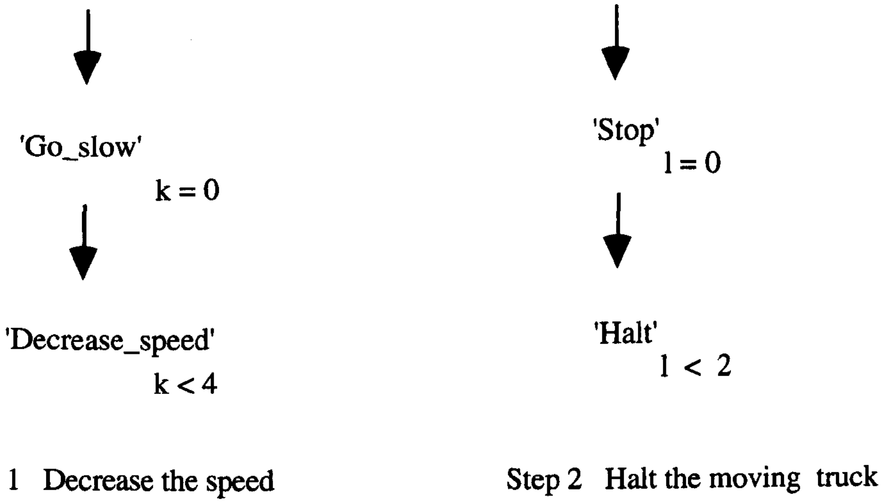


Figure 6.15(b) Scenarios of 'truck' while stopping at a platform with the stipulated constraints

Translating the behaviour of truck in TRL we have:

```
Process Truck
  begin

     s1 : if  (Move_forward, i1) then  (Go_forward, i2) ; (Increase_speed, i3)
                                  where (i3 > i2 + 2) and (i3 < i2 + 5) & s2

          elsif  (Move_reverse, j1) then  (Go_back, j2) ; (Increase_speed, j3)
                                  where (j3 > j2 + 2) and (j3 <j2 + 5)  &  s2  $

     s2 : if  (Go_slow, k1) then  (Decrease_speed, k2)
                                         where  (k2 < k1 + 4)  &  s3  $

     s3 : if  (Stop, l1) then  (Halt, l2)  where (l2 < l1 + 2)  $

 end
```

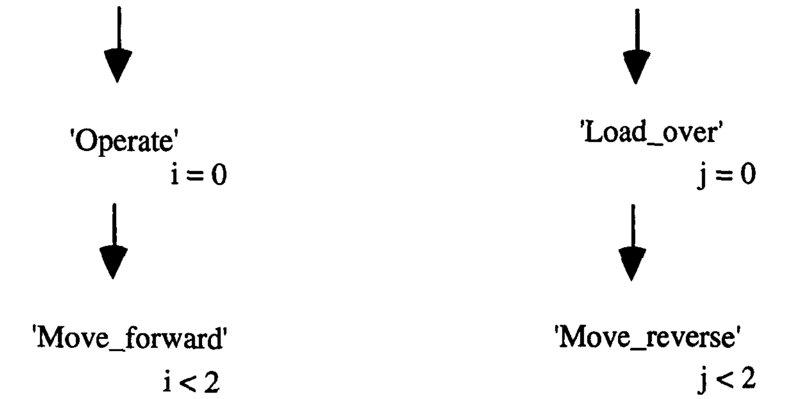
6.3.4.3 Monitor

Considering the scenario discussed in Figure 6.13, there are not any vital constraints on this. This behaviour in TRL is expressed as below.

```
Process Monitor
  begin
     s1 : if  (T_closer, i1) then  (Approach, i2)

          elsif  (T_enter, j1) then  (Arrived, j2)

          elsif  (Loading_done, k1) then  (Load_over, k2)   $

    end
```

6.3.4.4 Controller

The scenarios concerned with the controller is discussed in Figure 6.14(a) and Figure 6.14(b). The controller actions are time constrained. In Figure 6.16(a) the temporal requirements for initiating the loading and unloading operations are shown.

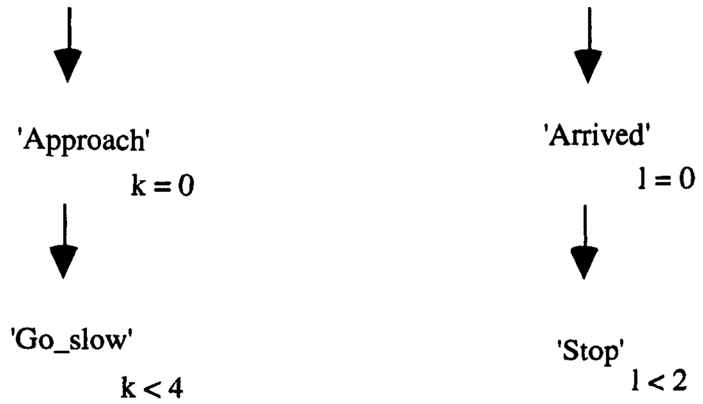


Scenario 1 Initiating the loading operation

Scenario 2 Initiating the unloading operation

Figure 6.16(a) Temporal requirements while moving the truck

Similarly Figure 6.16(b) describes the temporal requirements while stopping the truck at a platform.



Scenario 3 Reduce the speed of the Truck

Scenario 4 Stop the Truck

Figure 6.16(b) Temporal requirements while stopping the truck

These scenario of the controller are described in TRL as below.

```
Process Controller
   begin
      s1 : if  (Operate, i1) then  (Move_forward, i2)
                                                where (i2 <  i1 + 2)  &  s2

            elsif  (Load_over, j1) then  (Move_reverse, j2)
                                                 where (j2 <  j1 + 2)  &  s2  $

      s2 : if  (Approach, k1) then  (Go_slow, k2)
                                                 where  (k2 < k1 + 4)  &  s3  $

      s3 : if  (Arrived, l1) then  (Stop, l2)  where (l2 < l1 + 2)  $

   end
```

Here we have assumed that the material (to load into the truck) is always available at platform B, or the operator will have gathered that information before starting the operation. The entire system is then a composition of the agents discussed above. Thus the truck loading system as shown in Figure 6.17 is Operator || Truck || Monitor || Controller

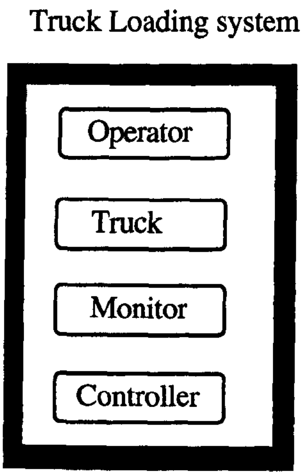


Figure 6.17 Representing the truck operating system

6.4 Observations

In the light of the arguments presented in the earlier chapters, and the case studies advanced here, we can deduce the following observations.

An understanding of the system can spring from concentrating on the *needs*, rather than concentrating on the finer points (the desires). The *needs* are the requirements that must be met under all circumstances. The desires are the requirements that must be taken into consideration. If we classify the desires, depending on their importance such as major, medium, and minor, then it may be of help to negotiate these requirements at a later stage.

Real-time systems control the physical processes. The needs can be better understood by understanding the domain of the controller, as the characteristics of the controller depends upon its domain.

The technical tasks are performed with the help of many technical artefacts, such as machines, and components. These artefacts have unique use in the system. The tasks of these components are normally too varied and complex. Depending on their use, the requirements engineer has to establish the particular purposes of these components. This helps to identify the agents in the system. The requirements determine the relationship between the agents. The functional relationship can be identified based on the needs. The combination of the agents results in a structure representing the overall needs.

This identification of agents allows a clear definition of the subsystems, so that they can be dealt separately. An agent has a purpose to the system. This purpose is perceived as a feature envisaged by the user. The feature is reported as a scenario. A scenario describes a purpose of an agent in a particular situation. This scenario can be abstracted as a sequence of events. Scenarios emphasise the important properties. The tasks of an agent can have task-specific constraints. These constraints are defined in the clearest possible terms.

The requirements model provides a platform, on which further discussions with the users can evolve. Such a discussion increases one's understanding of the system. A model described in the terminology of the users, helps in the validation of the model. Validation of the model is to determine the usefulness of the model with respect to the needs.

The requirements model addresses the abstraction. Some of the examples of this abstraction are:

- Do not design a rail-road crossing system, but look for the means of describing the objective of the system.

* Do not design a rail-road crossing system, but look for the means of describing the properties of the system.

From such formulations, the requirements can be derived in such a way that it does not prejudice the solution, and at the same time turns it into a function.

6.5 Summary

A well defined model provides a basis for formal communication among developers and the stakeholders. TRL provides such a model. The use of TRL permits the system to be described intuitively. TRL provides an approach for stating the requirements without the inclusion of unwarranted design details, ensures unambiguous communication of intent, and is responsive to the invariable changes of requirements.

Chapter 7

Evaluation

The approach is evaluated with other representative approaches discussed in Chapter 2. The evaluation of the approaches is driven through a case study.

7.1 Introduction

In the earlier chapter we studied the usefulness of the language TRL with case studies. The main impetus to the introduction of TRL arose from the awkwardness and poor readability of requirements caused by languages with arcane mathematical symbols, and by languages that included design level descriptions . We provided a rough genealogy of requirements and specification languages in Chapter 2. Requirements language must be chosen depending on the application in hand. The language should match the application as closely as possible. Real-time systems have specific requirements as studied in earlier chapters. Since requirements "maintainability" is often the largest desirable factor, the language must supplement the requirements without causing a sea of change in the whole of requirements document.

In the following sections we evaluate our approach with some of the other approaches with the help of a case study.

7.2 Cruise Control System

7.2.1 History

The problem was first posed by Ward (cf. Booch 86) and described in [Booch 86, Ward 85]. Booch uses the problem as a vehicle to explain object oriented concepts, while Ward Mellor 85 describe the problem with the Ward-Mellor approach.

7.2.2 Informal Problem Description

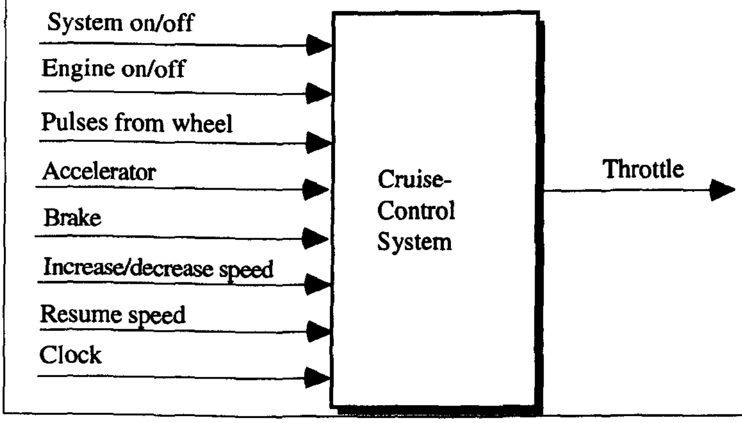
The input-output list as explained in [Booch 86] is as follows.

Inputs:

Engine on/off	If on, denotes that car engine is on
System on/off	If on, denotes that cruise-control is on if engine is on
Wheel pulse	A pulse is sent for every revolution of the wheel
Accelerator	Digital indicator of how far accelerator has been depressed
Brake	When brake is pressed, cruise-control reverts to manual control
Increase/decrease	Increase or decrease the maintained speed if cruise-control is on, and acts as initial *set* function for cruise-control
Resume	Resume the last maintained speed if cruise-control is on

Output:

Throttle	Digital value for engine throttle setting



The problem description following the above input-output list is as follows, and is adapted from [Ward 85].

A cruise control system relieves the driver of the responsibility for maintaining speed. The speed is maintained by monitoring the speed, and depressing, or accelerating to keep the actual speed close to the desired speed.

The Cruise Control System operates only when the engine is running, and is automatically reset to its "off" status when the engine is stopped. When the driver turns the system on, the speed at which the car is travelling at that instant is maintained. The system monitors the car's speed by sensing the rate at which the wheels are turning and maintains desired speed by monitoring and controlling the throttle position. The monitoring is accomplished by a sensor that produces a signal proportional to the throttle's position. The control is exercised by changing the degree of openness of a valve, which in turn controls a suction apparatus that draws on a chain to open the throttle. The throttle closes itself when not being actively controlled. After the system has been turned on, the driver may tell it to "start increasing speed", which causes the system to increase the speed at a fixed rate. When the driver tells the system to "stop increasing speed", it will maintain the speed reached at that point. Similarly, the driver may tell it to "start decreasing speed", which causes the system to decrease the speed at a fixed rate. When the driver tells the system to "stop decreasing speed", it will maintain the speed reached at that point.

Of course, the driver may turn the system off at any time. In addition, the driver can override the system to increase speed simply by depressing the accelerator pedal. This causes the chain controlling the throttle to go limp. During the period of greater speed, the system continues to attempt to maintain the speed previously set, and the system will return to the car to the previous speed when the driver releases the pedal. If the system is on and senses that the brake pedal has been depressed, it will cease maintaining speed but will not turn off. The driver may subsequently tell the system to resume speed (provided it hasn't been turned off in the interim), whereupon it will return at a fixed rate to the speed it was maintaining before braking and resume maintenance of that speed.

7.3 Application of Case Study

In the following sections we provide solution to the cruise control system in SREM, RTRL, PAISLey, and TRL.

7.3.1 SREM

As discussed in earlier chapters (see Chapter 2) SREM provides a set of tools to support the system development during the initial phase [Bell 77, Alford 77]. RSL is the base language of SREM. SREM approach is based on analysis of the data exchanged at the interfaces between the processing system and its peripheral hardware. Here it is assumed that each processing step involves receiving an input and transforming into an output. RSL expresses requirements in terms of processing paths. The processing path represents the sequence of data processing required to operate on an input stimulus and produce an output response. RSL provides information on the specification of requirements through the use of flow graphs. The flows through the system are specified by means of R-nets or requirements networks. The primary descriptive component of RSL is R-net (requirements network). Each R-net specifies the transformation of an input message to an output message. Each R-net is a graph with nodes representing structural and logical nodes. Subnets are used to shorten the length of an R-net. Each input message interface provides input to a distinct R-net, and the presence of data at that interface serves as an enabling condition for the R-net. An R-net can terminate producing an output message. The actual activity of an R-net is described in terms of processing tasks (called ALPHAs) and events (E-nodes), which describe the enabling of other R-nets.

R-net makes use of many symbolic representations. For example, the triangular nodes represent initiation and termination points. The hexagonal nodes are external input and output interfaces. The rectangular nodes represent the ALPHAs, and the circular nodes are the E-nodes. The graph structure on the R-net uses OR nodes to specify the conditions of processing. The AND nodes represent the paths that must be executed in any order.

7.3.1.1 Use of the Technique

The technique starts from identifying the stimulus-response, then creating R-nets and ALPHAs. RSL provides textual description of R-net. The system is thought of as a net that consumes the input and produces the output. The net can consist of sub-nets to allow for the expression of large requirements.

The SREM method follows the following phases for the production of specification:

- identification of the interface between controller and environment, and data description and processing;

- produce an initial description using R-nets.;

- specify data and behaviour of ALPHA functions in RSL;

- validate the specification using validation points;

- identify performance specifications like timing constraints.

Figure 7.1 shows how the network might look for the cruise control system. The circled plus indicates a condition for which the process may branch. In the example, either the left or the right branch may be taken. The circled ampersand indicates that processes must be followed in parallel, and in any order. The main tasks of cruise control system are to find the current speed, to calculate the desired speed, to get the brake status, and to calculate the throttle setting. To make the net readable we have used the sub-nets as shown in Figure 7.2, Figure 7.3, and Figure 7.4.

RNET: CRUISE CONTROL SYSTEM

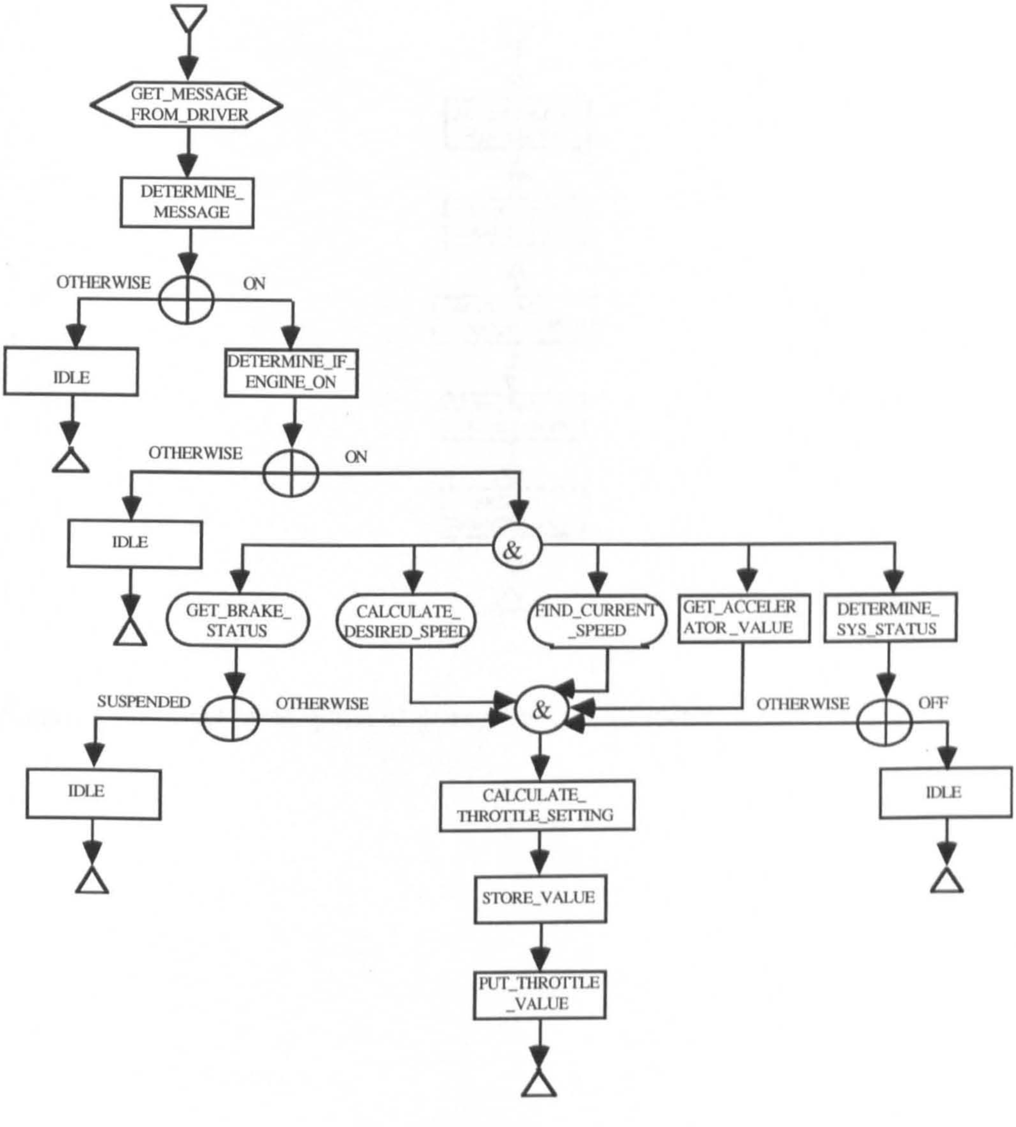


Figure 7.1 R-Net description of the system

SUBNET: FIND_CURRENT _SPEED

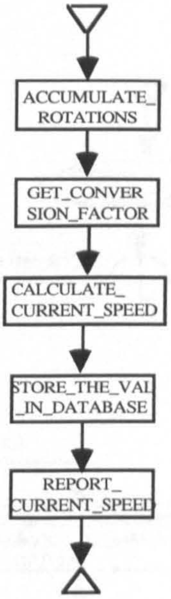


Figure 7.2 Subnet Description of getting the current speed

SUBNET: GET_BRAKE_STATUS

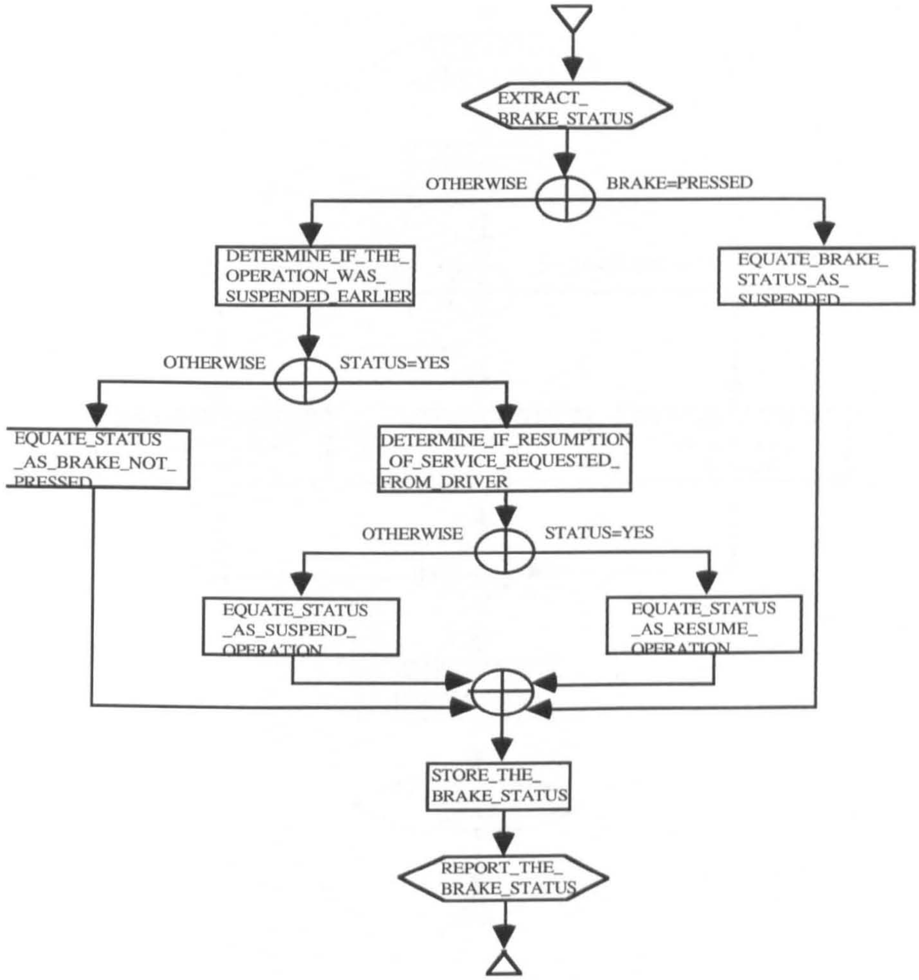


Figure 7.3 Subnet Description of setting the brake status

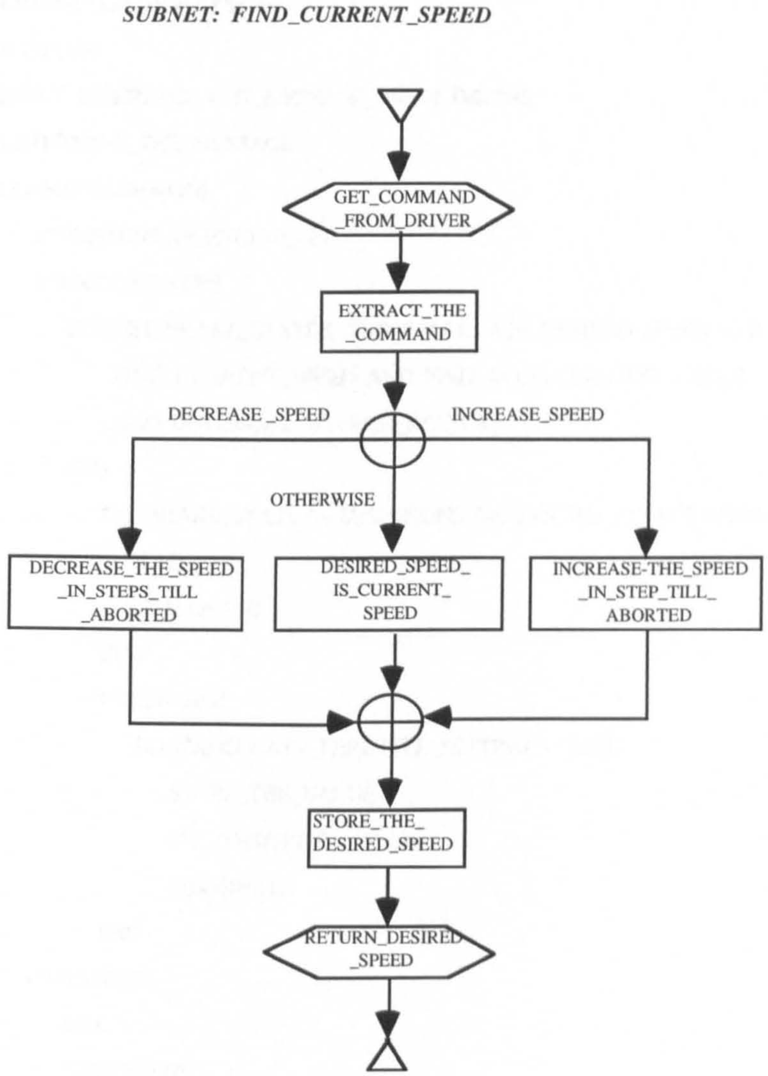

SUBNET: FIND_CURRENT_SPEED
GET_COMMAND_FROM_DRIVER
EXTRACT_THE_COMMAND
DECREASE_SPEED
INCREASE_SPEED
OTHERWISE
DECREASE_THE_SPEED_IN_STEPS_TILL_ABORTED
DESIRED_SPEED_IS_CURRENT_SPEED
INCREASE-THE_SPEED_IN_STEP_TILL_ABORTED
STORE_THE_DESIRED_SPEED
RETURN_DESIRED_SPEED

Figure 7.4 Subnet Description of getting the desired speed

7.3.1.2 RSL Description

After the R-net diagrams are written, then the components of each diagram are translated into their corresponding RSL statements. For example the R-net depicted in Figure 7.1 is written in RSL language as shown in Figure 7.5. Similarly the R-net in Figure 7.2 is depicted in Figure 7.6, Figure 7.3 in Figure 7.7, and Figure 7.4 in Figure 7.8.

```
R_NET: CRUISE CONTROL SYSTEM
  STRUCTURE:
    INPUT_INTERFACE GET_MESSAGE_FROM_DRIVER
    DETERMINE_THE_MESSAGE
    DO (MESSAGE = ON)
      DETERMINE_IF_ENGINE_ON
      DO (STATUS = ON)
        DO (GET_BRAKE_STATUS AND CALCULATE_DESIRED_SPEED AND
           FIND_CURRENT_SPEED AND FIND_ACCELLERATOR_VALUE
           AND DETERMINE_SYSTEM_STATUS)
        END
          DO (BRAKE_STATUS = SUSPENDED OR SYSTEM_STATUS = OFF)
            IDLE
            TERMINATE
          END
          OTHERWISE
            DO (CALCULATE_THROTTLE_SETTING_VALUE)
              STORE_THE_VALUE
              PUT_THROTTLE_VALUE
              TERMINATE
          END
      OTHERWISE
        IDLE
        TERMINATE
      END
    OTHERWISE
      IDLE
      TERMINATE
    END
  END
```

Figure 7.5 RSL description of R-net shown in Figure 7.1

```
SUBNET: CALCULATE_DESIRED_SPEED
    STRUCTURE
        INPUT INTERFACE  GET_COMMAND_FROM_DRIVER
        EXTRACT_THE_COMMAND
        DO (COMMAND = INCREASE SPEED)
            INCREASE_THE_SPEED_IN_STEP_TILL_ABORTED
        DO (COMMAND = DECREASE_SPEED)
            DECREASE_SPEED_IN_STEPS_TILL_ABORTED
        OTHERWISE
            EQUATE_DESIRED_SPEED_AS_CURRENT_SPEED
        END
        STORE_THE_DESIRED_SPEED
        OUTPUT INTERFACE   RETURN_THE_DESIRED_SPEED
        TERMINATE
        END
```

Figure 7.6 RSL description of R-net shown in Figure 7.2

```
SUBNET: FIND_CURRENT_SPEED
    STRUCTURE
        ACCUMULATE_WHEEL_ROTATIOIN
        GET_CONVERSION_FACTOR
        CALCULATE_THE_CURRENT_SPEED
        STORE_THE_VALUE_IN_DATABASE
        OUTPUT INTERFACE  REPORT_CURRENT_SPEED
        TERMINATE
    END
```

Figure 7.7 RSL description of R-net shown in Figure 7.3

```
SUBNET: GET_BRAKE_STATUS
   STRUCTURE
     INPUT INTERFACE  EXTRACT_THE_BRAKE_STATUS
     DO (BRAKE = PRESSED)
       EQUATE_THE_BRAKE_STATUS
     OTHERWISE
       DETERMINE_IF_THE_OPERATION_WAS_SUSPENDED_EARLIER
       DO (STATUS = YES)
        DETERMINE_IF_RESUMPTION_OF_SERVICE_REQUESTED_FROM_DRIVER
         DO (STATUS = YES)
           EQUATE_STATUS_AS_RESUME_OPERATION
         OTHERWISE
           EQUATE_STATUS_AS_SUSPEND_OPERATION
       OTHERWISE
         EQUATE_STATUS_AS_BRAKE_NOT_PRESSED
         END
       END
    END
  STORE_THE_BRAKE_STATUS
  OUTPUT INTERFACE   REPORT_BRAKE_STATUS
  TERMINATE
  END
```

Figure 7.8 RSL description of R-net shown in Figure 7.4

7.3.2 RTRL

RTRL was developed by GTE Laboratories for expressing the requirements of telecommunication systems [Taylor 83, Dasarathy 85, Chandrasekharan 85, Casey 82]. RTRL provides the textual description of finite-state machine (FSM). RTRL makes use of explicit use of states, transitions, and decision points (check points). Like RSL the system is analysed by stimulus-response sequences. The system after receiving a stimulus moves to a new state while providing a response. RTRL

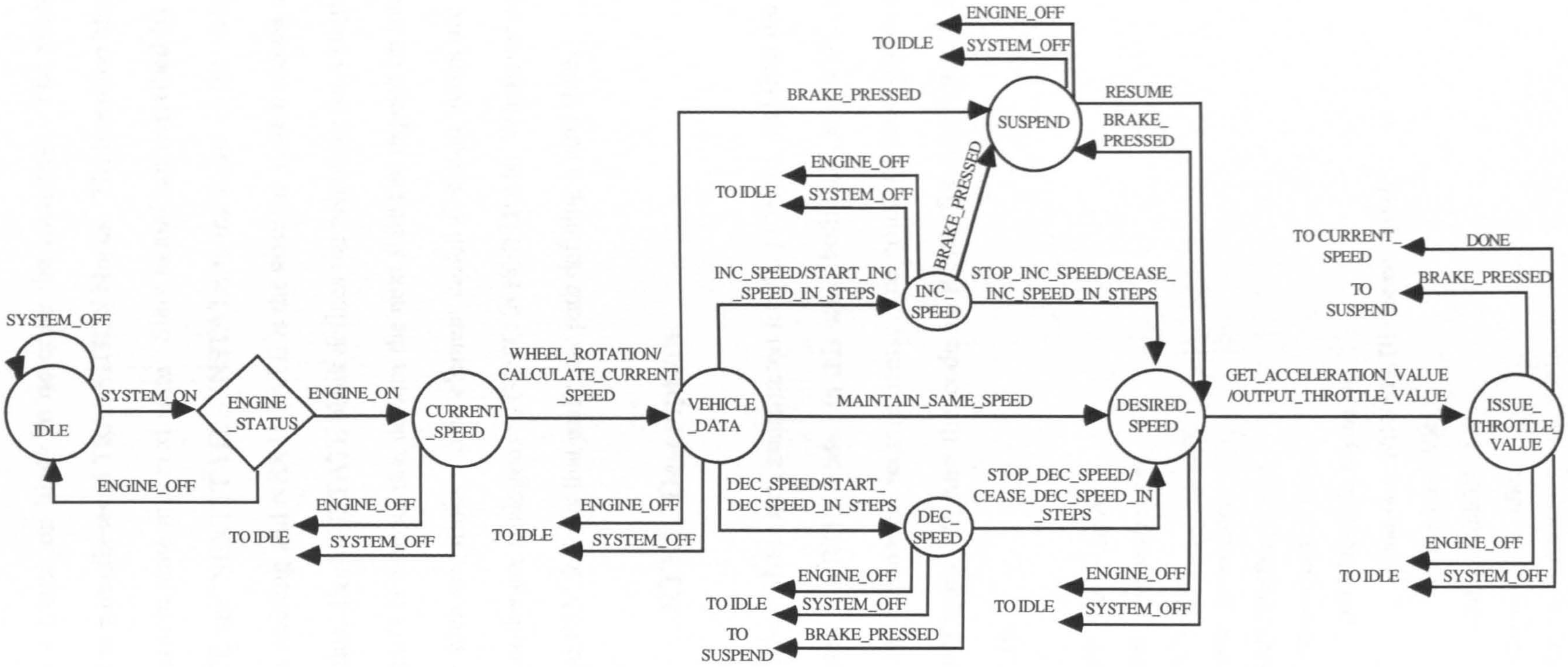


Fig. 7.9 RTRL Description of the System

provides a transition block to describe the transition. The stimuli names are enclosed in parentheses in TRANSITION blocks. The transition block is delimited by the present state and next state. State names are described by the keywords INSTATE and NEXTSTATE. INSTATE is the name of the state at which the system is residing and NEXTSTATE is the state the system moves upon receiving the stimulus. Each INSTATE block defines the behaviour for a single system state. The system's behaviour that defines the next state can depend on internal data. To describe such situations a new element called decision nodes are defined. The decision nodes are analogous to GOTO in FORTRAN. Whenever a possible result is selected FSM follows that particular path defining a next state.

7.3.2.1 RTRL Description

In RTRL the FSM is first constructed for the problem, and then the FSM network is translated into RTRL code. In this sense both RSL, and RTRL share the same view of developing the requirements. The cruise control system in RTRL is as shown in Figure 7.9, and the code in RTRL for the Figure 7.9 is shown in Figure 7.10.

```
%FEATURE  idle_to_active;
%FEATURE  turn_system_on;
    INSTATE  idle;
        SEND  system_on;
        TRANSITION;
            (system_on):
                DECISION  is_engine_on;
                    (engine_on):  NEWSTATE  current_speed;
                    (engine_off):  idle;
                ENDDECISION;
            (system_off):  idle;
        ENDTRANSITION;
```

```
INSTATE  current_speed;
    SEND  request_for_wheel_rotations;
    TRANSITION;
        (wheel_rotation):
            SEND  calculate_the_current_speed;
            NEWSTATE  vehicle_data;
            (system_off):  NEWSTATE  idle;
            (engine_off):  NEWSTATE  idle;
    ENDTRANSITION;
INSTATE  vehicle_data;
    TRANSITION;
        (brake_pressed):  NEWSTATE  suspended;
        (increase_speed):
            SEND  start_increasing_speed_in_steps;
            NEWSTATE:  increase_speed;
        (maintain_same_speed):  NEWSTATE  (desired_speed);
        (decrease_speed):
            SEND  start_decreasing_speed_in_steps;
            NEWSTATE:  decrease_speed;
        (system_off):  NEWSTATE  idle;
        (engine_off):  NEWSTATE  idle;
    ENDTRANSITION;
        INSTATE  increase_speed;
            TRANSITION;
                (stop_increasing_speed):
                    SEND  cease_increase_speed;
                    NEWSTATE:  desired_speed;
                    (brake_pressed):  NEWSTATE  suspended;
                    (system_off):  NEWSTATE  idle;
                    (engine_off):  NEWSTATE  idle;
            ENDTRASITION;
        INSTATE  decrease_speed;
            TRASITION;
                (stop_decreasing_speed):
                    SEND  cease_decraese_speed;
                    NEWSTATE:  desired_speed;
                (brake_pressed):  NEWSTATE  suspend;
            (system_off): NEWSTATE idle;
            (engine_off): NEWSTATE idle;
         ENDTRANSITION
      INSTATE desired_speed;
         SEND request_for_acceleration_value;
         TRANSITION;
            (get_acc_value):
               SEND output_throttle_value;
               NEWSTATE: issue_throttle_value;
            (brake_pressed): NEWSTATE suspend;
            (system_off): NEWSTATE idle;
            (engine_off): NEWSTATE idle;
         ENDTRANSITION;
      INSTATE suspend;
         TRANSITION;
            (resume): NEWSTATE desired_speed;
            (system_off): NEWSTATE idle;
            (engine_off): NEWSTATE idle;
         ENDTRANSITION;
      INSTATE issue_throttle_setting;
         TRANSITION;
            (brake_pressed): NEWSTATE suspend;
            (engine_off) NEWSTATE idle;
            (system_off) NEWSTATE idle;
            (done) NEWSTATE current_speed;
         ENDTRANSITION;
```

Figure 7.10 RTRL description of the system shown in Figure 7.9

7.3.3 PAISLey

The Process-oriented Applicative and Interpretable Specification Language was developed by Zave [Zave 82]. PAISLey was targeted towards the specification of embedded systems. The detailed design description feature of the language was described as an operational approach to the system specification [Zave 84]. The requirements is specified by the interacting model of system and its environment processes. The specification in PAISLey involves writing the code for the processes. Understanding of the system's behaviour is achieved by executing the code. A system is a structure of cyclic processes. Some of the real-time features like sequencing and control flow is difficult to specify [Zave 91b].

As shown in Figure 7.11 the system and its environment is decomposed into sets of interacting processes. The processes are further defined by defining the state space of the process, and by declaring and defining the successor function (the function that defines how process changes the state), and exchange function (the function that defines how process interact). Figure 7.12 the definition of processes is shown. In figure 7.12:

- The overall speed controlling process is defined as a function mapping the controller state into controller state; for example, it is capable of changing the state from "idle" to "output the throttle value to control the speed". To take any action the system depends upon the driver command, the engine status, the current speed data, the desired speed data, the accelerator data, and the brake state.

- The update of the current speed data base depends upon the speed data measured from the wheel. The wheel speed measuring function reads hardware sensors attached to the wheel and converts into actual speed data. This data is stored in the database.

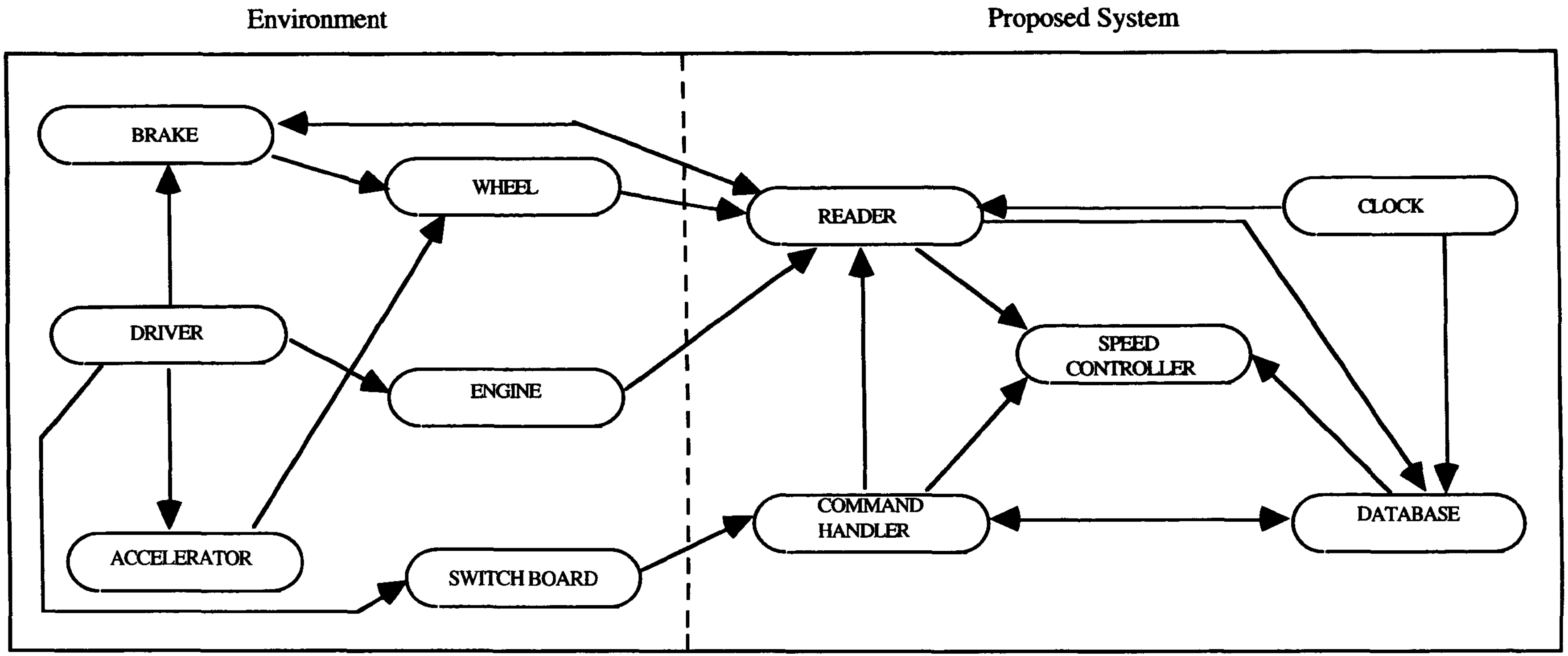


Fig 7.11 Primary Processors for the System and its Environment

- The update of the required speed data base depends upon the command from the driver. The driver can set the required (desired) speed either above or below the current speed.

- The update of the brake status depends upon the command (actions) from the driver. The brake status can be either idle, or pressed. The state changes depending on the driver action.

- The engine state is independent of controller. Engine state can change from on to off. The system is operative only when the engine is on.

Controller Cycle:	CONTROLLER STATE * DRIVER COMMAND DATA * CURRENT SPEED DATABASE * REQUIRED SPEED DATABASE * BRAKE STATE * ACCELERATOR DATA * ENGINE STATUS → CONTROLLER STATE
Current Speed Update:	CURRENT SPEED DATABASE * WHEEL DATA → CURRENT SPEED DATABASE
Required Speed Update:	DESIRED SPEED DATABASE * DRIVER SPEED DATA → DESIRED SPEED DATABASE
Brake Status Update:	BRAKE STATE * DRIVER COMMAND DATA → BRAKE STATE
Monitor Engine state:	ENGINE STATE → ENGINE STATE

Figure 7.12 Declaration of PAISLey processes

7.3.4 TRL

TRL is aimed at the conceptualisation of real-time systems. In the literature, extensive studies have pointed out that most of the eventual system errors could be traced to problems in the requirements definition, due largely to the complexity of extracting the requirements from volumes of narrative system description. TRL focuses on the needs and objectives of the system, and provides a framework in which the descriptions are expressed with the much needed simplicity.

Framework for Modelling the System

TRL establishes a framework by analysing and identifying the agents relevant to the system. As we begin to analyse a system, we find many parts that interact. These parts involve details of inescapable complexity. The fundamental task of requirements engineer is to mask this complexity while focusing on the aims of the system. TRL identifies agents that help us to make intelligent decisions regarding the separation of concerns, and provides an economy of expression.

Modelling Agents

In this section, we briefly describe the role played by each agent. An agent has a specific responsibility to the system. This responsibility defines its use. The detailed narration of the use of an agent is provided by scenarios. As explained in earlier chapters, the scenarios are characterised by events.

Driver

The main function of the driver can be abstracted by following scenarios. The driver can bring the system into operation by pressing a switch, and similarly can take the system out of operation by pressing another switch. While system is on, the driver can ask the system either to increase or decrease the speed in steps, and later aborts the process of varying the speed.

The scenarios are abstracted by events, and are as shown below.

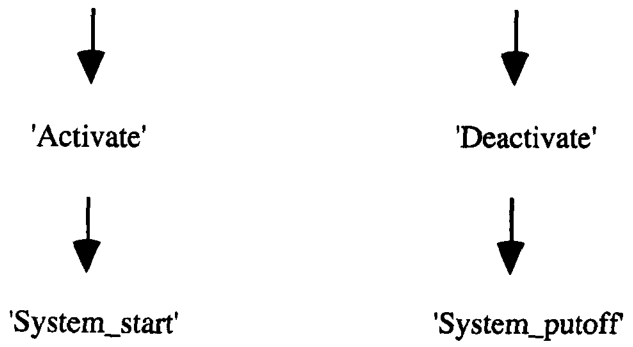


Figure 7.13(a) Scenario of driver activating/deactivating the system

Similarly the driver can increase or decrease the speed in steps. The corresponding scenarios are as shown below.

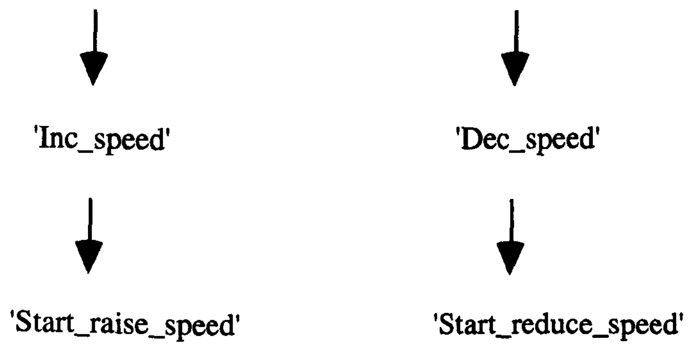


Figure 7.13(b) Scenario of driver initiating the process of varying the speed

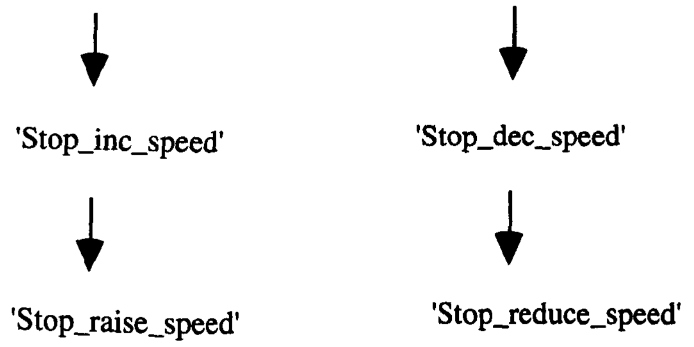


Figure 7.13(c) Scenario of driver terminating the process of varying the speed

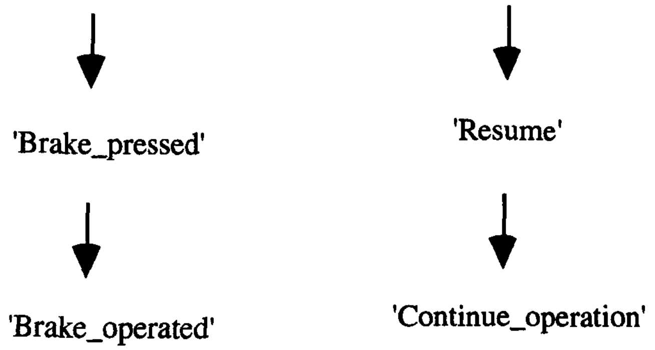


Figure 7.13(d) Scenario of driver operating the brake

The signature of the Driver is:

Driver initiates the system to operate - Activate

System status turned on - System_on

Driver instructs the system to shut-down - Deactivate

System status turned off - System_putoff

Driver instructs the system to increase the speed in steps - Inc_speed

Message sent to increase the speed in steps - Start_raise_speed

Driver instructs the system to decrease the speed in steps - Dec_speed

Message sent to decrease the speed in steps - Start_reduce_speed

Driver instructs the system to stop increasing the speed - Stop_inc_speed

Message sent to stop increasing the speed in steps - Stop_raise_speed

Driver instructs the system to stop decreasing the speed - Stop_dec_speed

Message sent to stop decreasing the speed in steps - Stop_reduce_speed

Driver presses the brake - Brake_pressed

Message sent to denote that the brake is operated - Brake_operated

Driver requests for the resumption of service - Resume

Message sent to resume the operation - Continue_operation

Static Constraints associated are:

Engine is on - engine_on

Translating the above scenarios in TRL we have:

```
Process Driver
 begin
   s1: if (Activate, i1) and (engine_on) then (Sys_on, i2) & s2 %

   s2: if (Deactivate, j1) then (Sys_putoff, j2)
          elsif (Inc_speed, k1) then (Start_raise_speed, k2) & s3
          elsif (Dec_speed, l1) then (Start_reduce_speed, l2) & s4
          elsif (Brake_pressed, m1) then (Brake_operated, m2) & s5 %

   s3: if (Stop_inc_speed, n1) then (Stop_raise_speed, n2) %

   s4: if (Stop_dec_speed, p1) then (Stop_reduce_speed, p2) %

   s5: if (Resume, q1) then (Continue_operation, q2) %
end
```

Speed Sensor

The Speed sensor, measures the current speed by monitoring the wheel rotations. A conversion factor is used to calculate the current speed with the accumulated wheel rotations.

The scenario is as shown in Figure 7.14.

The signature of Speed_Sensor is:

System status turned on - System_on

Wheel rotations are collected to measure the current speed - Get_wheel_rotations

Conversion factor is recalled to calculate current speed - Get_conversion_factor

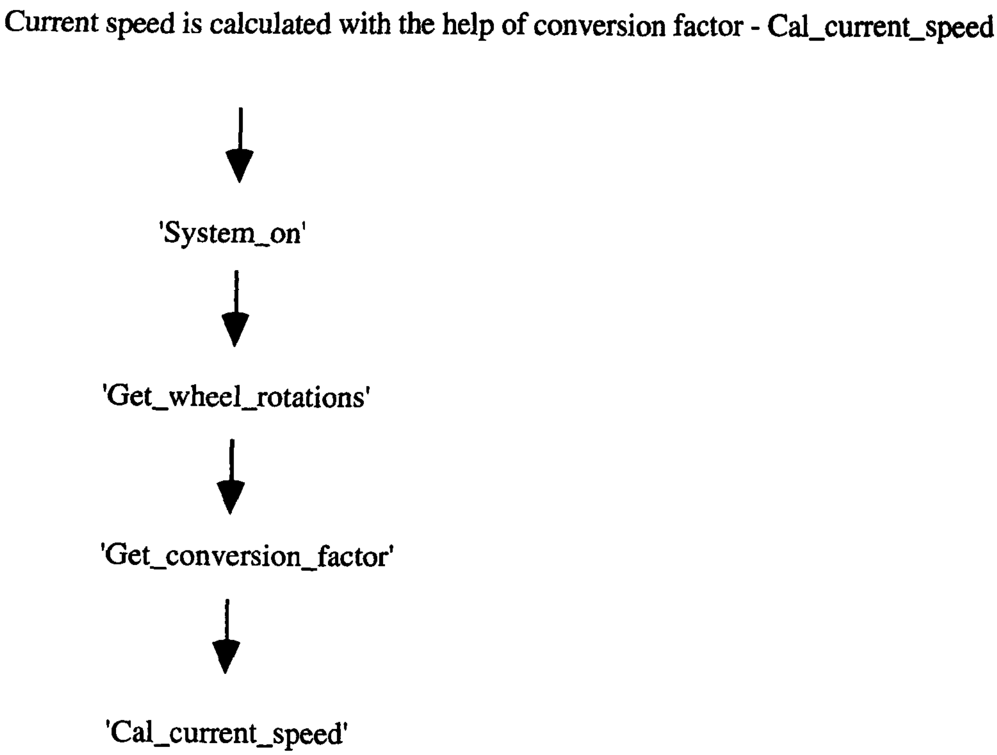

Current speed is calculated with the help of conversion factor - Cal_current_speed
'System_on'
'Get_wheel_rotations'
'Get_conversion_factor'
'Cal_current_speed'

Figure 7.14 Scenario of 'speed_sensor' monitoring the current speed

Expressing the behaviour in TRL we have:

```
Process Speed_Sensor
 begin
   s1: if (System_on, t1) and (engine_on)  then (Get_wheel_rotations, t2) ;
                        (Get_conversion_factor, t3) ; (Cal_current_speed, t4)  %
 end
```

Monitor

The monitor, monitors the operator command to vary the speed. The desired speed varies, when the driver wishes either to increase or decrease the current speed. The speed is varied in steps till aborted.

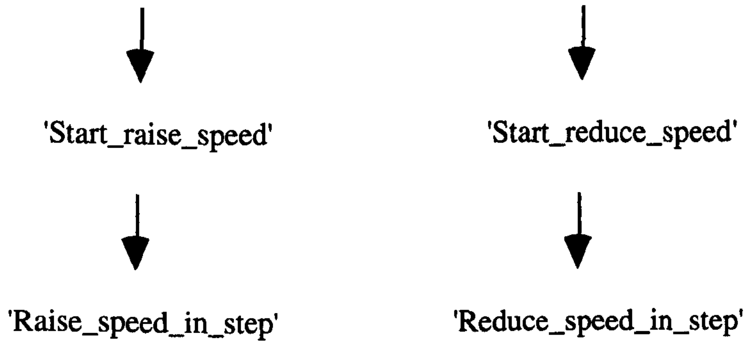


Figure 7.15 (a) Scenario of 'monitor' start varying the speed

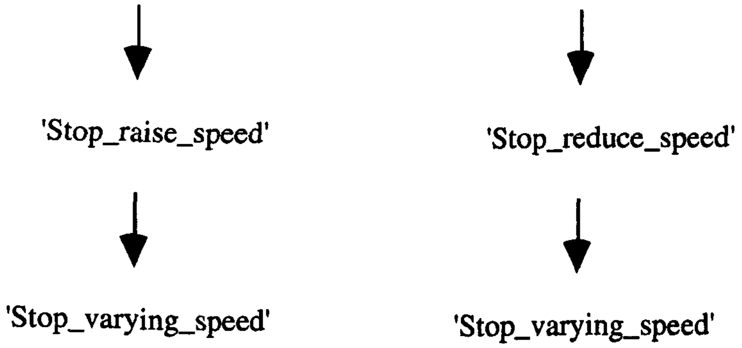


Figure 7.15 (b) Scenario of 'monitor' stop varying the speed

The signature of Monitor is:

Monitor recognises the request to start increase speed - Start_raise_speed

Monitor takes action to raise the speed in step - Raise_speed_in_step

Monitor recognises the request to start decrease speed - Start_reduce_speed

Monitor takes action to decrease the speed in step - Reduce_speed_in_step

Monitor recognises the request to stop increasing speed - Stop_raise_speed

Monitor recognises the request to stop decreasing speed - Stop_reduce_speed

Monitor takes action to stop varying the speed in step - Stop_varying_speed

The above scenarios are translated in TRL as below.

```
Process Monitor
  begin
     s1: if (Start_raise_speed, e1) and (engine_on) then
                                        (Raise_speed_in_step, e2) & s2
            elsif (Start_reduce_speed, f1) and (engine_on) then
                                        (Reduce_speed_in_step, f2) & s3  %

    s2: if (Stop_raise_speed, g1) then (Stop_varying_speed, g2) %

    s3: if (Stop_reduce_speed, h1) then (Stop_varying_speed, h2) %

end
```

Controller

Controller controls the vehicle speed by issuing the value for the throttle position. The throttle value is decided depending on the current speed, the desired speed, and the accelerator value.

The scenario is as shown in Figure 7.16.

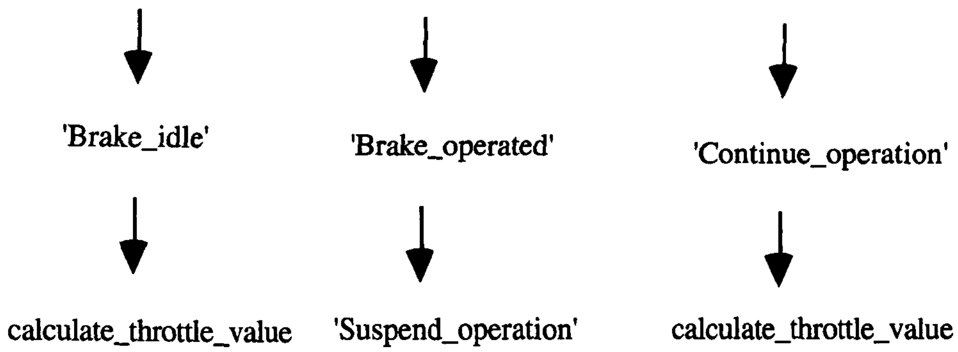


Figure 7.16 Scenario of 'controller' as an agent

The signature of Controller is:

Controller recognises that the break is idle - Break_idle

Controller recognises that the break is operated - Break_operated

Controller recognises the continue operation request - Continue_operation

Controller suspends the operation - Suspend_operation

Controller collects the current speed data - Get_cs

Controller collects the desired speed data - Get_ds

Controller collects the accelerator value - Get_acc_value

Calculate and output throttle value - Issue_th_value

The static constraint associated can be expressed as:

system live = (sytem_active) and (engine_on)

Translating the behaviour in TRL we have,

```
Process Controller
 begin
   s1: if (Brake_idle, a1) and (system_live) then (Get_cs, a2) ; (Get_ds, a3) ;
                                       (Get_acc_value, a4) ; (Issue_th_value, a5)
        elsif (Brake_operated, b1) and (system_live) then
                                                    (Suspend_operation, b2) & s2 %

  s2: if (Continue_operation, c1) and (system_live) then (Get_cs, c2) ;
                             (Get_ds, c3) ; (Get_acc_value, c4) ; (Issue_th_value, c5) %
end
```

The entire system is then a composition of the agents discussed above. Thus the entire system is as shown in Figure 7.17

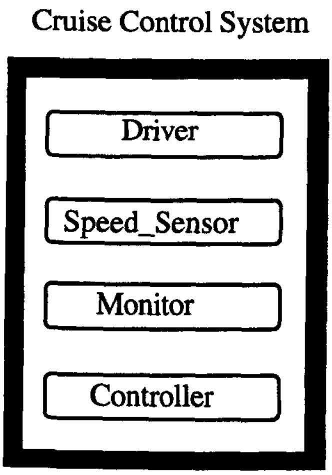


Figure 7.17 Representing cruise control system

7.4 Evaluation of What and for What?

An evaluation is meant to determine the usefulness of a solution with respect to a given objective. An objective is fundamental here. The importance of a solution must be gauged in terms of these objectives. Here we recall the evaluation criteria discussed in Chapter 2.

The main criterion of requirements modelling is to promote the circumstances, in which requirements engineers can develop a clear understanding of the system. If this is achieved a requirements engineer is more likely to develop an accurate representation of the system. Identification of user's overall perception of the system helps to promote the requirements engineers' and users' understanding of how the system works. It is essential that the requirements engineer builds an accurate conceptual representation of the system. This is essential as the end product should meet expectations - the customer likely to assert.

This conveys the factors such as:

- is it possible to take this representation back to the users, in order for them to usefully comment and further explain, if necessary?

- is this representation able to provide a generic description of the system?

As discussed in Chapter 3, timing constraints involve both safety and temporal requirements. As explained in Chapter 5, it is necessary for the formalism to state all types of timing constraints that may arise in a system. Also it is necessary that the timing requirements be described easily, as these requirements are used by various parties including users. This conveys the factors such as:

- how easy it is to reason about time in the formalism? and

- is it possible to state all types of timing properties?

The problems with requirements (as explained in earlier chapters) are:

- ill-disciplined requirements;
- disassociation of validation from users;
- bulk of information; and
- inappropriateness of description languages.

As computer controlled systems are expanding unabatedly, the systems become more specific, and the requirements language plays an increasingly significant role. Conceptual models are the abstract representations of the system which omit the fine details of the system, and faithfully reflect its externally observable characteristics. In this representation, whatever is represented is done so to the level of rigour and accuracy that there is an adequate basis for suitability assessment. Also, the representations are required to be in a medium, in which alterations can be more easily investigated. The representation medium must include:

- ability to mirror real-time systems requirements, and to support a mechanism for formal communication of requirements within the structure identified by the model;

- ability to capture the realities of real-time system, with emphasis on dynamic constraints; and

- ability to describe the performance implications of timing constraints. The description of timing constraint must recognise not only the importance of the time with which the job has to be done, but also the hidden fact of what if the job could not be completed within the prescribed time. Temporal requirements describe the constraints to be observed in a system. To consider the temporal issue seriously, the formalism must also incorporate a mechanism to describe the timing exceptions in the syntax of the language.

7.4.1 Comparison of the Approaches

Every technique is likely to have some kind of framework for rationalising concepts and requirements in order to allow clarity and professional communication. The same system can have different types of descriptions as shown above (with the cruise control system). We discussed the salient features of the techniques above. The example discussed above, highlights differences in philosophy and the approach taken by each of the techniques.

As explained in Chapter 2, the requirements language has at least three goals: as an analysis tool; as a vehicle for human communication; and as a vehicle towards automation. In the following sections we drive our discussion of the techniques through these goals. Here we recall our earlier discussion of the approaches, and also the solution of the case study to comment on these characteristics.

7.4.1.1 Analysis Tool

Analysis comprises of descriptive representations that depict the motivation of the model. The basic motivation of the model is to describe the objectives. The objectives are achieved by strategies, the kind of descriptions that achieve the conceived goal. Traditional approaches have a bias towards functional decomposition. Such an approach normally results in a system that is rigid, and often unmaintainable [Heitmeyer 83].

As explained in Chapter 2, a model can represent different perspectives. The three perspectives provide a representation for each class of player involved in the development of the system: requirements engineer, specifier, and designer. The requirements engineer has a conceptual representation of the product that serves some purpose. The specifier transcribes this perception of product into the operational model. Next the designer translates this representation into a solution model.

As each perspective reflects a different set of constraints, the meaning or definition of the modelling language changes depending upon the emphasis of the perspective. For example, the RSL statements provide meaning for the specifier, while the constructs of PAISLey have a meaning for the designer. Since the emphasis of each perspective is different, the structure and the objective of the model is likely to be different. Since each language has a unique basic model, the meaning (and thus its usefulness) of the model is unique.

In the following section we describe the approaches with dimensions of analysis, such as abstraction, localisation, uniformity, and temporal reasoning. The notation has to provide facilities that makes it easier to model the essential properties without getting into its details. Such a notation allows the important properties to be expressed and distinguished.

As observed by many researchers (for example Luqi 88) SREM does not support abstractions very well. When a modeller tries to understand a system, the way the system is to be designed, then the computing aspects become more important than the conceptual understanding of the requirements. Commenting on the practical use of SREM, Scheffer *et al* [Scheffer 85] observe that the description of the system takes the form of initial system specification. They express that, "it was developed to specify software requirements after the system requirements analysis phase has been completed, but before any detailed processing algorithms have been formed". As shown in the example, R-net provides a mechanism to decompose the requirements, in terms of ALPHAs. The RSL notation is used to define data access properties. The structures in RSL consists of primitives whose meaning may be unclear [Bell 77]. SREM does not provide a unified approach to define requirements. With any change in requirements more than one document needs modification.

The abstraction level in RTRL depends on stimulus-response sequence. In large systems, it is difficult to think of a system's features in terms of stimulus-response [Davis 88]. Also as Taylor observes [Taylor 82] many of the system failures can be attributed to the system being viewed merely as a transformation of stimulus to response. RTRL does not provide any mechanism to decompose the requirements. The descriptions in RTRL follows the FSM description. Without the FSM drawing it is difficult to understand the description. RTRL as a modelling language is very inefficient. It is difficult to express the requirements of large systems as a monolithic state machine. Also such a description is difficult for the user and requirements engineer to visualise the activities in a system.

PAISLey needs complete description of the way the functions are achieved. Zave [Zave 84] defends the criticism by stating the approach as operational. Zave [Zave 91b] considers the three major activities in software development process as construction, validation, and implementation. A PAISLey specification is written as a set of function definitions. The activities like sequencing, or control flow is difficult to specify [Zave 91b]. PAISLey provides a unified approach to specify the requirements.

TRL stresses that requirements elicitation is not a design of the system, it is rather a statement of the need. (Here we recall some of the salient features of our approach discussed in earlier chapters). TRL consists of a defined approach to requirements elicitation. TRL provides a description of the real-world model. The system structure provided by the recognition of agents assists in evaluating the explanations obtained by the customers. Explanations obtained by the customers are in the form of scenarios. Scenarios comprise of events. The following descriptive elements provide an objective manner to determine how events interact:

(1) **What** event occurred?

(2) **Who** performed an event?

(3) **With** what aim was an event performed?

(4) **What** caused an event to occur?

(5) **Under what circumstances** did an event occur?

(6) **By what means** was an event performed?

(7) **When (at what time)** was an event performed?

Interpreting the explanations of an event with the above criteria provides an excellent understanding of the events occurring in the system. The above list of criteria is further explained below.

What event occurred; is answered by the event name. An event name symbolises the unit of work performed.

Who performed an event; is answered by the name of an agent. The list of events performed by an agent provides the signature of an agent.

With what aim was an event performed; is answered by the reason in performing this event. It provides an explanation on the importance of this event.

What caused an event to occur; provides a reasonable explanation on the reactivity of the system.

Under what circumstances did an event occur; provides explanation on the static constraints. For example, an event 'e' can occur only if the water temperature exceeds twenty five degrees.

By what means was an event performed; provides explanation on the mechanics of the system. This provides details of 'how the event was performed'.

When was an event performed; provides explanation on the temporal constraint of the event. Temporal requirement must be described at the application level as a requirement. A timing constraint described at the application level (as a requirement) makes one to think of the implications of satisfying or not satisfying a timing constraint, and this is the focal issue of referring to a timing constraint.

Temporal Requirements: Rationale and Description

Temporal requirements are the primary determinants of the functional correctness with real-time systems. In the earlier chapters we categorised and explained the influences of temporal requirements. The impact of timing constraints is felt in various stages of the system development cycle. Well defined timing constraints do not influence a solution method, and provide more information about the desirable solution. Leveson [Leveson 90] points out that, simple primitives for timing, such as a time-out, do not adequately capture the complexities of time and therefore are inadequate for fully specifying and modelling timing requirements.

In SREM the timing constraints are described over R-nets. A timing constraint is described with the help of validation points. Validation points are drawn as circles and inserted over R-nets. Validation points are labelled. A validation path is a series of validation points. A timing constraint is described over a validation path. For example in Figure 7.2 consider a timing constraint such that, the current speed is calculated within 2 seconds of accumulating the wheel rotations. To define a timing constraint over this path, a validation point is inserted after the alpha 'accumulate wheel rotations' and after the alpha 'calculate current speed'. Then our 2 second requirement is a descriptor of the path from one validation point to another. Description of timing constraint in R-net can span over several R-nets, and thus becomes difficult to visualise the requirement.

Timing constraints in RTRL are expressed as a timer alarm. When the alarm expires, it causes a transition. This is similar to SDL way of defining timing constraints. The time-out (timer alarm) feature can describe the timing constraint between two events, and cannot describe timing constraints over several events. The description of timing constraint over several events in RTRL is similar to the description in SREM. For example, refer to the diagram described below (Figure 7.18). If the timing constraint over the path ABCD is 2 seconds, then it is described as

```
LATENCY    ABCD       2
```

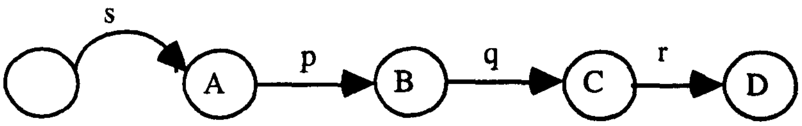


Figure 7.18 Timing Constraint on several events in RTRL

The temporal requirements in PAISLey is again a time-out feature. There is no mechanism to state timing constraints over several events [Zave 82]. When the specification is executed, the simulator prints the timing of each event. The timing constraints, like time-out can be expressed. It is not possible to describe other complex timing constraints. As explained in Chapter 2, a timing constraint in PAISLey is described as a comment.

TRL framework provides a means to better plan and integrate the real-time considerations. As demonstrated in Chapter 5 TRL handles all types of complicated timing constraints, and are stated very easily. For example, the two second timing requirement we discussed in this section, is a timing relation between the two events accumulate wheel rotations, and calculate current speed. TRL provides a concise notation for defining all complicated timing constraints. The description of temporal requirements is handled at a high level. The rigorous and extensive ability to handle all types of timing constraints is of particular concern to the requirements engineers. Real-time systems are often safety critical systems. Safety concerns the implication of timing constraints. The language as a tool of thought provides means to think in these aspects.

Criterion	RSL	RTRL	PAISLey	TRL
Primary application emphasis	Real-time systems	Telecommunication systems. Can be used with all real-time systems.	Embedded systems	Real-time systems
Perspective	Specifier	User level	Designer level	User level
Abstraction	Detailed description. Description oriented towards specification.	Stimulus-response sequence	Design orientation. Complete code must be written.	A the right granularity for users and engineers.
Localisation	Hierarchies of ALPHAs	No localisation	Process described at implementation level	Agents characterised by scenarios
Uniformity	Makes use of variety of descriptions, like R-net, and RSL.	Descriptions are at least two level: the FSM diagram, and the code.	Unified approach	See PAISLey
Temporal reasoning	Time values can be represented over ALPHA. Timing description can span several R-nets	Informal extension of timer facility to FSM.	Restricted temporal requirements	Can describe all types of timing constraints
Representation of timing exception	Not addressed	Not addressed	Not addressed	Addressed

Table 7.1 Comparison highlighting the differences in the approaches

7.4.1.2 Human Communication Tool

Various people are involved during system development. This means that the requirements document has to be communicated explicitly among a number of people. As pointed out in earlier chapters (see Chapter 1 and Chapter 2) the primary purpose of requirements document is to promote effective communication among developers and stakeholders. Given the various roles different people play, it is essential to communicate in order to ensure a common understanding of the desired system. Communication is needed among users, requirements engineers, specifiers, and developers in order to obtain a system that will reflect users' requirements.

Constructing a large software system is not merely a matter of technical capability, but also a matter of communication. Problems arise because many people are involved in this phase from marketing, technical, financial, and the user group. The simplicity of the language aided with the absence of arcane mathematical symbols assists the persons in communication. People communicate but not very effectively. As described in Chapter 2, human communication improves with two vital characteristics: understandability, and modifiability.

Understandability

Understandability is an important but a difficult criterion to measure. Understandability is a subjective connotation. However understandability is often related with complexity. This means that anything that is highly complex is difficult to understand. If the complexity is simplified, it is made more accessible to a wider community, and more easily understood. The term 'complexity' is used in an informal way. As we are referring to the representation technique, simplicity (the absence of complexity) means that the technique must be both easy and fast to use. For the users, and acceptance testers the representation formalism must be easy to recall. Understandability is a means to achieve a clear idea of the concepts.

As described in Chapter 2, understandability is related to a number of characteristics, like structuredness, conciseness, self-descriptiveness, and readability. The stakeholders, and acceptance testers read the document for the purpose of evaluation, and these factors are important for them.

Self-Descriptiveness: This quality is directly related the syntactic aspects of the language. Requirements document must use the terminology of the environment, the way the users interact with the system. The requirements document is *descriptive*, rather than *prescriptive*. The requirements language must provide suitable constructs for defining various constraints that the system must satisfy, and these constructs must reflect the nature of the environment. This results in a close correspondence between reality and notation.

As expressed by Bell [Bell 77] the constructs in RSL may be unclear. The same is true for RTRL, as it is a description of FSM. PAISLey is very cryptic. As commented by many persons the descriptions in PAISLey is difficult to read and understand (for example, Davis 90, Stokes 91). The constructs in TRL follow the working rules of the user, and are meaningful and realistic in the context of real-time system and its environment.

Conciseness: The representation of the model influences the way in which different people perceive the system. A well chosen representation technique induces a clear conceptual understanding of the system for all concerned persons. When modelling complex systems, it is necessary to avoid detailed design descriptions, as it may be obtrusive to understanding the objectives of the system.

Descriptions in RSL are detailed, and provides specification of the system [Scheffer 85]. Descriptions in RTRL provide a description of monolithic finite state machine. PAISLey statements provide explicit design details. The framework in TRL acknowledges that the entities in real-time systems are reactive, and descriptions in TRL contain details relevant to that level.

Structuredness: Human learning, and problem solving are greatly facilitated by meaningful structure. Meaningful structure is beneficial for representing environmental and computer concepts. A structure is meaningful if the users can relate the concepts with the components they work in every day life. A system represented as an organised set of components is in harmony with the mechanism the users can relate to, and (such a structure) is meaningful to assimilate advanced features.

The structuring mechanism in SREM is by means of R-net and subnets. Each net or subnet describe a function or sub function. RTRL lacks any modularity. PAISLey provides modularity by defining the processes. TRL provides a structure of agents, a structure that the users can identify with the environment.

Readability: As pointed out in Chapter 2, the ability to read a fragment of requirements is more important than the ability to write the same piece of fragment. Requirements document is likely to be read by many persons, and must be comprehensible. It is difficult to expect people to agree to a document if they cannot read the same.

Both in SREM, and RTRL the requirements are described with the help of the diagrams: R-net in case of SREM, and FSM in the case of RTRL. PAISLey code is difficult to read. The descriptions in TRL mimic the causal nature of reactive systems, and is easy to read in the absence of any cryptic declarations.

Modifiability

It is simplistic to assume that all the requirements of a system can be captured at once, and then be described in the requirements document. In fact requirements are not captured, requirements are negotiated and agreed upon after a number of meetings and discussions. Requirements document like the development of a system follows an evolutionary process. Requirements evolve, changes must be made to the requirements document. As noted earlier (in Chapter 2) modifiability is associated with features such as structuredness, self-descriptiveness, extendability, and writeability. Modifiability provides means whereby changes in requirements are controlled.

Extendability: As pointed out earlier, changes do occur in requirements, and needs to be accommodated in the document. Extendability ensures that the resulting changes are managed. With the associated features such as structuredness, and self-descriptiveness extendability simplifies the evolution.

For example, consider a change to the requirements of cruise-control system described above. In the above description, when the driver presses the accelerator pedal (while the system is on) the system continues to attempt to maintain the previous speed. This requirement may later be recognised as a hindrance to the intentions of the driver, and a change may occur. The changed requirement, suggests that whenever the driver presses the accelerator pedal (while the system is on) the operation of the system must be suspended, and the system be brought to service, with the request (resume) from the driver.

In SREM the changes occur in R-net, the R-net has to be changed to introduce the check the status of the accelerator alpha, and then decision be taken depending on the accelerator status. RSL requires detailed information, and requires suitable modification in RSL. The data description in RSL requires modification.

In RTRL the change is difficult to implement, as it lacks the notion of modularity. The FSM has to be re-written creating new states, and this distorts the FSM. Changes have to be incorporated in to the textual description of RTRL also. The error may creep in while changing the requirement, as the unchanged states may not reflect the belief held earlier.

The changes to the requirements occur on the basis of discussion with the stakeholders. With descriptions in PAISLey the possibility of such a discussion may not be possible, as the descriptions are difficult to understand. Once the changes are identified, the detailed instructions in PAISLey needs to be modified.

TRL approach identifies the agents, and its documentation is readable and localised. The description of agents has a high level of visibility, and is self-contained. An agent cooperates with other agents to achieve the desired goal. Modifications to the function of agent, makes change only in one part of the document and needs no change in the other parts of the document. Even if totally new functions are to be added, then a new agent may be created to achieve the desired goal. The structure of TRL advocates adaptability.

For example, the change that we mentioned in the requirements of cruise control system can be incorporated in the description of 'controller' agent. The descriptions of 'controller' agent needs only another statement to be added to it, stating that if the 'accelerator pedal' is pressed then the system operation is suspended.

An important problem in large projects is the changes occurring in requirements. With our technique these changes can be incorporated easily without changing the whole organisation of the requirements document. The advantage comes from the localisation of the information. While in RSL and RTRL the information is processed in sequence, and any change is reflected in disturbing the whole organisation of the requirements document. With our technique the changes have to be determined with only the affected agents. While in other approaches the changes may have to be carefully determined as it can affect the whole definition of the system.

Writeability: The information that is captured and documented is likely to change with the increased knowledge about the environment and the intended system. The requirements document must be easily updatable. The writeability dimension denotes how easy it is to create and update the document. Writeability improves with computer understandable languages. All the approaches considered here provide writeability. Once the requirements or change in the requirements are identified, the information can easily be documented.

The above discussion is summarised in table 7.2.

Criterion	RSL	RTRL	PAISLey	TRL
Selfdescriptive-ness	Descriptions in RSL may be unclear	Descriptions follow the FSM structure	Descriptions are not intuitive and difficult to understand	Descriptions by means of user defined scenarios
Conciseness	Detailed design description	Monolithic finite state machine	Complete code must be written	Concise scenario description.
Structuredness	R-net and ALPHAs	Not addressed	Processes described at implementation level	Structure of agents categories scenarios into groups
Readability	Like FSM diagrams	See RSL	Concepts are not intuitive, and constructs are difficult to read	Scenario description with readable constructs
Extendability	A small change may disturb the whole net. Hierarchies of ALPHA provide some control.	Monolithic nature of the document is difficult to extend. A small change can disturb the whole net.	The hierarchies of process description cope with changes	Scenarios that fall into groups make the evolution easier
Writeability	Document can easily be updated	See RSL	See RSL	See RSL

Table 7.2 Comparison highlighting the differences in the approaches

7.4.1.3 Vehicle towards Automation

System development is regarded as a series of model building activities. A good conceptual modelling approach accommodates the evolution of the model. As pointed out earlier in Chapter 2, the computer understandable language helps in propagating the changes, that occur in requirements. The issue of representing the requirements document is complicated by the frequent changes that occur in requirements. Automated tools can be of help in propagating the changes, because of their ability to handle volume of data. Automated tools help in supporting the evolution of the model in few steps, like supporting the creation of test data. In general automated tool can support in the following functions:

- capturing new requirements;
- updating the requirements;
- inquiring the requirements (to check whether a requirement has been considered); and
- evaluating the requirements.

Such a language helps in document creation, document polishing, and document production. Such a document can also be used to create test data automatically. The modelling approaches like SREM, RTRL, PAISLey, and TRL are a suitable approach to use it as a vehicle towards automation. In these approaches requirements can be checked against a validation criteria, for example timing violations, or syntax errors.

7.5 An Overview of TRL

Requirements modelling roughly deals with the following activities:

- grasp of the problem;
- familiarisation with the problem;
- presentation of the problem, and
- validation of the problem.

Requirements modelling as seen by the above activities, is a user centred activity. During the system development a gulf arises between the users and system developers. The vocabulary of the two is entirely different. In the design of TRL, requirements modelling is projected as a bridge between the two different worlds, the world of customer, and the world of developers.

The conceptual modelling process discussed earlier (see Chapter 3), provides an engineering approach to problem understanding. In this approach, when we use an agent, we use a representation. The power of an agent comes from the concept of a representational device. In fact I define an agent as an artificial device that serves a representational function. Agents are mediators between the customers, and the developers. A set of scenarios provides a concrete representation of the use to which the agent will be put. Our approach enumerates critical and typical scenarios that users want to do and need to do, to achieve the desired goal. The scenario description helps to reveal any mismatch. Scenarios are descriptive stories about the intended use of the system. Scenarios is a valuable tool for validating the requirements. The scenario concept is a powerful one. It allows the user to know in advance whether the 'requirements engineer' has understood their needs. The importance of such a step cannot be over emphasised. Bubenko [Bubenko 86] terms the trend in system development as 'You Don't See What You Get'. In traditional system development phase, a user does often not have a reasonable understanding and feeling of what kind of a system he/she will get until the system is operational. Such a situation is mitigated here. Scenarios are used as a mechanism for mental prototyping. The various facets of this modelling approach is reported in [Sateesh 95a, Sateesh 95b, Sateesh 95c, and Sateesh 95d].

This concept is a well suited tool for constructing descriptions, and abstractions about the system. It is particularly well suited for the iterative nature of the conceptual modelling process. The description language TRL is used with a variety of problems. The use of TRL in MMI (Man-Machine Interaction) is reported in [Sateesh 94a], and the practicality of the language is also reported in [Sateesh 94b, Sateesh 94c, and Sateesh 94d].

7.6 Summary

The point is not that one representation is superior to another, but that the different approaches have different properties, and priorities. We are of the opinion that the users must be able to comment on the proposed description of what the system does. This aspect of representation is of prime importance because requirements description is of constructing, analysing and documenting the description. After all, the primary purpose of the document is a description of the system that fulfils the client's needs. The document must be easily maintainable to take care of the changes occurring in the requirements.

The TRL modelling technique is an effective approach. The technique allows the modeller to focus his/her attention on the needs and objectives of the system. The concept of agent provides better understanding of the requirements without any influence on the implementable aspect of the system. The means of description serve for describing the agents as well as for analysing, and documenting the specific use of them.

Requirements description is intellectually tough. A requirements language can at most, alleviate the difficulty of the task. The important aspects of a system can only be discussed effectively by defining the use of a system. The use of a system is captured through scenarios. Users can become frustrated, and confused if they are not able to visualise the proposed use of the system, which they expect. This is necessary to comment usefully and in detail, on specific features. This analysis provides a sound understanding of the work to be carried out by the system. These are the basic underlying viewpoints in our approach.

Chapter 8

Summary and Conclusions

A system has a well defined use to its community. The requirements must reflect those needs. A requirements model of a system must rely on the user level activities, and aid the human understanding and communication. In this thesis we proposed a novel approach suitable for the description of requirements of real-time systems.

Engineering in general is directed to build things according to the requirements - with needed functional capability. Requirements of real-time systems are becoming increasingly complex. This increase in complexity is partly due to the increasing capability of microprocessors. Real-time systems are penetrating a wide range of applications like, industrial applications, military applications, and health care systems. Requirements of these systems are difficult to understand. This thesis has described the results of research into the problems of modelling the requirements of real-time systems.

8.1 Thesis Summary

Chapter 1 argued that the future systems are likely to be characterised by the desirable property of real-time systems. Real-time systems are time critical and reactive. These systems interact with physical devices, and perform complex functions. The requirements description of these systems must encompass these characteristics. Requirements is different from specification. Requirements description produces the conceptual model of a system, while specification produces the empirical model of a system. A requirements description must include the facilities to describe the dynamic nature of real-time systems. We contend that the requirements engineers must be provided with an approach that supports the description, analysis, and validation of the requirements.

Chapter 2 reviews the requirements modelling area, that is of primary interest to this thesis. It gave an insight into the position of requirements modelling languages, and into the common ancestry of these languages. We examined the research efforts in the area, and noted that there is an issue to be addressed. With the traditional approach of modelling, it is difficult to visualise the application's requirements. A conceptual model must describe both the static and dynamic aspects of the system. A conceptual model must reflect the collective (stakeholders, and requirements engineer) perception of the system. It is important that the requirements description reflects the stakeholders perception of the problem to validate the understanding of the problem.

In chapter 3 a real-time system is viewed within a broad operational environment, with user as an integral part of the system. We present an engineering approach to derive the conceptual model of real-time system. Each phase is associated with an objective. This analysis suggests what information to look for, and what analysis to be performed during each phase. The aim of this approach is the clear formulation of the system needs. The splitting of the conceptual modelling process into steps ensures that the essential links between problem definition, and objectives are maintained. The goals provide important criteria for the exploration of requirements. The important aspects of a system can be discussed effectively by defining the use of a system. Such an approach provides formal expression to the goals of the system. A systematic approach avoids time consuming errors due to the lack of information or bias. The purpose of an agent is to accomplish something. Scenarios include a simple elaboration of the way an agent accomplishes a goal. Scenarios describe the use of agents, and reflect a user's belief about the system.

Real-time systems are event-driven, and requires explicit description of temporal properties. Chapter 4 discussed a formal event model of a system. In this model we make use of dense time. The model describes the functional and temporal restrictions, using the same framework. This is done by enriching the elements in the domain, with an explicit time component.

The events are regarded as atomic. The events that overlap in time (continuous events) are represented by treating their initiation and termination as distinct atomic events. We associate an event symbol to each event. In our model the following phenomena can be noted:

- events trigger the operations,
- operations modify the state of the agents, and
- agents are characterised by scenarios.

Real-world systems often consist of several agents. The behaviour of an agent is then a possible event sequence, over the alphabet of event symbols. A system is regarded as a combination of concurrently acting agents.

Chapter 5 discusses the description language TRL. A problem associated with requirements elicitation is the communication barrier between the two parties, the stakeholders and system engineers. Every one is aware how difficult it is to understand the description of a well known system, with complex descriptions. It is even more difficult to understand a non-existent system, with a complex description. There is a fair amount of evidence that the specification languages create more harm than good during requirements [Fraser 94]. The user level requirements are not readily recognisable when confined into such a structure. To alleviate such problems, we present TRL. Real-time systems are time sensitive. Qualitative approaches such as Allen's interval relations [Allen 83] face difficulties in representing and reasoning about metric constraints (restricting the distance between temporal events). Our model allows the representation of all types of timing constraints that may arise in a system. We also present a general classification of timing constraints, noting the limitation of classification provided by Dasarathy [Dasarathy 85]. Our formalism conveniently handles all forms of temporal constraints. The syntax of the language also provides features to describe the time-related exception scenario.

Chapter 6 demonstrates the practical use of our approach. TRL has been used for a number of problems discussed in the literature. In this chapter two standard case studies are reported.

Chapter 7 discusses evaluation of our approach with the representative techniques discussed in Chapter 2. Evaluation is driven with the help of another standard case study.

Chapter 8 is the conclusion and summarises the thesis. It also explores avenues for further research.

8.2 Contributions

A new approach to describe the requirements of real-time systems was presented. Among the salient features of the TRL model, is a fundamental notion of time, and compositionality. The payoff for this dual treatment is manifold. Requirements become more structured since they can constrain the system events. This model allows the representation of external environment and the controller along with the available resources, in a unique framework making it possible to study the properties of the system. The description language - TRL is small and simple. As Hoare [Hoare 81] observes, if the basic tool, the language is itself complicated, then it becomes part of the problem, rather than part of its solution. As Wirth [Wirth 95] expresses 'increasingly, people seem to misinterpret complexity as sophistication, which is baffling - the incomprehensible should cause suspicion rather than admiration'. The simplicity reduces the number of errors made by the requirements engineer. As noted by Parnas [Parnas 94] a common error found in formal specifications is that the specifiers write down something that does not correspond to their intent. It is simple to describe the structure of a system and its properties in TRL, as it allows a requirements engineer to express the real world scenarios easily. TRL formalism helps to organise and understand a complex system, as it supports abstractions, and hierarchical decompositions. The constructs and abstractions provided by TRL are useful for modelling real-time systems, and controlling the complexity of large systems.

A real-time system is viewed within a broad operational environment. An engineering approach is presented to derive the conceptual model of real-time systems. This layered approach ensures that the essential links between problem definition, and objectives are maintained. The approach enables an efficient interaction with the users. It encourages the user to reason on the requirements. The important aspects of real-time systems are the static and dynamic constraints. TRL handles both of these aspects well. TRL also provides a natural facility to describe the time related exception scenarios. This increases confidence in the users, and requirements engineer, as such decisions are not left alone to the imagination of designers.

In process controlled systems the failures of computing system is often due to unexpected scenarios that arise between the environment and controller [Leveson 86]. The main difficulty in studying the interactions between controller and environment, is the lack of an approach, that provides a graceful transition from real world (non-computing) to concrete world (computing). The TRL formalism bridges this gap. Requirements engineer works between two worlds, the world of user, and the world of system developers. Gougen [Gougen 92] terms the two worlds as 'the dry' and 'the wet' aspects. Gougen argues that reconciling the two worlds has a strong practical need, and this reconciliation may be the essence of requirements engineering. TRL as summarised in Chapter 7 essentially achieves this.

In TRL the information is presented in a way the user handles it. The benefits of the approach discussed allows the requirements engineer to speak in the users' language, and to view the interaction from the users' perspective. The advantages with TRL are twofold. First, concentrating on the user level terminology focuses the attention away from design issues. Secondly, the identification of agents allows the abstraction of key features without being lost in a mass of detail. The detail is available, but it is localised. This makes the requirements description hierarchically structured in terms of the levels of abstraction of the goals of the agents, and this undoubtedly is an aid to understanding.

The TRL formalism captures naturally many aspects of the real world, while encapsulating the notion of discrete event system. During requirements analysis, validation is a process to gain confidence in the model. Validation is performed with stakeholders. There are many aspects of software requirements that can be most effectively validated by user inspection of the scenarios. Scenarios include something the agent wants to accomplish, which is associated with action. This essentially poses the question, what must be done in order to accomplish the mission? A scenario is an encapsulated description of achieving a specific outcome, under specified circumstances. Scenarios have two main uses: First scenarios can be used to understand the needs. Second, scenarios can be used for validating those needs. The scenarios can increase the confidence in the requirements engineers, and users. This can be visualised as a process of mechanically reducing the requirements into a unique simple form.

In a goal directed reasoning the engineer analyses the ways of achieving a desired goal. This reasoning provides an opportunity to identify unforeseen consequences. While in a data directed reasoning the engineer is attempting to interpret the data (associated with the situation) to identify the course of action. Jackson [Jackson 94] observes that concentration on solution is widespread, and all methods place their emphasis on describing a solution. According to Shemer [Shemer 87] such solution specification is the cause of many failures of software systems development.

The approach advocated addresses the key issues. They are:

1. The approach focuses on the features conceived by the users, and requirements are derived from these features. This need based concept reduces the tendency of requirements engineers to over specify the system, and thus step into the system design. The over specification cuts into the design freedom of system developers, making the system overly restrictive.

2. The description focuses on the important objectives, while emphasising the constraints to be met by the system.

3. As the description of the requirements is in the terminology of the users, it allows the users to provide valuable comments on the model.

Thus the approach is effective for requirements description. The benefits of this approach are that it enables the stakeholders and requirements engineers to develop a shared understanding of the needs.

8.3 Directions for Future Research

It is common in every research that, as some progress is made, a substantial amount of further work is generated. The work described here is no exception. In this section we highlight the areas in which the research presented here can be further extended.

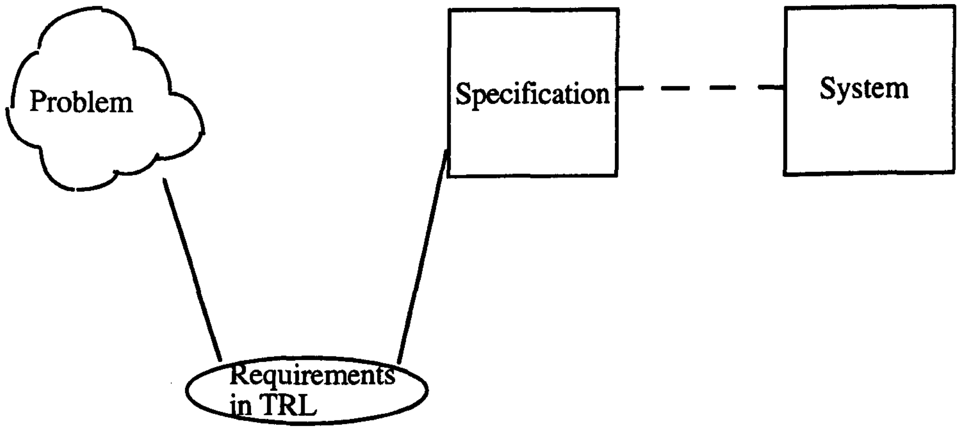


Figure 8.1 Position of TRL in system development

As described in Figure 8.1 TRL provides a bridge between the scruffy world (real world), and the neat world[34] (of specification). Please note in Figure 8.1, we are not advocating any life cycle, but are highlighting the main activities during the development of a system. Reviewing the Figure 8.1, we can identify two areas for

[34] We are using the terminology as used in 3rd European Software Engineering Conference, 1991 (for example see [Greenspan 91]).

further research, as shown in Figure 8.2. The objectives of the areas are (1) to make sure, and (2) to know more. These are explained below.

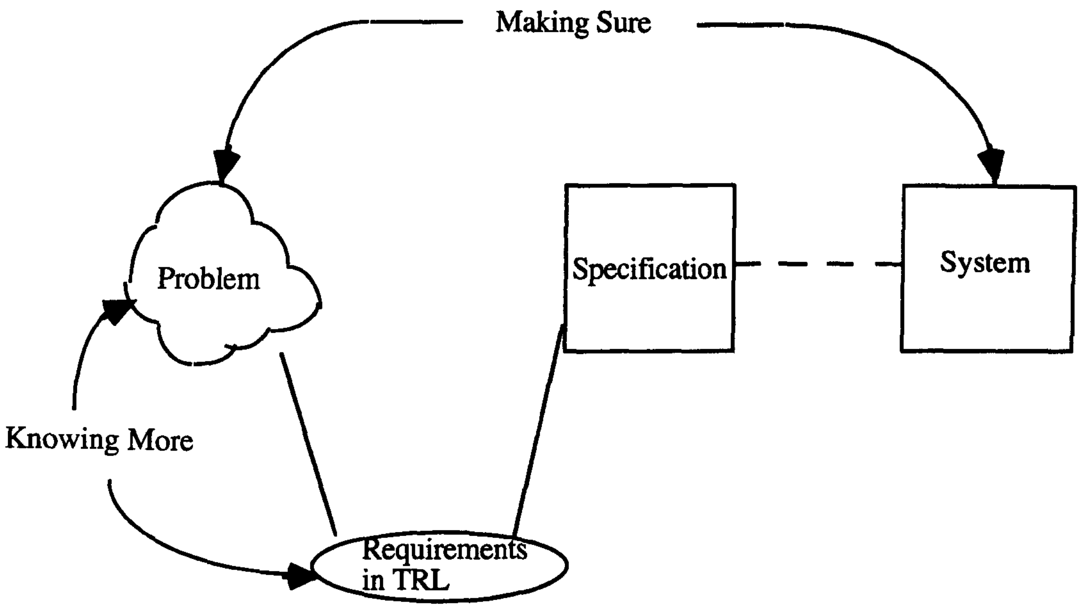


Figure 8.2 Identifying the areas for further work

8.3.1 Making Sure

Any software system, undergoes mainly three types of testing, the module testing, integration testing, and acceptance testing. In this section we are interested in acceptance testing. Acceptance testing (AT) is normally carried out with the stakeholders. This testing is carried out on the real hardware, normally in the same environment where the system is likely to be installed. Some times, an external agency may also be involved in validating the system. This external agency comprises of specialists appointed by the stakeholders to validate the system.

This process (AT) consists of three basic activities: (1) generation of detailed test plans, (2) the documentation of test results to check the progress, and

(3) agreement on the resolution of test results, and procedure on retest of the defect tests. In this process the generation of test data is dependent upon the requirements description of the application. In this sense, acceptance testing, is the testing of the system against the needs of the stakeholder. The purpose of generating the test data is to establish a means to formally demonstrate that the system to be delivered performs according to the requirements. As shown in Figure 8.3, it is possible to generate the test data automatically [Weyuker 94].

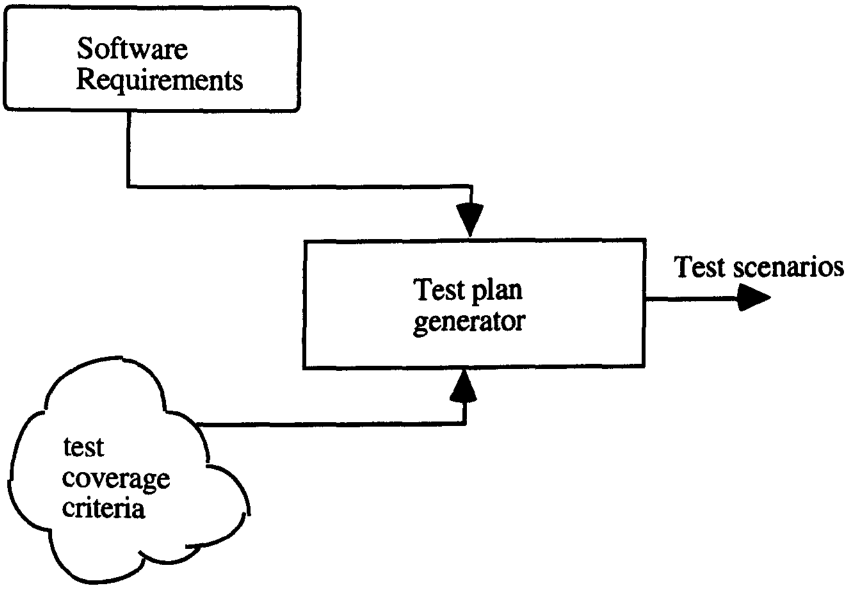


Figure 8.3 Automatic generation of test data

Another interesting area is the design of a query language (knowing more) which is discussed below.

8.3.2 Knowing More

The goal of requirements engineering is to increase the understanding of the requirements. This is partly achieved by the design of TRL which is both understandable, and executable. However the understanding of the requirements both by the stakeholders, and system engineers increase in the presence of a querying language [Potts 94]. The purpose of querying language (now on referred as RQL - Requirements Querying Language) is to construct queries against the requirements expressed in TRL. RQL makes it possible to gather information on properties of the system. A query singles out a behaviour based on the properties supplied by the query. RQL increases the capabilities of requirements inspection[35]. Requirements inspection, is a process to provide information, whether a particular property, or a scenario has been considered in the requirements description. A real-time system exhibits a great variety of behaviour, which may become difficult to analyse the properties of a system manually. The presence of RQL can mitigate this problem.

8.4 Conclusion

The study relating to the modelling of real-time systems presented here has struck an important chord in learning more about the requirements modelling of real-time systems. The research discussed here can be extended with a number of tools to span the various fields of requirements engineering. This research holds great potential for further work.

[35] We are borrowing the terminology of [Fagan 86]. The details of Fagan's work is outside the scope of this thesis, and can also be found in [Sommerville 92], [Pressman 87].

Appendix A

Published Works

The following publications have originated from the work reported in this thesis:

[Sateesh 95a] T.K. Sateesh, "Real World Model for Real-Time Systems", in the *Requirements Engineering and Knowledge Engineering track of KAW '95, (Ninth Knowledge Acquisition Workshop)* Banff, Canada, March 1995.

[Sateesh 95b] T.K. Sateesh, "Conceptual Model for Real-Time Systems: A Perspective", in *proceedings of the 10th Annual ACM Symposium on Applied Computing (SAC '95)*, Nashville, Tennessee, February 1995

[Sateesh 95c] T.K. Sateesh, "Making the Requirements of Process Controlled Systems Explicit", in *proceedings of the 28th Annual Hawaii International Conference on System Sciences (HICSS-28)* Maui, Hawaii, January, 1995

[Sateesh 95d] T.K. Sateesh, "Representing the Conceptual Model of a Time Critical System", in *proceedings of Groningen Information Technology Conference (GRONICS '95)*, Netherlands, February 1995

[Sateesh 94a] T.K. Sateesh, "Modelling the Temporal Requirements of Man-Machine Interaction", in *proceedings of the 1994 Workshop on Information Technology and Systems (WITS '94)* Vancouver, Canada, December, 1994, pp. 252 - 261

[Sateesh 94b] T.K. Sateesh and P.A.V. Hall, "Eliciting the Requirements for Process Controlled Systems", in *proceedings of the 1994 International Computer Symposium (ICS '94)* Hsinchu, Taiwan, 1994

[Sateesh 94c] T.K. Sateesh and P.A.V. Hall, "Modelling the Requirements for Process Controlled Systems", in *Software Quality and Productivity: Theory, Practice, Education and Training*, Edited by Matthew Lee, Ben-Zion Barta and Peter Juliff, Chapman and Hall, pp. 88-91

[Sateesh 94d] T.K. Sateesh, "Expressing Temporal Requirements of Man-Machine Interaction", in *Integrating Human Factors with Software Engineering*, Ed. by William E. Hefley, Human Computer Interaction Institute and Software Engineering Institute, Carnegie Mellon University, Pittsburgh, pp. 123 - 140

Bibliography

[Abbott 90] R.J. Abbott, "Resourceful Systems for Fault Tolerance, Reliability, and Safety", *ACM Computing Surveys*, vol. 22, No. 1, March 1990, pp. 35-68

[Abbott 81] R.J. Abbott and D.K. Moorhead, "Software Requirements and Specifications: A Survey of Needs and Languages", *Journal of Systems and Software*, 2(4) Dec. 1981, pp. 297-316

[Abbott 83] R.J. Abbott, "Program Design by Informal English Descriptions", *Communications of the ACM,* vol.26 (1), Nov. 1983, pp. 882-894

[Abbott 88] R. Abbott and H. Garcia-Molina, "Scheduling Real-time Transactions", *Sigmod Record*, vol. 17, No. 1, March 1988, pp. 71-81

[Adhame 89] E. Adhame, R.E.M. Champion, and R. Pyburn, "A Knowledge Based Approach to Requirements Engineering", in *Software Engineering Environments: Research and Practice*, (Ed) K. Bennett, Ellis Horwood Ltd., Chichester, 1989, pp. 187-202

[Agerwala 79] T. Agerwala, "Putting Petri Nets to Work", *IEEE Computer*, December 1979, pp. 85-94

[Aho 86] A.V. Aho, R. Sethi, and J.D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley 1986

[Alford 77] M.W. Alford, "A Requirements Engineering Methodology for Real-time Processing Requirements", *IEEE Transactions On Software Engineering*, January 1977, pp. 60-69

[Alford 85] M. Alford, "SREM at the Age of Eight: The Distributed Computing Design System", *IEEE Computer*, April 1985, pp. 36-46

[Allen 83] J.F. Allen, "Maintaining Knowledge About Temporal Intervals", *Communication of the ACM,* vol. 26, No. 11, Nov. 1983, pp. 832-843

[Alpern 85] B. Alpern, and F.B. Schneider, "Defining Liveness", *Information Processing Letters,* 21, 1985, pp. 181-185

[Alpern 89] B. Alpern, and F.B. Schneider, "Verifying Temporal Properties without Temporal Logic", *ACM Transactions on Programming Languages and Systems*, 11 (1), January 1989, pp.147-167

[Alur 90] R. Alur, C. Courcoubetis and D. Dill, "Model Checking for Real-Time Systems", in *Fifth Annual IEEE Symposium on Logic in Computer Science*, 1990, pp. 414-425

[Alur 92] R. Alur and D. Dill, "The Theory of Timed Automata", *Lecture Notes in Computer Science-600,* Springer-Verlag, 1992 pp. 45-73

[Anderson 81] T. Anderson and P.A. Lee, *Fault Tolerance: Principle and Practice*, Prentice-Hall, Englewood Cliffs, USA, 1981

[Anderson 89] J.S. Anderson, and S. Fickas, "A Proposed Perspective Shift: Viewing Specification Design as a Planning Problem", in *Proceedings of the 5th International Workshop on Software Specification, and Design*, Pittsburgh, USA, 1989

[Arnold 80] A. Arnold, and M. Nivat, "Formal Computations of Nondeterministic Recursive Programme Schemes", *Math. Systems Theory*, 13, 1980, pp. 219-236

[Arnold 94] A. Arnold (translated by John Plaice), *Finite Transition Systems: Semantics of Communicating Systems,* Prentice Hall, 1994

[Auernheimer 86] S.B. Auernheimer, and R.A. Kemmerer, "RT-ASLAN: A specification Language for Real-Time Systems", *IEEE Transactions on Software Engineering*, 12(9), September 1986, pp. 879-889

[Balzer 79] R.M. Balzer and N.M. Goldman, "Principles of Good Software Specification and their Implication for Specification Languages", *Proceedings of the Specification of Reliable Software Conference*, April 1979 pp. 58-67

[Balzer 82] R.M. Balzer, N.M. Goldman, and D.S. Wile, "Operational Specification as the Basis for Rapid Prototyping", *ACM Sigsoft Software Engineering Notes*, vol. 7 (5), 1982, pp. 3-16

[Balzer 83] R. Balzer, T.E. Cheatam and C. Green, "Software Technology in the 1990's: Using New Paradigm", *IEEE Computer*, November 1983, pp. 34-45

[Basili 88] V.R. Basili and H.D. Rombach, "The TAME Project: Towards Improvement-Oriented Software Environments", *IEEE Transactions on Software Engineering*, SE-14, 6, June 1988, pp. 758-773

[Bell 77] T.E. Bell, D.C. Bixler, and M.E. Dyer, "An Extendable Approach to Computer-Aided Software Requirrements Engineering", *IEEE Transactions on Software Engineering*, Jan 1977, pp.49-60

[Bennett 91] P.A. Bennett, *Safety*, in *Software Engineering Reference Book*, Ed. By J.A. McDermid, Butterworth-Heinemann, 1991

[Benthem 91] J. van Benthem, *The Logic Of Time*, Kluwer Academic Publications, Second Edition, 1991

[Berzins 85] V. Berzins, and M. Gray, "Analysis and Design in MSG.84: Formalizing Functional Specifications", *IEEE Transactions* *on Software Engineering*, vol.SE-11, No. 8, August 1985, pp. 657-670

[Bestavros 91] A. Bestavros, "Specification and Verification of Real-time Embedded Systems using Time-constrained Reactive Automata", *IEEE Real-Time Systems Symposium,* 1991, pp. 244-253

[Boehm 76] B. Boehm, "Software Engineering", *IEEE Transactions On Computers,* C-25, 1976, pp. 1226-1241

[Boehm 81] B. Boehm, *Software Engineering Economics*, Englewood Cliffs, NJ: Prentice Hall, 1981

[Bolognesi 88] T. Bolognesi, and H. Brinksma, "Introduction to the ISO Specification Language LOTOS", *Computer Networks and ISDN Systems,* 14(1), 1988, pp. 25-29

[Booch 83] G. Booch, *Software Engineering with ADA*, First Edition, Benjamin/Cummings, Menlo park, California, 1983

[Booch 86] G. Booch, "Object-Oriented Development", *IEEE Transactions on Software Engineering*, Vol 12, No. 2, Feb 1986, pp. 211-221

[Borgida 85] A. Borgida, S. Greenspan, and J. Mylopoulos, "Knowledge Representation as the Basis for Requirements Specifications", *IEEE Computer,* (18), 1985, pp. 82-91

[Borriello 90] G. Borriello and T. Amon, "On the Specification of Timing Behaviour", in *proceedings of TAU'90: The 1990 ACM International workshop on Timing Issues in the Specification and Synthesis of Digital Systems,* Vancouver, Canada, August 1990

[Bowen 93] J. Bowen, and V. Stavridou, "Safety-critical systems, formal methods and standards, *Software Engineering Journal,* July, 1993, pp. 189-209

[Bowen 95] J.P. Bowen, and M.G. Hinchey, "Ten Commandments of Formal Methods", *IEEE Computer*, April 1995, pp. 56-63

[Brodie 84] M.L. Brodie, J. Mylopoulos, and J.W. Schimdt (Ed), *On Conceptual Modelling: Perspectives from artificial Intelligence, Databases, and Programming Languages,* Springer-Verlag, 1984

[Brooks 87] F.P. Brooks, Jr. "No Silver Bullet Essence and Accidents of Software Engineering", *IEEE Computer*, April 1987, pp. 10-19

[Bubenko 80] J.A. Bubenko, "Information Modelling in the Context of System Development", in *Information Processing 80,* Ed. S.H. Lavington, North-Holland, Amsterdam, 1980, pp. 395-411

[Bubenko 86] J.A. Bubenko jr., "Information System Methodologies-A Research View", in *Information System Design Methodologies: Improving the Practice*, Ed. by T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, North-Holland, 1986, pp. 289-318

[Büchi 62] J.R. Büchi, "On a Decision Method in Restricted Second Order Arithmetic", *Logic, Methodology And Philosophy Of Science,* Stanford University Press, 1962, pp. 1-11

[Burns 90] A. Burns and A. Wellings, *Real-Time Systems and Their Programming Languages*, Addison-Wesley, 1990

[Burns 91] A. Burns, "Scheduling Hard Real-Time Systems: A Review", *Software Engineering* Journal, May 1991, pp.116-128

[Carroll 92] J.M. Carroll, and M.B. Rosson, "Getting Around the Task-Artifact Cycle: How to Make Claims and Design by Scenario", *ACM Transactions on Information Systems*, 10(2), April 1992, pp. 181-212

[Casey 82] B.E. Casey and B. Dasarathy, "Modelling and Validating the Man-Machine Interface", *Software-Practice And Experience*, vol. 12, 1982, pp. 557-569

[CCITT 88] CCITT Z.100-Z.104, *Specification and Description Language*, CCITT, Geneva, 1988

[Chandrasekaran 85] M. Chandrasekaran, B. Dasarathy and Z. Kishimoto, "Requirements-Based Testing of Real-Time Systems: Modeling for Testability", *IEEE Computer*, vol. 18, No. 4, Apr. 1985, pp. 71-80

[Chandrasekaran 91] B. Chandrasekaran, R. Bhatnagar, D.D. Sharma, "Real-Time Disturbance Control", *Communication of the ACM*, vol. 34 (8) August 1991 pp. 32-47

[Chandy 89] K.M. Chandy and J. Misra, *Parallel Program Design*, Addison-Wesley Publishing company, May 1989

[Checkland 81] P. Checkland, *Systems Thinking, Systems Practice*, Wiley, 1981

[Chen 76] P.S. Chen, "The Entity Relationship Model: Towards a Unified View of Data", *ACM Transactions on Database Systems* 1 (1) , March 1976 pp. 9-36

[Choueka 74] Y. Choueka, "Theories of Automata on ω-Tapes: A simplified Approach", *Journal of Computer and System Sciences*, 8, 1974 pp. 117-141

[Ciapessoni 93] E. Ciapessoni, E. Corsetti, A. Montanari, and P.S. Pietro, "Embedding time granualarity in a logical specification language for a synchronous real-time systems", in *Science of Computer Programming*, 20, 1993, pp. 141-171

[Clarke 86] E.M. Clarke, E.A. Emerson, and A.P. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications", *ACM TOPLAS*, 8(2) 1986, pp. 244-263

[Coad 90] P. Coad, and E. Yourdon, *Object-Oriented Analysis*, Prentice-Hall, 1990

[Coolahan 83] J.E. Coolahan Jr, and N. Roussopoulos, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets", *IEEE Transactions on Software Engineering*, 9(5), Sept. 1983, pp. 603-616

[Coombes 93] A. Coombes and J. McDermid, "Specifying temporal requirements for distributed real-time systems in Z", *Software Engineering Journal*, September 1993, pp.273-283

[Dardenne 93] A. Dardenne, A. van Lamsweerde, and S. Fickas, "Goal directed requirements acquisition", in *Science of Computer Programming*, 20, 1993, pp. 3-50

[Dasarathy 85] B. Dasarathy, "Timing Constructs of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them", *IEEE Transactions On Software Engineering*, vol. SE-11, No. 1, January 1985, pp. 80-86

[Dasgupta 91] S. Dasgupta, *Design Theory and Computer Science*, Cambridge University Press, 1991

[David 92] R. David and H. Alla , *Petri Nets and Grafcet*, Prentice Hall, New Jersey , 1992

[Davies 89] J. Davies and S. Schneider, "An Introduction to Timed CSP", *Technical Monograph PRG-75*, Oxford University, August 1989

[Davis 77] C.G. Davis and C.R. Vick, "The Software Development System", *IEEE Transactions on Software Engineering*, SE-3 (1), Jan 1977, pp. 69-84

[Davis 79] A.M. Davis and T.G. Rauscher, "Formal Techniques And Automatic Processing to Ensure Correctness In Requirements Specification", *Proceedings of Specification of Reliable Software Conference*, IEEE Computer Society, 1979, pp. 15-35

[Davis 82a] A.M. Davis, "The Design Of a Family Of Application-Oriented Requirements Languages", *IEEE Computer*, May 1982, pp. 21-28

[Davis 82b] G.B. Davis, "Strategies for Information Requirements Determination", *IBM Systems Journal*, vol. 21 (1), 1982, pp. 4-30

[Davis 88] A.M. Davis, "A Comparison of Techniques for the Specification of External Behaviour of Systems", *Communications of the ACM*, 31(9), Sept. 1988, pp.1098-1115

[Davis 90] A.M. Davis, *Software Requirements: Analysis and Specification*, Prentice Hall, 1990

[De Marco 78] T. De Marco, *Structured Analysis and System Specification*, Yourdon Press, New York, 1978

[Deutsch 88] M.S. Deutsch, "Focusing Real-Time Systems Analysis On User Operations," *IEEE Software*, September 1988, pp. 39-50

[Dijkstra 68] E.W. Dijkstra, "Go to statements considered harmful", *Communications of the ACM*, 11 (3), March 1968, pp. 147-148

[Dill 89] D.L. Dill, "Timing Assumptions and Verification of Finite~State concurrent Systems", *Lecture Notes in Computer Science-407*, Springer-Verlag, 1989, pp. 197-212

[Dobson 93] J. Dobson, and R. Strens, "A methodology for requirements management applied to safety requirements", in *Safety-critical Systems: Current issues techniques, and standards*, (Ed) F. Redmill, and T. Anderson, Chapman and Hall, London, 1993, pp.123-136

[Dubois 86] E. Dubois, J. Hagelstein, E. Lahou, F. Ponsaert, A. Rifaut and F. Williams, "The ERAE Model: A Case Study", in *Information Systems Design Methodologies: Improving the*

Practice, Ed.by T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart, North-holland, 1986, pp. 87-105

[Dubois 87] E. Dubois and J. Hagelstein, "Reasoning on Formal Requirements: A Lift Control system", *Proceedings of the International Workshop on Software Specification and Design*, 1987, pp. 161-168

[Easterbrook 93] S.M. Easterbrook, 'Domain Modelling with Hierarchies of Alternative Viewpoints", in *Proceedings of IEEE Symposium on Requirements Engineering*, San Diego, California, 1993

[Eilenberg 74] S. Eilenberg, *Automata, Languages, And Machines Volume-A*, Academic Press, 1974

[Elmendorf 74] W.R. Elmendorf, "Functional Analysis using Cause-Effect Graphs", in *Proceedings of SHARE XLIII*, New York, 1974, pp. 567-576

[Emerson 89] E.A. Emerson, A. Mok, A.P. Sistla and J. Srinivasan, *Quantitative Temporal Reasoning,* in *First Workshop on Computer Aided Verification* , Grenoble, France, 1989

[Faci 91] M. Faci, L. Logrippo, and B. Stepien, "Formal specification of telephone systems in LOTOS: the constraint-oriented style approach", *Computer Networks and ISDN Systems*, vol. 21, 1991, pp. 53-67

[Fagan 86] M.E. Fagan, "Advances in Software Inspections", *IEEE Transactions on Software Engineering*, 12(7), 1986, pp. 744-751

[Faulk 83] S.R. Faulk and D.L. Parnas, "On the Uses of Synchronisation in Hard-Real-time Systems", *Proceedings of the Real-Time Systems Symposium*, Arlington, 1983, pp. 101-109

[Feather 87] M.S. Feather, "Language Support for the Specification and Development of Composite Systems", *ACM Transactions on*

Programming Languages and Systems, vol. 9, Nov. 1987, pp. 198-234

[Feather 89] M.S. Feather, "Constructing Specifications by Combining Parallel Elaborations", *IEEE Transactions on Software Engineering*, 15(2), 1989, pp. 198-208

[Feather 93] M. Feather, "Requirements Engineering: Getting Right from Wrong", in *Europen Software Engineering Conference*, Springer-Verlag, 1993, pp. 485-488

[Fickas 87] S. Fickas, "Automating the Analysis Process", in *Proceedings of the 4th International Workshop on Software Specification and Design*, IEEE, Monterrey, 1987, pp.58-67

[Fickas 92] S. Fickas, and B.R. Helm, "Knowledge Representation and Reasoning in the Design of Composite Systems", *IEEE Transactions on Software Engineering*, 18(6) June 1992, pp.470-482

[Finkelstein 87] A. Finkelstein and C. Potts, "Building Formal Specifications Using Structured Common Sense", *Proceedings of the International Workshop on Software Specification and Design*, 1987, pp. 108-113

[Finkelstein 88] A. Finkelstein, "Reuse of Formatted Requirements Specifications", in *Software Engineering Journal*, September 1988, pp. 186-197

[Finkelstein 92] A. Finkelstein, J. Krammer, B. Nuseibeh, L. Finkelstein, and M. Goedicke, "Viewpoints: A Framework for Multiple Perspectives in System Development", *International Journal of Software Engineering and Knowledge Engineering*, vol. 2 (1), March 1992, pp. 31-57

[Foster 81] C.C. Foster, *Real Time Programming-Neglected Topics*, Addison-Wesley series 'the joy of computing', 1981

[France 92] R.B. France, "Semantically Extended Data Flow Diagrams: A Formal Specification Tool", *IEEE Transactions on Software Engineering*, vol.18 (4) Apr. 1992, pp. 329-346

[Fraser 91] M.D. Fraser, K. Kumar, and V.K. Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap", *IEEE Transactions on Software Engineering*, vol.17 (5), May 1991, pp. 454-466

[Fraser 94] M.D. Fraser, K. Kumar, and V.K. Vaishnavi, "Strategies for Incorporating Formal Specifications in software Development", *Communications of the ACM*, vol. 37 (10) Oct. 1994, pp. 74-86

[Freeman 87] P. Freeman, *Software Perspectives: The System is the Message*, Reading MA: Addison-Wesley, 1987

[Freeman 89] P. Freeman, in the Foreword to the book, *Managing the Software Process*, By W.S. Humphrey, Addison-Wesley Publishing Company, 1989, pp.v-vi

[Fuggetta 93] A. Fuggetta, C. Ghezzi, D. Mandrioli, and A. Morzenti, "Executable Specifications with Data-flow Diagrams", *Software-Practice and Experience*, vol. 23(6), June, 1993, pp.629-653

[Gabrielian 91] A. Gabrielian, and M. Franklin, "Multilevel specification of real-time systems", *Communications of the ACM*, 34(5), 1991, pp. 50-60

[Gane 79] C.P. Gane and T. Sarson, *Structured System Analysis : Tools and Techniques*, Prentice-Hall International, Englewood Cliffs, NJ, 1979

[Gehani 86] *Software Specification techniques*, Edited by N. Gehani, and A.D. McGettrick, Addison-Wesley Publishing Co., 1986

[Ghezzi 91] C. Ghezzi, D. Mandrioli, S. Morasca, and M. Pezzé, "A Unified high-level Petri net Model for Time Critical

Systems", *IEEE Transactions on Software Engineering*, 17(2), February 1991, pp.160-172

[Glass 80] R.L. Glass, "Real-Time: The Last World of Software Debugging and Testing", *Communications of the ACM*, vol. 23(5), May 1980, pp. 264-271

[Glass 83] R.L. Glass, *Real-Time Software*, Prentice-Hall, 1983

[Goldman 80] N. Goldman and D. Wile, "A relational database foundation for process specification", in *Entity relationship approach to systems analysis and design*, North-Holland, 1980, pp. 413-432

[Goldsack 91] S.J. Goldsack and A.C.W. Finkelstein, "Requirements engineering for real-time systems", *Software Engineering journal*, May 1991, pp. 101-115

[Gomaa 84] H. Gomaa, "A Software Design Method for Real-Time Systems", *Communications of the ACM*, 27(9), 1984, pp.938-984

[Gomaa 86] H. Gomaa, "Software Development of Real-Time Systems", *Communications of the ACM*, 29(7), 1986, pp.657-668

[Gordon 79] M.J.C. Gordon, *The Denotational Description of Programming Languages: An Introduction,* Springer-Verlag, New York, 1979

[Gorski 89] J. Gorski, "Formal approach to Development of Critical Computer Applications", *Proceedings of the 22nd annual Hawaii International Conference on System Sciences*, 1989, pp. 243-251

[Gougen 92] J.A. Gougen, "The dry and the wet", *Technical Monograph PRG-100,* Oxford University Computing Laboratory, March 1992

[Gougen 93] J.A. Gougen, and C. Linde, "Techniques for Requirements Elicitation", *Proceedings of IEEE Symposium on Requirements Engineering*, San Diego, California, 1993

[Gray 91] E.M. Gray and R.H. Thayer, "Requirements", in *Aerospace Software Engineering A Collection of Concepts*, Ed. by C. Anderson and M. Dorfman, Washington: AIAA, 1991 pp. 89-121

[Greenspan 86] Sol J. Greenspan, A. Borgida, and J. Mylopoulos, "A Requirements Modeling Language And Its Logic", *Information Systems*, vol. 11, (1), 1986, pp. 9-23

[Greenspan 91] Sol Greenspan, "The Scruffy Side of Requirements Engineering", *Lecture Notes in Computer Science-550,* Springer-Verlag, 1991, pp. 492-494

[Greenspan 94] S. Greenspan, J. Mylopoulos, and A. Borgida, "On Formal Requirements Modelling Languages: RML Revisited", in *Proceedings of the 16th International Conference on Software Engineering*, May 16-21, Sorrento, Italy, 1994, pp. 135-147

[Guinan 86] P.J. Guinan, and R.P. Bostrom, "Development of Computer-Based Information Systems: A Communication Framework", *DataBase*, Spring, 1986, pp. 3-16

[Hall 76] P. Hall, J. Owlett, and S. Todd, *Relations and Entities*, in *Modelling in Data-base Management Systems*, (Ed) G.M. Nijssen, North-Holland, 1976, pp.201-220

[Hall 90] A. Hall, Seven Myths of Formal Methods, *IEEE Software*, 7(5), 1990, pp. 11-20

[Harel 85] D. Harel and A. Pnueli, "On the Development of Reactive Systems", in K.R. Apt (ed), *Logics and Models of Concurrent Systems*, NATO ASI Series F 13, Springer-Verlag, New York, 1985, pp. 477-498

[Harel 87] D. Harel, "Statecharts: A Visual Formalism For Complex Systems", *Science of Computer Programming* , 8 (1987) pp. 23-274

[Harel 92] D. Harel, "Biting the Silver Bullet Toward a Brighter Future for System Development ", *IEEE Computer*, January 1992, pp. 8-20

[Hatley 87] D.J. Hatley, and I.A. Pirbhai, *Strategies for Real-time System Specification*, Dorset House Publishing New-York, 1987

[Hayes 87] Ian Hayes (Ed), *Specification Case Studies*, Prentice-Hall International, 1987

[Heitmeyer 83] C.L. Heitmeyer and J.D. McLean, "Abstract Requirements Specification: A New Approach and Its Application", *IEEE Transactions on Software Engineering,* vol. SE-9, No. 5, September 1983, pp. 580-589

[Heitmeyer 93] C. Heitmeyer, R. Jeffords, and B. Labaw, "A Benchmark for Comparing Different Approaches for Specifying and Verifying Real-Time Systems", in *IEEE Workshop on Real-Time Operating Systems and Software.*, 1993

[Heninger 80] K.L. Heninger, "Specifying Software Requirements for Complex Systems: New Techniques and Their Applications", *IEEE Transactions On Software Engineering*, SE-6 (1), 2-12, Jan 1980

[Hilbrook 90] Capt H. Hilbrook III, "A Scenario-Based Methodology for Conducting Requirements Elicitation", *ACM Sigsoft Software Engineering Notes*, vol. 15(1), Jan 1990, pp. 95-104

[Hirschheim 89] R.A. Hirschheim, and H.K. Klein, "Four Paradigms of Information Systems development", *Communications of the ACM*, 32(10), 1989, pp. 1199-1216

[Hoare 78] C.A.R. Hoare, "Communicating Sequential Processes", *Communications of the ACM*, 21(8), 1978 , pp. 666-677

[Hoare 81] C.A.R. Hoare, "The Emperor's Old Clothes", *Communications of the ACM*, 24(2), February 1981, pp.75-83

[Hoare 85] C.A.R. Hoare, *Communicating Sequential Processes*, Prentice-Hall International, U.K, 1985

[Hoare 90] C.A.R. Hoare, "Let's Make Models", in *Lecture Notes in Computer Science-458*, Ed. J.C.M. Baeten and J.W. Klop, CONCUR-90, Theories of Concurrency: Unification and Extension, Springer-Verlag, 90 pp. 33

[Holliday 87] M.A. Holliday and M.K. Vernon, "A Generalized Petri Net Model for Performance Analysis", *IEEE Transactions on Software Engineering*, 13(12) December 1987, pp. 1297-1310

[Hoogeboom 86] H.J. Hoogeboom and G. Rozenberg, "Infinitary Languages: Basic Theory And Applications To Concurrent Systems", *Lecture Notes in Computer Science-224*, 1986, Springer-Verlag, pp. 266-342

[Hooper 82] J.W. Hooper and P. Hsia, "Scenario-Based Prototyping for Requirements Identification", *ACM Sigsoft Software Engineering Notes*, vol. 7 (5), 1982, pp. 88-92

[Hopcroft 79] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata theory, Languages and Computation*, Addition-Wesley, Reading, Mass, 1979

[Hsia 88] P. Hsia and A.T. Yaung, "Another Approach to system decomposition: Requirements Clustering", *COMPSAC*, 1988, pp. 75-82

[Humphrey 89] W.S. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Company, 1989

[Inan 88] K. Inan and Varaiya, "Finitely Recursive Process Models for Discrete Event Systems", *IEEE Transactions on Automatic Control* vol. 33 (7), July 1988, pp. 626-639

[ISO 87] *Information Processing systems-Concepts and terminology for the conceptual schema and the information base, International Organization for Standardization,* 1987, Ref.No. ISO/TR 9007: 1987(E), 1987

[Jackson 83] M.A. Jackson, *System Development,* Prentice Hall, 1983

[Jackson 94] Michael Jackson, "Problems, methods, and specialisation", *Software Engineering Journal,* November 1994, pp.249-255

[Jacob 83] R.J.K. Jacob, Using Formal specifications in the Design of a Human-computer Interaface, *Communication of the ACM,* vol. 26 (4) April 1983 pp. 259-264

[Jacobson 92] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard, *Object-Oriented Software Engineering: A Use Case Driven Approach,* Addison Wesley, 1992

[Jaffe 91] M.S. Jaffe, N.G. Leveson, M.P.E. Heimdahl and B.E. Melhart, "Software Requirements Analysis for Real-Time Process-Control Systems", *IEEE Transactions On Software Engineering*, vol.17, No. 3, Mar 1991, pp. 241-258

[Jahanian 86] F. Jahanian and A.K. Mok, "Safety Analysis of Timing Properties in Real-Time Systems", *IEEE Transactions on Software Engineering*, SE-12(9), Sept. 1986, pp. 890-904

[Jahanian 87] F. Jahanian and A.K. Mok, "A Graph-Theoretic Approach For Timing Analysis and its Implementation", *IEEE Transactions on Computers*, vol-36, Aug 1987, pp. 961-975

[Jahanian 88] F. Jahanian and D.A. Stuart, "A Method for Verifying Properties of Modechart Specifications", *Proceedings of the*

Real-Time Systems Symposium , Dec. 1988 , Huntsville, pp. 12-21

[Jahanian 94] F. Jahanian and A.K. Mok, "Modechart: A Specification Language for Real-Time Systems", *IEEE Transactions on Software Engineering*, 20(12) December 1994, pp.933

[Jarke 94] M. Jarke and K. Pohl, "Requirements Engineering in 2001: (virtually) managing a changing reality", in *Software Engineering Journal*, November 1994, pp. 257-266

[Jensen 85] E. Jensen D. Locke, and H. Tokuda, "A Time-Driven Scheduling Model for Real-Time Operating Systems", *Proceedings of the 6th Real-Time Systems Symposium*, 1985, San Diego, CA, December 1985, pp. 112-122

[Jones 90] C.B. Jones, *Systematic Software Development using VDM*, Prentice Hall, London, Second Edition, 1990

[Joseph 92] M. Joseph, "Problems, promises and performance: some questions for real-time system specification", *Lecture Notes in Computer Science-600*, Springer-Verlag 1992, pp. 315-324

[Kaposi 93] A. Kaposi, and I. Pyle, "Systems are not only software", *Software Engineering Journal*, January 1993, pp. 31-40

[Kesten 91] Y. Kesten, and A. Pnueli, "Timed and Hybrid Statecharts and their Textual Representation", in *Lecture Notes in Computer Science-571*, Springer-Verlag, 1991, pp. 591-620

[Koymans 88] R. Koymans, R. Kuiper and E. Zilstra, "Paradigms for real-time systems", *Proceeding of symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, Warwick, U.K. Sept, 1988, Springer-Verlag

[Koymans 92] R. Koymans, "(Real) Time: A Philosophical Perspective", *Lecture Notes in Computer Science-600*, Springer-Verlag 1992, pp. 353-370

[Koymans 90] R. Koymans, "Specifying real-time properties with metric temporal logic", *Journal of Real-Time systems*, 2, 1990

[Kung 83] C.H. Kung, *An Analysis of Three Conceptual Models with Time Perspective, in Information systems Design Methodologies-A Feature Analysis*, Ed. by T.W. Olle *et al*, North-Holland, Amsterdam, 1983

[Kung 89] C.H. Kung, "Conceptual Modelling in the Context of Software Development", *IEEE Transactions on Software Engineering*, vol. 15, Oct. 1989, pp. 1176-1187

[Kuo 67] B.C. Kuo, *Automatic Control Systems*, 2nd Edition, Prentice-Hall, 1967

[Kurki-Suonio 92] R. Kurki-Suonio, "Operational Specification With Joint Actions: Serializable Databases", *Distributed Computing*, 6, 1992, pp. 19-37

[Kurki-Suonio 93] R. Kurki-Suonio, K. Systä, and J. Vain, "Real-time specification and modeling with joint actions", in *Science of Computer Programming*, 20, 1993, pp. 113-140

[Kurki-Suonio 94] R. Kurki-Suonio, "Real Time: Further Misconceptions (or Half-Truths)", *IEEE Computer*, June 1994, pp. 71-76

[Kurshan 87] R.P. Kurshan, "Complementing Deterministic Büchi Automata in Polynomial Time", J*ournal of Computer System Sciences*, (35), 1987, pp. 59-71

[Kurshan 90] R.P. Kurshan, "Analysis of Discrete Event Coordination", *Lecture Notes in Computer Scienc*e vol. 430, Stepwise Refinement of Distributed Systems Models, Formalisms, Correctness, Ed. J.W. deBakker, W.-P. deRoever and G. Rozenbeg, Springer-verlag, 1990, pp. 414-453

[Lala 91] J.H. Lala, R.E. Harper and L.S. Alger, "A Design Approach for Ultrareliable Real-Time Systems", *IEEE Computer* 1991 pp. 12-22

[Lam 90] S.S. Lam, and A. Udaya Shankar, "A Relational Notation for State Transition Systems", *IEEE Transactions on Software Engineering*, 16(7) July1990, pp.755-775

[Lamport 77] L. Lamport, "Proving the correctness of multiprocess programs", *IEEE Transactions of Software Engineering*, 3, 1977, pp.125-143

[Lamport 78] L. Lamport, "Time, Clocks and the Ordering of Events in a Distributed System", *Communication of the ACM*, vol. 21 (7), July 1978, pp. 558-565

[Lamport 83] L. Lamport, "What Good Is Temporal Logic", *Proceedings of IFIP 9th World Computer Congress,* 1983, pp. 657-668

[Lamport 89] L. Lamport, "A Simple Approach to Specify Concurrent Systems", *Communications of the ACM*, vol. 32 (1), January 1989, pp. 32-45

[Ledru 93] Y. Ledru, "Developing Reactive systems in a VDM Framework", in *Science of Computer Programming,* 20, 1993, pp. 51-71

[Leveson 83] N.G. Leveson and P.R. Harvey, "Software Fault Tree Analysis", *The Journal of Systems and Software*, (3) , 1983, pp. 173-181

[Leveson 86] N. Leveson, "Software Safety: Why, What and How", *ACM Computer Surveys,* 18(2), June 1986 pp. 125-163

[Leveson 87] Nancy Leveson, and Janice Stolzy, "Safety Analysis Using Petri Nets", *IEEE Transactions on Software Engineering,* vol. 13 (3) March 1987, pp. 386-397

[Leveson 90] N.G. Leveson, "The Challenge of Building Process-control Software", *IEEE Software*, November 1990 , pp. 55-62

[Leveson 91] N.G. Leveson, "Software Safety in Embedded Computer Systems", *Communication of the ACM*, Feb. 1991, 34 (2), pp. 34-46

[Leveson 94] N.G. Leveson, M.P.E. Heimdahl, H. Hildreth. and J.D. Reese, "Requirements Specification for Process-control systems", *IEEE Transactions on Software Engineering*, 20(9) Sept 1994, pp. 684

[Lewis 90] Harry Lewis, "A Logic of Concrete Time Intervals", in *proceedings of the 5th annual IEEE Symposium on Logic in Computer Science*, Philadelphia, PA, June, 1990, IEEE Computer Society Press

[Luqi 88] Luqi, V. Berzins, and R.T. Yeh, "A Prototyping Language for Real-Time Software", *IEEE Transactions on Software Engineering*, 14(10), 1988, pp.1409-1423

[Lustman 94] F. Lustman, "Specifying Transaction-Based Information Systems with Regular Expressions", *IEEE Transactions on Software Engineering*, vol.20, No.3, March 1994, pp. 207-217

[Lynch 88] Nancy Lynch and Mark Tuttle, "An Introduction to Input/Output Automata", *Technical Report MIT/LCS/TM-373*, MIT, Cambridge, Massachusetts, Nov. 1988

[Lynch 90] N. Lynch, and H. Attiya, "Using mappings to prove timing properties", in *Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing*, 1990, pp.265-280

[Mahony 92] B.P. Mahony and I.J. Hayes, "A Case-Study in Timed Refinement: A Mine Pump", *IEEE Transactions on Software Engineering*, 18(9), September 1992, pp. 817-826

[Malhotra 80] A. Malhotra, J.M. Carroll, J.C. Thomas, and L.A. Miller, "Cognitive Processes in Design", *International Journal Man-Machine Studies*, vol. 12, 1980, pp.119-140

[Manna 91] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag 1991

[McDermid 93] J. A. McDermid, "Object Objection Sustained", *Proceedings of Requirements Engineering*, 1993, pp. 228

[McMenamin 84] S.M. McMenamin, and J.F. Palmer, *Essential Systems Analysis*, Englewood Cliffs, NJ: Yourdon Press, 1984

[McNaughton 66] R. McNaughton, "Testing and Generating Infinite Sequences By a Finite Automaton", *Information And Control*, (9), 1966, pp. 521-530

[Mellichamp 83] D. Mellichamp, *Real Time Computing with Applications to Data Acquisition and Control*, Van Nostrand Reinhold, 1983

[Merlin 76a] P.M. Merlin and D.J. Farber, "Recoverability of Communication Protocols", *IEEE Transaction on Communications*, September 1976

[Merlin 76b] P.M. Merlin, "A Methodology for Design And Implementation of Protocols", *IEEE Transaction on Communications*, June 1976, pp. 614-621

[Merlin 83] P.V. Merlin and G. Bochmann, "On the Construction of Submodule Specifications and Communication Protocols", *ACM Transactions on Programming Languages and Systems*, vol. 5 (1), January, 1983, pp. 1-25

[Meyer 90] B. Meyer, *Introduction to the Theory of Programming Languages*, Prentice Hall, 1990

[MIL-STD 84] MIL-STD-822B *System Safety Program Requirements*, Department of Air Force, Government Printing Office, Washington, USA, March 1984

[Mok 84] A.K. Mok, "The Decomposition of Real-Time System Requirements into Process Models", *Proceedings of Real-Time Systems Symposium*, 1984 pp. 125-134

[Mok 91] A.K. Mok, "Towards Mechanisation of Real-time System Design", in *Foundations of Real-Time Computing: Formal Specifications and Methods*, Kluwer Press, 1991

[Moller 89] F. Moller, and C. Tofts, "A Temporal Calculus of Communicating Systems", *Technical Report,* University of Edinburgh, 1989

[Monarchi 92] D.E. Monarchi and G.I. Puhr, "A Research Typology for Object-Oriented Analysis and Design", *Communications of the ACM,* Sept. 1992, vol. 35 (9), pp. 35-47

[Mullery 79] G.P. Mullery, "CORE-A Method for Controlled Requirement Specification," *4th International Conference On Software Engineering,* Sept. 17-19, 1979, pp. 126-135

[Mylopoulos 80] J. Mylopoulos, P.A. Bernstein, and H.K. Wong, "A Language Facility for Designing Data-Intensive Applications", *ACM Transacations on Database Systems,* 5(2), June 1980

[Nordström 84] B. Nordström and J. Smith, "Propositions And Specifications of Programs in Martin-Löf's Type Theory", *BIT,* 24(1984) pp. 28-301

[Nuseibeh 94] B. Nuseibeh, J. Kramer, and A. Finkelstein, "A Framework for Expressing the Relationships Between Multiple Views in Requirements Specification", *IEEE Transactions on Software Engineering,* 20(10), October 1994, pp. 760-773

[Ogata 90] K. Ogata, *Modern Control Engineering,* 2nd Edition, Prentice-Hall International, 1990

[Olle 82] T.W. Olle, H.G. Sol, and A.A. Verrijn-Stuart (Eds), *Information Systems Design Methodologies: a Comparative Review,* North-Holland, Amsterdam, 1982

[Olle 83] T.W. Olle, H.G. Sol, and C.J. Tully (Eds), *Information Systems Design Methodologies: a Feature Analysis,* North-Holland, Amsterdam, 1983

[Ostroff 87] J.S. Ostroff and W.M. Wonham, "Modelling, Specifying and Verifying Real-Time Embedded Computer Systems", *Proceedings of the Real-Time Systems Symposium*, Dec. 1987, pp. 124-132

[Ostroff 89] J.S. Ostroff, *Temporal Logic for Real-Time Systems*, Research Studies Press Ltd., Somerset, England, 1989

[Ostroff 92] J.S. Ostroff, "Formal Methods for the Specification and Design of Real-Time Safety Critical Systems", *Journal of Systems Software*, 18, 1992, pp. 33-60

[Ould 94] M.A. Ould, "Systems will be people too", *Software Engineering Journal*, November 1994, pp. 244-248

[Oxford 89] The Oxford English Dictionary, Second Edition, Clarendon Press-Oxford, Prepared by J.A. Simpson, and E.S.C. Weiner, 1989

[Oxford 90] Dictionary of Computing, Oxford, Oxford University Press, 3rd Edition, (Oxford Science Publications), Editors: V. Illingworth, E.L. Glaser, and I.C. Pyle, 1990

[Pagan 81] F.G. Pagan, *Formal Specification of Programming Languages: A Panoramic Primer*, Prentice-Hall, New Jersey, 1981

[Parnas 69] D.L. Parnas, "On the Use of Transition Diagrams in the Design of a User Interface for an Interactive Computer System", in *proceedings of the 24th ACM Conference*, New York: ACM Press, 1969, pp.379-385

[Parnas 72] D.L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *Communication of the ACM*, vol. 15, December 1972, pp. 1053-1058

[Parnas 74] David Parnas, "On A 'Buzzword': Hierarchical Structure", *proceedings of IFIP 74*, North-Holland, pp. 336-339

[Parnas 86] D.L. Parnas, and P.C. Clements, "A Rational Design Process : How and Why to Fake It ", *IEEE Transactions On*

Software Engineering, vol. SE-12, No. 2, Feb. 1986, pp 251-257

[Parnas 90] D.L. Parnas, J.V. Schouwen, and S.P. Kwan, "Evaluation of Safety-Critical Software", *Communication of the ACM*, 33(6) June 1990, pp. 636-648

[Parnas 94] Y. Wang and D.L. Parnas, "Simulating the Behaviour of Software Modules by Trace Rewriting", *IEEE Transactions of Software Engineering*, 20 (10), 1994, pp. 750-759

[Perrow 84] C. Perrow, *Normal Accidents: Living With High Risk Technologies,* Basic Books, New York, USA, 1984

[Peterson 77] J.L. Peterson "Petri Nets," *ACM Computing Surveys*, vol. 9, No. 3, September 1977, pp. 223-252

[Peterson 81] J.L. Peterson, *Petri Net Theory And The Modeling of Systems*, Prentice Hall Inc., 1981

[Pnueli 77] A. Pnueli, "The Temporal Logic of Programs", *Proceedings of the 18th IEEE Symposium on foundations of Computer Science*, 1977, pp. 46-77

[Pnueli 86] A. Pnueli, "Applications of Temporal Logic to the Specification and Verification of Reactive Systems: A Survey of Current Trends", in *Lecture Notes in Computer Science -244,* Springer-Verlag, 1986, pp. 510-584

[Pohl 94] K. Pohl, "The three dimensions of requirements engineering: a framework and its application", *Information Systems*, vol. 19 No. 3 1994

[Potts 91] C. Potts, "Expediency and Appropriate Technology: An agenda for requirements engineering research" in the 1990s, in *Lecture Notes in Computer Science-550,* Springer-Verlag, 1991, pp. 495-496

[Potts 94] C. Potts, K. Takahashi, and A.I. Antón, "Inquiry-Based Requirements Analysis", *IEEE Software*, March 1994, pp.21-32

[Pressman 87] R.S. Pressman, *Software Engineering : A Practitioner's Approach*, Second Edition, McGraw Hill, New York, 1987

[Pressman 94] R.S. Pressman, Adapted by Darrel Ince, *Software Engineering : A Practitioner's Approach*, Third Edition, European Edition, McGraw Hill, 1994

[Raju 94] S.C.V. Raju, and A.C. Shaw, "A Prototyping Environment for Specifying, Executing and Checking Communicating Real-Time State Machines", *Software Practice and Experience,* 24(2) February 1994, pp. 175-195

[Ramadge 89] P.J. Ramadge, "Some Tractable Supervisory Control Problems for Discrete event Systems modeled by Büchi Automata", *IEEE Transactions on Automatic Control*, vol. 34 (1), January, 1989, pp. 10-19

[Ramamoorthy 78] C.V. Ramamoorthy and H.H. So, "Software Requirements and Specifications: Status and Perspectives", in *Tutorial Software Methodology*, IEEE catalog no. EHO 142-0, 1978, pp. 43-164

[Ramchandani 74] C. Ramchandani, "Analysis of Asynchronous Concurrent Systems By Timed Petri Nets", *Technical Report 120*, Massachussets Institute of Technology, Feb. 1974

[Ramesh 92] B. Ramesh, and V. Dhar, "Supporting Systems Development Using Knowledge Captured During Requirements Engineering", *IEEE Transactions on Software Engineering*, 18(6), June 1992, pp. 498-510

[Ramesh 93] B. Ramesh, and M. Edwards, "Issues in the Development of a Requirements Traceability Model", *Proceedings of IEEE Symposium on Requirements Engineering*, San Diego, California, 1993

[Reisig 85] W. Reisig, "Petri nets: An Introduction", in *EATCS Monograph on Theoretical Computer Science*, New York, Springer-Verlag, 1985

[IEEE 87] "Challenges to Control: A Collective View", Report of the Workshop held at the University of Santa Barbara, September 18-19, 1986, *IEEE Transactions on Automatic Control,* vol. AC-32(4), April 1987, pp. 275-285

[Reubenstein 91] H.B. Reubenstein, and R.C. Waters, "The Requirements Apprentice: Automated Assistance for Requirements Acquisition", *IEEE Transactions on Software Engineering*, vol. SE- 17, No. 3, March 1991, pp. 226-240

[Richter 86] C.A. Richter, "An Assessment of Structured Analysis and Structured Design", *SIGSOFT Software Engineering Notes*, 11(4), 1986

[Riddle 79] W.E. Riddle, "An Approach to Software System Behaviour Description", *Computer Languages*, vol. 4, 1979, pp. 29-47

[Rockström 83] A. Rockström, and R. Saracco, "SDL-CCITT Specification and Description Language", *IEEE Transaction on Communications*, vol. COM-30 (6), June, 1983, pp. 1310-1318

[Roman 85] G. Roman, "A Taxonomy of Current Issues in Requirements Engineering", *IEEE Computer*, April 1985, pp. 14-22

[Ross 77a] D.T. Ross and K.E. Schoman Jr., "Structured Analysis for Requirements Specification", *IEEE Transactions on Software Engineering*, vol. SE-3, Jan 1977, pp. 6-15

[Ross 77b] D.T. Ross, "Structured Analysis (SA): A Language for Communicating Ideas", *IEEE Transactions on Software Engineering*, vol. SE-3, Jan 1977, pp. 16-34

[Ross 85] Interview: "Douglas Ross Talks about Structured Analysis", *IEEE Computer*, July 1985, pp. 80-88

[Rouse 81] W.B. Rouse, "Human-Computer Interaction in the Control of Dynamic Systems", *ACM Computing Surveys*, (13), 1981, pp. 13-31

[Ruggles 90] C.L.N. Ruggles (Ed), "Formal Methods in Standards" *A Report from the BCS Working Group*, Springer-Verlag, 1990

[Safra 88] S. Safra, "On The Complexity of ω-automata", *29th Annual Symposium on foundations of Computer science*, October 1988, pp. 319-327

[Sateesh 94a] T.K. Sateesh, "Modelling the Temporal Requirements of Man-Machine Interaction", in *proceedings of the 1994 Workshop on Information Technology and Systems (WITS '94)* Vancouver, Canada, December, 1994, pp. 252-261

[Sateesh 94b] T.K. Sateesh and P.A.V. Hall, "Eliciting the Requirements for Process Controlled Systems", in *proceedings of the 1994 International Computer Symposium (ICS '94)* Hsinchu, Taiwan, December, 1994

[Sateesh 94c] T.K. Sateesh and P.A.V. Hall, "Modelling the Requirements for Process Controlled Systems", in *Software Quality and Productivity: Theory, Practice, Education and Training*, Edited by Matthew Lee, Ben-Zion Barta and Peter Juliff, Chapman and Hall, 1994, pp. 88-91

[Sateesh 94d] T.K. Sateesh, "Expressing Temporal Requirements of Man-Machine Interaction", in *Integrating Human Factors with Software Engineering*, Ed. by William E. Hefley, Human Computer Interaction Institute and Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1994, pp. 123-140

[Sateesh 95a] T.K. Sateesh, "Real World Model for Real-Time Systems", in the *Requirements Engineering and Knowledge Engineering track of KAW '95*, (Ninth Knowledge Acquisition Workshop) Banff, Canada,March, 1995.

[Sateesh 95b] T.K. Sateesh, "Conceptual Model for Real-Time Systems: A Perspective", in *proceedings of the 10th Annual ACM Symposium on Applied Computing (SAC '95),* Nashville, Tennessee, February, 1995

[Sateesh 95c] T.K. Sateesh, "Making the Requirements of Process Controlled Systems Explicit", in *proceedings of the 28th Annual Hawaii International Conference on System Sciences (HICSS-28)* Maui, Hawaii, January, 1995

[Sateesh 95d] T.K. Sateesh, "Representing the Conceptual Model of a Time Critical System", in *proceedings of Groningen Information Technology Conference (GRONICS '95),* Netherlands, February, 1995

[Scharer 81] Laura Scharer, "Pinpointing Requirements", *Datamation,* April 1981, Reprinted in [Thayer 90a], pp. 17-34

[Scheffer 85] P.A. Scheffer, A.H. Stone and W.E. Rzepka, "A Case Study of SREM", *IEEE Computer,* April 1985, pp. 47-54

[Schobbens 93] P. Schobbens, "Exceptions for algebraic specifications: on the meaning of 'but'", in *Science of Computer Programming,* 20, 1993, pp. 73-111

[Sennett 89] C. Sennett (Ed), *High Integrity Software,* Pittman, 1989

[Shaler 88] S. Shaler and S.J. Mellor, *Object-oriented Systems Analysis,* Yourdon Press, 1988

[Shankar 93] A. Udaya Shankar, "An Introduction to Assertional Reasoning for Concurrent Systems", *ACM Computing Surveys,* 25(3), September, 1993, pp.225-262

[Shaw 92] A.C. Shaw, "Communicating Real-Time State Machines", *IEEE Transactions on Software Engineering,* 18(9), September 1992, pp. 805-816

[Shemer 87] I. Shemer, "Systems Analysis: A Systematic Analysis of a Conceptual Model", *Communication of the ACM*, vol. 30, No. 6, June 1987, pp. 506-512

[Shin 1987] K.G. Shin, "Introduction to Special Issue on Real-Time Systems," *IEEE Transactions on Computers*, vol. C-36, No. 8, August 1987, pp. 901-902

[Sistla 87] A.P. Sistla, M.Y. Vardi, and P. Wolper, "The Complementation Problem for Büchi automata with Application to Temporal Logic", *Theoretical Computer Science*, 49, 1987, pp. 217-237

[Smoliar 81] S.W. Smoliar, "Operational Requirements Accommodation in Distributed Systems Design", *IEEE Transactions on Software Engineering*, vol. SE-7, (6), Nov. 1981, pp. 531-537

[Snepscheut 85] Jan L.A. van de Snepscheut, "Trace Theory and VLSI Design", *Lecture Notes in Computer Science-200*, Springer-Verlag, 1985

[Sol 83] H.G. Sol, "A Feature Analysis of Information Systems Design Methodologies: Methodological Considerations", in *Information Systems Design Methodologies: A Feature Analysis*, (Ed) by T.W. Olle, H.G. Sol, and C.J. Tully, North-Holland, 1983 pp. 1-8

[Sølveberg 80] A. Sølveberg, "A Contribution to the Definition of Concepts for Expressing User's Information Systems Requirements", in *Entity-Relationship Approach to Systems and Design* (Ed. by P.P. Chen) Elsevier, Amsterdam, 1980, pp. 359-380

[Sommerville 92] I. Sommerville, *Software Engineering*, Fourth Edition, Addison-Wesley, Reading, MA, 1992

[Sommerville 93] I. Sommerville, T. Rodden, P. Sawyer, R. Bentley, and M. Twidale, "Integrating Ethnography into the Requirements

Engineering Process", *Proceedings of IEEE Symposium on Requirements Engineering*, San Diego, California, 1993

[Spivey 88] J.M. Spivey, *Understanding Z, A Specification Language and its Formal Semantics,* Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, Cambridge, England, 1988

[Stankovic 88a] J.A. Stankovic and K. Ramamritham, *Hard Real-Time Systems*, Computer Society Press of IEEE 1988

[Stankovic 88b] J.A. Stankovic, "Misconceptions about Real-Time Computing: A serious Problem for Next Generation Systems", *IEEE Computer* vol. 21 (10), 1988, pp. 10-19

[Stavely 83] A.M. Stavely, "Modeling and Projection in Software Development", *The Journal of Systems and software*, vol. 3, 1983, pp. 137-146

[Stewart 87] D.V. Stewart, *Software Engineering With Systems Analysis and Design,* Brooks/Cole Publishing Co., Monterey, 1987

[Stokes 91] D.A. Stokes, *Requirements Analysis, in Software Engineering Reference Book,* Ed. By J.A. McDermid, Butterworth-Heinemann Ltd, 1991

[Taggart 77] W.M. Taggart Jr., and M.O. Tharp, "A Survey of Information Requirements analysis Techniques", *ACM Computing Surveys*, 9(4), Dec. 1977, pp.273-290

[Taylor 80] B. Taylor, "A Method for expressing the Functional Requirements of Real-Time Systems", *IFAC Real-Time Programming*, Austria 1980 , pp. 111-120

[Taylor 82] J.R. Taylor, *An Integrated Approach to the Treatment of Design and Specification Errors in Electronic Systems and Software, in Electronic Components and Systems*, (Eds). E. Lauger and J. Motoft, North-Holland, Amsterdam, 1982

[Tennent 81] R.D. Tennent, *Principles of Programming Languages*, Prentice-Hall International, Inc., 1981

[Terwilliger 87] R. Terwilliger and R. Campbell, "PLEASE: A language for incremental software development", in *Proceedings of the 4th International Workshop on Software Specification and Design*, April 1987

[Thayer 90a] R. Thayer and M. Dorfman, *Standard, guidelines, and examples on system and software requirements engineering*, Tutorial, IEEE Computer Society Press, California, 1990.

[Thayer 90b] R. Thayer and M. Dorfman, S*ystem and software requirements engineering*, Tutorial, IEEE Computer Society Press, California, 1990.

[Thomas 81] W. Thomas, "A Combinatorial Approach to the Theory of ω-Automata", *Information and Control*, 48, 1981, pp. 261-283

[Tse 91] T.H. Tse, and L. Pong, "An Examination of Requirements Specification Languages", *The Computer Journal,* 34(2), 1991, pp. 143-152

[Turski 86] W.M. Turski, "And No Philosophers' Stone, Either", *Information Processing 86*, North-Holland, 1986, pp.1077-1080

[Turski 88] W.M. Turski, "Time Considered Irrelevant for Real-time Systems", *BIT*, 28, 1988 , pp. 473-486

[Vardi 86] M.Y. Vardi and P. Wolper, "An automata Theoretic approach to Automatic Program Verification", In *Proceedings of the Symposium on Logic in Computer Science*, Cambridge, June 1986, pp. 322-331

[Verrijn-Stuart 87] A. Verrijn-Stuart, "Themes and Trends in Information systems: TC8", 1975-1985, *Computer Journal*, vol. 30 (2), 1987, pp. 97-109

[Wang 91] Yi Wang, "CCS + Time = an interleaving model for real-time systems", in *ICALP'91, Lecture Notes in Computer Science-510*, Springer-Verlag, 1991, pp.217-228

[Ward 85] P.T. Ward and S.J. Mellor, *Structured Development for Real-Time Systems*, Yourdon Press, 1985

[Ward 86] P.T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing", *IEEE Transactions on Software Engineering*, vol. SE-12, 1986, pp. 198-210

[Ward 89] P.T. Ward, "How to Integrate Object Orientation with Structured analysis and Design", *IEEE Software*, March 1989, pp. 74-82

[Wasserman 79] A.I. Wasserman, and S.K. Stintson, "A Specification Method for Interactive Information Systems", in *proceedings of the Symposium on Specification of Reliable Software*, 1979, pp. 68-79

[Wasserman 90] A.I. Wasserman, P.A. Pircher, and R.J. Muller, "The Object-Oriented Design Notation for Software Design Representation", *IEEE Computer*, vol.23 (3), March 1990, pp. 50-63

[Welke 92] R.J. Welke, "The Case Repository: More than Another Database Application", in *Challenges and Strategies for Research in Systems Development*, Ed. by W.W. Cotterman and J.A. Senn, John-Wiley and Sons Ltd, 1992, pp. 181-218

[Weyuker 94] E. Weyuker, T. Goradia, and A. Singh, "Automatically Generating Test Data from a Boolean Specification", *IEEE Transactions on Software Engineering*, vol. SE-20, No.5, May 1994, pp. 353-363

[Wing 88] J.M. Wing, "A Study of 12 Specifications of the Library Problem", *IEEE Software,* July 1988, pp.66-76

[Wing 90] J. Wing, "Specifiers Introduction to Formal Methods", *IEEE Computer* 23(9), Sept. 1990, pp. 8-26

[Wirfs-Brock 90] R. Wirfs-Brock, B. Wilkerson, and L. Wiener, *Designing Object-Oriented Software*, Prentice-Hall Inc., 1990

[Wirth 77] N. Wirth, "Towards a Discipline of Real-Time Programming", *Communication of the ACM*, Aug. 1977, 20 (8) , pp. 577-583

[Wirth 95] N. Wirth, "A Plea for Lean Software", *IEEE Computer*, February 1995, pp.64-68

[Witrow 80] G.J. Witrow, *The Natural Philosophy of Time*, Clarendon Press, Oxford, 1980

[Yeh 84] R.T. Yeh, P. Zave, A.P. Conn and G.E. Cole jr., "Software Requirements: New Directions and Perspectives," in *Handbook of Software Engineering* edited by C.R. Vick and C.V. Ramamoorthy, Van Nostrand Reinhold Co. 1984, pp. 519-543

[Yourdon 79] E. Yourdon and L. Constantine, *Structured Design*, Prentice Hall International, Englewood Cliffs, N.J, 1979

[Yourdon 90] E. Yourdon, "Auld Lang Syne", *BYTE*, October 1990, pp. 257-263

[Zave 81] P. Zave and R.T. Yeh, "Executable Requirements for Embeded Systems", in *Proceedings of the 5th International Workshop on Software Engineering*, IEEE New York, March 1981 pp. 295-304

[Zave 82] P. Zave, "An Operational Approach to Requirements Specification for Embedded Systems", *IEEE Transactions on Software Engineering*, vol. SE-8, No.3, May 1982, pp. 250-269

[Zave 84] P. Zave, "The Operational Versus The Conventional Approach To Software Development", *Communication of the ACM*, Feb. 1984, vol.27, No.2 pp. 104-118

[Zave 86] P. Zave and W. Schell, "Salient Features of an Executable Specification Language and Its Environment", *IEEE Transactions on Software Engineering*, vol. SE-12, No.2, Feb 1986, pp. 312-325

[Zave 91a] P. Zave, "A Comparison of the Major Approaches to Software Specification and Design", in *System and Software Requirements* vol.1, Ed. By R. Thayer and M. Dorfman, IEEE Computer Society Press, 1990, pp. 199

[Zave 91b] P. Zave, "An Insider's Evaluation of PAISLey", *IEEE Transactions on Software Engineering*, vol. SE-17, No.3, March 1991, pp. 212-225

[Zeigler 76] B.P. Zeigler , *Theory of Modelling and Simulation*, Wiley, New York, 1976

[Zeigler 84] B.P. Zeigler, *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, Orlando, Florida, 1984