A Methodology for the Requirements Analysis of Critical Real–Time Systems

Rogério de Lemos

Ph.D. Thesis

November 1994

University of Newcastle upon Tyne

Department of Computing Science

Abstract

This thesis describes a methodology for the requirements analysis of critical real–time systems. The methodology is based on formal methods, and provides a systematic way in which requirements can be analysed and specifications produced. The proposed methodology consists of a framework with distinct phases of analysis, a set of techniques appropriate for the issues to be analysed at each phase of the framework, a hierarchical structure of the specifications obtained from the process of analysis, and techniques to perform quality assessment of the specifications.

The phases of the framework, which are abstraction levels for the analysis of the requirements, follow directly from a general structure adopted for critical real–time systems. The intention is to define abstraction levels, or domains, in which the analysis of requirements can be performed in terms of specific properties of the system, thus reducing the inherent complexity of the analysis.

Depending on the issues to be analysed in each domain, the choice of the appropriate formalism is determined by the set of features, related to that domain, that a formalism should possess. In this work, instead of proposing new formalisms we concentrate on identifying and enumerating those features that a formalism should have.

The specifications produced at each phase of the framework are organised by means of a specification hierarchy, which facilitates our assessment of the quality of the requirements specifications, and their traceability. Such an assessment should be performed by qualitative and quantitative means in order to obtain high confidence (assurance) that the level of safety is acceptable.

In order to exemplify the proposed methodology for the requirements analysis of critical real–time systems we discuss a case study based on a crossing of two rail tracks (in a model railway), which raises safety issues that are similar to those found at a traditional level crossing (i.e. rail–road).

Acknowledgements

"Procuro unir as pontas meio rotas através do Tempo real ou inventado enquanto fico me perguntando o que isto tudo tem a ver..."
Lygia Fagundes Telles, *As Horas Nuas.*

It is my pleasure to thank firstly my supervisor Professor Tom Anderson for his constructive criticism of my research work, and for his many helpful comments upon the contents of this thesis. I would also like to thank Dr C R Snow and Dr N Speirs for their help in the early years of my research.

My special thanks go to my colleagues Dr Paul Ezhilchelvan, for our short but productive co-operation in the area of group membership for distributed real-time systems, and Dr Amer Saeed, for his immense patience to endure me for so long while we have conducted the research work in the area of requirements analysis for safety-critical systems. I would also like to extend my thanks to all my colleagues of the department of Computing Science, and my special thanks go to my colleagues in the CSR, who made my tenure at this department more pleasant.

My studies at the University of Newcastle were made possible through the financial support of CAPES/Ministry of Education (Brazil) for which I am sincerely grateful.

Table of Contents

Chapter 1

Introduction

A critical real–time system is a real–time system for which there exists at least one failure that can be adjudged to cause an accident (e.g. loss of life) /PDCS 90/. (We employ the term "critical real–time system", instead of "safety–critical system", in order to emphasize the real–time characteristics of the type of systems that we are concerned with; it can be the case that a safety–critical system might not have any timing characteristics at all.) The development of such systems is usually controlled by regulatory authorities, specific to the area of application, such as transportation, aerospace, energy industry, medicine, and defence. These regulatory boards impose certification criteria, in accordance with the safety standards established for the different sectors, which must be satisfied before the system can be put into service.

As the use of computers increases in critical applications and the level of criticality of the roles performed by the computers also increases, currently available methods have been proven inadequate for the development of critical real–time systems because they do not provide the required level of confidence /Leveson 91b/. Hence there is a need to search for new methods for the development of software for such systems. Within the context of software development this thesis is essentially concerned with the requirements phase, where we adopt the usual practice of dividing the development process into requirements, design, and implementation. Furthermore, we are only concerned with issues related to "software safety", although "safety" is an attribute of the system rather than just software. The requirements phase is characterised by intensive communication between the customer, or a group of experts, and the analyst. It comprises the elicitation of the requirements (i.e. acquisition and expression of the customer requirements), the analysis of the requirements (i.e. checking for ambiguities, contradictions, and incompleteness of the requirements), and the explicit definition of

the requirements (i.e. the construction of the requirements specification). Experience has shown that faults in the requirements specification can corrupt later stages of software development, introducing faults into the final system which can subsequently cause an accident /Ericsson 81/. Furthermore, the longer it takes (in terms of system development) to identify a fault, the more expensive it will be to nullify the effects of the fault /Boehm 81/.

One approach that has been advocated for achieving an improvement in the dependability of critical real−time systems is the utilization of formal methods /MoD 91a, Moser 90/. The potential advantages of using formal methods, apart from contributing to improving the understanding of the requirements specification, include unambiguity, refinement consistency, and the opportunity to check for completeness (with respect to a key set of questions and inferences based on the information specified /Jaffe 1991/). However, the utilization of formal methods has its limitations in the sense that faults cannot be completely eliminated. A first limitation concerns those faults that can arise during the process of formalising the requirements, either due to misunderstandings of the user needs or when expressing those needs. A second limitation is related to the assumptions that are usually made in order, for instance, to obtain a mathematical model of the real world; although these assumptions are useful (or even essential) in simplifying the analysis process, they may also introduce imperfections into the model. Finally, a third limitation arises from observing past experience in the utilization of formal methods which shows that a formal verification may itself contain faults. To maximise the benefits that can be obtained from the application of a formal method, a set of guide-lines should be provided in order to advise the user of the circumstances under which the method can and should be applied, as well as how it can be applied most effectively /Wing 90/. Although there are a great number of formal methods, they can be roughly classified into a few broad types in accordance with their expressive power /Pnueli 86, Wing 90/. The utilization in a development of more than one type of formal method has become increasingly common because of their different characteristics and expressive power, which enables the representation and

analysis of complementary views of a system, with each formal method working to its own strengths /Pnueli 86, Olderog 91/.

With respect to the requirements phase, the motivation for the application of formal methods is to locate and remove faults introduced during this phase, before proceeding to any subsequent phase of development. Within the requirements phase, our concern is specifically with the analysis of the requirements, a multidisciplinary activity where there is a need to analyse certain views of the system which have different, and sometimes unrelated, characteristics. Employing a multi–formalism approach, we are able to choose appropriate formalisms in order to analyse each view in terms of its specific characteristics. In our proposed methodology, a set of formal methods are employed in accordance with the characteristics of the system, and the type of analysis to be performed.

As a general structure for critical real–time systems, in particular process control systems, we adopt a commonly accepted structure of partitioning the system into three different components: the *operator*, the *controller*, and the *physical process* (or *plant*) /Saeed 91/. Apart from these, we also identify the components *plant interface* and *operator interface* which contain those components of the controller that support the interface to the plant and the operator, respectively. The entire system has an external environment, which is that part of the rest of the world which may affect, or be affected by, the system.

The contents of this chapter are organized as follows. The next section presents a set of definitions which are basic for the rest of the thesis. In section 2 we outline the proposed methodology within the context of the above general structure for critical real–time systems. Section 3 presents a survey of current techniques for requirements analysis. Section 4 summarises the technical contributions of this work, and section 5 provides an outline of the subsequent chapters of the thesis.

1.1. Basic Definitions

Our basic concern is to ensure that an *accident* (an unintended event or sequence of events that causes death, injury, environmental or material damage) does not occur. However, we encounter the problem that we cannot directly influence all of the factors that could conceivably lead to an accident. To deal with this problem the notion of a *hazard* (a physical situation or state, expressed as a system condition) is typically used to describe a circumstance from which an accident could ensue, even though it might be prevented by the controller. While a system is in a hazard state, whether an accident might ensue or not, might depend on issues which are outside the scope of the controller /Leveson 91b/. Thus, to ensure safety, we strive to identify all of the hazards and then apply techniques to reduce the probability of the system getting into a hazard state. The effort that should be employed to prevent a hazard depends on the risk associated with that hazard. The *risk* of a hazard is related to the likelihood of the system entering that hazard state, the likelihood that the hazard will actually lead to an accident, and the expected potential loss associated with such an accident, i.e. the accident severity /Leveson 86/.

An *initiating event* (analogous to an erroneous transition /Lee 90/) is an event to which a subsequent hazard state could be attributed /MoD 91b/. An *unsafe state* is a state which could lead to a hazard, in the absence of corrective action by the safety controller and/or safety operator and in the absence of any subsequent initiating events. If a state is not an unsafe state then it is said to be *safe*. These definitions ensure if a system is in a safe state then it will not subsequently enter a hazard state unless an initiating event occurs.

1.2. A Methodology for Requirements Analysis

The methodology for the software requirements analysis for critical real–time systems presented here is based on establishing a clear separation of the mission and the safety requirements. The mission requirements focus on what the system is supposed to achieve in terms of function and timeliness requirements, and the safety requirements focus on

the elimination and control of hazards and on the limitation of damage in the case of an accident. The benefits of making this distinction, which is essentially logical rather than physical, during requirements analysis are: resolution of potential conflicts, detection of omissions and inconsistencies between the mission and safety issues, the ability to focus on the safety–critical issues, and simplification of safety certification.

In the following we discuss a methodology, based on formal methods, which provides a systematic way in which the safety requirements for critical real–time systems can be analysed. The proposed methodology consists of a framework with distinct phases of analysis, a set of techniques appropriate for the issues to be analysed at each phase of the framework, a hierarchical structure of the specifications obtained from the process of analysis, and techniques to perform the quality assessment of the safety specifications.

The phases of the framework correspond to the domains of analysis which follow directly from a general structure adopted for critical real–time systems. The intention is to define domains of analysis in which the analysis of requirements can be performed in terms of specific issues of the system, thus reducing the inherent complexity of the analysis. In the *conceptual domain* we specify the role that the system has in its environment, and identify those failure behaviours of the system that constitute accidents. In the *plant domain* we identify the state variables representing the physical state of the plant, the properties of the physical process, namely the physical laws, rules of operation, and hazards, together with the conditions over the physical process which prevent the system from entering into a hazard state. In the *plant interface domain* we identify the properties of sensors and actuators, such as their failure rates and the imperfections that are introduced while monitoring and controlling the variables of the plant. Finally, in the *controller domain* the properties of the controller are identified, such as the top level organization of the components of the controller and the failure rates of these components.

An advantage in performing requirements analysis in different domains of analysis is that, depending on the issues to be analysed in each domain, an appropriate formalism

can be selected. By applying the appropriate formal method, we are able to emphasize specific characteristics of the domain in which the analysis is performed. An inconvenience of an approach based on more than one formalism is the need to find a common formal framework which can be used to link the different formalisms, a problem which does not exist when only a single formalism is adopted. However, such "single–minded" approaches, for instance durational calculus /Ravn 93/, do not provide the required notational flexibility for the whole task of requirements analysis and, moreover, attempts to overcome this limitation necessarily increase the complexity of the single formalism. Clearly, there exists a tradeoff between these two approaches. In the single formalism approach, a formal verification is more easily obtained; on the other hand, in the multi–formalism approach we are able to perform requirements analysis in a more thorough manner. However, the number of approaches which make use of multi–formalisms, by combining the benefits that each individually can provide, is increasing /Barroca 92, Fidge 92, He 90, Morasca 89/; this could be taken as an indication that current limitations might soon be overcome.

In this work, instead of proposing new formalisms we concentrate on identifying the features that a formalism should have in order to be applied in the requirements analysis of a specific domain. There were two reasons for taking this approach. First, there are already enough formalisms that could fulfil the set of features that are required in order to perform the analysis in each domain, and second, not very much is known concerning the application of formal methods in the requirement analysis of critical real–time systems. In other words, the usual practice has been to identify, for a given formalism, the features which justify its application to requirements analysis /Ravn 93, Scholefield 92/, rather than the opposite approach of identifying the salient features that a formalism should possess to be suitable for requirements analysis. Within this thesis, the formalisms adopted for each of the domains of analysis, identified above, were selected solely to illustrate the fundamentals of our approach to requirements analysis.

Now we proceed to introduce, briefly, some of the features that formalisms should have for each of the domains of analysis. For the conceptual domain, we seek notations which allow the reasoning about and the representation of a very high level description of the preferred behaviour of the system within its enclosing environment. A formalism for this domain should support the representation of causal relationships between states of the system and environment. For the plant domain, we have to analyse systems which contain both continuous and discrete variables. Hence it is necessary during this phase to employ both differential equations and discrete mathematics. For this domain, a descriptive formalism that enables behaviour to be specified in terms of axioms over a model of the system is most appropriate. Such formalisms allow specifications to have a conjunctive nature, in the sense that new requirements can be added without the need to reconstruct the full specification. For the plant interface domain, we are concerned with modelling the properties of sensors and actuators. The specifications obtained from the plant domain are re–written in terms of the imperfections of sensors and actuators. For this analysis we need formalisms which have the same characteristics as the formalisms specified for the plant domain. For the controller domain, we have to analyse the high level architectural design of the controller, modelling the interactions between its components and the operations that must be performed by these components. To perform this analysis we need a formalism which explicitly embodies concurrency and non–determinism.

The utilization of more than one formalism for requirements analysis has another drawback, apart from those already mentioned, which is the need to work with different notations. In order to handle this, we introduced the event/action model (E/A Model), which is based on primitive concepts such as events, states and actions, and which allows the discrete behaviour of systems to be described in terms of a set of variables representing the state of the system. The aim is to employ these primitive concepts as abstract notions when using different formalisms. To formally express the primitive concepts of the E/A model and to facilitate the formal analysis, we employ the predicate

event/action notation (PEA notation). In this notation, the primitive functions are related to predicates over system variables.

The specifications produced at each phase of the framework are located and organised by means of a specification hierarchy, which facilitates assessment of the quality of the requirements specifications and their traceability. Such an assessment should be performed by both qualitative and quantitative means in order to obtain high confidence (assurance) that the level of safety is acceptable. The qualitative approach to assurance is based on formal and informal analyses of the specifications, which are performed by applying verification and validation techniques, respectively. The quantitative approach to assurance either measures or estimates the probability that the system will enter in hazard state both due to the violation of assumptions made when constructing mathematical models of the plant, and as a consequence of the measured or predicted failure rate of sensors, actuators and controller components.

A general characteristic of the methodology discussed in this thesis is the utilization of formal methods from as early a stage as possible during requirements analysis. We achieve this by formalising the properties of the physical process, namely the rules of operation, physical laws and hazards, and subsequently obtaining the other requirements specifications by successive formal refinements. By adopting this approach we aim to facilitate the qualitative assessment of the requirements specifications by minimizing the validation process (checking the correctness of a formal specification against an informal one), and maximizing the verification process (checking the correctness between two formal specifications); the latter process can (in principle) be performed mechanically with the help of proof checkers, such as HOL /Gordon 88/ and EHDM /Rushby 91/. The proposed utilization of formal methods differs from existing approaches in that the usual practice is to obtain first the informal specification of the complete system, and then formalise only that part of the specification which is related to the controller /Dauchy 91/.

1.3. Current Requirements Analysis Techniques

In this section, we present an overview of some related work.

The approach suggested in this thesis, of dividing the requirements analysis into domains of analysis, and using the appropriate notation for the different tasks and domains of analysis, has not previously been investigated. What has usually been presented is the utilization of a single formalism such as Invariants /Bishop 86/, Temporal Logic /Gorski 86/, Petri nets /Leveson 87/, and THL /Saeed 90/. However, there are two approaches in the literature which use more than one formalism for requirements analysis /Jahanian 88/, /Ostroff 90/. These papers are primarily concerned with the analysis of timeliness requirements; they do not seek to establish a methodology for the analysis of the whole set of safety requirements of critical real–time systems.

In /Jahanian 88/, the specification of the system is realized in terms of Real–Time Logic (RTL) and Modecharts; Modecharts produce a decision procedure for classes of properties expressed as RTL formulas. The system properties are verified using Computation Graphs, obtained from the Modecharts, to check if the corresponding RTL formulas comply with the Modecharts.

In /Ostroff 90/ a methodology is presented for the specification and verification of real–time systems; this makes use of multiple formal methods, for different domains of analysis. A Timed Transition Model (TTM), with an extension to finite state machines, is used to model "plant–controller processes", whereas Real Time Temporal Logic (RTTL) is used for specifying properties over the trajectories of a TTM. A weakness of this approach is that a TTM specification of a system must be constructed before RTTL formulae can be stated; this causes difficulties in validation since the initial formalization step is large, and the ability to develop a descriptive specification is lost.

The Formal Requirements Specification Techniques (FOREST) /Cunningham 86, Goldsack 91/ were developed to support requirements engineering for real–time systems; the approach consists of a methodology called Structured Common Sense

(SCS), based upon a number of existing techniques such as CORE and JSD, and a formal method called Modal Action Logic (MAL), based upon a many–sorted first order logic. The FOREST approach is noteworthy since it placed emphasis on an overall methodology and allows modelling of the operational environment.

In terms of analysis procedures for localizing faults in requirements specifications, a general correctness criteria which must be satisfied by process control systems was presented in /Jaffe 91/. However, the application of this set of criterion is more a technique to validate requirements specifications than a methodology to produce them, because issues such as how the specifications are obtained and what methods are employed are not addressed. The same observation could be applied to the work by Parnas & Madey /Parnas 91/, where a systematic way of producing documentation in the development of computer systems is discussed. Comparing with our approach, although both approaches adopt very similar domains of analysis in which requirements specifications are produced, the cited publications only discuss how the specifications should be represented for documentation purposes, and say little or nothing about the methods which should be employed to obtain the specifications.

The ProCos project (Provably Correct Systems) has conducted research in the area of specifying and verifying system requirements /Ravn 93/. The aim of this work is to specify requirements and verify the system design using mathematical models. There is no treatment of the issues that arise during analysis of the requirements. Although the approach identifies domains of analysis, in terms of the physical process and controller, a single formal method, Durational Calculus, is employed.

Instead of employing formal methods, tools based on rigorous approaches have been employed for the production of requirements specifications; STATEMATE /Harel 90/ is such a tool. In STATEMATE, requirements specifications are produced from three perspectives: the functional view as data flow diagrams, the behavioural view as statecharts, and the structural view as module charts. One of the most powerful features of STATEMATE is its simulation capability.

1.4. Contribution of the Thesis

In the following, we outline the main contributions of this thesis; references to work already published by the author are provided.

The essential contribution of this thesis is the proposal of a methodology for the requirements analysis of critical real–time systems. For this methodology we introduced a framework in which the analysis is performed in domains with the aim of simplifying the analysis and thereby leading to more accurate specifications. These domains of analysis are associated with the components of the system, and for each domain, depending on the perspective being considered /Finkelstein 92/, we can perform different types of analysis. For each domain, depending on the issues to be analysed, we defined a set of features that a formal method should possess. The bases of the framework were introduced in /Saeed 91a/, and elaborated for use in the requirements analysis of a train set crossing case study /de Lemos 92b/. The overall methodology, including a revised version of the framework, was presented in /Anderson 93/.

Another contribution is related to the approach taken in performing the quality assessment of the requirements specifications. Essentially, the hierarchical way in which the specifications are structured, facilitates how the qualitative and the quantitative analysis are performed in an integrated manner /Saeed 92/. The former, by applying verification and validation techniques, and the latter, by measuring the probability of a hazard occurring due to the impact of violations of the assumptions and the failure rates of sensors and actuators.

For temporal analysis, we introduced the event/action model (E/A model) based on work by Jahanian & Mok /Jahanian 86/, with some additional features that make it applicable to requirements analysis; specifically, the model is now flexible enough to support both discrete and dense time structures, and can depict timing analysis graphically /de Lemos 92a/. In a further development, the role of the E/A model was extended by using its basic concepts (events, states and actions) to describe the behaviour of systems within the

different domains of analysis. In order to support this wider role of the E/A model, we also introduced the predicate event/action notation (PEA notation) which provides a compact notation and facilitates formal analysis of the system behaviour /Anderson 93/. In this thesis we present another version of the PEA notation which allows temporal analysis to be performed in both qualitative (ordering of events) and quantitative (time of event occurrence) terms.

Two other contributions should be mentioned (although not directly related to the requirements methodology) because of the influence they had on the main research work. The first of these dealt with the issue of value inconsistencies resulting from time uncertainties in distributed real–time systems. The aim was to identify the influence that time inconsistencies within the system had on the value domain of a system variable, and to define the minimum conditions, depending on whether we are dealing with continuous or discrete variables, which should be imposed on the time domain in order to obtain consistency in the value domain /de Lemos 91/. In the second contribution, we extended the framework for requirements analysis to the next stage in software development by performing the link between requirements and design of real–time software /de Lemos 92c/. In the approach, which was object–based, we employed Petri nets with Objects (PNO) /Sibertin 85/ for the controller analysis, and for design we employed an object–based notation that offers support for active objects /Shrivastava 91/. Also in this work we introduced an approach for the interface analysis, based on the specification of standard, exceptional and failure behaviours for the sensors and actuators; this served as a basis for the interface analysis which is presented in this thesis.

Apart from the above, but less directly related to the subject of requirements analysis, is another contribution which relates to research work conducted in the area of distributed real–time systems /de Lemos 90, Ezhilchelvan 90/. The problem addressed in these papers is that of how to maintain a consistent and timely knowledge of the group of non–faulty processors by those processors which are members of the group.

Compared with the solution initially proposed /Kopetz 89/, our group membership algorithm was much simpler and could tolerate a greater number of processor failures.

As can be seen from the bibliographical references made to other publications, part of the work discussed in this thesis arises from joint work with colleagues from the department of Computing Science. Work on distributed real–time systems (which was not elaborated in this thesis) was conducted jointly with Paul Ezhilchelvan. In the area of requirements analysis for critical real–time systems, the work has benefitted greatly from interaction with Amer Saeed, who was responsible for sowing the first seeds in this area of research in this department /Saeed 90b/, most notably by developing the formalism Timed History Logic (THL) which we have used extensively in this thesis.

1.5. Outline of the Thesis

The rest of the thesis is organized as follows.

Chapter 2 introduces process control systems as the type of critical real–time systems with which we are concerned. As a general structure, we partition critical real–time systems into three distinct components: the operator, the controller and the physical process (or plant). This structure is further decomposed to reflect the decision to separate the mission and safety issues. The roles of each of these components are then described.

In Chapter 3 we provide the basis from which the analysis of timeliness requirements is performed. Initially, we discuss how the flow of time can be modelled, by defining two types of time structures which will be used in the framework. Finally, we discuss the issue of value inconsistencies due to timing uncertainties in the context of distributed real–time systems design, and provide the conditions to be imposed on the time domain in order to obtain consistency in the value domain.

In Chapter 4 we introduce the event/action model (E/A model) which describes the behaviour of systems at the various domains of analysis in terms of primitive concepts

such as events, actions and states, and the concept of a timeline. The utilization of these concepts provides considerable flexibility, enabling descriptions of system behaviour to be expressed ranging from the activities of the physical entities of the plant to the temporal ordering of the computational tasks of the controller. However, to formally express the primitive concepts of the E/A model and to facilitate the formal analysis, we employ the PEA notation. The concepts of the E/A model are then formalised in terms of THL and PrT nets.

Chapter 5 presents a framework for requirements analysis and a hierarchical structure for the safety specifications. The framework is presented in terms of phases of analysis which correspond to the domains of analysis which follow directly from the general structure adopted for critical real–time systems. For each phase of the framework we discuss the appropriate formalism to be employed, the time structure which should be used, and what the E/A model should represent.

Chapter 6 discusses how the quality of the safety specifications can be assessed qualitatively and quantitatively in order to obtain high confidence (assurance) that the level of risk is acceptable. The qualitative approach to assurance is based on formal and informal analysis of the specifications, whereas the quantitative approach is based on an evaluation of the probability that the system will enter a hazard state (because of faulty assumptions about the behaviour of the physical process, or failures of the components of the interface and controller).

In order to exemplify our methodology for the requirements analysis of critical real–time systems, in Chapter 7 we discuss a case study based on a train set crossing, which raises safety issues that are similar to those found at the traditional level crossing (i.e. rail–road).

Finally, in Chapter 8 we summarise the main issues presented in this thesis, and suggest directions for future investigation.

Chapter 2

General Structure of Critical Real–Time Systems

2.1. Introduction

The adoption of a general structure for a critical real–time system will be a useful guide for their requirements analysis, since it establishes the domains in which different methods and techniques for the analysis can be applied. In this chapter we present such a general structure, identifying the basic components of the system and describing their respective role within the system. The type of systems under consideration are process control systems, also known as embedded systems /Leveson 91b, Ostroff 87/. A process control system has been defined as an arrangement of components interconnected in such a way as to maintain, or to affect in a prescribed manner, some physical quantity or condition of the process /Lowe 71/.

The restrictive view that we have taken in considering only process control systems, instead of adopting a more general approach which could also include information processing systems /Shrivastava 90/, has two essential motivations: the first is to exploit existing dissimilarities between the structures of process control and information processing systems in terms of the role (or behaviour) that the components of these structures might have in the respective systems, and the second is that we would like to use the knowledge already available, and which has been successfully applied, in developing process control systems /Åström 85, Parnas 91/.

The contents of this chapter are organized as follows. In the next section we present a short historical evolution of the application of computers in process control systems. Section 3 describes the basic components of a critical real–time system. In section 4 we justify the roles of the mission and safety components. Finally, we present some concluding remarks on the issues presented in this chapter.

2.2. Evolution of Critical Real–Time Systems

In this section we present a historical overview of the utilization of computers in process control systems. The aim is to present the application domain that we are concerned with, and to confirm that computers have been applied for some time in process control systems. The major difference between the earlier applications and those only now emerging, which we label in this thesis as "critical", is the higher risk of occurrence of an accident associated with the latter.

The so–called real–time computing systems which initially appeared were utilised in large commercial applications, usually transaction–based, such as airline booking systems, bank automation and information systems for stock trading. These systems were based on large centralized computer configurations where the software had a relatively minor role, resulting in very complex systems with low reliability /Stankovic 89/. The essential feature of these systems was the delivery of the required service within a predictable and acceptable interval of time. However, issues like reliability, high performance and functionality were limited or impossible to achieve because of the lack of development methods and design disciplines. The consequences of a system failure were never catastrophic because non–computer based systems were invariably available as back-ups.

Apart from transaction–based systems, another application area from which real–time systems also emerged was that of process control systems: the automation of applications such as petro–chemical plants, electrical power generation and distribution, pulp and paper industries, manufacturing systems, mining, and aircraft systems. The utilization of computers in these applications started in the mid fifties with the development of a digital computer for a military aeroplane, firstly, for supervisory control, and later for automatic flight and weapon control.

Computer systems in commercial applications of process control were introduced more as a result of pressure from the computer industries in order to expand their markets

rather than due to any initiative from the process and manufacturing industries; the first commercial control computer was installed, in 1959, in a oil refinery at Texas, by TRW (the aerospace company that lost the contract for the military aeroplane mentioned above) /Williams 84/. In addition to monitoring the plant, the digital computer was also responsible for the closed–loop control of some of the plant's functions. The computers used in this period, called the pioneering period /Åström 85/, were very large, slow and unreliable. Because the computers were so unreliable, they had to be used in supervisory control only, while conventional analog controllers were used for the primary control functions. One of the aims of this period was to gain acceptance of the new techniques by the plant's personnel.

The next period, the centralized period, was characterized by the replacement of analog instrumentation, so that the computer could directly access the plant – direct digital control (DDC). The installation of the first DDC systems was achieved, independently, in the year 1962 by ICI at a soda ash plant in England, and the Monsanto company at an ethylane plant in Texas /Williams 84/. The results obtained from early DDC installations generated a great deal of interest which gave a major impetus to the development and further utilization of the minicomputer in the field of computerized process control. Many of the drawbacks of the computer systems used in DDC, in terms of reliability and processing time, were due to limitations of the available technology at that time. The standard solution was the design of much larger and faster computer systems, with the high costs of these systems justified by the incorporation of all computer functions, both supervisory and DDC, in one centrally located computer. This period was characterized by two features: first, the vast communication system, needed to bring signals from the plant to the centralized computer and to return control signals back to the plant, was very expensive and prone to electrical noise problems which were a major cause of unreliability; second, depending on the level of criticality of the application, there was the need to have a complete analog system, redundant to the DDC, in order to improve the reliability of the overall system.

The evolution of computerized process control and the corresponding increase in the number of applications was made possible by progress in understanding physical processes, and by advances in computer and control technology. The third period, the microcomputer and distributed control period, was characterized by the introduction of microprocessors and local area networks. The advent of the microprocessor meant that computers were smaller and relatively inexpensive, influencing drastically, in terms of size, performance and cost, the control equipment, and enabling the same system to embody both control (DDC) and logic (PLC) functions. (PLC stands for "programmable logic controller": special purpose computers which execute the logic and sequence control functions, originally performed by separate relay systems.) Local area networks were first introduced to provide high speed data links enabling data to be exchanged between the many small and distributed computers, each one controlling just one or a few loops of the physical process. The development of distributed control resulted in architectures incorporating hierarchical control systems with different levels of control: the upper system levels depending on the lower levels for the physical process data, and the lower system levels in turn depending on the higher levels for more sophisticated control functions. This type of configuration, which is physically dispersed but logically central, has been used in most process control applications, particularly manufacturing systems /Chintamaneni 88/.

Recent years have seen increasing deployment of computers in process control applications considered safety–critical, where the consequences of failure may entail danger to human life, property, and the environment. Examples of such applications include control of commercial aircraft /Rouquet 86, Traverse 89/, air traffic control systems /Avizienis 87/, emergency shut down systems for nuclear reactors /Sayet 90/, railway signalling systems /Hachiga 92/, and medical systems /Leveson 93/.

2.3. Structure of the System

A system is any identifiable mechanism which maintains a pattern of behaviour at an interface between the mechanism and its environment, where an interface is a place of

interaction between two systems /Lee 90/. Hence the environment of a system is another system that interacts at an interface with the given system. From the structural viewpoint, a system is a set of components bound together in order to interact. Recursively, a component of a system is another system /Laprie 89/.

For applications referred to as process control systems, the primary focus of this thesis, a commonly accepted structure is to partition the system into three distinct components: the *operator*, the *controller*, and the *physical process* (or *plant*) /Kopetz 89, Saeed 91/. The system has an environment, which is that part of the rest of the world which may affect, or be affected by, the system. Within this structure the controller, for example, might be a computer based system which has a distributed architecture.

Within this structure for critical real–time systems, as shown in figure 2.1, we identify two interfaces: the interface between the operator and the controller, and the interface between the plant and the controller. The former is the place of interaction between the operator and the controller, and the latter is the place of interaction between the plant and the controller. This structure can be considered a simplified version because the subsystems within the controller which are dedicated to providing support for operations at the controller interfaces have not (yet) been separately identified. Refining the structure of the controller we identify three distinct components: the *operator interface*, the *plant interface* and the *controlling system* (in the sequel, we sometimes reuse the term "controller" instead of "controlling system"). The plant interface contains those components of the controller that supports the interface between the controlling system and the plant, namely sensors and actuators, and the operator interface contains those components of the controlling system that supports the interface between the controlling system and the operator, such as consoles and dials. Another simplification introduced in this model, is the absence of an interface between the operator and the physical process. It might be the case that the operator has to have direct control over some variables of the physical process which the controller might not have any influence. We have adopted this simplification because of the current tendency in computer based

process control systems of forcing all the interactions between the operator and the plant to pass through the controlling system /Traverse 89/. This allows to check whether the attitudes of the operator do not violate any pre–established operational envelop; a reason for this measure is to avoid possible human errors in the control of the system.

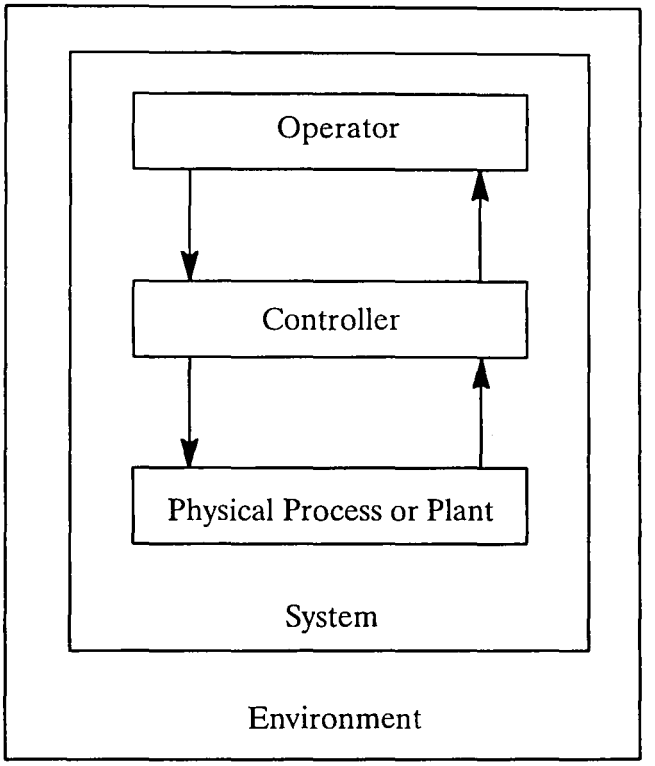


Figure 2.1. General structure for critical real–time system.

The interaction between the operator, controller and plant can be described as follows. The controller receives data from the physical process, through the sensors, and setpoints fixed by the operator – the latter are inputs, such as parameters and conditions, which establish how the system should operate. This data is processed by the controller, and the results are output, through the actuators, to the physical process. The outputs of the controller could influence the behaviour of the physical process in such a way that changes can be observed by the controller through the sensors – thus closing the loop. For correct behaviour from a critical real–time system, it is extremely important to avoid inconsistencies between the states of these three components. The states of the components are: the "real" state of the physical process, the state of the model (adopted

in the controller) which represents the physical process, and finally, the state of the physical process as perceived by the operator (in another model) via the information produced by the controller.

Now we proceed to describe each of these three main components of a critical real–time system, together with the plant interface (the operator interface is outside the scope of the present work).

2.3.1. Physical Process or Plant

The physical process or plant is the component within the general structure whose behaviour is to be either monitored or controlled. The wide range of relevant physical processes can be classified according to the dynamics that their physical variables present, which can be either continuous or discrete. *Continuous* or *analog variables* are those variables whose changes in the value domain are always continuous, such as temperature, velocity and pressure. For these variables, from a known state it is possible to predict other future states (values) of the variable. *Discrete variables* are those variables whose changes in the value domain are in terms of discontinuities, such as the position of a switch. For these variables, knowing the current state, it does not help in predicting subsequent states of the variable.

Systems whose variables are continuous are known as *continuous variable dynamic systems* (CVDS) and systems whose variables are discrete are known as *discrete event dynamic systems* (DEDS) /Ostroff 86, Ho 89/. However, it is worth noting that there is a third category of physical processes, which includes combinations of these two types of systems, in various forms. Such systems, containing both continuous and discrete variables, are known as *hybrid systems* /Maler 92, Nerode 93/. When analysing a system, it is important to identify which parts are CVDS and DEDS because of the different techniques that have to be employed in their development. In the following, we present some of the features that distinguish the two type of systems.

2.3.1.1. Continuous Variable Dynamic Systems

The behaviour of continuous variable dynamic systems (CVDS) can be described by ordinary and partial differential equations which express the *physical laws*, that is the essential dynamic features of the physical process. These systems can tolerate brief state inconsistencies (of limited magnitude) between the actual state of the physical process and the state of the model by which the controller represents the physical process. After resuming from some disruption which has generated an inconsistency of states, the system is able to recover to a consistent state because the controller can determine an estimate of the state of the physical process while the system was disrupted. On the other hand, if the system suffers a long duration disruption then the states of the controller and the physical process might become inconsistent, impairing a timely and useful recovery of the state of the controller. The following example illustrates how CVDS deal with state inconsistencies. If the controller of a train misses a reading of the speed of the train, the controller can either simply continue to use the previous reading of the speed or estimate a new value based on a sequence of past readings – this is possible because the train's speed is a continuous variable and thus its behaviour can be predicted within known margins of confidence. In this example, the safety of the train should not be affected so long as the model of the physical process and confidence margins are properly employed. However, the safety of the system could be affected if the controller fails to obtain updated readings for sequence of values of the speed, since the state of the controller might start to diverge substantially (become inconsistent) from the actual state of the physical process. Other examples of CVDS systems are chemical plants /Wilkie 90/ and commercial aeroplanes /Traverse 89/.

2.3.1.2. Discrete Event Dynamic Systems

The behaviour of discrete event dynamic systems (DEDS) can be modelled by automata /Leveson 88/ or discrete mathematics /Jahanian 86/. Since DEDS are invariably man–made systems, there are no physical laws which constrain the system configuration other than natural limits of material and ergonomics, which are the *rules of operation*.

These systems do not necessarily tolerate momentary state inconsistencies between the actual state of the physical process and the state maintained by the controller. The system cannot then recover to a consistent state after a disruption because the controller is not able to estimate any state changes that might have occurred in the physical process while the system was disrupted. However, long disruptions in the system can sometimes be tolerated, mainly when the system remains in a state which is considered to be safe. The following example illustrates how DEDS deal with state inconsistencies. If the controller of a train misses the reading of a track–side signal, the controller will not be able to maintain its representation of the physical process to be consistent with reality. In this situation the safety of the train could be severely affected. But if the state inconsistency can be detected (or anticipated), the usual practice is to disrupt the operation of the system completely by forcing the system into a safe state (e.g. stop the train), in order to avoid the system getting into some arbitrary but undesirable state whose consequences might be catastrophic. Other examples of such systems are manufacturing systems /Cassidy 85/, rail and road crossings /de Lemos 92/, and high–voltage power substations /CIGRÉ 83/.

2.3.2. Controller

The controller is designed to generate appropriate signals which make the plant behave in the desired manner by ordering the operations in the plant and regulating the values of the physical variables. If the controller is computer based we do not make any restriction as to whether the architecture is centralized or distributed. The implementation of the controller can either be analog or digital. Typical examples of controller sub–components are proportional–integral–derivative controllers to implement the control algorithms, and programmable logic controllers to implement the sequencing of events.

The control algorithms to be implemented by the controller can use feedback and/or feedforward techniques. Feedback techniques cause the outputs of the plant to follow a desired path accurately, in spite of external disturbances or internal parameter

changes, by monitoring the plant variables to be controlled and applying the required adjustments. On the other hand, feedforward techniques measure and estimate the external disturbances (inputs that do not participate in feedback control) and try to predict their effect on the plant. Corrective actions are taken by adjusting the control signals generated by the controller. In feedback control, unknown "outputs" are measured and compared with known (desired) inputs to generate control signals. In feedforward control, unknown "inputs" are measured and that information, along with the desired inputs, is used to generate control signals that can reduce errors due to these unknown inputs or variations in them /de Silva 89/.

The activity of modern controllers is not limited to maintaining and regulating the variables in a plant, but also includes supervision and planning by means of scheduling the operations in the plant /Jaffe 91/.

2.3.3. Operator

The operator is responsible for any human actions to be applied on the physical process or controller. These actions include the initialisation of parameters, plant operation and maintenance. Within the adopted general structure we consider only those activities concerned with the normal operation of the system, ignoring maintenance issues, and furthermore, we assume that the greater the role of computing systems in all levels of control, the less is the influence that operators can have on the physical process; this tendency is confirmed by the latest developments in airborne systems /Rouquet 86/. Thus, in our structure the operator only has direct access to the controller in order to set its initial parameters, and to operate the physical process.

Although the details of the interaction between the operator and the controller are dependent on the type of application under consideration, two general features can be outlined. The first of these concerns the asynchronous nature of the interactions between the two components, and the second concerns the long time duration between successive activities from the operator on the controller, in relation to the number of activities

performed by the controller over this period. This latter feature derives from the fact that in automatic critical real–time systems the influence of the operator is minimal; the operator only interferes when the behaviour of the physical process starts to deviate from the required behaviour. Examples of such applications, which seldom require the interference of an human operator, are (usually) physically large plants, such as manufacturing production lines, petrochemical plants and nuclear power plants. In these applications the operator activities are normally restricted to handling emergency situations, and the initial and final steps in the operation of these physical processes.

2.3.4. Plant Interface

The plant interface is made up of sensors and actuators /de Silva 89/. Refining the controller of figure 2.1, in order to incorporate explicitly the plant interface, we obtain the schematic layout of figure 2.2. Sensors are devices used to monitor the behaviour of

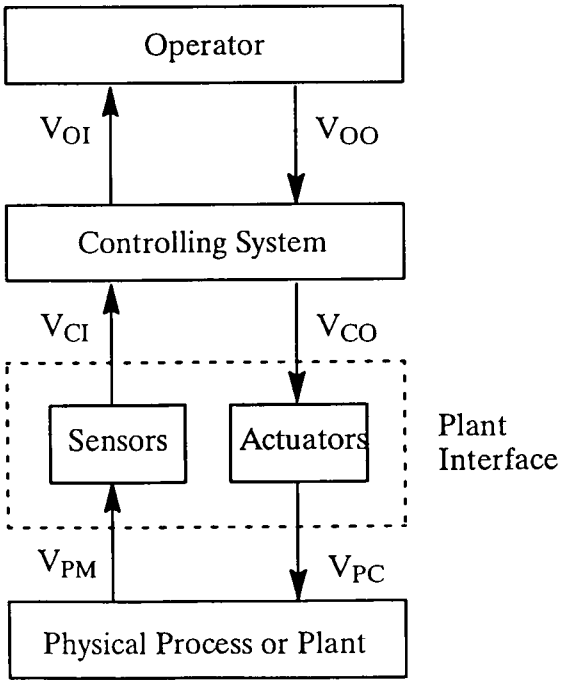


Figure 2.2. Structure of the plant interface.

the plant: strain gauges, thermocouples, thermistors, and tachometers are examples of sensors. Actuators are devices that can influence the behaviour of the plant: solenoids, valves, relays, and DC motors are examples of actuators.

The selection of sensors and actuators to be deployed in the plant interface depends on which variables of the plant it is important to control. Moreover, the selection must also consider which variables can physically be measured and which variables can be controlled, respectively, for sensors and actuators. Other factors that can influence the choice of sensors and actuators are /Franklin 86/: technology, functional performance, quality factors and cost.

2.4. System Variables

In the general structure of a critical real–time system, for each level of abstraction a state space, in terms of state variables, is defined. At the level of abstraction of the plant, the state variables represent the physical state of the plant; at the level of abstraction of the plant interface, the state space of the plant is reduced to those state variables which are monitored and controlled by the sensors and actuators; at the level of abstraction of the controller, the state variables represent the state that the model of the controller has of the physical process; at the level of abstraction of the operator, the state variables represent the physical state of the operator; at the level of abstraction of the operator interface, the state variables represent the state of the components of the operator interface.

From the system structure shown in figure 2.2, we define the following variables. The set of variables of the system – *system variables* (V), consists of the variables of the physical process – *physical variables* (V_P), the variables of the plant interface – *plant interface variables* (V_{PI}), the variables of the controller – *controller variables* (V_C), and the variables of the operator – *operator variables* (V_O). In the following, we will not considered the operator as a component of the system, and thus we are only concerned with the physical variables, the plant interface variables and the controller variables ($V_P \cup V_{PI} \cup V_C \subseteq V$).

The variables of the physical process can be partitioned into the set of *monitored variables* (V_{PM}) and the set of *controlled variables* (V_{PC}). The monitored variables are those

physical variables whose values are measured by the controller, and the controlled variables are those physical variables whose values the controller intends to modify. Some physical variables can be both monitored and controlled. The variables which are neither monitored nor controlled, i.e. those which are irrelevant for the controller, are not considered in this model. The union of the monitored and controlled variables is a subset of the physical variables ($V_{PM} \cup V_{PC} \subseteq V_P$).

The variables of the controller can be partitioned into the set of *input variables* (V_{CI}) and the set of *output variables* (V_{CO}). The input variables are those controller variables which correspond to the monitored variables, and the output variables are those controller variables which correspond to the controlled variables. The variables which are neither input nor output, i.e. those which are not related to the model that the controller has of the physical process, are not considered in this model. The union of the input and output variables is a subset of the controller variables ($V_{CI} \cup V_{CO} \subseteq V_C$).

The variables of the plant interface are a subset of the plant and controller variables. For instance, not all the variables of the physical process are read by the sensors of the plant interface; it might be the case that it is unfeasible to build a sensor which measures the value of a particular physical variable.

Each variable (v_i) of the set of system variables ($V=\{v_1, v_2, v_3, ...\}$) can be represented as a function which maps the timeline (T) into the range of values of the variable (RV_{v_i}), thus we have $v_i(t)$: $T \rightarrow RV_{v_i}$. This range, or set, of values is the collection of values that a system variable can assume, and it is partitioned into two subsets, the anticipated values ($RV^a_{v_i}$) and the unanticipated values ($RV^u_{v_i}$): $RV_{v_i}=RV^a_{v_i} \cup RV^u_{v_i}$. The anticipated values of a variable are those values which are expected to occur, and the unanticipated ones are the rest. The concept of unanticipated values is similar to that of noncode value errors presented and discussed in /Powell 92/.

PAGE
MISSING
IN
ORIGINAL

be satisfied at run–time by preventing the mission controller from issuing any control commands to the safety controller; it is then up to the safety controller to monitor the operating conditions of the mission controller and, on the basis of its own observations, to issue the appropriate control commands to the mission controller. Thus, the mission controller can influence the safety controller only via the observations made by the safety controller (either over the state of the physical process or mission controller). The safety controller must assume sole control of the system, that is, it must override the mission controller, whenever the system enters into an unsafe state. Also, the safety controller endeavours to prevent the occurrence of hazardous states, and, in the worst case, if an accident becomes inevitable, must minimize the amount of damage that will be caused to the system and the environment. On the other hand, the role of the safety operator is to set up the safety limits of the safety controller and the plant, and to interfere in the operation of the plant whenever it enters into an unsafe state which the safety controller was not able to handle.

With reference to figure 2.3, it would be preferable to have two totally independent controllers, with separate sensors and actuators at the plant interface, to avoid the risk of any common failures of both the mission and the safety controller. Two fully independent subsystems can be utilized in those situations, for example, where such a separation is already enforced at the physical process level /Sayet 90/. Unfortunately, for some applications this ideal structure is difficult to obtain due to limitations found in the physical process, or restrictions imposed by the design of the system. Figure 2.3 depicts this latter situation where the safety controller makes (possibly restricted) use of the sensors and actuators of the mission controller.

2.6. Concluding Remarks

In this chapter we have introduced process control systems as the type of critical real–time systems that we are concerned with. The approach proposed in this thesis for conducting requirements analysis of critical real–time systems is based on the structure of a system in order to identify the levels of abstraction, or domains of analysis, in which

the analysis is to be conducted. From the structure of a system, and the relations between the components of the system, we define the domains of analysis and the sequential order of the domains in which the analysis is to be conducted. In the study presented in this chapter, we have identified, for an initial level of abstraction in which a system should be analysed, the plant, operator and controller, as being the main components of a critical real–time system from which the requirements analysis should be conducted. The issues related to the requirements analysis will be discussed in more detail in Chapter 5. Within the scope of this thesis, we restrict the overall analysis of the system to the specification of the safety controller, taking into consideration the restrictions imposed by the activities to be performed by the operator, the physical laws and/or rules of operation of the physical process, and the particular characteristics of the environment of the system.

Chapter 3

Time Domain in Critical Real–Time Systems

3.1. Introduction

The service delivered by a system can be defined in terms of the *value* or information content of the service, and the *time* or instant of delivery of the service /Powell 92/. Moreover, the service can be described with respect to a set of system variables which are defined in terms of a *value domain* and a *time domain*. In order to model the "flow of time" in a time domain, we define a time structure which is based on temporal entities connected by relations. As temporal entities, we can consider *points* (instants, moments that imply no duration), *periods* (stretches of time that imply duration) and *events* (composite entities, usually associated with linguistic constructs, which describe an activity)[1], and for each of these entities we associate different relations, such as identity and precedence, in order to define a time structure /van Benthem 90/. (Periods as temporal entities should not be confused with intervals, the former implies duration however there are no defined boundaries, i.e. points (they are obtained by maximal nests of periods /Whitehead 29/), while the latter is referred as a time stretch between two time points.) The choice of which temporal entity should be regarded as most suitable as the basis for a time structure is a controversial issue /Allen 84, Newton–Smith 81, van Benthem 90/. A representative of a time structure based on points as temporal entities is the "physical time" which is the time as employed by physicists to represent their theories. The "physical time" is considered to be isomorphic to the Euclidean line, in the sense that the points in time have the same order, as well as the same topological and metrical properties as the points on straight line. Time structures which make use of

1. This is a very broad definition, which is used, in the context of this thesis, to present the different temporal entities which represent the passage of time; in the next chapter, 'events' will be redefined as temporal entities of no duration.

periods as temporal entities have been used in knowledge representation of temporal information /Allen 83/. Finally, the usage of events as temporal entities is more rare to represent the passage of time, perhaps due to their compositional nature, i.e. events are spatio–temporal entities whose logical analysis cannot remain purely temporal /van Benthem 90/.

In order to show the necessity of completely specifying (during requirements analysis) the time domain of system variables, we also present in this chapter a study relating the consequences that uncertainties in the time domain of a system variable can have on its value domain. This study of the analysis and synthesis of time uncertainties is presented in the context of distributed real–time systems.

The contents of this chapter are organized as follows. In the next section we present a general model of time which will serve as a basis for the analysis of timeliness requirements. In section 3, we present a study of the consequences that uncertainties in the time domain can have on the value domain of system variables. Finally, the last section briefly reviews this chapter.

3.2. General Model of Time

Two classes of temporal relationships are distinguished for real–time systems, in accordance with the causal relationships between events /Koymans 88/:

> *qualitative* temporal relationships are only concerned with the ordering of events in time, which means that there is no explicit time reference for the occurrence of an event;
>
> *quantitative* temporal relationships, referred to in this thesis as *timing constraints*, are concerned with event occurrences where explicitly time appears as a parameter.

Timing constraints can be further subdivided into *timeliness requirements* and *performance requirements*. Timeliness requirements are intervals of time which dictate

the response time of the system to stimuli (either physical or logical). Performance requirements are the amount of processing that a system performs within an interval of time. (In this work we are only concerned with timeliness requirements.) A timeliness requirement concerning a sequence of events can be specified in one of the following forms:

> an *absolute* time representation, associated with one of the events in the sequence;
>
> a *relative* time representation, stipulating a limit on the elapsed time between the occurrence of two events in the sequence.

For our general model of time we assume that time (in an abstract sense) is a single and absolute phenomenon, and that Newtonian time (rather than of relativistic time) is sufficient to model the flow of time in the type of systems with which we are concerned. For the formal definition of time we adopt the concept of point structures, instead of period structures, since they are more widespread in their usage and more convenient, in mathematical terms, for modelling the physics of the real–world. Supplementing from the definition of time structures for the general model of time, we also define the concept of an interval in terms of a time structure, and a containment relation between time structures.

3.2.1. Time Structures

In this section, two traditional time structures are defined in terms of time points ordered by a relation of precedence ($<$, less than). This relation generates a temporal ordering of the type "earlier than" or "before" between time points of the structure /van Benthem 90/.

A *point structure* $\mathcal{P}$ is an ordered couple $\langle T, < \rangle$ of a non–empty set T of time points with a binary relation $<$ on T.

In order to define a time structure, the following conditions should be imposed on the point structure defined above. The transitivity condition stipulates the formal aspect of the flow of time, and irreflexivity stipulates the absence of stops in the modelling of the flow of time.

Transitivity

For any three time points t_1, t_2, t_3 in T, if t_1 precedes t_2 and t_2 precedes t_3 then t_1 precedes t_3.

$$\forall t_1, t_2, t_3 \in T: t_1 < t_2 \wedge t_2 < t_3 \Rightarrow t_1 < t_3.$$

Irreflexivity

A time point in T cannot precede itself.

$$\forall t \in T: \neg(t < t).$$

A point structure satisfying the transitivity and irreflexivity conditions is a strict partial order. However, if time has to flow in a single stream instead of multiple (or branching) streams, then a linearity condition should also be imposed on the point structure.

Linearity

For any two time points in T, either one precedes the other or they are the same point.

$$\forall t_1, t_2 \in T: t_1 < t_2 \vee t_2 < t_1 \vee t_1 = t_2.$$

A point structure satisfying the transitivity, irreflexivity and linearity conditions is a strict total order (or irreflexive total order /Goswami 92/). Thus the set of time points T can be referred to as a *timeline*, an abstract concept which captures the passage (or flow) of time. The next condition stipulates that there is a continual succession of time points towards the past and the future of a timeline.

Succession

For any time point, there is always a point that precedes it and another that succeeds it.

$$\forall t_1 \in T: \exists t_2 \in T: t_1 < t_2,$$
$$\forall t_1 \in T: \exists t_2 \in T: t_2 < t_1.$$

Rather than defining an endless succession of points, we could alternatively define a begin and an end of a timeline.

$$\exists t_1 \in T: \neg\exists t_2 \in T: t_2 < t_1,$$
$$\exists t_1 \in T: \neg\exists t_2 \in T: t_1 < t_2.$$

The next step is to establish how dense are the points on the timeline. There are two alternatives, either to adopt "infinite divisibility" in which case the timeline is dense, or adopt "step−wise succession" in which case the timeline is discrete. These two conditions are defined as follows.

Denseness

Between any two time points there is an intermediate time point.

$$\forall t_1, t_2 \in T: (t_1 < t_2 \Rightarrow \exists t_3 \in T: t_1 < t_3 < t_2).$$

Discreteness

Every time point has an adjacent timepoint; between two adjacent time points no intermediate time point exists.

$$\forall t_1, t_2 \in T: (t_1 < t_2 \Rightarrow \exists t_3 \in T:(t_1 < t_3 \wedge \neg\exists t_4 \in T: t_1 < t_4 < t_3)) \wedge$$
$$\forall t_1, t_2 \in T: (t_1 < t_2 \Rightarrow \exists t_3 \in T:(t_3 < t_2 \wedge \neg\exists t_4 \in T: t_3 < t_4 < t_2)).$$

From the conditions, defined above, imposed on a point structure, we are able to identify two types of time structure, namely, discrete and dense. However, before that we introduce the concept of a *time structure* as being a tuple $\mathcal{T} = \langle \mathcal{P}, + \rangle$, where $\mathcal{P}$ is a point structure, and $+$ the addition operation on T.

A *dense time structure* is defined as a tuple $\mathcal{T}_{DE} = \langle \mathcal{P}, + \rangle$, which satisfies the conditions of *transitivity*, *irreflexivity*, *linearity*, *succession*, and *denseness*. Either the Real numbers

(**R**) or the Rational numbers (**Q**) can be used to model dense time structures. Time structures that are isomorphic to the reals are known as *continuous time structures*.

A *discrete time structure* is defined as a tuple $\mathcal{T}_{DI} = \langle \mathcal{P}, + \rangle$, which satisfies the conditions of *transitivity*, *irreflexivity*, *linearity*, *succession*, and *discreteness*. In order to establish the notion of distance between two adjacent points of a timeline, we introduce the concept of a *time grid* which is a set of equidistant points on the timeline; each point on the time grid is referred as a *tick*, and the distance between any two adjacent points is the *granularity* (Δ) of the time grid. For discrete time structures the granularity is greater than zero ($\Delta > 0$) and allows to quantify the discreteness of the time grid. The Δ–distance is another condition that can be added to the definition of a discrete timeline to denote the distance that separates two adjacent time points.

Δ–distance

There exists a time point t_3 that lies between two other time points t_1, t_2 in T, if and only if t_1 precedes t_2 by more than Δ.

$$\forall t_1, t_2 \in T: t_1 + \Delta < t_2 \Leftrightarrow \exists t_3 \in T: t_1 < t_3 < t_2.$$

Instead of a tuple, we can define a discrete time structure as a triple $\mathcal{T}_{DI} = \langle \mathcal{P}, \Delta, + \rangle$, with $\Delta > 0$, and satisfying the same conditions as before. A discrete time structure becomes isomorphic to the integers (**I**) when $\Delta = 1$. A discrete time structure may be realized by a physical clock, to which we associate exactly one time grid. We assume the existence of an abstract (and hypothetical) *reference clock* which has a sufficiently fine granularity to enable the measurement (in principle) of the duration between ticks of any other clock.

3.2.2. Relationships Between Time Structures

The relationship between time structures is captured through the containment relation. This condition is important when passing information between time structures in order to assess the amount of information that might be lost.

A time structure $\mathcal{T}_2$ contains another time structure $\mathcal{T}_1$ (denoted by $\mathcal{T}_2$ **con** $\mathcal{T}_1$) iff $T(\mathcal{T}_1)$, the set of time points of $\mathcal{T}_1$, is a subset of $T(\mathcal{T}_2)$, the set of time points of $\mathcal{T}_2$.

$$\mathcal{T}_2 \textbf{ con } \mathcal{T}_1 \stackrel{def}{\Leftrightarrow} T(\mathcal{T}_1) \subseteq T(\mathcal{T}_2).$$

This relation of containment is only applied to discrete time structures since dense time structures are already at the finest granularity, which implies that the set of time points of any other time structure must be (isomorphic to) a subset of the time points of a dense time structure.

3.2.3. Time Intervals

An interval implies duration and refers to what lies between two time points, and is defined as a subset of the point set T of a time structure. We define two sorts of intervals.

A *convex interval* (*Int*) is a subset of T that represents an uninterrupted stretch of time.

$$\forall t_1, t_2 \in Int: \forall t_3 \in T: (t_1 < t_3 < t_2 \Rightarrow t_3 \in Int).$$

A *non−convex interval* (*NInt*) is a union of disjoint convex intervals.

$$\forall Int_1, Int_2 \in NInt: Int_1 \neq Int_2 \Rightarrow Int_1 \cap Int_2 = \emptyset.$$

From now on, the terms convex interval and interval are interchangeable.

3.3. Uncertainties in the Time Domain

In this section, we present a study, in the context of distributed real−time systems, which relates inconsistencies in the value domain with the time uncertainties that exist within the components of a system or between the system and its environment. (In the following, the use of the term "system" will refer to "control system" of the system structure, previously presented.) After performing the analysis, in order to identify the causes of the value inconsistencies, we present the conditions which should be imposed on the system in order to avoid these inconsistencies.

A characteristic of a real–time system is that the service to be delivered by the system must satisfy requirements in both the value and time domains if the system is not to be judged as having failed. This implies that a real–time system must operate properly despite the presence of possible uncertainties in both these domains. Uncertainties in the value domain do not present any new problems; for example, it is common practice to account for the imperfections of an instrument (or measuring device) through standard performance parameters. However, uncertainties in the time domain do present a new set of fundamental problems in distributed real–time systems, due to the time uncertainties that exist between the timelines of the system and the environment and, within the system, between the timelines of its components /Kopetz 90/. The source of these latter time uncertainties is the clock synchronization imperfections that exist between the various system components. For example, when two system components attempt to measure the same physical variable at a specific time, they may in fact (as a result of these time uncertainties) measure the variable at two different times, leading to value discrepancies.

The rest of this section discusses the conditions under which two values (which can be either a real world value and a system value, or two system values) representing the same physical variable are considered to be mutually consistent, given that the time instants at which these values can be obtained lie within a known bound. In /Kopetz 90/ a condition for mutual consistency was discussed in terms of the maximum rate of change of the value domain. Here, we extend that work to include a condition in terms of the time domain which is sufficient to guarantee consistency between the two values.

The analysis performed is restricted to the case of value uncertainties which arise when components of a system read the value of a physical variable from the system environment; however, the analysis can easily be adapted to the case where the system components output control signals to the environment.

3.3.1. System Model

As a model of time, we assume a discrete time structure, as already defined. With each component in the distributed real–time system we associate a local clock which has its own time grid. All of these local clocks are assumed to be synchronised by implementing a synchronised time grid /Kopetz 87/, which permits us to interpret all system–wide timestamps. This ensures that the timestamps of any two nodes, related to the occurrence of the same event, are at most one tick apart on the synchronised time grid, implying that the order in time of any two events can be deduced consistently at all nodes if the events occur at least two ticks apart. We assume the existence of a reference clock which enables us to quantify the duration between ticks of any other clock.

Within a distributed real–time system we consider two components, C_a and C_b, interconnected by a communication network, as shown in figure 3.1. (Only two components are necessary for the pairwise analysis of uncertainties presented here.) These two components are part of the system/environment interface. They represent sensors which measure the same physical variable, and exchange these measured values with each other.

The following notation will be used to model uncertainties in both value and time domains, which may exist in any measurement:

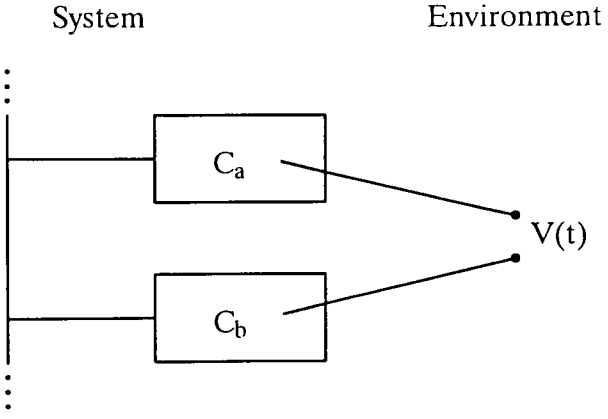


Figure 3.1. Model of the system.

x – represents the subscript of a system component, $x \in \{a, b\}$;

t, t_x – timelines of the environment and C_x respectively;

$t(i), t_x(i)$ – the *ith* tick on the timelines of the environment and C_x respectively;

$r(t(i)), r(t_x(i))$ – the *ith* tick on the timelines of the environment and C_x respectively, as timed by the reference clock;

$V(t)$ – the function that describes the behaviour of variable V of the environment;

$V_x(t_x)$ – functions that describes the behaviour of variable V as measured by C_x;

ε – represents the error factor in the measurements which is either systematic or random, $\varepsilon \in \{s, r\}$;

τ – represents uncertainty in the time domain;

τ_ε^x – represents uncertainty in the time domain, for C_x and error factor ε;

v – represents uncertainty in the value domain;

v_ε^x – represents uncertainty in the value domain, for C_x and error factor ε.

3.3.2. Uncertainties in measuring Value and Time

Uncertainties represent errors in the knowledge of the actual value of a physical variable obtained by a measurement. That is, errors are algebraic differences between a measurement of a value and the actual value of the corresponding physical variable /de Silva 89/. The actual value is considered to be defined with respect to a precise reference scale of measurement, known as a standard, which is based on an arbitrary chosen reference of suitable magnitude that is assumed to be unvarying.

Measurement errors are typically decomposed into two factors: *systematic* and *random*. The former can be characterised as a mean error – the difference between the actual value and the mean value of a set of test measurements – and the latter by a confidence interval around the mean value where a specific percentage of all test measurements lie (commonly expressed in terms of the standard deviation). *Accuracy* and *precision*, respectively, are the two terms usually employed to express the extent of systematic and random errors of a measuring component (instrument), and they may be defined as follows:

> *Accuracy* is the conformity of a measured value to an actual value, or standard value;
>
> *Precision* is the maximum difference between any two measured values of the same actual value.

Accuracy and precision are properties of a measuring component (instrument)[2], and they influence any measurement performed by that instrument.

The concepts of accuracy and precision will be employed in this section as a basis for the analysis to be performed on the type of uncertainties that may arise either in measuring time or the values of physical variables, in distributed real–time systems. In fact, the expression "measuring time" actually means measurement of the value of a physical variable which represents the passage of time.

As mentioned before, the discrepancies that can arise in measuring the value of a physical variable, at a particular point in time, stem from the limitations of accuracy and precision associated with the instruments employed in the measurement of time and value. In order to examine relationships related to the tolerances in measuring time and value, we consider the two functions $V_a(t_a)$ and $V_b(t_b)$ which represent two system values

2. Other properties may be associated with a measuring instrument which are not considered here, such as resolution, repeatability and reproducibility.

of the same physical variable. (The same analysis can be modified to cover the case of a real world value $V(t)$ and a system value $V_a(t_a)$.)

In the following, we present the four types of discrepancies, in the time and value domains, that can arise when two system components measure the value of a physical variable, at a particular point in time.

a. No uncertainties in either time or value measurements, hence the variables $V_a(t_a)$ and $V_b(t_b)$ can be considered identical:

$\forall i: r(t_a(i)) = r(t_b(i)) \wedge V_a(t_a(i)) = V_b(t_b(i))$;

b. Uncertainty in value measurement, but not in time measurement:

$\forall i: r(t_a(i)) = r(t_b(i)) \wedge |V_a(t_a(i)) - V_b(t_b(i))| \leq v$.

Remark: In the above two cases the timelines t_a and t_b are considered to be exactly synchronous.

c. Uncertainty in time measurement, but not in value measurement:

$\forall i: |r(t_a(i)) - r(t_b(i))| \leq \tau \wedge (\forall j: r(t_a(j)) = r(t_b(j)) \Rightarrow V_a(t_a(j)) = V_b(t_b(j)))$;

d. Uncertainty in both value and time measurements:

$\forall i: |r(t_a(i)) - r(t_b(i))| \leq \tau \wedge |V_a(t_a(i)) - V_b(t_b(i))| \leq v$.

In this study we are concerned with those cases that are represented by relationship *c*, where value uncertainties arise solely because of time uncertainties that may exist between the clocks of the measuring instruments. That is, we assume that the measurement of the value of a physical variable is not subject to errors, but we cannot guarantee that two corresponding measurements are made at exactly the same time. Because the value of the physical variable can change, the two values obtained from the measurements could be different.

3.3.3. Value Uncertainties due to Time Uncertainties

The analysis of value uncertainties due to time uncertainties will be realised in two parts: firstly, in terms of the individual system components, and secondly, in terms of the whole system, by combining the uncertainties of the system components.

The time measurement performed by a component concerns the basic operation of reading the value of its own local clock at specific time intervals corresponding to the granularity of the synchronised time grid. The time uncertainties that can exist at each component are caused by the time differences between the occurrence of the ticks of the local clock and the corresponding ticks of the synchronised grid. These time uncertainties are due to skews in the components' local clocks, and the limitations of the process used to synchronise the clocks of the system; for each tick of the synchronised time grid the time uncertainty varies.

In the following, we initially identify, in terms of a component, how the systematic and random errors in measuring time can affect the measurement of the value. After combining the two errors, we present how value discrepancies can occur, between measurements performed by two system components, because of the time uncertainties that exist between the components.

In terms of a system component, the systematic timing error, which is obtained from the mean value $t_x^{\mu}(i)$ for a set of time measurements performed by C_x, may be represented as follows:

$$(r(t(i)) - r(t_x^{\mu}(i))) = \tau_s^x \qquad |V(t(i)) - V_x(t_x^{\mu}(i))| \leq v_s^x.$$

The same applies to the random error which may be represented as follows:

$$|r(t_x{}^{-}(i)) - r(t_x{}^{+}(i))| \leq \tau_r^x \qquad |V_x(t_x{}^{-}(i)) - V_x(t_x{}^{+}(i))| \leq v_r^x.$$

The notation $t_x{}^{-}(i)$ and $t_x{}^{+}(i)$ represents the extreme values of the confidence interval around the mean value of the readings of the components' local clocks.

The total time uncertainties (τ^x) of a component may be expressed in terms of the systematic error τ_s^x and the random error τ_r^x:

$$\tau^x = |\tau_s^x| + \tau_r^x/2.$$

The total value uncertainty (ν^x) is constrained by the maximum absolute rate of change ($K=max\ |dV/dt|$) in value of the physical variable being measured

$$\nu^x = K\tau^x.$$

For a component C_x, the value uncertainty due to time uncertainty, associated with the measurement by C_x of a physical variable, is bounded as follows:

$$|r(t(i)) - r(t_x(i))| \leq \tau^x \qquad |V(t(i)) - V_x(t_x(i))| \leq \nu^x.$$

An upper limit for the value uncertainty (ν) and the time uncertainty (τ) are determined by:

$$\tau = \tau^a + \tau^b \qquad \nu = K\tau.$$

The value uncertainty between the components C_a and C_b due to the time uncertainty between the two components, is bounded as follows:

$$|r(t_a(i)) - r(t_b(i))| \leq \tau \qquad |V_a(t_a(i)) - V_b(t_b(i))| \leq \nu. \qquad (*)$$

In the above analysis we have not taken into account the clock synchronization mechanism; this will bring the times of the clocks of the components closer together, reducing τ. In fact, clock synchronization must ensure that τ is less than the granularity g of the synchronized grid.

3.3.4. Value Inconsistencies due to Time Uncertainties

Two values of the same physical variable are said to be mutually consistent if their uncertainties in time and value are within known and bounded ranges.

One approach to determining a bound on the difference between two values representing the measurement of the same physical variable, is to establish a maximum value uncertainty in terms of the maximum time uncertainty, which in our case refers to the granularity of the synchronised time grid. This notion was called the timestamp – inconsistency of an observation in /Kopetz 90/. This approach was employed in the previous section in order to calculate the maximum value uncertainty due to time uncertainty. Although it is necessary to have such a condition on the value domain, this is not sufficient because it does not guarantee that the signal of the original variable is correctly reconstructed from the measurements (samplings) performed by the components. A condition should also be imposed over the time domain, in terms of the rate of measurements performed by a component.

For continuous variables, an appropriate condition can be defined as a ratio of the highest frequency of the signal of the physical variable, or the Nyquist frequency (f_N), being measured. For instance, we could apply the sampling theorem, also known as the Shannon theorem, which states that a sampled variable may be reconstructed from its samples only if the frequency of the variable is lower than half of the sampling frequency. However, in practice, the Shannon theorem does not give a condition which enables us to maintain the consistency of two values representing the same measured physical variable; it is rather difficult to formulate a uniform method for the selection of an appropriate measuring rate. For instance, there are some applications which require a measuring (sampling) rate ten times the highest frequency of the physical variable. In terms of the model of the distributed system under consideration, the frequency of ticks (the inverse of the granularity) of the synchronised time grid should be at least twice the sampling frequency (f_S) of the signal of the continuous variable being measured.

For discrete variables, a condition on the time domain to achieve mutual consistency between two values can be formulated as a dual of the condition for continuous variables. The condition can be defined in terms of the smallest time period in which the discrete variable being measured is stable at a value. To allow the sampled variable to be

reconstructed from its samples, the period between samplings should be smaller than half of the stable time period of the discrete variable. Stating this in terms of the model of the distributed system under consideration, the granularity of the synchronised time grid should be at least half of the sampling period (t_S) of the signal of the discrete variable being measured.

From the above analysis, for both continuous and discrete variables, we deduce that the time uncertainty, recalling equation (*), may be expressed as:

$$\tau < g < 1 / (2 f_S).$$

The granularity of the synchronised time grid must be at least half the rate of the sampling interval, because we can only deduce the order of two events if their occurrence is at least two ticks apart.

3.3.5. Continuous and Discrete Variables

Here we consider separately the mutual consistency of continuous and discrete variables. The separation is necessary because of the different dynamic characteristics of both types of variables. For continuous variables the change in value is bounded by a certain rate which is related to the Nyquist frequency of the physical variable, whereas for discrete variables the change in value is (ideally) instantaneous. For the purpose of checking for consistency, these differences impose two distinct approaches: in the former, we are concerned with approximate equalities with bounded errors, and in the latter, with strict equalities /Kopetz 90/. In the following we analyse both cases, giving conditions which enable consistency to be checked; these conditions are obtained from equation (*).

For continuous variables, any two measured values representing the same physical variable are mutually consistent ($\approx$) at the *ith* tick if and only if both measures, realised within the time uncertainty τ, are within the value uncertainty v. This may be expressed as follows:

$$V_a(t_a(i)) \approx V_b(t_b(i)) \text{ iff } |r(t_a(i)) - r(t_b(i))| \leq \tau \wedge |V_a(t_a(i)) - V_b(t_b(i))| \leq \nu.$$

Remark: For continuous variables, if the frequency of the synchronised clock is much greater than the sampling frequency, the value uncertainty due to time uncertainty ceases to be significant.

For discrete variables, any two measured values representing the same physical variable are mutually consistent ($\approx$) at the *ith* tick if and only if both measures, realised within the time uncertainty τ, are equal. This may be expressed as follows:

$$V_a(t_a(i)) \approx V_b(t_b(i)) \text{ iff } |r(t_a(i)) - r(t_b(i))| \leq \tau \wedge V_a(t_a(i)) = V_b(t_b(i)).$$

The time uncertainty parameter (τ), in the case of discrete variables, assumes a slightly different meaning; instead of being related to the highest frequency of the physical variable, it is related to the smallest time period in which the discrete variable is stable. This arises from the fact that an instantaneous change in state implies a whole range of frequencies up to infinity, which means that no measuring rate that satisfies the Shannon theorem can be found.

Concluding, to achieve mutual consistency between the values of the two components, for continuous variables we have to establish the rate of measurement performed by a component as a ratio of the highest frequency of the physical variable being measured, and for discrete variables we have to establish the rate of measurement performed by a component in the smallest time period in which the discrete variable is stable. For instance, in the case of continuous variables, if we apply the ratio proposed by Shannon, then the frequency of ticks (the inverse of the granularity) of the synchronised time grid should be at least four times the highest frequency of the signal of the physical variable being measured.

3.4. Concluding Remarks

In the first part of this Chapter, we have presented a general model of time, in which a point–based structure was adopted to represent the passage of time. A time structure

was defined in terms of a set of logical conditions, which enable us to define dense and discrete time structures. These time structures provide a basis for a framework for analysing timeliness requirements to be discussed on following two chapters. The result of the study presented in this Chapter provided the basis from which we could argue that the choice of a time structure should rely essentially on the subject and the type of analysis to be conducted, rather than on the technique that has to be employed to conduct the analysis. Although this seems a trivial conclusion to arrive, there are still technical works which suggest the technique to be employed, without providing the choice of the time structure to be used for the analysis.

In the second part of this chapter, we presented the conditions to be imposed on a system in order to avoid value inconsistencies in the system variables, due to time uncertainties that exist in the system /de Lemos 91/. These conditions depend on whether the system variables are continuous or discrete.

Chapter 4

Behaviour Description in Critical Real–Time Systems

4.1. Introduction

The aim of this chapter is to define a model, based on simple concepts, which enables us to describe the behaviour of critical real–time systems. The model should be sufficiently general in order to allow the behavioural description not only of the software functions, but also of the activities performed at other levels of abstraction of the system.

The model proposed here was originally discussed within the context of an analysis of timeliness requirements /de Lemos 92a/, and was intended to provide the means by which informal analysis could be performed (for example, by depicting the timing constraints in terms of graphics), before starting to formalise system requirements. This would enable a more accurate analysis to be performed, from which we could obtain specifications which were unambiguous and consistent. In this chapter we extend that initial work by elaborating the basic model – leading to the event/action model (E/A model) – and by proposing a formal notation for the concepts developed for the basic model – the predicate event/action notation (PEA notation).

The purpose of our work is not merely to propose a new formalism for describing the type of systems that we are concerned with; instead, we aim to define a set of basic concepts which can be incorporated by other formalisms. The choice of the type the formalism to be employed, apart from other possible factors, will depend essentially on the type of application and the analysis to be performed. This approach is consistent with with the broad notions introduced in the first chapter, concerning the utilization of formal methods in the requirements analysis of critical real–time systems.

At this stage of our work, although we are advocating the utilization of more than one formal method to perform the formal analysis of the requirements, we are not concerned with how the different formalisms should be (formally) linked. Instead, as already mentioned, our major objective is to characterise, for the type of system with which we are concerned, the concepts that are most appropriate adequate for the description of system behaviour at the different levels of analysis, and which can be employed across a range of different formalisms. The model proposed to describe the behaviour of systems relies on events, actions, states and their associated time uncertainties, and is based on similar models presented in the literature /Heninger 80, Jahanian 86/; however, it contains some considerable differences which will be discussed at the end of this chapter. In order to show the flexibility of the concepts adopted in our model we show how they can be incorporated in logic formalisms such as Timed History Logic (THL) /Saeed 90/, and net formalisms such as Predicate Transition nets (PrT nets) /Genrich 81/.

The contents of this chapter are organized as follows. In the following two sections we present, respectively, the E/A model and the PEA notation. In section 4, the E/A model is formalised in terms of two different formalisms, THL and PrT nets, with the aim of showing the flexibility of its concepts. In section 5, in order to clarify some concepts introduced in the previous sections we apply those concepts to an example based on a simplified nuclear reactor control system. Section 6 compares the PEA notation with some related models. Finally, the last section presents concluding remarks for this chapter.

4.2. An Event/Action Model (E/A Model)

In order to describe the behaviour of real-time safety-critical systems, which exhibit both continuous and discrete behaviours, we introduce the *event/action model* (E/A model). The E/A model provides a set of primitive concepts which enable system behaviour to be modelled in terms of system predicates (predicates over system variables). In the approach taken, those variables which are continuous have their

behaviour discretised according to imposed thresholds and the discontinuities that they are subject to.

The E/A model is based on primitive concepts such as events, actions and states, and the concept of a time structure (or timeline). The *state* of a system is the information that, together with the system input, determines the behaviour of the system. A *transition* represents a transformation in the system state. The system state is modified by the occurrence of events and the execution of actions. An *event* is a temporal marker of no duration which causes or marks a transition. (Instead of the broad definition given in the previous chapter for an event, here, an event has a more restrictive intepretation: it marks a point in time which is of significance in describing an activity.) An *action* is the basic unit of activity which implies duration. The *duration* of an interval is the distance in time between the two events that define the interval. (An *interval*, as already defined, is a time stretch between two time points.) Apart from events, actions and states which describe the behaviour of process control systems, the E/A model also takes into account the timing uncertainties associated with them.

The motivation for selecting these primitive concepts is twofold: they have been used as primitives in several real–time specification languages /Henninger 80, Jahanian 86/, and they have meaningful interpretations at different levels of abstraction. These concepts provide flexibility, enabling descriptions to be given of system behaviour ranging from the activities of the physical entities of the plant to the temporal ordering of the computational tasks of the control system. The key features of the E/A model are:

- the primitive concepts can be expressed in different classes of formalisms,
- both discrete and dense time structures are supported, and
- timing constraints can be depicted graphically.

4.2.1. Definition of the E/A Model

The E/A model is defined in terms of an extended first order logic that includes variables and predicates that are interpreted as functions from a time domain. The predicates of

the extended first order logic are also referred to as state predicates and those predicates that are used to specify a transition (i.e. mark a time point) referred to as transition predicates. In the following we present the syntax and semantics of the E/A model.

4.2.1.1. Syntax of the E/A Model

The underlying model consists of a time independent and a time dependent part. The time independent part consists of a set of variable names, a set of predicate names and a set of function names. Each variable name denotes a value from an associated type, each predicate name a predicate with a fixed arity n and associated domain, and each function name a function with a fixed arity n and the associated domain and range. The time dependent part consists of a set of time dependent variables and a set of time dependent predicates. Each variable name denotes a function from the time domain to a type, each predicate name denotes a function from the time domain to a predicate. The passage of time (time domain) is represented by a time structure $\boldsymbol{T}$ which is isomorphic to the non−negative reals or a subset of them (e.g. natural numbers).

For the the above model, a term is either a variable or a n−ary function evaluated over n terms, and a predicate is defined inductively as:

1. A well−defined expression over n terms using the relational operators (e.g. $<$ and $\leq$), a time independent predicate or a time dependent predicate is an atomic predicate.

2. If p and q are predicates, so are $\neg p$ and $p \wedge q$.

3. If p is a predicate and x is a time independent variable of type V_x, $\forall x \in V_x$: p is a predicate.

All time dependent variables and predicates are piecewise continuous (i.e. there are a finite number of discontinuities and limits exist from both the left and right at each discontinuity). Predicates satisfy the *finite variability* property, since a predicate can only change at a discontinuity.

We associate *instance number* to a predicate to specify the number of times for which the predicate has been true. Instance numbers are a convenient way for describing the behaviour of discrete variables, or continuous variables that have been discretised, because references to the past and future behaviours of predicates are allowed. In order to express the instance number of a state predicate we introduce the following notation $sp(t)^i$, where $sp(t)$ is a state predicate and the superscript index i is the instance number.

A state predicate holds true for the first time at a particular time point t (the first instance of a state predicate) when it never holds true before a time point t_1 and holds true since then until time point t (inclusive). The first instance of a state predicate is defined as follows:

$$\forall t \in T: [sp(t)^1 \Leftrightarrow sp(t) \wedge (\exists t_1 \in T: t_1 \leq t \wedge (\forall t_2 \in T: t_2 < t_1 \Rightarrow \neg sp(t_2)) \wedge (\forall t_3 \in T: t_1 \leq t_3 \leq t \Rightarrow sp(t_3)))].$$

A state predicate holds true for the ith time at a particular time point t (the ith instance of a state predicate) when it holds true for the $(i-1)$th time between t_1 and t_2, it holds false between t_2 and t_3, and it holds true between t_3 and time point t (inclusive). The ith instance of a state predicate, represented by the superscript index i ($i > 1$), is defined as follows:

$$\forall i \in I^+: i > 1 \Rightarrow \forall t \in T: [\, sp(t)^i \Leftrightarrow sp(t) \wedge \exists t_1, t_2, t_3 \in T: t_1 < t_2 < t_3 < t \wedge \forall t_4 \in T: t_1 \leq t_4 < t_2 \Rightarrow sp(t_4)^{i-1} \wedge \forall t_5 \in T: t_2 \leq t_5 < t_3 \Rightarrow \neg sp(t_5) \wedge \forall t_6 \in T: t_3 \leq t_6 \leq t \Rightarrow sp(t_6)].$$

In order to capture a *transition predicate* we introduce the bar operator "|". Figure 4.1 depicts four diagrams that represent the possible transition predicates that can be obtained from a state predicate when the predicate is either true or false for the first or

last time. Clockwise direction, starting from the diagram on the top left corner, the four cases are: true for the first time, false for the last time, true for the last time and false for the first time. In the diagrams the following conventions were adopted: full circle represents a predicate that is true at time t, and the empty circle represents a predicate that is false at time t.

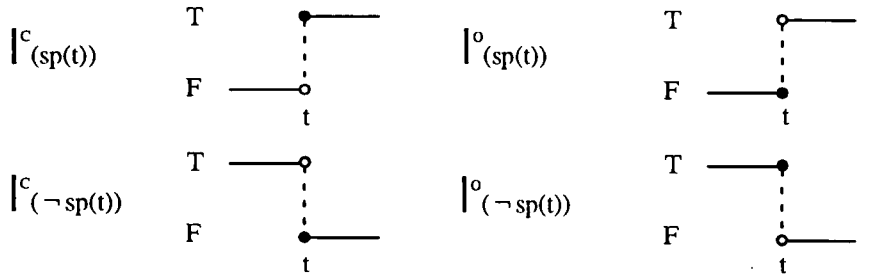


Figure 4.1. Transitions over the state predicates sp(t) and ¬sp(t).

By applying the operator "$|^c$" to a state predicate we capture the first point in time at which a state predicate becomes true (or false). This transition, known as a *closed transition*, is defined as follows:

$$\forall t \in T: [|^c(sp(t)) \Leftrightarrow sp(t) \wedge$$
$$(\exists \delta \in T: \forall t_1 \in T: t-\delta < t_1 < t \Rightarrow \neg sp(t_1) \wedge$$
$$\forall t_2 \in T: t < t_2 < t+\delta \Rightarrow sp(t_2))].$$

The value of δ is defined between the time separation (distance) between two consecutive observations (or samplings) of the value of the variables, used to construct the predicate. For continuous variables, δ has to be defined as a factor of the highest frequency of the variable (or the Nyquist frequency), and for discrete variables, δ has to be defined in terms of the smallest time period for which the variable remains stable /de Lemos 91/.

By applying the operator "$|^o$" to a state predicate we capture the last point in time just before a predicate becomes true (or false). This transition, known as an *open transition*, is defined as follows:

$$\forall t \in T: [\mid^{o}(sp(t)) \Leftrightarrow \neg sp(t) \wedge$$
$$(\exists \delta \in T: \forall t_1 \in T: t-\delta < t_1 < t \Rightarrow \neg sp(t_1) \wedge$$
$$\forall t_2 \in T: t < t_2 < t+\delta \Rightarrow sp(t_2))].$$

In order to capture the instance number of a transition, we introduce the bar operator with a subscript index i added to it "$\mid_i$". The first instance of a closed transition is defined as follows:

$$\forall t \in T: [\mid^{c}_{1}(sp(t)) \Leftrightarrow \mid^{c}(sp(t)) \wedge (\forall t_1 \in T: t_1 < t \Rightarrow \neg \mid^{c}(sp(t_1))].$$

The ith instance of a closed transition, represented by the index i ($i > 1$), is defined as follows:

$$\forall t \in T: \forall i \in I^{+}: [i > 1 \Rightarrow \mid^{c}_{i}(sp(t)) \Leftrightarrow \mid^{c}(sp(t)) \wedge$$
$$(\exists t_1 \in T: t_1 < t \wedge \mid^{c}_{i-1}(sp(t_1)) \wedge (\forall t_2 \in T: t_1 < t_2 < t \Rightarrow \neg \mid^{c}(sp(t_2))))].$$

The definition of the bar operator for the open transition is similar to the one for the closed transition. In the sequel, unless otherwise mentioned, a bar operator without a superscript will refer to a closed transition. For example, the transition $\mid(sp(t))$ captures the point in time when $sp(t)$ becomes true for the first time.

4.2.1.2. Semantics of the E/A Model

For the time independent part, the valuation functions $\mathcal{V}$, $\mathcal{P}$ and $\mathcal{F}$ are respectively defined for the variables, predicates and functions in the traditional way as for a first-order logic. For the time dependent part the valuation functions $\mathcal{V}_{\mathcal{T}}$ and $\mathcal{P}_{\mathcal{T}}$ are defined for the variables and predicates as follows:

for a variable v_i with the range V_{v_i}, $\mathcal{V}_{\mathcal{T}}(v_i)$: $T \rightarrow V_{v_i}$

for a predicate p_i with range V_j for term tr_j, $\mathcal{P}_{\mathcal{T}}(p_i(tr_1, ..., tr_n))$: $T \rightarrow V_1 \times ...$
$\times V_n \rightarrow B$.

The interpretation $\mathcal{I}$ for the underlying model is defined by the structure $(\mathcal{V}, \mathcal{P}, \mathcal{F}, \mathcal{V}_{\mathcal{T}}, \mathcal{P}_{\mathcal{T}})$, and is defined pointwise.

The interpretation of a term *tr* is defined as follows:

$\mathcal{I}(tr)(t) = \mathcal{V}(tr)$, if *tr* is a time independent variable

$\mathcal{I}(tr)(t) = \mathcal{V}_{\mathcal{T}}(tr)(t)$, if *tr* is a time dependent variable

$\mathcal{I}(tr)(t) = \mathcal{F}(f_i)(\mathcal{I}(tr_1)(t), ..., \mathcal{I}(tr_n)(t))$, if *tr* is a function $f_i(tr_1, ..., tr_n)$.

The interpretation of the basic predicates is defined as follows:

$\mathcal{I}(p(tr_1, ..., tr_n))(t) = \mathcal{P}(p)(\mathcal{I}(tr_1)(t), ..., \mathcal{I}(tr_n)(t))$, if *p* is a time independent predicate

$\mathcal{I}(p(tr_1, ..., tr_n))(t) = \mathcal{P}_{\mathcal{T}}(p)(t)(\mathcal{I}(tr_1)(t), ..., \mathcal{I}(tr_n)(t))$, if *p* is a time dependent predicate.

For a given interpretation $\mathcal{I}$, each predicate *p* is evaluated by the function $\mathcal{I}(p)$: $T \rightarrow B$.

$\mathcal{I}(\neg p)(t)$=true iff $\mathcal{I}(p)(t)$=false

$\mathcal{I}(p \wedge q)(t)$=true iff $\mathcal{I}(p)(t)$=true and $\mathcal{I}(q)(t)$=true

$\mathcal{I}(\forall x \in V_x: p)(t)$=true iff $\mathcal{I}'(p)(t)$=true

for every $\mathcal{I}'$ that has the same valuation functions as $\mathcal{I}$ except for $\mathcal{V}'(y)$ which is the same as $\mathcal{V}(y)$ for every time independent variable not equal to *x*.

4.2.2. Primitive Functions of the E/A Model

The primitive concepts of the E/A model are related to the timeline by two types of primitive function: point and interval. A *point function* is a temporal marker of no duration, represented as a cut in the timeline, which models events. An *interval function* denotes a duration, represented as a contiguous section of the timeline, which models states and actions. The timing uncertainties associated with point and interval functions can be represented in terms of a *utility function*. The primitive functions have time and instance number as parameters.

In the definition of the primitive functions, conditions (state and transition predicates) capture the value domain properties of the functions, and time points capture the time domain properties of the functions. In order to distinguish the value domain from the time domain definitions, the subscript "V" or "T", respectively, is added to the name of the function. The definitions of the primitive functions, in the value domain, will be made only in terms of closed transitions (although the open transition could have been used instead). As a consequence, the lower and upper boundaries of a time interval will be respectively closed ("[") and open (")"). Other combinations could be obtained by employing a different usage of the transitions.

4.2.2.1. Point Function

A *point function* $E(t,i)$ is a function which maps the timeline (T the set of all of its points) and the number of instances of an event (I^+ the set of nonnegative integers) into a Boolean ($B=\{\text{true, false}\}$).

In the value domain, the definition of the point function is in terms of the ith instance of a condition (state predicate $sp_E(t)$) which is specified by a transition predicate:

$$E_V(t,i)\colon T \times I^+ \rightarrow B$$

$$\forall t \in T\colon \forall i \in I^+\colon [E_V(t,i) \Leftrightarrow \downarrow_i(sp_E(t))].$$

In the time domain, the definition of the point function is in terms of the time point constants t_E^i which mark the ith instance of the event:

$$E_T(t,i)\colon T \times I^+ \rightarrow B$$

$$\forall t \in T\colon \forall i \in I^+\colon [E_T(t,i) \Leftrightarrow t = t_E^i] \qquad t_E^i \in T_E = \{t_E^1, t_E^2, \ldots\}.$$

Instead of the notation (t_E^i), the occurrence function ($@(E,i)$) from RTL could have been employed /Jahanian 86/.

The value and time domain definitions of a point function are alternative forms for describing the occurrence of the same event:

$$\forall t \in T: \forall i \in I^{+}: [E_V(t,i) \Leftrightarrow E_T(t,i)].$$

A relative time representation of the timeliness requirements of a sequence of events can impose one of two basic types of timing constraint /Dasarathy 85/:

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

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

A third basic timing constraint presented in /Dasarathy 85/ is the *durational* timing constraint which states that an activity must occur for t amount of time. However, an activity which has duration can be represented by two events which mark the begining and end of the activity (i.e. an interval function which the topic of the next section). Thus the durational timing constraint could alternatively be stated as exactly t time units must elapse between two events, which is just an application of the other two basic timing constraints.

In a similar way as is presented for RTL /Jahanian 86/, we state below two *monotonicity* properties which are associated with the point function:

uniqueness property – at most one time point can be associated with each occurrence of an event, i.e. the same instance number of an event cannot happen at two distinct time points:

$$\forall t, t' \in T: \forall i \in I^{+}: [E(t,i) \wedge E(t',i) \Rightarrow t = t'].$$

ordering property – if the ith occurrence of an event happens, then the previous occurrences of same event must have happened earlier, hence two distinct occurrences of the same event must happen at different time points:

$$\forall t, t' \in T: \forall i, j \in I^{+}: [E(t,i) \wedge E(t',j) \wedge i < j \Rightarrow t < t'].$$

For discrete timelines, in order to observe the occurrence of all instances of any particular event in the 'real–world' (assuming to be a dense timeline), the granularity of the discrete timeline should be smaller than the smallest time interval between the occurrence of any two events in the 'real–world'. If any event occurs more than once between two ticks of the discrete timeline, only one of the occurrences will be observed.

4.2.2.2. Interval Function

An *interval function* $A(t,i)$ is a function which maps the timeline and the number of instances of an action (or a state) into a Boolean. (In the following, the definition of the interval function will be restricted to the execution of an action, however, it could be extended to represent the time interval in which a system state holds true (or false).) An action is manifested in the system by its associated events: the *start event* that marks the initiation of an action, and the *finish event* that marks the completion of an action.

In the value domain, the interval function is defined in terms of the transition predicates corresponding to the start ($sp_{\uparrow A}(t)$) and finish ($sp_{\downarrow A}(t)$) conditions, and the state predicate corresponding to the invariant condition ($sp_{INV}(t)$). This is represented as follows:

$$A_V(t)\colon \boldsymbol{T} \rightarrow \boldsymbol{B}$$

$$\forall t \in T\colon [A_V(t) \Leftrightarrow \exists t_1, t_2 \in T\colon t_1 \leq t < t_2 \wedge (\mid\!(sp_{\uparrow A}(t_1)) \wedge \mid\!(sp_{\downarrow A}(t_2)) \wedge$$

$$\forall t_3 \in T\colon t_1 \leq t_3 < t_2 \Rightarrow sp_{INV}(t_3) \wedge (\neg\, sp_{\downarrow A}(t_3))].$$

The first instance of an interval function is defined as follows:

$$A_V(t,1)\colon \boldsymbol{T} \times \boldsymbol{I}^+ \rightarrow \boldsymbol{B}$$

$$\forall t \in T\colon [A_V(t,1) \Leftrightarrow A_V(t) \wedge (\exists t_1 \in T\colon t_1 \leq t) \wedge$$

$$(\forall t_2 \in T\colon t_2 < t_1 \Rightarrow \neg A_V(t_2)) \wedge (\forall t_3 \in T\colon t_1 \leq t_3 \leq t \Rightarrow A_V(t_3))].$$

The ith instance of an interval function is defined as follows:

$$A_V(t,i): T \times I^+ \rightarrow B$$

$$\forall i \in I^+: i > 1 \Rightarrow$$

$$\forall t \in T: [A_V(t,i) \Leftrightarrow A_V(t) \wedge \exists t_1, t_2, t_3 \in T: t_1 < t_2 < t_3 < t \wedge$$

$$\forall t_4 \in T: t_1 \leq t_4 < t_2 \Rightarrow A_V(t_4 i-1) \wedge \forall t_5 \in T: t_2 \leq t_5 < t_3 \Rightarrow \neg A_V(t_5) \wedge$$

$$\forall t_6 \in T: t_3 \leq t_6 \leq t \Rightarrow A_V(t_6)].$$

In the value domain, the start and finish events are defined, respectively, as the *start condition* and the *finish condition*. The start condition for an action A defines the condition that triggers the execution of an action; it is denoted by $\uparrow A_V(t,i)$ and defined as follows:

$$\uparrow A_V(t,i): T \times I^+ \rightarrow B$$

$$\forall t \in T: [\uparrow A_V(t,i) \Leftrightarrow \mid (A_V(t,i))].$$

The finish condition defines the condition in which an action A terminates its execution; it is denoted by $\downarrow A_V(t,i)$ and defined as follows:

$$\downarrow A_V(t,i): T \times I^+ \rightarrow B$$

$$\forall t \in T: [\downarrow A_V(t,i) \Leftrightarrow \mid (\neg A_V(t,i))].$$

In the time domain, the start and finish events are defined as temporal markers, respectively, the *start time* and the *finish time*. The definition of the start and finish times follows directly from the time domain definition of the point function. They are defined in terms of time point constants of the form $t^i_{\uparrow A}$ and $t^i_{\downarrow A}$ at which the respective events $\uparrow A_T(t,i)$ and $\downarrow A_T(t,i)$ have occurred for the ith time.

In the time domain, the interval function $A_T(t,i)$ is defined, as follows, in terms of the start and finish times of an action:

$$A_T(t,i): T \times I^+ \rightarrow B$$

$$\forall t \in T: \forall i \in I^+: [A_T(t,i) \Leftrightarrow t^i_{\uparrow A} \leq t < t^i_{\downarrow A}].$$

The *start time* refers to the time point at which the start event of an action occurs:

$$\uparrow A_T(t,i)\colon T \times I^+ \rightarrow B$$

$$\forall t \in T\colon \forall i \in I^+\colon [\uparrow A_T(t,i) \Leftrightarrow t = t^i_{\uparrow A}] \qquad t^i_{\uparrow A} \in T_{\uparrow A} = \{t^1_{\uparrow A}, t^2_{\uparrow A}, \ldots\}.$$

The *finish time* refers to the time point at which the finish event of an action occurs:

$$\downarrow A_T(t,i)\colon T \times I^+ \rightarrow B$$

$$\forall t \in T\colon \forall i \in I^+\colon [\downarrow A_T(t,i) \Leftrightarrow t = t^i_{\downarrow A}] \qquad t^i_{\downarrow A} \in T_{\downarrow A} = \{t^1_{\downarrow A}, t^2_{\downarrow A}, \ldots\}.$$

Similar to the point function definition, the value and time domain definitions of an interval function are alternative forms for describing the execution of the same action:

$$\forall t \in T\colon \forall i \in I^+\colon [A_V(t,i) \Leftrightarrow A_T(t,i)].$$

The *execution time* of an action is the duration of the interval during which the action is executed.

We now state two properties that are associated with the interval function:

duration property – the finish event of an action cannot precede the start event of the action:

$$\forall t, t' \in T\colon \forall i \in I^+\colon [\downarrow A(t,i) \Rightarrow \uparrow A(t',i) \wedge t > t'].$$

If the start and finish events occur at the same time point, implying a durationless action, then the action is modelled by a point function.

ordering property – two instances of the same action cannot be executed at the same time:

$$\forall t, t' \in T\colon \forall i, j \in I^+\colon [\downarrow A(t,i) \wedge \uparrow A(t',j) \wedge i < j \Rightarrow t < t'].$$

4.2.2.3. Utility Function

The specification of time uncertainties for the two functions defined above can be represented by means of a *utility function* $U(t,i)$, or "value" function (in the sense of monetary value) /Jensen 85/. We are concerned with the class of utility functions which

are typically applicable to critical real–time systems – *discrete* or *critical utility functions*. In these systems the utility, or usefulness, can only assume two values, either maximal or minimal – hence $U(t,i)$ is a Boolean function.

The utility function $U_E(t,i)$ associated with the point function $E(t,i)$ which models the occurrence of an event, maps the timeline and the instance number of an event into the usefulness of its occurrence. The maximal usefulness, in time domain, in the occurrence of an event is established by the time interval defined by the following two times:

earliest occurring time (*eot*) – the first point in time at or after which an event can occur;

latest occurring time (*lot*) – the last point in time before which the event can occur.

If an event occurs either its usefulness is maximal if it occurs during the interval of time defined by these two time points, or minimal if the event occurs outside the time interval. A utility function representing the usefulness of the occurrence of an event can be defined as follows:

$$U_E(t,i): T \times I^+ \rightarrow B$$

$$\forall t \in T: \forall i \in I^+: [U_E(t,i) \Leftrightarrow (t^i_{E_{eot}} \leq t < t^i_{E_{lot}})]$$

$$t^i_{E_{eot}} \in T_{E_{eot}} = \{t^1_{E_{eot}}, t^2_{E_{eot}}, \ldots\}$$

$$t^i_{E_{lot}} \in T_{E_{lot}} = \{t^1_{E_{lot}}, t^2_{E_{lot}}, \ldots\}.$$

The utility function $U_A(t,i)$ associated with the interval function, which models the execution of an action, maps the timeline and the instance number of an action into the usefulness of its execution. In order to define the maximal usefulness in the execution of an action, in the time domain, we have to consider the start and finish events of that action and associate utility functions to these events. The time attributes of the utility function associated with the execution of an action are the following:

earliest starting time (*est*) – the first point in time at or after which an action can start its execution;

latest starting time (*lst*) – the last point in time before which an action can start its execution;

earliest finishing time (*eft*) – the first point in time at or after which an action can finish its execution;

latest finishing time (*lft*) – the last point in time before which an action can finish its execution.

The time attributes *est* and *lft* are often referred to as "delay" and "deadline", respectively. When plotting an utility function that represents these time attributes, they are expressed by step utility functions, respectively positive and negative step functions, as it was depicted in figure 4.1.

A utility function representing the usefulness of the execution of an action is defined as follows, in terms of the utility functions of the start and finish events of the action:

$$U_{\uparrow A}(t',i): T \times I^{+} \rightarrow B$$

$$\forall t' \in T: \forall i \in I^{+}: [U_{\uparrow A}(t',i) \Leftrightarrow (t^{i}_{A_{est}} \leq t' < t^{i}_{A_{lst}})]$$

$$t^{i}_{A_{est}} \in T_{A_{est}} = \{t^{1}_{A_{est}}, t^{2}_{A_{est}}, \ldots\}$$

$$t^{i}_{A_{lst}} \in T_{A_{lst}} = \{t^{1}_{A_{lst}}, t^{2}_{A_{lst}}, \ldots\};$$

$$U_{\downarrow A}(t'',i): T \times I^{+} \rightarrow B$$

$$\forall t'' \in T: \forall i \in I^{+}: [U_{\downarrow A}(t'',i) \Leftrightarrow (t^{i}_{A_{eft}} \leq t'' < t^{i}_{A_{lft}})]$$

$$t^{i}_{A_{eft}} \in T_{A_{eft}} = \{t^{1}_{A_{eft}}, t^{2}_{A_{eft}}, \ldots\}$$

$$t^{i}_{A_{lft}} \in T_{A_{lft}} = \{t^{1}_{A_{lft}}, t^{2}_{A_{lft}}, \ldots\};$$

$$U_{A}(t,i): T \times I^{+} \rightarrow B$$

$$\forall t, t', t'' \in T: \forall i \in I^{+}: [U_{A}(t,i) \Leftrightarrow U_{\uparrow A}(t',i) \wedge U_{\downarrow A}(t'',i)].$$

In the following, we present an example, illustrated in figure 4.2, which shows how the functions defined above are interrelated in the specification of timeliness requirements associated with the execution of an action. Taking as a reference point the occurrence of an event at time t_E^1, depicted by function $E(t,i)$, and knowing beforehand the earliest time (*min*) and the latest time (*max*) for an action, of duration *dur*, to start its execution, we are able to construct the utility function $U_A(t,i)$ associated with the execution of the action. This utility function represents the time uncertainties associated with the execution of the action which were derived from the given basic timing relations. Finally, one of the possible executions of the action is represented by the function $A(t,i)$, which is executed within the timing constraints imposed by the utility function.

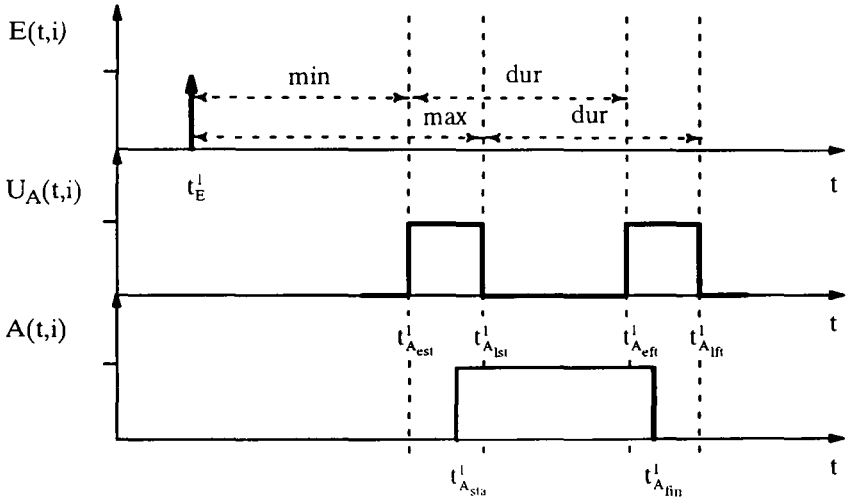


Figure 4.2. E/A model in the specification of timeliness requirements.

If two or more utility functions represent either the occurrence of an event or the execution of an action, then the resulting utility function should embody the timing constraints of all of the utility functions. Below we present a lemma which allows us to define a utility function when there exists more than one utility function associated with an action. (The lemma can also be applied for the occurrence of events.)

Lemma 4.1

When there exists more than one utility function representing the usefulness of executing the same action, the resulting utility function reflects the following timing constraints on the start time and finish time of an action:

$$t_{A_{sta}} \in \{t \mid max(t^1_{A_{est}}, ..., t^n_{A_{est}}) \leq t \leq min(t^1_{A_{lst}}, ..., t^n_{A_{lst}})\};$$

$$t_{A_{fin}} \in \{t \mid max(t^1_{A_{eft}}, ..., t^n_{A_{eft}}) \leq t \leq min(t^1_{A_{lft}}, ..., t^n_{A_{lft}})\}.$$

Sketch of the Proof: If the start and finish events of the action to be executed have the respective time occurrences of $t_{A_{sta}}$ and $t_{A_{fin}}$, then for these two events we associate their respective utility functions that correspond to the various utility functions representing the usefulness of executing the action. If we consider, for instance, only $t_{A_{sta}}$ (the analysis for $t_{A_{fin}}$ is identical) then the time attributes of the resulting utility function should not violate of the time attributes of any of the utility functions. For this condition to be satisfied the resulting utility function must have its $t_{A_{est}}$ the maximum of all the $t_{A_{est}}$'s of the utility functions, and its $t_{A_{lst}}$ the minimum of all the $t_{A_{lst}}$'s of the utility functions.

An example of having more than one utility function representing the usefulness in the execution of an action, is the situation where the same exception handler is used to treat two or more exceptions which impose different timing constraints upon the execution of the handler. In practical terms this could represent the occurrence of two different alarms which are treated by the same action.

4.2.3. Event/Action Model and Time Structures

One of the characteristics of the E/A model is that it can support both dense and discrete time structures. However, depending on the time structure there are some dissimilarities on how the occurrence of an event is observed. We say that an event is observed when a point on the timeline is associated with its occurrence.

In dense time structures the occurrence and observation of an event is always simultaneous. Thus, for distinct events we associate distinct time points, except for the case where events occur simultaneously. In discrete time structures the occurrence and observation of an event is not simultaneous. In discrete time structures, once an event occurs, it is not possible to observe an event itself, only its consequence(s) may be

observed. The observation instants are restricted to the time points of the timeline. Hence an event cannot be observed between any two time points on the timeline; the time point we associate with the observation of an event might not be coincident with the actual occurrence of the event.

Depending on the granularity of a discrete time structure, an action might be represented either by a point function or an interval function. If we consider, for example, a time structure $\mathcal{T}_2$ which contains another time structure $\mathcal{T}_1$, as shown in figure 4.3, then an action $A(t,1)$ when measured in $\mathcal{T}_2$ has a duration of less than $\Delta(\mathcal{T}_1)$, but when measured in $\mathcal{T}_1$ it becomes durationless. However, for another instance of the same action – $A(t,2)$, we can have a different situation, as shown in figure 4.3. In summary, an action of fixed duration in one time structure ($\mathcal{T}_2$) may have different duration in another time structure ($\mathcal{T}_1$) with a coarser granularity – $\Delta(\mathcal{T}_1)>\Delta(\mathcal{T}_2)$, depending on the time points associated with the respective starting and finishing events of the action. (In order to maintain compatibility with the closed/open interval being employed, we have opted to "observe" the occurrence of an event at the time point before its occurrence. The reason for this is that in order to establish an closed/open interval, the left point of the interval should be observable. Whether the observation of an event is at time point before or after the actual occurrence of the event is irrelevant because in both cases the same inevitable imprecisions are introduced.)

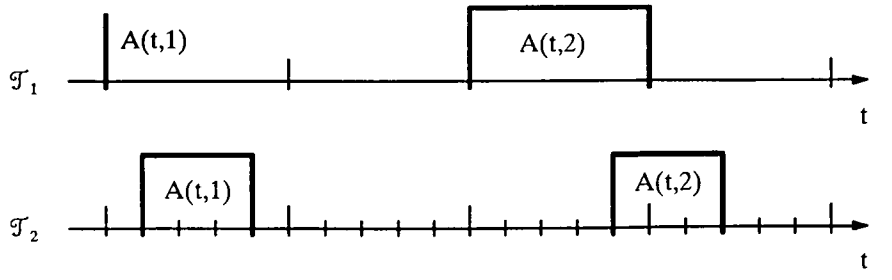


Figure 4.3. E/A model and the granularity of discrete time structures.

Apart from the above mentioned time uncertainties which are inherent in the utilization of discrete time structures with different time granularities, there are those time uncertainties related to the refinement of the timeline to which a specification of a timing

constraint was originally made /Corsetti 91/. These time uncertainties can easily lead to the following type of ambiguity. If we state that a predicate holds at time t_1 of a coarser timeline T_1, then we assume that the same predicate also holds in a subset of the interval corresponding to t_1 in a finer timeline T_2; this subset might correspond to the whole interval, or a single instant, or even a scattered sequence of smaller intervals. For example, this can be the case for a specification which merely states that two trains, while in motion, must be kept at least 10 minutes apart from each other; this specification does not say anything about the time uncertainty around the time point which defines the end of the interval of "10 minutes", if, for instance, the timing constraint was to be considered in terms of a microsecond timeline of the controller of the system.

4.3. Predicate Event/Action Notation (PEA Notation)

The concern in the E/A model was to identify a set of primitive concepts and define a set of primitive functions that formalise these concepts. In order to express the primitive concepts of the E/A model concisely and to facilitate formal analysis of the system behaviour, we introduce the Predicate Event/Action notation (PEA notation). In the following, the primitive functions of the E/A model will be defined as primitive predicates of the PEA notation.

4.3.1. Primitive Predicates of the PEA Notation

4.3.1.1. Point Predicate

For the value domain definition of the point predicate the notation "$E(i)$" is introduced to denote the ith instance (i.e. occurrence) of an event E. The point predicate $E(i)$ is true iff there exists a time point t at which the point function $E_V(t,i)$ holds true:

$$\forall i \in I^+: [E(i) \Leftrightarrow \exists t \in T: E_V(t,i)].$$

For the time domain definition of the point predicate the notation "$E(i)@t$" is introduced to denote that the ith instance of an event E occurs at time point t. The point predicate $E(i)@t$ is true iff at the time point t the event E has occurred for the ith time:

$$\forall t \in T: \forall i \in I^+: [E(i)@t \Leftrightarrow E_T(t,i)].$$

From the above definition, an observation that could be made is that the notation $E(i)@t$ is just an alternative for the notation $E_T(t,i)$.

4.3.1.2. Interval Predicate

For the value domain definition of the interval predicate the notation "$A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle$" is introduced to denote the ith execution of action A. The start condition "$\uparrow A$" and finish condition "$\downarrow A$" are transition predicates, and the invariant condition "A_{INV}" is a state predicate. An action is executed between "$\uparrow A$" and "$\downarrow A$" while "A_{INV}" holds true. The conditions associated with an interval predicate are defined as follows:

$$\forall i \in I^+: [\uparrow A(i) \Leftrightarrow \exists t_{\uparrow A} \in T: \uparrow A_V(t_{\uparrow A}, i)],$$

$$\forall i \in I^+: [A_{INV}(i) \Leftrightarrow \exists t \in T: A_V{}^{INV}(t,i)], \text{ and}$$

$$\forall i \in I^+: [\downarrow A(i) \Leftrightarrow \exists t_{\downarrow A} \in T: \downarrow A_V(t_{\downarrow A}, i)].$$

The interval predicate $A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle$ is true iff for all time points between $t_{\uparrow A}$ and $t_{\downarrow A}$ the interval function $A_V(t,i)$ holds true. The interval predicate is defined in the value domain as follows:

$$\forall i \in I^+: [A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \Leftrightarrow$$
$$\exists t_{\uparrow A}, t_{\downarrow A} \in T: \forall t \in T: (t_{\uparrow A} \leq t < t_{\downarrow A} \Leftrightarrow A_V(t,i))].$$

For the time domain definition of the interval predicate the notation "$A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle$" is introduced to denote the ith execution of an action A between the time points $t_{\uparrow A}$ and $t_{\downarrow A}$. These time points represent the times at which the events associated with action A have occurred: the start event at time $t_{\uparrow A}$ and the finish event at time $t_{\downarrow A}$. The two events are respectively defined as follows, in terms of the point function:

$$\forall t_{\uparrow A} \in T: \forall i \in I^+: [\uparrow A(i)@t_{\uparrow A} \Leftrightarrow \uparrow A_T(t_{\uparrow A}, i)], \text{ and}$$

$$\forall t_{\downarrow A} \in T: \forall i \in I^+: [\downarrow A(i)@t_{\downarrow A} \Leftrightarrow \downarrow A_T(t_{\downarrow A}, i)].$$

The interval predicate is true iff for all time points between $t_{\uparrow A}$ and $t_{\downarrow A}$ the interval function $A_T(t,i)$ holds true. The interval predicate is defined in the time domain as follows:

$$\forall t_{\uparrow A}, t_{\downarrow A} \in T: \forall i \in I^+: [A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \Leftrightarrow \forall t \in T: (t_{\uparrow A} \leq t < t_{\downarrow A} \Leftrightarrow A_T(t,i))].$$

4.3.1.3. Utility Predicate

In the following we define the utility predicates that will be employed in reasoning about time uncertainties.

The utility predicate "$U_E(i)@\langle t_{eot}, t_{lot}\rangle$" is introduced to denote the time uncertainty associated with the ith occurrence of event E. The utility predicate is true iff for all time points between t_{eot} and t_{lot} the utility function $U_E(t,i)$ holds true. The utility predicate representing the usefulness of the occurrence of an event is represented as:

$$\forall t_{eot}, t_{lot} \in T: \forall i \in I^+: [U_E(i)@\langle t_{eot}, t_{lot}\rangle \Leftrightarrow$$
$$\forall t \in T: (t_{eot} \leq t < t_{lot} \Leftrightarrow U_E(t,i))].$$

For the utility predicate of an action A, the notation "$U_A(i)@\langle t_{\uparrow A_{est}}, t_{\uparrow A_{lst}}, t_{\downarrow A_{eft}}, t_{\downarrow A_{lft}}\rangle$" is introduced to denote the time uncertainty associated with the ith execution of the action. The utility predicate is true iff for all time points between $t_{\uparrow A_{est}}$ and $t_{\uparrow A_{lst}}$ the utility function $U_{\uparrow A}(t',i)$ holds true, and for all time points between $t_{\downarrow A_{eft}}$ and $t_{\downarrow A_{lft}}$ the utility function $U_{\downarrow A}(t'',i)$ holds true. The utility predicate representing the usefulness in the execution of an action is represented as:

$$\forall t_{\uparrow A_{est}}, t_{\uparrow A_{lst}}, t_{\downarrow A_{eft}}, t_{\downarrow A_{lft}} \in T: \forall i \in I^{+}: [\ U_A(i)@\langle t_{\uparrow A_{est}}, t_{\uparrow A_{lst}}, t_{\downarrow A_{eft}}, t_{\downarrow A_{lft}}\rangle \Leftrightarrow$$

$$\forall t' \in T: (t_{\uparrow A_{est}} \leq t' < t_{\uparrow A_{lst}} \Leftrightarrow U_{\uparrow A}(t',i)) \wedge$$

$$\forall t'' \in T: (t_{\downarrow A_{eft}} \leq t'' < t_{\downarrow A_{lft}} \Leftrightarrow U_{\downarrow A}(t'',i))].$$

4.3.2. Operators of the PEA Notation

In order to compose point and interval predicates of the PEA notation, a set of logical operators are defined in both value and time domains. In the following, we define some of the PEA notation operators in terms of the functions of the E/A model; the full range of operators is defined in Appendix D.

From two standard logical operators, negation ($\neg$) and conjunction ($\wedge$), the other logical operators can be defined, such as disjunction ($\vee$), implication ($\Rightarrow$) and equivalence ($\Leftrightarrow$). As an example of these standard logical operators, we define, in the value domain, the conjunction of two interval predicates as follows:

$$\forall i, j \in I^{+}: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=} \exists t \in T: A_V(t,i) \wedge B_V(t,j).$$

Four additional logical (binary) operators are defined over the interval predicates of the PEA notation: choice ($+$) when either one of the predicates can become true (or false), meet ($\prec$) when a predicate becomes true (or false) at the same time point that other predicate becomes true (or false), overlap ($\|$) when two predicates are true (or false) at the same time point, and disjoint (∇) when two predicates neither meet nor overlap. These operators are very convinient in composing primitive predicates in order to describing the operational behaviour of systems. As an example of these additional logical operators, we define, in both value and time domains, the meet and overlap operators.

The meet operator ($\prec$):

$$\forall i, j \in I^{+}: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \prec B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$

$$A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \wedge \downarrow A(i) \Leftrightarrow \uparrow B(j);$$

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i,j \in I^{+}: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \prec B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$

$$A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \wedge (t_{\downarrow A} = t_{\uparrow B}).$$

The overlap operator ($\|$):

$$\forall i,j \in I^{+}: A(i)\langle \uparrow A, A_{INV}, \downarrow A\rangle \| B(j)\langle \uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$

$$A(i)\langle \uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle \uparrow B, B_{INV}, \downarrow B\rangle \wedge$$

$$\uparrow A(i) \Rightarrow B(j)\langle \uparrow B, B_{INV}, \downarrow B\rangle \vee \uparrow B(j) \Rightarrow A(i)\langle \uparrow A, A_{INV}, \downarrow A\rangle;$$

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i,j \in I^{+}: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \| B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$

$$(A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle) \wedge ((t_{\uparrow B} \leq t_{\uparrow A} < t_{\downarrow B}) \vee (t_{\uparrow A} \leq t_{\uparrow B} < t_{\downarrow A})).$$

Apart from the above two sets of logical operators, we now define an algebraic operator that is only applicable to the time domain; the *duration operator* (or δ*−operator*) measures the time distance between two time points (associated with the occurrence of events). The duration between the occurrence of the events $E(i)@t_E$ and $F(i)@t_F$ is given, as follows, by the duration operator whose arguments are t_E and t_F:

$$\delta_{(t,t)} : T \times T \rightarrow T$$

$$\delta_{(t_E,t_F)} = t_F - t_E$$

The minimum and maximum durations between the occurrence of two events is obtained from their respective utility predicates, by using the following two variants of the duration operator:

$$\delta^{\min}_{(t,t)} : T \times T \rightarrow T \qquad \delta^{\max}_{(t,t)} : T \times T \rightarrow T$$

$$\delta^{\min}_{(t_E,t_F)} = t_{F_{eot}} - t_{E_{tot}} \qquad \delta^{\max}_{(t_E,t_F)} = t_{F_{tot}} - t_{E_{eot}}$$

If the two time points of the duration operator refer to the time of occurrence of the start and finish events of an action, then the notation $\delta_{A(i)}$ can be used instead of $\delta_{(t^i_{\uparrow A}, t^i_{\downarrow A})}$.

4.3.3. Examples

In order to exemplify the concepts and the notation introduced so far we now present two examples based on the timing templates given in /Jahanian 86/.

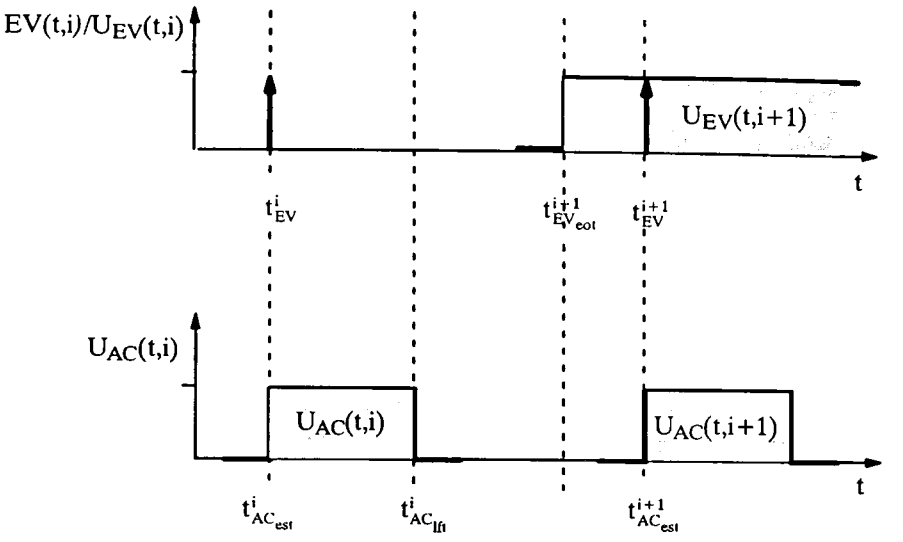
For the first example, let us consider a sporadic timing constraint in which after the occurrence of an event EV, an action AC must be executed with the deadline of 60 ms; between the occurrence of two consecutive events a minimum separation of 100 ms must be imposed. In figure 4.4 we depict this situation in terms of the functions defined for the E/A model. The top diagram of figure 4.4 represents the point function $EV(t,i)$ and the utility function $U_{EV}(t,i)$ for the event EV; they are respectively represented by an uparrow and the shaded area. The time point constants t^{i}_{EV} and t^{i+1}_{EV}, on the time axis, are related to the ith and $(i+1)$th occurrence of the event EV, and $t^{i+1}_{EV_{eol}}$ is the earliest occurrence time for the $(i+1)$th occurrence of event EV. The bottom diagram of figure 4.4 depicts the utility function $U_{AC}(t,i)$ of the action $AC(t,i)$ that has to be executed whenever an event EV occurs. The timing equations below the two diagrams represent the timing relations that have to be satisfied.

This sporadic timing constraint is formalised into the following logic assertions:

$$\forall i \in I^{+}: \forall t_{EV} \in T: [EV(i)@t_{EV} \Rightarrow \quad U_{EV}(i+1)@\langle t_{EV}+100, - \rangle];$$

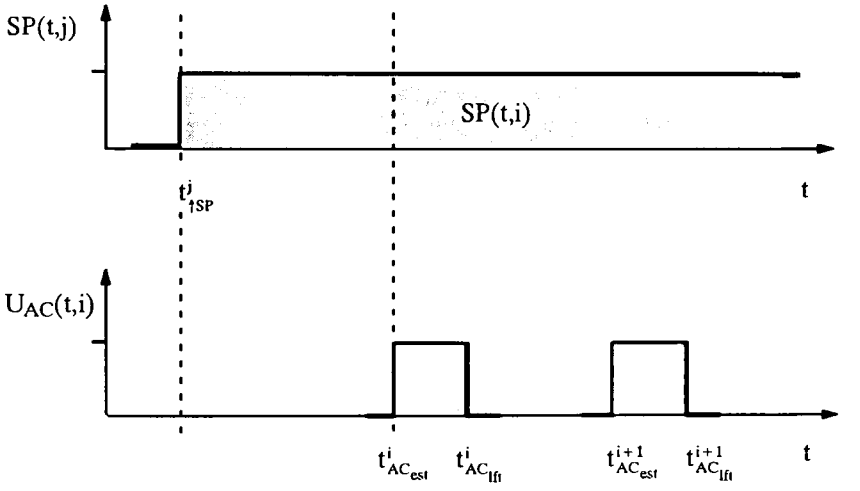
$$\forall i \in I^{+}: \forall t_{EV} \in T: [EV(i)@t_{EV} \Rightarrow U_{AC}(i)@\langle t_{EV}, t_{EV}, -, t_{EV}+60 \rangle];$$

For the second example, let us consider a periodic timing constraint in which while the state predicate SP is true at least for 200 ms, an action AC must be executed periodically every 200 ms with a completion deadline of 100 ms. In figure 4.5 we depict this situation in terms of the functions defined for the E/A model. The top diagram represents the interval function $SP(t,j)$ related to the state SP which becomes true for the jth time at the time point $t^{j}_{\uparrow SP}$. The bottom diagram depicts the utility function $U_{AC}(t,i)$ of the action $AC(t,i)$ that has to be executed periodically.



$$t^{i+1}_{EV_{eot}} = t^i_{EV} + 100$$
$$t^i_{AC_{est}} = t^i_{EV}$$
$$t^i_{AC_{lft}} = t^i_{EV} + 60$$

Figure 4.4. E/A model of a sporadic timing constraint.



$$t^i_{AC_{est}} \geq t^j_{\uparrow SP} + 200 * i$$
$$t^i_{AC_{lft}} < t^j_{\uparrow SP} + 200 * i + 100$$

Figure 4.5. E/A model of a periodic timing constraint.

This periodic timing constraint is formalised into the following logic assertion:

$$\forall i,j \in I^+: \forall t_{\uparrow SP}, t_{\downarrow SP} \in T: [SP(j)@\langle t_{\uparrow SP}, t_{\downarrow SP}\rangle \wedge$$
$$t_{\downarrow SP} > t_{\uparrow SP} + 200*i \Rightarrow U_{AC}(i)@\langle t_{\uparrow SP} + 200*i, -, -, t_{\uparrow SP} + 200*i + 100\rangle].$$

The purpose of these examples was to show that the PEA notation can represent the same timing templates that are used by RTL /Jahanian 86/ in a much compact form which is more adequate for conducting the requirements analysis. The same comparison cannot be made in terms of the value domain because RTL only handles uninterpreted predicates.

4.4. Formalisation of the E/A Model

The E/A model has introduced a set of concepts which are flexible enough in order to be incorporated by different formalisms. In this section, we illustrate how the primitive concepts of the E/A model can be incorporated other into existing formalisms. The general approach adopted is to define the primitive functions in terms of the primitives of the formalisms, and then impose restrictions over these primitives to capture the basic properties of the point and interval functions. We do not aim to provide automatic transformation rules, or formal linking, between the different formalisms; our aim is to show how different primitives of the E/A model can be expressed in other formalisms. This approach to formalisation is applied to the classes of formalisms identified in this paper: descriptive and operational. To do so, we employ, respectively, THL /Saeed 90a/ and PrT nets /Genrich 87/.

4.4.1. E/A Model in THL

In this section we present how the primitive functions of the E/A model can be incorporated into THL /Saeed 90a/. (The basics of THL are described in Appendix A.)

The THL description of an E/A model of a system is obtained by extending the THL state space Γ, in terms of the primitive functions. That is, for a system with q point functions and r interval functions the state vector is extended by the following variables:

$$\langle E_1(t), ..., E_q(t), \uparrow A_1(t), ..., \uparrow A_r(t), \downarrow A_1(t), ..., \downarrow A_r(t), A_1(t), ..., A_r(t) \rangle.$$

The above functions are of the form $E(t)$: $T \rightarrow B \times I^+$. However, to be consistent with the E/A model the following convention is adopted to introduce the parameter "i":

$$\forall H \in \Gamma\mathbf{H}: \forall t \in T: \forall i \in I^{+}: [H.E(t,i) \equiv H.E(t)=(true,i)].$$

Axioms are then introduced into the THL description, to ensure that the *uniqueness* and *ordering* properties of point functions and the *duration* and *ordering* properties of interval functions are satisfied for all well–defined histories. For example, we say that a history H is well–defined for a point function E if and only if:

$$\forall t, t' \in T: \forall i \in I^{+}: [H.E(t,i) \wedge H.E(t',i) \Rightarrow t = t'].$$

The bar operator "$|^{c}$" of the E/A model is defined in THL as the following history relation:

$$\forall t \in T: |^{c}(sp(t)) \Leftrightarrow$$
$$\forall T_1 \in T: (T_1=t \wedge sp(T_1) \wedge \exists t_1 \in T: \forall t_2 \in T: t_1<t_2<T_1 \Rightarrow \neg sp(t_2)).$$

4.4.2. E/A Model in PrT Nets

In the following we present the primitive functions of the E/A model, in terms of Predicate–Transition nets (PrT nets) /Genrich 87/. PrT nets is a form of high–level Petri net which is mainly used for the modelling and analysis of discrete–event systems which are concurrent, asynchronous, and non–deterministic. The use of high–level nets, instead of (Time) Petri nets /Leveson 87/, adds to the modelling power of the latter the formal treatment of *individuals* (i.e. the notion of token identity) and their changing properties and relations. (The basics of PrT nets are described in Appendix B.)

The representation of the E/A model using PrT nets, and the implicit timed firing rule, is straight forward because of the similarity between the firing rules of a transition and the properties associated with the occurrence of events and execution of actions. The timing constraints to be modelled are expressed as relational expressions which must be satisfied for a transition to become enabled. To allow the complete representation of the E/A model in terms of PrT nets, we assume a Weak Time Semantics (WTS) rather than a Strong Time Semantics (STS) for the firing of the transitions which means that a

transition does not have to fire when its maximum firing time has been reached; if it does fire it does so within the time interval specified by the time condition /Ghezzi 91/.

The occurrence of an event in the E/A model (i.e. a cut in the timeline when a condition becomes true) corresponds in PrT nets to the instantaneous time that a transition takes to fire. If an explicit reference to time is made, by considering an event as a temporal marker, then the definition of PrT nets has to be modified in order to incorporate the notion of the passage of time. (To represent the passage of time in PrT nets we adopt the approach described in Appendix B; alternative approaches, such as Time ER nets /Ghezzi 91/, could also have been employed.) From the underlying model of time, when a transition is fired a timestamp is associated with the tuple (or token), representing the time that the tuple was produced. Figure 4.6 shows the concept of an event in terms of PrT nets. When transition *tr* fires the tuple *tup* is removed from predicate P_1 and placed in P_2 with a timestamp $tstp(P_2)$ corresponding to the time that the transition had fired. The variable *tstamp*, whose value is of numerical type, is associated with the labels of the net and represents the timestamp of the tuple. The utility predicate of an event is represented by the timing relation associated with transition *tr*, in other words, once transition *tr* is enabled, it is allowed to fire only within the time interval established by the timing relation associated with the transition.

The execution of an action in the E/A model implies a time interval between the occurrence of the two events associated with the action: the start and finish events. In PrT nets (corresponding to the representation of an event) an action is denoted by two

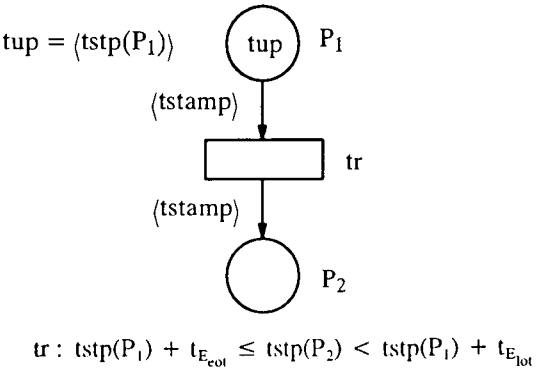
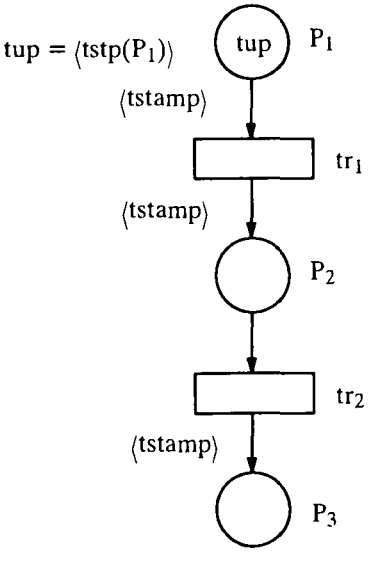


Figure 4.6. The occurrence of an event in PrT nets.

transitions representing the two events, and a predicate representing the execution of the action. The precondition and postcondition of an action are expressed by first order logic formula annotating the transitions. Figure 4.7 shows the concept of an action in terms of PrT nets. When transition tr_1 fires, the variable *tstamp* is updated with the time that the tuple was produced, which corresponds to the start time of the action. While the tuple *tup* is at predicate P_2 this means that the action is being executed, until transition tr_2 fires, corresponding the finish event of an action. The annotations in transitions tr_1 and tr_2 represent the time uncertainties associated, respectively, with the start and finish events of an action, that is, they represent the utility function associated with the execution of an action.

Apart from specifying the normal behaviour of the system in terms of the timing constraints that have to be met, it might also be important to specify the abnormal behaviour of the system for the case when timing constraints are violated /Cristian 89, de Lemos 92c/. For violations of the timing constraints which can be anticipated we specify the exceptional behaviour of the system. The PrT net of the model presented in figure 4.8 is an extension of the model of figure 4.7, in the sense that we also consider the

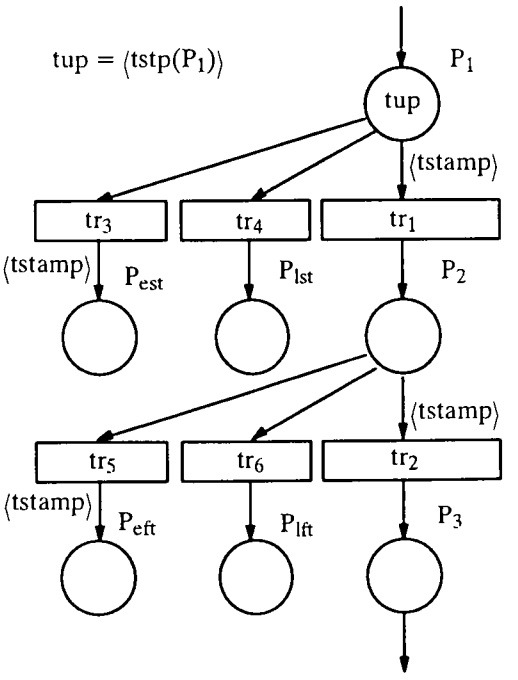


$$tr_1 : tstp(P_1) + t_{A_{est}} \leq tstp(P_2) < tstp(P_1) + t_{A_{lst}}$$

$$tr_2 : tstp(P_2) + t_{A_{eft}} \leq tstp(P_3) < tstp(P_2) + t_{A_{lft}}$$

Figure 4.7. The execution of an action in PrT nets.

specification of the exceptional behaviour for the case when the basic time attributes associated with the execution of an action are violated. If we consider, for example, the state in which an action is about to start its execution (P_1), then either an action starts its execution within the timing attributes associated with its respective utility function (tr_1), or exceptions are raised. These exceptions are associated with the cases when an action starts its execution earlier than specified (tr_3), and when an action does not start its execution within the specified time interval (tr_4). The exception handlers associated with these two exceptions are represented by the net predicates P_{est} and P_{lst}, respectively. The same PrT net structure is also applied to the case when an action is about to finish its execution, as shown in figure 4.8.



$tr_1 : tstp(P_1) + t_{A_{est}} \leq tstp(P_2) < tstp(P_1) + t_{A_{lst}}$

$tr_3 : tstp(P_1) \leq tstp(P_{est}) < tstp(P_1) + t_{A_{est}}$

$tr_4 : tstp(P_1) + t_{A_{lst}} \leq tstp(P_{lst}) < \infty$

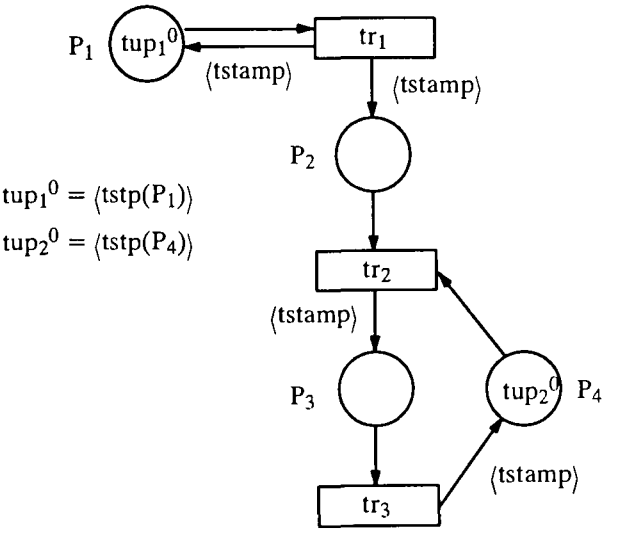
$tr_2 : tstp(P_2) + t_{A_{eft}} \leq tstp(P_3) < tstp(P_2) + t_{A_{lft}}$

$tr_5 : tstp(P_2) \leq tstp(P_{eft}) < tstp(P_2) + t_{A_{eft}}$

$tr_6 : tstp(P_2) + t_{A_{lft}} \leq tstp(P_{lft}) < \infty$

Figure 4.8. Timing exceptions in PrT nets.

In the following, we present the two examples of section 4.3.5 in terms of PrT nets. The first example, shown in figure 4.9, represents a sporadic timing constraint. The event *EV* is represented by transition tr_1, and the action *AC* is represented by transitions tr_3 and tr_4, and the net predicate P_3. The event *EV* is only allowed to occur at intervals of 100 ms; this timing constraint is imposed by the relational expression of transition tr_1. Once *EV* occurs, action *AC* has to be executed within a deadline of 60 ms. The timing relation of transition tr_2 shows that there are no time uncertainties associated with the occurrence of the start event of action *AC*; the deadline for the execution of the action is represented by the timing relation of transition tr_3. We assume that the value of the timestamp in the initial marking $tstp(P_1)=tstp(P_4)=0$.



tr_1 : $tstp(P_1) + 100 \leq tstp(P_2) < \infty$

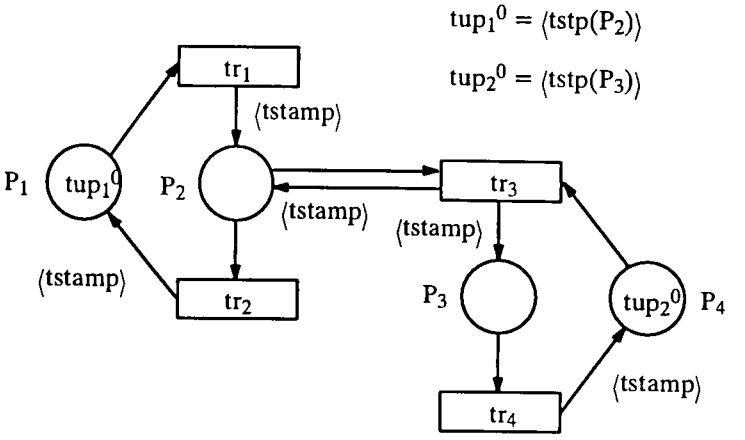
tr_2 : $tstp(P_2) \leq tstp(P_3) < tstp(P_2)$

tr_3 : $tstp(P_3) \leq tstp(P_4) < tstp(P_2) + 60$

Figure 4.9. A sporadic timing constraint in PrT nets.

In figure 4.10, we present (in terms of PrT nets) the second example of section 4.3.5 which involves a periodic timing constraint. The state predicate *SP* is represented by transitions tr_1 and tr_2, and the net predicate P_2, and the action AC is represented by transitions tr_3 and tr_4, and the net predicate P_3. Once the state *SP* holds true for at least 200 ms, action *AC* may start its execution periodically at every 200 ms and with a deadline of 100ms; the periodicity of *AC* is imposed by the timing relation of transition tr_3, and the deadline by

the timing relation of transition tr_4. We assume that the value of the timestamps in the initial marking $tstp(P_2)=tstp(P_3)=0$.



tr_1 : $tstp(P_1) < tstp(P_2) < \infty$

tr_2 : $tstp(P_2) + 200 < tstp(P_1) < \infty$

tr_3 : $tstp(P_2) \leq tstp(P_3) < \infty$

tr_4 : $tstp(P_3) + 100 < tstp(P_4) < tstp(P_3) + 100$

Figure 4.10. A periodic timing constraint in PrT nets.

4.5. Extract of a Case Study

To clarify some concepts introduced so far, an extract of a case study based on a simplified nuclear reactor control system was selected as an example /Saeed 93a/. Specifically, the example will serve to illustrate how the concepts of the E/A model can be incorporated within different classes of formalisms in order to describe system behaviour.

The example involves a system used to control the temperature of a nuclear reactor, the rods of the reactor have to be moved down when the temperature reaches the pre−defined threshold (5000K). The activity of moving down the rods takes 20 time units, and the start of consecutive movements of the rods should be at least 30 time units apart. To simplify the concepts, we make the (strong) assumption that the movement of the rods is not constrained by any physical limitation.

4.5.1. The Case Study from the PEA Notation Perspective

In a very simplified form, the nuclear plant can be defined, in the PEA notation, as the parallel composition of the actions describing the physical process and safety controller. ($A(i)\langle\,\rangle$ and $A(i)@\langle\,\rangle$ are abbreviations for the notation previously introduced.)

$$\forall i \in I^{+}: Nuclear\_Plant(i)\langle\,\rangle \Leftrightarrow Physical\_Process(i)\langle\,\rangle \parallel Safety\_Controller(i)\langle\,\rangle.$$

The physical process is defined by the sequential composition of actions representing the down movement of the rods ($PDMovRods(i)\langle\,\rangle$) and no movement of the rods ($PNDMovRods(i)\langle\,\rangle$).

$$\forall i \in I^{+}: Physical\_Process(i)\langle\,\rangle \Leftrightarrow PNDMovRods(i)\langle\,\rangle \prec PDMovRods(i)\langle\,\rangle.$$

In order to capture the timing requirements imposed on the actions the δ−operator can be employed to specify that the execution of $PDMovRods(i)\langle\,\rangle$ should take exactly 20 units of time, and consecutive executions should be at least 30 units of time apart:

$$\forall i \in I^{+}: \delta_{PDMovRods(i)} = 20.$$

$$\forall i \in I^{+}: \delta^{\min}_{(t^{i}_{\uparrow PDMovRods}, t^{i+1}_{\uparrow PDMovRods})} = 30.$$

The safety controller is defined by sequential composition of actions representing the down movement of the rods ($CoDMovRods(i)\langle\,\rangle$), no movement ($CoNDMovRods(i)\langle\,\rangle$), and a wait action on the down movement of the rods ($CoWMovRods(i)\langle\,\rangle$).

$$\forall i \in I^{+}: Safety\_Controller(i)\langle\,\rangle \Leftrightarrow$$
$$CoNDMovRods(i)\langle\,\rangle \prec CoDMovRods(i)\langle\,\rangle \prec CoWMovRods(i)\langle\,\rangle.$$

In the next two sections we show how the PEA notation specifications can be used as templates for specifying the behaviour in a descriptive formalism (THL) and an operational formalism (PrT nets).

4.5.2. The Case Study from the THL Perspective

The behaviour of the nuclear plant is specified in THL by imposing invariant and history relations over the start and finish conditions of the actions of the physical process and safety controller.

4.5.2.1. Physical Process

Pr1. The initial condition of the physical process is that *PNDMovRods* holds true.

$$\forall T_1 \in T\text{: } PNDMovRods(1)(T_1) \Leftrightarrow (T_1=0).$$

Pr2. The rods must start to move down if and only if the temperature (*PTemp*) is above the threshold (5000K) and 30 time units have elapsed since the previous time the rods started to move down, the execution of the action should take exactly 20 time units.

$$\forall T_1 \in T\text{: } \uparrow PDMovRods(1)(T_1) \Leftrightarrow$$
$$(PTemp>5000)(T_1) \wedge \forall t \in T\text{: } t<T_1 \Rightarrow \neg (PTemp>5000)(t).$$

$$\forall T_1 \in T\text{: } \forall i \in I^+\text{: } i>1 \Rightarrow \uparrow PDMovRods(i)(T_1) \Leftrightarrow$$
$$(PTemp>5000)(T_1) \wedge \exists t_1 \in T\text{: } (t_1 \leq T_1 - 30 \wedge \uparrow PDMovRods(i-1)(t_1) \wedge$$
$$\forall t \in T\text{: } t_1+30 \leq t < T_1 \Rightarrow \neg (PTemp>5000)(t)).$$

$$\forall T_1 \in T\text{: } \forall i \in I^+\text{: } \downarrow PDMovRods(i)(T_1+20) \Leftrightarrow \uparrow PDMovRods(i)(T_1).$$

Pr3. *PNDMovRods* immediately follows the action *PDMovRods*, as specified by the sequential composition of the actions in the PEA notation description of *Physical_Process*.

$$\forall i \in I^+\text{: } \uparrow PNDMovRods(i+1) \Leftrightarrow \downarrow PDMovRods(i).$$

$$\forall i \in I^+\text{: } \downarrow PNDMovRods(i) \Leftrightarrow \uparrow PDMovRods(i).$$

4.5.2.2. Safety Controller

In this example, we assume that provided the reactor temperature (*PTemp*) is within its anticipated range then it is equal to the thermometer reading (*CiTemp*).

Cr1. The initial condition of the safety controller is that *CoNDMovRods* holds true.

$$\forall T_1 \in T\text{: } CoNDMovRods(1)(T_1) \Leftrightarrow (T_1 = 0).$$

Cr2. The safety controller must start to move the rods down at the instant the temperature (*CiTemp*) rises above the threshold (5000K) and 30 time units have elapsed since the previous time the rods started to move down, the execution of the action *CoDMovRods* should take exactly 20 time units.

$$\forall T_1 \in T\text{: } \uparrow CoDMovRods(1)(T_1) \Leftrightarrow$$
$$(CiTemp > 5000)(T_1) \wedge \forall t \in T\text{: } t < T_1 \Rightarrow \neg (CiTemp > 5000)(t).$$

$$\forall T_1 \in T\text{: } \forall i \in I^+\text{: } i > 1 \Rightarrow \uparrow CoDMovRods(i)(T_1) \Leftrightarrow$$
$$(CiTemp > 5000)(T_1) \wedge \exists t_1 \in T\text{: } (t_1 \leq T_1 - 30 \wedge \uparrow CoDMovRods(i-1)(t_1) \wedge$$
$$\forall t \in T\text{: } t_1 + 30 \leq t < T_1 \Rightarrow \neg (CiTemp > 5000)(t)).$$

$$\forall T_1 \in T\text{: } \forall i \in I^+\text{: } \downarrow CoDMovRods(i)(T_1 + 20) \Leftrightarrow \uparrow CoDMovRods(i)(T_1).$$

Cr3. *CoWMovRods* immediately follows action *CoDMovRods*, and is of duration exactly 10 time units.

$$\forall i \in I^+\text{: } \uparrow CoWMovRods(i) \Leftrightarrow \downarrow CoDMovRods(i).$$

$$\forall T_1 \in T\text{: } \forall i \in I^+\text{: } \downarrow CoWMovRods(i)(T_1 + 10) \Leftrightarrow \uparrow CoWMovRods(i)(T_1).$$

Cr4. *CoNDMovRods* must immediately follow *CoWMovRods* and precede *CoDMovRods*, as specified by the sequential composition of the actions in the PEA notation description of *Safety_Controller*.

$\forall i \in I^+$: $\uparrow CoNDMovRods(i+1) \Leftrightarrow \downarrow CoWMovRods(i)$.

$\forall i \in I^+$: $\downarrow CoNDMovRods(i) \Leftrightarrow \uparrow CoDMovRods(i)$.

4.5.3. The Case Study from the PrT Nets Perspective

In order to exemplify the utilization of the E/A model primitive functions as modelling concepts for an operational formalism, in the following, we discuss the PrT net model of the movement of the rods to control the temperature of a nuclear reactor. The PrT net model is shown in figure 4.11 and includes models of the physical process and the safety controller. The name and the role of the places of the PrT net correspond to the actions specified by the descriptive formalism, presented in the previous section. The relations associated with the transitions of the PrT net are defined in figure 4.11, and correspond to the conditions of the actions defined in THL. In the PrT net model, the place *CiTemp* contains the last measurement of the reactor temperature which is periodically updated from the simulated values contained in *PSimTemp*. The subnet containing *PDMovRods* and *PNDMovRods* is obtained from the PEA notation formula that defines the sequence of actions that are associated with the rods; the subnet containing *CoDMovRods*, *CoWDMovRods* and *CoNDMovRods* is obtained from the PEA notation formula that defines the sequence of actions that have to be performed by the safety controller in order to meet the requirements imposed on the movement of the rods depending on the temperature of the reactor; the places *ActDMovRods* and *ActNDMovRods* represent the outputs of the (logical) actuator that moves the rods. The predicates associated with the transitions *T3*, *T4* and *T5* are derived from the start and finish conditions defined for the actions to be executed by the safety controller.

4.6. Comparison of the PEA Notation with similar Models

The fundamental notions adopted for the PEA notation were based on the event–action model initially proposed by Jahanian & Mok /Jahanian 86/. This model attempted to capture the temporal ordering of computational actions, and was based on the following

PAGE
MISSING
IN
ORIGINAL

Although some of the notions adopted for the PEA notation are similar to those presented in /Jahanian 86/, there are some differences which makes the former model more suitable for the type of behavioural representations needed for requirements analysis. An essential difference between the two models is the fact that the PEA notation considers time as an independent variable. (In more recent work /Jahanian 88a/ the occurrence function "@", which had time as dependent variable, is replaced by the (Boolean) occurrence relation "Φ" which has time as a parameter.) The PEA notation approach allows events, actions and states, and their respective utility functions, to be easily depicted as graphics, which is consistent with the techniques usually employed in process control engineering. Another difference between the two models concerns the expressiveness of their notations. In the Jahanian & Mok approach, by restricting its expressiveness, emphasis was given to a notation that would simplify the mechanical analysis of the specifications, while in the PEA notation an attempt was made to identify and clearly represent each of the concepts defined. Another issue that shows the flexibility of the PEA notation, by comparision with the Jahanian & Mok approach, is that while the former model supports both dense and discrete time structures, the latter only supports discrete time structures. As far as similarities between the models are concerned, the logic assertions of the PEA notation can also be transformed into a set of formulas in Real–Time Logic, in a similar way to that performed for the Jahanian & Mok event–action model /Jahanian 86/.

Another feature of the PEA notation, which has no equivalent in the Jahanian & Mok event–action model /Jahanian 86/, is that in the PEA notation we are also able to describe the behaviour of a system, and its components, by abstracting from their timing properties, in a similar way that of ISL /Goswami 92/. Interval Specification Logic (ISL) is a first order linear–time temporal logic which is used for specifying the properties of programs in execution intervals which are sequence of states. In ISL a timed description of the system behaviour can be developed, by refinement, from an untimed description.

The PEA notation can also be compared with two other similar techniques, TRIO /Ghezzi 90/ and VVSL /Middleburg 89/. TRIO is a first–order temporal logic language for specifying and verifying timing requirements; its proof theory and its executability have been mathematically defined. The language provides operators that allow to check the truth or falsity of a proposition at particular time instants. TRIO can accommodate either dense or discrete time structures. However, TRIO does not provide mechanisms for modularizing complex specifications, and it is hard to read and understand. VVSL defines operations in terms of temporal predicates over state variables; in addition to the pre–condition and post–condition of VDM /Jones 86/, VVSL introduces an *inter–condition* which imposes constraints during execution (i.e. acts as a state–invariant). However, unlike the PEA notation, VVSL does not permit quantitative timing analysis.

With respect to the timing constraints of the PEA notation, instead of adopting an exhaustive list of all possible timing relations /Dasarathy 85/, we have only considered those that are essential for representing events, actions and states, and their associated time uncertainties.

In comparision with other models mentioned above, the PEA notation is more flexible in its application to requirements analysis because its features allows different issues in the behaviour of the system, and its components, to be modelled from different perspectives of the analysis.

4.7. Concluding Remarks

The aim of this chapter was to introduce the E/A model, which is based on a set of concepts that allow us to describe the behaviour of systems at different levels of abstraction, and using different formalisms. On top of the E/A model we proposed the PEA notation in order to provide a compact notation for the E/A model, and to capture the behaviour of critical real–time systems in terms of predicates over state variables. Apart from proposing the PEA notation we also formalised the E/A model in terms of

THL and PrT nets with the aim of demonstrating the flexibility of the concepts being employed.

However, there are some important issues which need to be resolved in order to obtain a complete model from which we would be able to perform a formal analysis of system timeliness requirements, the theme which was given more emphasis in this chapter. One of these issues is how to obtain confidence that the specifications performed in terms of the E/A model are complete; a possible way to tackle this issue is to provide a set of criteria similar to the work done by Jaffe et al /Jaffe 91/. Another issue is to justify formally the extension proposed for PrT nets, which would allow the modelling and analysis of timing constraints. A final issue, which is related to the approach of using THL and PrT nets to perform a formal analysis of the requirements, is the need to provide some means by which the verification of the PrT net model could be checked against the original specification in THL. A hint to be followed up is to try to extract from the simulation of the PrT net model its E/A model, and to verify this E/A model against the initial E/A model specified using THL.

Chapter 5

A Methodology for Requirements Analysis

5.1. Introduction

In this chapter we describe a methodology, based on formal techniques, for the requirements analysis of critical real–time systems. The approach, which provides a systematic way in which safety requirements can be analysed, consists of a framework with distinct phases of analysis, a set of techniques appropriate for the issues to be analysed at each phase of the framework, a graph to record the safety specifications obtained from the process of analysis, and a set of techniques for conducting a quality analysis of the safety specifications.

The proposed approach for the requirements analysis of critical real–time systems is based on five key principles:

- *Focus on safety.* The approach is based on maintaining a separation between the mission and the safety requirements. (Safety requirements focus on the elimination and control of hazards, and on the limitation of damage in an accident.) For a particular system, the extent to which safety features can be separated from the mission implementation will depend on the trade–offs that can be made between maintaining safe behaviour and optimising the system's ability to fulfil its mission. These trade–offs will take account of the risks associated with the system hazards and the benefits gained from the mission.

- *Techniques for the analysis.* The techniques to be employed for a specific domain of analysis are selected in accordance with the characteristics of the properties of interest of that domain (and the perspective adopted). This leads to the utilization of a range of specialized informal and formal techniques. The problem of linking

specifications obtained from different techniques is tackled by employing a primitive set of modelling concepts that are appropriate for the properties being analysed.

- *Framework for the analysis.* A systematic approach is provided by deriving a framework from a system structure that is general for an application domain. This framework partitions the requirements analysis into phases, and defines dependencies between the phases. Each phase focuses on a specific domain of analysis, providing partial knowledge of the overall system. During the analysis, a domain is viewed from different perspectives; each perspective captures a specific description of the domain.

- *Formal record of the requirements specifications.* The safety specifications obtained from the requirements analysis are recorded in terms of a directed graph, in which nodes represent the specifications, and edges denote the relationships between the specifications. The graph provides a succinct representation which is amenable to formal analysis.

- *Quality analysis.* The key quality factor for critical real–time systems is the risk to which the environment is subjected by the system. To obtain confidence that the risks associated with the requirements specifications are acceptable, quality analysis techniques are included in the proposed approach. The quality of the requirements specifications is confirmed by: firstly, applying *verification* and *validation* techniques to ensure that the specifications are consistent and maintain safe behaviour, and secondly, conducting a risk analysis of the requirements specifications. The approach enables the integration of formal techniques with traditional safety techniques to yield a coherent method for risk analysis.

The contents of the chapter are organised as follows. The next section justifies the adopted separation between mission and safety issues of systems, specifically, during requirements analysis. Then the role of different techniques during requirements analysis is discussed in section 3, emphasising the contribution of formal techniques. In

section 4 we describe in more detail the phases of the framework for requirements analysis, covering the issues to be analysed, the techniques to be employed, and the time bases to be used. Section 5 presents the graph in which the requirements specifications are organised. In section 6 we present how the quality analysis of requirements specifications is conducted by qualitative and quantitative means. Finally, we present some concluding remarks on the methodology presented in this chapter.

5.2. Separation between the Mission and the Safety Requirements

The methodology presented here for software requirements analysis for critical real–time systems is based on establishing a clear separation of the mission and the safety requirements. On the one hand, the mission requirements focus on what the system is supposed to achieve in terms of function, timeliness and some dependability requirements – namely the attributes of reliability, availability and security /Laprie 90/. On the other hand, the safety requirements focus on the elimination and control of hazards, and the limitation of damage in the case of an accident; in other words, they address what the system should not do /Leveson 86/, and are thus related to the safety attribute of dependability.

This separation at the requirements level is essentially logical rather than physical; moreover, we do not have any intention to force a particular structure on the implementation. The benefits of making this distinction during requirements analysis are: the ability to focus on the safety–critical issues, the simplification of safety certification, the resolution of potential conflicts, and detection of omissions and inconsistencies between the mission and safety issues.

Although logical at the requirements analysis phase, this separation between the mission and safety can, when appropriate, be enforced – at the plant domain, as exemplified by the shut down systems of nuclear reactors, and at the controller domain, by implementing separate controllers for mission and safety as is done in interlocking systems for railways /Chandra 91/, or by the application of design techniques, such as

"firewalls", which are intended to prevent the mission factors from interfering with the safety factors once the controller is implemented. Whether it is feasible to separate completely the mission from the safety requirements for a particular critical real–time system depends on the ease with which a distinction can be drawn between safe and unsafe states. The mission controller is concerned with system behaviour while the system is in a safe state, and the safety controller when the system is in an unsafe state. Furthermore, the aims of the two controllers can be considered to be complementary. The mission controller seeks to ensure that the mission is accomplished – this will also require that the system does not enter into an unsafe state. The safety controller is concerned with avoiding hazardous states by explicitly dealing with the unsafe states that precede these states.

5.3. Formal Methods for the Framework

As already mentioned in the Introduction, although there are potential advantages in employing formal methods, there are also a number of limitations concerning their applicability in the requirements analysis of critical real–time systems. In this section, we enumerate the goals that should to be pursued in order to make the application of formal methods feasible, and the features that a formalism should possess in order to be applied effectively in the context of the framework. But first, we define what we mean by a "formal method".

Formal is a term applied to a notation or technique if it is amenable to mechanical manipulation according to some calculus. A method is a set of procedures and guidelines for the application of a notation or technique to system development. When the notation or technique is formal, the method is referred to as a formal method.

In the following, we describe the demands that have to be considered when applying formal methods to the process of software development /Anderson 93, Tsai 92/.

- During requirements analysis the objective is to capture the main ideas of the system being developed rather than represent minute details of the system. Hence,

it is necessary to employ techniques which can improve the comprehension and assimilation of the specifications. This can be achieved by employing techniques that are accessible and suitable. Accessibility of a technique refers to the experience and easiness that an analyst has when employing the technique, and to the usefulness of the resulting documentation (produced by applying the technique) as a means of communication between the analyst and the client. Suitability refers to the appropriateness of a specific technique in performing a particular aspect of analysis at the relevant stage of requirements analysis.

- The utilization of a technique during requirements analysis should be disassociated from a prescribed specific method because the method might restrict the possible implementation choices. In the sense that while a method should be tailored for the subject and type of analysis to be conducted, a technique should be sufficiently flexible in order to be used across different subjects and types of analysis.

- Formal techniques, apart from allowing verification to be performed, should also enable specifications to be validated either by testing /Dauchy 91/ or by symbolic execution /Harel 90/.

- Consistency has to be achieved between emerging techniques and those currently adopted in practice, fundamentally, those techniques that are related to plant analysis, such as Hazard and Operability Studies (HAZOPS) /Lawley 73/. Safety is essentially an attribute of the system rather than just software, which implies that the requirements analysis of critical real–time systems should always consider the system in which the software is to be embedded.

- The formal method should support traceability in order to control any changes applied to the product during its development, and to relate these changes back to the relevant requirements. This is particularly important when conducting safety certification.

- The formal method should also allow the system to be structured into subsystems which can be dealt with separately from each other. This structuring can be in terms

of levels of abstraction, where higher levels are associated with the early stages of requirements analysis, thus reducing complexity.

Formal support for the range of different activities concerning the framework for requirements analysis demands the utilisation of a set of formal methods, whose features and expressive power should match the characteristics of the different activities. Selecting an appropriate formal technique for an individual activity allows emphasis to be placed on pertinent characteristics of the system, enabling a technique to work to its own strengths. In the proposed approach we employ formal techniques in accordance with the characteristics of the system to be analysed, during the different phases of the framework, and define the linkage between the different techniques, resulting in a coherent approach.

Although formal methods have been classified in a different number of ways /de Roever 91, Olderog 91, Pnueli 86, Wing 90/, for the purpose of the framework we identify two kinds of formalisms: descriptive and operational.

When using descriptive formalisms, system behaviour is specified in terms of system properties that have to be satisfied. The specification is axiomatic, hence it should have a conjunctive nature, which thus allows new requirements to be added while performing the analysis, without the need to modify or reconstruct the entire specification. In order not to restrict the set of possible implementations which can be obtained from a specification, the specification should not state more than the necessary minimal constraints on the system behaviour. In the context of the framework, as an example of descriptive formalisms, we will only be considering logical formalisms, such as Temporal Logic /Gorski 86/, Real–Time Logic (RTL) /Jahanian 86/, Real–Time Temporal Logic (RTTL) /Ostroff 87/, and Timed History Logic (THL) /Saeed 90/.

When using operational formalisms, system behaviour is specified by constructing a model of the system in terms of mathematical structures such as tuples, relations, functions, sets and sequences. Another feature of this class of formalisms is that they

should be able to explicitly specify concurrency and non-determinism, in order to model the interaction between specifications. In the context of the framework, as an example of operational formalisms, we will only be considering graphical formalisms, such as Petri nets /Murata 89/, Extended State Machines (ESM) /Ostroff 87/, Statecharts /Harel 87/, and Modecharts /Jahanian 88/. Also, algebraic formalisms that have constructs supporting non-determinism and concurrency can be employed, such as CSP /Hoare 85/ and CCS /Milner 83/.

In order to facilitate the description of system behaviour using different formal methods, we adopt the E/A model as a common foundation for the models of system behaviour. These basic concepts of the E/A models (events, actions and states) are sufficiently general that they may be expressed either in descriptive or operational formalisms. In other words, the basic concepts of the E/A model will be used to model the behaviour of the system being analysed by employing the proposed formalisms.

5.4. Framework for the Requirements Analysis

When developing critical real-time systems, a key concern is to ensure that accidents do not occur. The usual practice for ensuring safety is to strive to identify all of the hazards and then apply techniques which reduce the probability of these hazards occurring. The level of effort that should be employed to handle a hazard depends on the risk associated with that hazard.

In order to facilitate systematic analysis, for each level of abstraction of the system we derive the phases of a framework from the components of that level, and their interactions: the components define the domains of analysis and the interactions between the components define the dependencies that exist between the domains. The organization of the frameworks, in terms of the phases and their ordering, follows (respectively) from the domains of analysis and their dependencies.

Our general framework for the requirements analysis of critical real-time systems is shown in Figure 5.1. This is obtained by, first, splitting the mission requirements from

the safety requirements, and second, subdividing the analysis of the safety requirements into a sequence of phases. The phases are convenient notions when describing the framework and serve to show that the level of abstraction decreases from descriptive to operational as the analysis proceeds. The results of a phase can, of course, be influenced by feedback from subsequent phases, since these take additional information into account. For example, the plant interface analysis can take into account limitations of sensors and actuators which may enforce changes to the specifications produced at the plant level. The proposed framework forces safety specifications to be formal from the initial stages of requirements analysis. This has the benefit that, once the essential requirements of the system are formalised (e.g. hazards) the subsequent safety specifications and designs can be obtained by formal refinement.

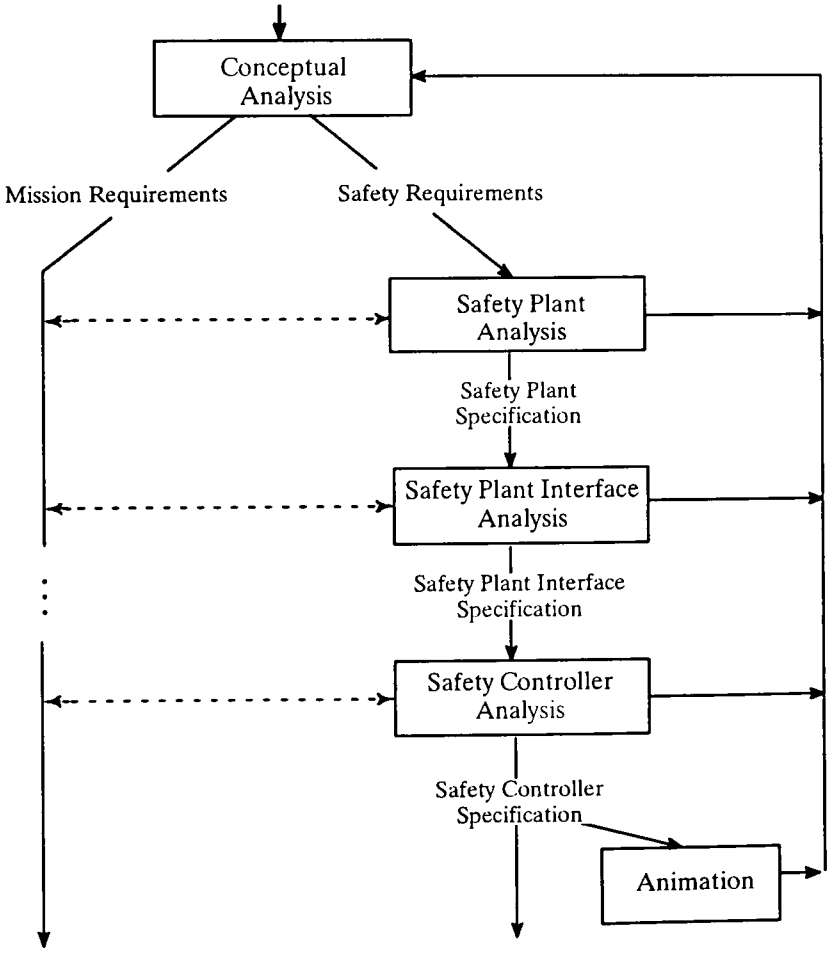


Figure 5.1. A framework for requirements analysis.

According to figure 5.1, in order to conduct the requirements analysis, the ordering in the phases analysis will be as follows. The initial phase, *Conceptual Analysis,* produces a statement of the aim and purpose of the system and determines those failure behaviours of the system which constitute accidents. The second phase, *Safety Plant Analysis,* identifies the plant properties relevant to the safety requirements, such as the physical laws and rules of operation that govern plant behaviour. The third phase, *Safety Plant Interface Analysis*, delineates the interface between the plant and controller, and specifies the behaviour that must be exhibited at that interface. The final phase, *Safety Controller Analysis*, establishes a top level organization for the controller in terms of the properties of its components and their interactions. (On its own animation is not considered a phase of analysis, is part of the Safety Controller Analysis. Animation was given a certain distinction in the framework of the figure 5.1 because of its importance in the validation of the requirements specifications.)

In the following, we describe in more detail each of the identified phases by identifying the formalisms to be employed for the analysis, the time structure in which the analysis is more likely to be conducted, and how the concepts of the E/A model will be applied in order to facilitate the modelling of the behaviour of the system.

5.4.1. Conceptual Analysis

The objective of this phase is to produce an initial, informal statement of the aim and purpose of the system, with particular emphasis on what the impact the environment can have on the system and what can be the consequences of the behaviour of the system on the environment. During this phase what is meant by "safety" for the system is also determined (i.e. the failure behaviours of the system which constitute accidents are identified). As a product (i.e. output) of the Conceptual Analysis we obtain the Safety Requirements, enumerating the potential *accidents* (an unintended event or sequence of events that causes death, injury, environmental or material damage). Identification of the possible accidents a system could cause would be conducted using standard techniques, such as Comparative Methods /ICE 85/. The identification of the accidents

provides a basis from which a distinction between the mission and safety issues of the system can be obtained. Another activity to be performed during this phase is the identification of the primary modes of operation of the physical process; these modes are classes of states which group together related functions.

At the conceptual level, we are looking for notations which would allow the representation of, and reasoning about, a very high level description of the preferred behaviour of the system within its enclosing environment. A formal notation which allows the representation of causality seems the most adequate because at this level of description there is no intention in conducting a qualitative analysis of the activities of the system. Currently, we describe system behaviour in terms of interpreted predicate symbols in the PEA notation. This enables causality to be modelled in terms of temporal ordering and modes of operation to be defined as actions or states, and also has the advantage of allowing us to specify the main state transitions and activities of the system at a high level, providing a link to the more detailed analysis at the plant level.

5.4.2. Safety Plant Analysis

The objective of this phase is to identify those properties of the physical process which are relevant to the Safety Requirements. An essential starting point for the formal analysis of a critical real–time system is to identify the physical entities that make up the plant, and to construct a formal model of the physical process which captures its characteristics as dictated by physical laws and rules of operation. We assume that a model of the plant can be obtained that accurately represents the behaviour of the plant necessary to conduct the requirements analysis of the software to be embedded in the system (i.e. the safety controller). The plant analysis must also support the identification of *hazards* (a condition over the physical process describing states from which an accident might ensue). As the product of the plant analysis we obtain the Safety Plant Specification which contains the *safety constraints* (a safety constraint is a condition imposed on the physical process: usually the negation of a hazard modified to incorporate safety margins) and the *safety strategies* (a safety strategy is a way of

maintaining a safety constraint, and is defined as a set of conditions, in terms of controllable factors, over the physical process). The conjunction of all of the safety constraints of a system characterises the specified safe behaviour for that system.

5.4.2.1. Proposed Formalism

At the plant level, we have to analyse both continuous variable and discrete event dynamic systems. Hence it is necessary to employ both differential equations and discrete mathematics at this level. After the analysis of continuous systems is performed, and the appropriate safety thresholds of the plant behaviour defined, we need a descriptive formalism in order to specify behaviour in terms of axioms over a model of the system. In this thesis, as a descriptive formalism, we use THL during the Safety Plant Analysis, primarily because the behaviour of systems expressed in THL is defined by imposing constraints over the set of all sequences of states that the system can exhibit. This property gives THL formulae a conjunctive nature. Hence, by using THL, the Safety Plant Specification can be constructed by considering each safety constraint and safety strategy separately.

5.4.2.2. Proposed Time Structure

We propose the use of a dense time structure during the Safety Plant Analysis which is compatible with THL, the formalism being proposed for this phase of analysis. Several other formalisms use a dense time structure; these include extensions to Temporal Logic /Pnueli 88/, CSP /Reed 86/ and CCS /Milner 83/. Some justification for selecting a dense time structure for the Safety Plant Analysis is presented below.

Physical laws govern the properties of the environment in which a system works, and therefore impinge directly on the behaviour of the system. To perform a thorough analysis of system behaviour these laws must be expressed and their impact on the system analysed. Many of these physical laws are defined in terms of formalisms that assume a (continuous) physical time, e.g. the distance travelled by a train for a duration of time

is given by an integral over the velocity of the train. Hence, a dense representation of time is more suitable for the plant analysis.

The plant time structure is a dense time structure (defined as a tuple $\mathcal{T}_{DE} = \langle \mathcal{P}, + \rangle$) isomorphic to the reals (**R**).

5.4.2.3. The E/A Model in the Safety Plant Analysis

During the Safety Plant Analysis, after the physical entities of the plant are identified, we define, by employing the E/A model, the activities associated with the physical entities and the relationships that might exist between these entities. From the behavioural analysis of the entities, in terms of their discrete and continuous variables, we identify their respective thresholds in terms of both value and time domains. From these thresholds, which correspond to transitions in the continuous state of the system, we define the behaviour to be associated with the physical entities in terms of the basic concepts of the E/A model. In other words, by using the E/A model to describe the behaviour of the plant we are discretizing its behaviour with the aim of facilitating specification of the safety strategies and their subsequent verification. In terms of the E/A model the actions associated with a particular physical entity are exhaustive and exclusive, i.e. at any time point the physical entity must be performing exactly one action or be at the transition point between two consecutive actions. Another feature of the E/A model is that it facilitates timing analysis by providing formal and informal means to capture the behaviour exhibited by the physical entities in terms of the timing relationships between the occurrence of events.

5.4.3. Safety Plant Interface Analysis

The objective of this phase is to delineate the plant interface, and to specify the behaviour that must be observed at the identified interface. This involves stipulating the properties of the required sensors and actuators, including their failure rates, in order to determine the limitations (constraints) which these components impose on an "ideal"

implementation of the safety strategies by the controller. We perform an analysis of the safety strategies to determine their robustness in the presence of failures in the components of the plant interface, i.e. failures of sensors and actuators. This phase leads to the production of the Safety Plant Interface Specification, which contains the *robust safety strategies* (a robust safety strategy is a modification of a safety strategy incorporating the limitations of the sensors and actuators to be deployed).

5.4.3.1. Behaviour of Sensors and Actuators

In this section we are concerned with the analysis of those sensor or actuator failures that have an adverse affect on the safety of the system. For that the behaviour of sensors and actuators is specified in terms of their standard, exceptional and failure behaviours. The motivation for adopting such a partition is to attempt to characterise the complete behavioural space of sensors and actuators. The different categories of behaviour can be obtained by employing appropriate techniques. By specifying the failure behaviour of the sensors and actuators we are also able to establish how the system can be made more robust in the presence failures of these components. The specification of the exceptional behaviour of a sensor or actuator takes into account those situations when it becomes impossible for them to deliver the specified standard behaviour. A sensor or actuator instead of failing to provide any predictable service, notifies another component of the system by raising an exception.

The specification of the complete behavioural space of sensors and actuators involves specifying the anticipated values ($RV^{a}_{v_i}$) and unanticipated values ($RV^{u}_{v_i}$) of system variables, in terms of their ranges, and stipulating the standard, exceptional, and failure behaviour of the sensors and actuators. In order to sketch a model for the behaviour of any such component (either a sensor or an actuator) we define *vi* as the input variable and *vo* as the output variable of the component. The model follows an approach adopted for specifying the standard and exceptional behaviour of software components /Cristian 89/.

For component *CM* let $RV_{vi} = RV^{a}_{vi} \cup RV^{u}_{vi}$ be the set of values that may be inputs for *CM* and RV_{vo} be the set of values returned by *CM*. The behaviour of *CM* is defined in terms of partial relations over $RV^{a}_{vi} \times RV^{a}_{vo}$. The standard behaviour is defined by relation *SS(CM)*:

$$SS(CM) = \{(vi,vo) \mid vi \in RV^{a}_{vi} \wedge vo \text{ is an intended outcome for } vi\}.$$

For component *CM* we specify the set of exceptions as **E**. For each exception $e \in \mathbf{E}$ we specify the exceptional behaviour for *CM* by the relation $ES_e(CM)$:

$$ES_e(CM) = \{(vi,vo) \mid vi \in RV^{a}_{vi} \wedge vo \text{ is an intended outcome for } vi \text{ for } e\}.$$

We will say that component *CM* has failed if its behaviour is not contained within the relation *SS(CM)* or a relation $ES_e(CM)$. The failure behaviour is specified by the relation *FS(CM)*:

$$FS(CM) = \{(vi,vo) \mid vi \in RV^{a}_{vi} \wedge (vi,vo) \notin SS(CM) \wedge \forall e \in \mathbf{E}: (vi,vo) \notin ES_e(CM)\}.$$

The failure behaviour of a component is identified by conducting a failure modes and effect analysis (FMEA). A FMEA is used to analyse all component failure modes and to identify the resulting affect on the system /Wells 80/. We denote the set of anticipated failure modes as **FM**; and for each failure mode $af \in \mathbf{FM}$, we specify the failure behaviour by the relation $FS_{af}(CM)$:

$$FS_{af}(CM) = \{(vi,vo) \mid vi \in RV^{a}_{vi} \wedge vo \text{ is an outcome for } vi \text{ in } af\}.$$

The anticipated failure behaviour will be a subset of the failure behaviour:

$$\bigcup_{af \in \mathbf{FM}} FS_{af}(CM) \subseteq FS(CM).$$

In the above definitions, we have concentrated on relationships in the value domain. However, for many practical systems it will be necessary to extend the specification to incorporate the relationships that exist in the time domain /Powell 92/.

At the interface level, we are concerned with modelling the properties of sensors and actuators, the impact of failures occurring in either the physical process or controller, and their ability to maintain safe behaviour. Again, to perform this analysis we need a descriptive formalism in order to specify the behaviour of a component in isolation, either by using pre/post–conditions or relations.

5.4.4. Safety Controller Analysis

During this phase the specifications from the previous phase, namely the robust safety strategies, are analysed in terms of the properties of the controller, such as the top level organization of the controller components and their failure rate. This phase leads to the production of the Safety Controller Specification, containing the *safety controller strategies* (a safety controller strategy is a refinement of a robust safety strategy incorporating some of the components of the controller). The Safety Controller Specification is the main product of the requirements analysis, and provides the basis for subsequent development of the system.

5.4.4.1. Proposed Formalism

The general features which are required for a formalism to be appropriate for this phase are the following: the specification should be able to represent the architectural design of the controller, modelling the interactions between its components, and the operations that must be performed by these components. To perform this we need an operational formalism which explicitly specifies concurrency and non–determinism. In this thesis, as an operational formalism, we use Predicate–Transition nets (PrT nets) /Genrich 87/, a form of high–level Petri net, during the Safety Controller Analysis. Petri nets are mainly used for the modelling and analysis of discrete–event systems which are concurrent, asynchronous, and non–deterministic. The use of PrT nets, instead of (Timed) Petri nets /Leveson 87/, adds to the modelling power of the latter the formal treatment of *individuals* (i.e. the notion of token identity) and their changing properties and relations.

5.4.4.2. Proposed Time Structure

For the analysis of timeliness requirements during the Safety Controller Analysis we propose the use of a discrete time structure. The choice of using a discrete time structure seems natural, in the sense that it is the most convenient and expressive model of time for discrete computation. The timeliness analysis to be performed at this phase is essentially related to the safety requirements of the safety controller.

The controller behaviour can be viewed through a sequence of conceptual layers (or levels of abstraction) in which the passage of time is represented by discrete time structures with different granularities. The granularity of each layer is dictated by the timing constraints associated with the occurrence of events of that layer. The relationship between discrete time structures with different granuralities is discussed in section 3.2.2.

The safety controller time structure is a discrete time structure (defined as a tuple $\mathcal{T}_{DI} = \langle \mathcal{P}, + \rangle$) isomorphic to the Integers (**I**).

5.4.4.3. The E/A Model in the Safety Controller Analysis

In terms of timeliness requirements, during the Safety Plant Analysis, the E/A model was employed with the aim of understanding the temporal behaviour of the physical process, in terms of quantitative relationships, and to identify those timing thresholds which will be part of the safety strategy. The application of the E/A model during the Safety Controller Analysis has the purpose of capturing the behaviour of the components of the controller in terms of the tasks (actions) that they have to execute. The E/A model behaviour of the component is obtained by successively refining the safety plant interface specifications. In terms of the time domain, the behaviour of the components should take into account the timing thresholds defined during the Safety Plant Analysis, and the time uncertainties introduced by the sensors and actuators.

For every component, together with the specification of its standard behaviour, the component's failure behaviour should also be specified. (This failure behaviour might

be characterized by the component failing to execute an action within the specified timing constraints.) Once the failure behaviour of the component is known, error handling mechanisms, such as exception handling, can be incorporated with the component, in order to allow corrective measures to be taken whenever a error is detected. The advantage of considering such error handling mechanisms at this early stage is that these mechanisms can be specified and verified within the context of the safety controller strategies, instead of implementing *ad hoc* approaches to handle errors during the later phases of system development.

5.4.4.4. Animation

At this stage of the analysis, we can conduct the Animation of the Safety Controller Specification which enables the user of the system to check if the safety controller strategies do indeed capture the intended behaviour, and are consistent with the mission requirements. In our approach this is achieved by simulating the PrT net models of the physical process, plant interface, and safety controller. The animation provides a concrete manifestation of the abstract models, thus enabling the user to exercise the specifications in defined scenarios, and visualize different options in the refinement of the specifications. It is certainly possible to envisage a requirements validation system which would support and facilitate user interaction with the formally expressed requirements specification by presenting a user friendly graphical interpretation of an operational formalism.

5.5. Safety Specification Graph

The specifications produced during requirements analysis are recorded in a formal structure, referred to as a Safety Specification Graph (SSG). An SSG is a linear graph, in which a *node* represents a specification and an *edge* denotes a relationship between a pair of specifications. Since the analysis is by repeated refinement, an SSG is directed and acyclic. For those systems whose behaviour is described in terms of modes of operation, a separate SSG is constructed for each mode.

5.5.1. Structure of SSG

By examining how the specifications are obtained from the framework for requirements analysis, structural properties of an SSG can be deduced.

- *Distinct component graphs*. The derivation of the safety specifications starts from the identification of the potential accidents, hence each accident represents a root from which other safety specifications are derived. This is reflected in the SSG by the construction of a distinct component graph for each accident.

- *Evolutionary graph*. Each component graph is an evolutionary graph (a graph whose nodes and edges are formed by addition or removal /Marshall 71/), in which the evolution is related to the phases of the framework. At each phase, a set of nodes (representing the specifications produced) is added to the graph of the previous phase, and edges are introduced connecting the additional nodes to what were the terminal nodes of the graph.

- *Layered graph*. A component graph organizes the safety specifications into layers. From the framework presented in section 5.4, a six layered graph is envisaged. The top most layer, relating to the environment level, represents an accident (AC). Layers two, three and four, relating to the plant level, represent hazards (HZ), safety constraints (SC) and safety strategies (SS), respectively. The fifth layer, relating to the plant interface level, represents the robust safety strategies (RSS). The bottom layer, relating to the control system level, represents the safety controller strategies (SCS).

5.5.2. Relationships of SSG

The SSG encodes and records the relationship between safety specifications at consecutive layers of component graphs; three sorts of relationships are identified: the absence of all of the hazards associated with an accident should ensure that the accident does not occur; a safety constraint should imply the negation of the hazards which it

addresses; and a strategy is intended to maintain the specifications (of the previous layer) to which it is linked.

When more than one strategy is related to a specification in a previous layer, then either the strategies are exclusive alternatives and a choice will have to be made in later phases of development to select and refine a single strategy, or the strategies complement each other and all are needed to attain sufficient confidence that the risks are acceptable. These two situations are distinguished by annotating the edges with a "$\oplus$" in the exclusive case, and a "$\odot$" in the complementary case. The general structure of an example SSG is given in figure 5.2.

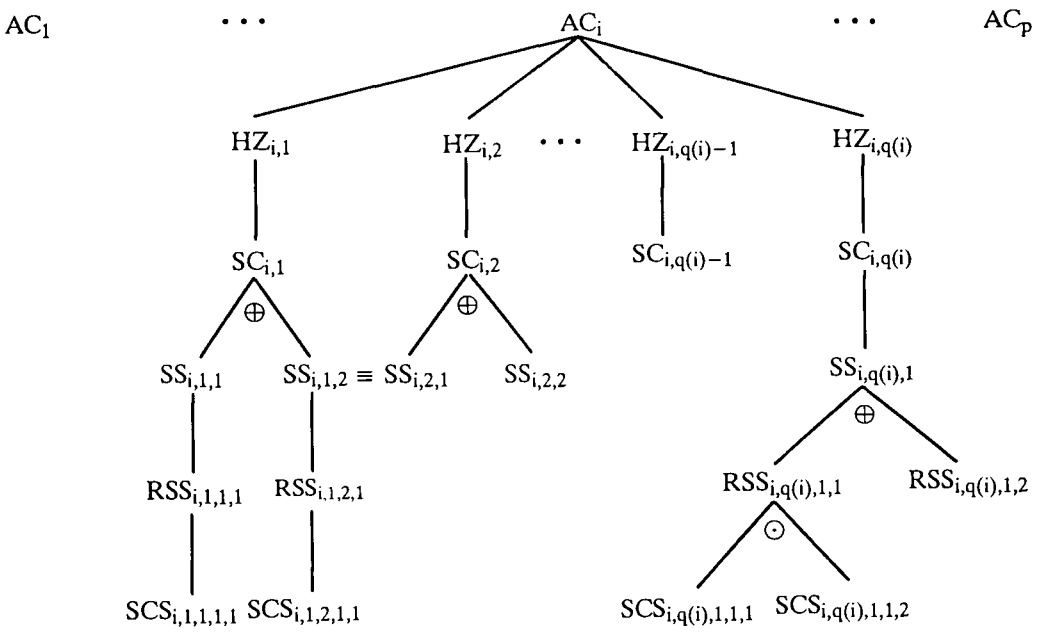


Figure 5.2 Example of a safety specification graph.

5.5.3. Role of SSG

The structured record of the safety specifications embodied in an SSG will provide assistance for subsequent certification and maintenance activities. The SSG can also assist in a number of specific tasks.

- *Qualitative Analysis*. Support is provided by establishing the relationships (which follow from the edges of the component graphs) that must be confirmed to ensure that the specifications maintain safe behaviour.

- *Modification*. A key concern when strategies are modified is *traceability*, that is the ability to trace back from a specification to its origins and to trace forwards to the specifications which are derived from it. Support for traceability is provided by constructing reachability matrices and adjacency matrices of an SSG, determining respectively the set of safety specifications reachable and adjacent to a given safety specification. This enables the localization of the side–effects of a modification and the identification of relationships that must be reconfirmed, thereby increasing the assurance that when changes are necessary they will be complete and consistent.

- *Analysis of Assumptions*. The structure embodied in the SSG can be used to record the dependencies of the relationships on the assumptions over the domains. This record of dependencies can also be used to trace the consequences of establishing specifications based upon discredited assumptions (i.e. those to which subsequent analysis associates low confidence), such as the need to reconfirm conditions or re–evaluate identified options.

5.6. Quality Analysis of Safety Specifications

One important aspect of determining the quality of the specifications for critical real–time systems is the risk analysis of the safety specifications; this aims to determine if the contribution of the software to the overall system risk is acceptable. In order to achieve this aim, a bridge has to be established between the risk analysis of the system and the software. In the context of the methodology described here, this bridge is established through the SSG by formal links between the system requirements and the software requirements. To perform the risk analysis, those circumstances which can violate a specification, and thereby cause the system to enter into a hazard state, have

to be identified and their probability of occurring calculated. Once the risk is quantified we are able to judge whether the risk associated with a specification is acceptable or not (risk assessment). If not, the specification has to be modified or combined with other specifications in order to reduce the risk. As a result, we obtain a more robust safety specification, a specification that can be violated only within an acceptable level of risk. It should be noted that the risk analysis presented in this section does not take into account the consequences of an accident, and thus all accidents are considered to be of equal severity.

During the operation of the system, the occurrence of an initiating event (an event which can lead the system into a hazard state) of an accident sequence /MoD 91/ distinguishes two kinds of system state: safe and unsafe states. The definition of an initiating event ensures that a hazard cannot occur subsequent to a safe state if no initiating event occurs. In terms of the requirements specifications, the concept of an initiating event thus refers to those circumstances which can initiate and lead to the violation of a safety specification.

The quality analysis of the requirements, in each domain of analysis, is performed from two different perspectives: qualitative and quantitative. The purpose of the qualitative analysis is to identify those circumstances which can lead to a specification being violated, and analyse the impact of these violations upon the safety of the system. Such circumstances must entail either the violation of the assumptions upon which a specification is based or the violation of safe states which are related to a particular specification. The quantitative analysis complements the qualitative analysis by attaching occurrence probabilities to these circumstances. In order to ensure that essential system behaviour is not precluded, the restrictions that a safety specification will impose on the mission of the system must also be considered.

5.6.1. Qualitative Risk Analysis

At each level of abstraction, analysis is conducted over safety specifications (descriptions of safe behaviour at the level) and assumptions (properties assumed at the level). In the proposed approach the qualitative analysis is conducted in two stages; firstly we perform a preliminary analysis based on verification and validation, and secondly we conduct a vulnerability analysis of the safety specifications.

5.6.1.1. Preliminary Analysis

Preliminary analysis aims to confirm the quality of the safety specifications by the *verification* and *validation* of the SSG. Verification is the process of checking that the safety specifications are internally consistent (i.e. the requirements captured by the specifications are not contradictory), and that they conform to the specifications stipulated at the previous layer. Validation is the process of checking that a safety specification accurately reflects the requirements and constraints imposed on the system by the user, or some other authority, and does not conflict with the mission requirements. Thus, the use of these terms conforms to the increasingly common interpretation: verification refers to checking that "the system is built right" whereas validation refers to checking that "the right system has been built".

A result of the preliminary analysis is that consistency conditions and safety conditions are confirmed. The *consistency conditions* are those conditions that must be established at each layer of the SSG to confirm that the safety specifications (which relate to the system in a particular mode) are not in conflict with each other (horizontal checks). The *safety conditions* are those conditions that must be established between different layers of the SSG to confirm that safety (absence of hazards) is maintained down the SSG (vertical checks); the actual relationships that need to be confirmed, to ensure compliance between the layers, are determined from the edges of the SSG.

At each layer, any assumptions required to confirm the relationships depicted by the edges of the SSG are recorded. As an example of the safety conditions that must be

verified, we examine the edge (from the SSG in figure 5.2) that connects the safety constraint $SC_{i,1}$ to the safety strategy $SS_{i,1,1}$. Let us suppose that the strategy is based upon assumption A (a property of the physical process); the relationship to be confirmed is then:

$$A \wedge SS_{i,1,1} \Rightarrow SC_{i,1} \qquad f1$$

A result of the preliminary analysis is that the circumstances under which safe behaviour is maintained are clearly scoped and organised to reflect their contribution to safe behaviour at the different levels of abstraction. This activity ensures that the knowledge gained during the development and validation/verification of the safety specifications can be applied effectively during the subsequent risk analysis.

5.6.1.2. Vulnerability Analysis

After performing the preliminary analysis of the safety specifications, the second aspect of the qualitative risk analysis consists of performing a vulnerability analysis of the specifications; this probes each safety specification, and its associated assumptions, to identify any circumstances under which the specification is unable to maintain safe behaviour, i.e. the violation of that specification. Once these circumstances are identified, the safety specifications can be modified to become more robust against possible violations. An initial step in the vulnerability analysis is to negate the relationships obtained during the preliminary analysis and to identify any system states which can lead to occurrence of these circumstances.

For the relationship *f1*, the logical assertion is negated and those plant states (PS) which can lead to the violation of $SC_{i,1}$ are identified:

$$\neg SC_{i,1} \Rightarrow \quad (\neg SS_{i,1,1} \wedge PS) \vee (\neg A \wedge PS) \qquad f2$$

For this relationship, which is associated with the plant level, the subsequent vulnerability analysis of the safety strategy $SS_{i,1,1}$ will identify those conditions that can lead to the violation of $SC_{i,1}$.

Although logical formulae are useful in obtaining a high–level view of the relationship between the specifications and assumptions, such formulae provide limited support for a failure analysis. A suitable representation, for such an analysis, is one which supports the identification of possible failure behaviours that can lead to the identified hazardous states. In this thesis, to perform the vulnerability analysis of the safety specifications, we employ fault tree analysis (FTA) /Vesely 81/ which has been used extensively in the analysis of system safety and more recently in the analysis of software safety /Leveson 91/. A key feature of fault tree analysis which makes it suitable is that the analysis is restricted to the identification of system components and conditions that lead to one particular undesired system state.

To construct a fault tree for the relationship *f2*, the initial step is to identify the undesired state, in this case the negation of the safety constraint $SC_{i,1}$, and then to determine the set of possible causes which can lead to the undesired state (refer to figure 5.3). For the logical formula *f2*, we identify the violation of the assumption and the violation of the safety strategy $SS_{i,1,1}$. The latter has to be further refined in order to identify its primary events.

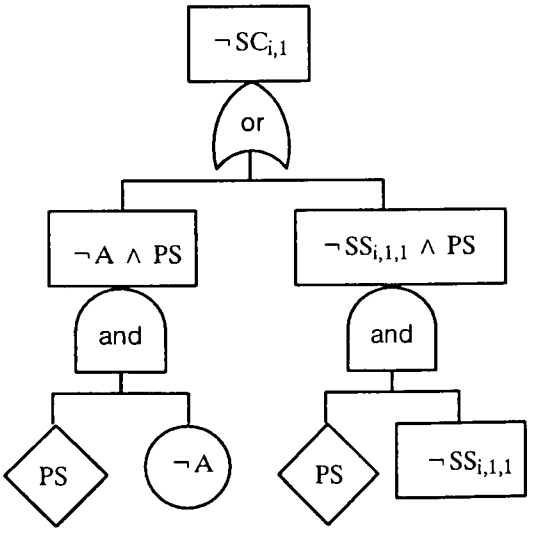


Figure 5.3. Fault tree for relationship f2.

Qualitative risk analysis provides a basis for obtaining more robust safety specifications which will lead to a risk abatement of the overall system. In the approach adopted, the

analysis is performed by employing both formal analysis and fault tree analysis in order to determine the weaknesses of the safety specifications. Once these weaknesses are identified the safety specifications can be modified to incorporate mechanisms which aim to reduce their vulnerability.

5.6.2. Quantitative Risk Analysis

In this section we discuss how a quantitative analysis can complement the qualitative analysis by introducing a measurement of confidence in the quality of the safety specifications. While the latter analysis identifies circumstances which can lead to the violation of the specifications, the former associates probabilities to these circumstances.

Although the qualitative approach strives to achieve total assurance for the safety specifications, there are three basic limitations which indicate that this aim may not be realised. A first limitation stems from the process of capturing user requirements: some faults introduced during the requirements stage may not be removed during the verification process, nor can it be guaranteed that all such faults will be removed during validation. A second limitation arises from observing past experience in the utilization of formal techniques, which shows that a formal verification may itself contain faults /Miller 90/. The third limitation is related to the confidence that can be placed on the assumptions upon which a specification is based. From these limitations we infer that even after performing the qualitative analysis we are still faced with uncertainties concerning the quality of the safety specifications, and hence it is still necessary to quantify these uncertainties in order to obtain a measurement of confidence in the quality of the safety specifications. In other words, the aim is to obtain an early prediction of the contribution of the software to the risk of the system.

The quantitative approach proposed here will not address the first two limitations mentioned above, which have been treated elsewhere /Ramamoorthy 85/. Instead, our concern is with the assumptions that are usually made in obtaining a mathematical model

of the real world. For example, for applications such as process control systems, if an assumption about the behaviour of the physical process is violated, the probability of a hazard occurring could very easily increase when the actual behaviour of the physical process starts to deviate from the behaviour expected on the basis of the defective assumption.

In this thesis, in order to perform the quantitative evaluation of the safety specifications we employ fault tree analysis techniques /Vesely 81/. Although fault tree analysis is mainly considered to be a technique for qualitative analysis, it is nevertheless often used for quantitative evaluations. In the qualitative approach, an undesired state (hazard state) is specified, and the system is analysed in terms of its requirements and its environment in order to find all possible ways that the undesired state can be reached. The quantitative approach calculates the probability that the system will enter an undesired state based on probabilistic information about the occurrence of primary events (events that have not been further refined, and for which probabilities have to be supplied). Once the *minimal cut sets* (the smallest combination of primary events which, if they all occur, will lead to an undesired state) are obtained the quantification of the fault tree is straightforward.

Fault tree analysis techniques have been employed in the various development stages of safety–critical software /Leveson 91b/. In this work fault tree analysis techniques were used in the quantitative evaluation of the safety specifications, however, other alternative methods, such as Markov models and stochastic Petri nets, could have been used. Another remark concerns the fact that, the type of quantitative evaluation performed on the safety specifications differs from other evaluations more usually performed on software, such as the use of reliability growth models to predict the reliability of software /Laprie 91/, in the sense that, our basic intention is to obtain a higher confidence in the quality of the safety specifications before proceeding to other phases of software development. However, only recently has some work been performed

in estimating the reliability of software at the early stages of the development /Wohlin 92/.

To associate occurrence probabilities to those circumstances which can lead to the violation of the specification, such as plant states and violation of assumptions, might not be a difficult task. On the other hand, associating occurrence probabilities to the violation of certain conditions which depend on a software implementation is more problematic because during the requirements phase of software development sufficient design and implementation information is not yet available. Instead of estimating the probability of a condition to be violated, target probabilities demanded from the higher level safety specifications such as hazards, can be used. However, once a specification is sufficiently detailed, currently available techniques which attempt to make early predictions of the software reliability can be used, such as: metrics /Ramamoorthy 85/, product−in−a−process /Laprie 92/, and execution of the specifications /Wohlin 92/.

In the following we describe an approach to conduct the qualitative analysis of the safety specifications which matches the framework for requirements analysis. During the first phase, the Safety Plant Analysis, the assumptions associated with the safety strategies are identified and quantified in terms of the probability of being violated. At this level of abstraction, these assumptions, which are the physical laws and the rules of operation, are violated when they cease to represent the actual behaviour of the plant. In the second phase, the Safety Plant Interface Analysis, the quantitative approach to assurance is performed in terms of the properties of sensors and actuators, such as the uncertainties in value and domain, and their probability of failure. In the third phase, the Safety Controller Analysis is performed in terms of the properties of the components of the system, and their probability of failure.

5.6.3. Risk Assessment

Another stage of the quality (risk) analysis is to perform the risk assessment of the safety specifications. This is a judgement based on the estimated risk which provides guidance

for high level decisions, usually associated with the process of requirements analysis. The results obtained from the quantitative analysis should be considered as a relative measurement of how effective a given strategy is in reducing the risk of a hazard, compared to the results obtained for alternative strategies. Hence it is most useful in determining which strategy or combination of strategies is most suitable for the risks involved (the choice of a strategy might also be influenced by constraints imposed by the implementation, e.g. availability of sensors and actuators with the required properties). Also, if the utilization of more than one strategy is required, this preliminary risk analysis facilitates the search for a suitable combination of the available strategies in order to avoid common mode failures.

By performing a quantitative assessment of the requirements specifications, we are able to analyse the feasibility of further development of the software. The feasibility analysis of a software specification represents the study performed to determine if the production of software that implements the software specification should proceed, or not. The primary concerns are mainly in terms of economic factors, such as whether the development costs will be recuperated from the service provided, and do the benefits outweigh the risks, as well as technical factors, such as system complexity and system novelty. The feasibility analysis of the software specification is based on:

a. the level of confidence, or the evaluation of effectiveness /Leveson 90/, of the formal and informal analysis of the safety requirements;

b. a prediction of the achievable and certifiable system reliability.

As far as the first issue is concerned, little work has been done until now; the challenge here is to find metrics which would allow a probabilistic measure to be derived from the approaches employed in the qualitative assessment of the safety requirements, so that we can quantify the level of confidence that can be placed on the qualitative assessment. The second issue is closely related to the application of software reliability models to the phase of requirement analysis; however, two major concerns may be raised /Musa 90/:

a. how accurate are the reliability predictions made at this time?

b. is the effort of modelling the reliability at this stage worth it?

For the first concern, new approaches have been proposed /Laprie 92/ which could allow the evaluation of software to levels above what can be attained using current methods; these new approaches include product–in–a–process assessment which incorporates information concerning the development process, and also utilises historical information from previous products when there is a gradual evolution in the products. For the second concern, a significant benefit obtained by performing a reliability assessment at the requirements stage is that it provides an early prediction of the attainable reliability for the system, which can be used as an early warning mechanism for the detection of difficult (and high risk) systems. The high cost of critical real–time systems and the risk associated with the production of the software for such systems provide support for the suggested use of reliability assessment at the requirements stage.

5.7. Mission and Safety Analysis

The primary aim of the quality analysis presented in this thesis is to reduce the system risk. However, it is usually impossible to maintain a complete dichotomy between the mission and safety aspects, and it would be futile to impose safety requirements which were so stringent that the system could not satisfy its mission. To complement the risk analysis, the impact of the safety specifications on the mission of the system must also be considered. Such an analysis involves relating the different safety specifications to the mission requirements that can be affected by them. If analysis of the mission requirements follows the framework described in section 5.3, leading to the construction of a *Mission Specification Graph* (MSG), a comparison between the safety specifications and mission specifications (requirements specifications for the mission) is made possible. During the development of a robust safety specification it would be possible to identify the mission specifications that may be influenced by that specification by inspecting the variables that it constraints and relating these variables to the mission

specifications at the same level of abstraction. Once the relevant mission specifications have been identified an informal analysis of the restrictions that the safety specification imposes on the mission can be conducted. An example of such an analysis is presented elsewhere /de Lemos 92b/.

5.8. Concluding Remarks

In conclusion, the aim of this chapter was to present a methodology for the requirements analysis of critical real–time systems, based on the application of formal techniques. The framework of the methodology is divided into domains of analysis, where for each domain we identify the activities to be performed. The general approach adopted for the methodology attempts to follow closely the usual practices in system engineering, specifically for process control systems. For each domain the analysis of different views of the requirements can be performed independently, by applying the most appropriate method.

Also in this chapter we have described the means which allow us to assess, qualitatively and quantitatively, the quality of the safety specifications. The qualitative analysis aims to verify and validate the safety specifications, and the quantitative evaluation aims to measure the probability of a hazard occurring due to the impact of violations of the assumptions, and the failure rates of sensors and actuators and controller components. The adoption of a framework for requirements analysis and a hierarchical structure for the safety specifications facilitates this quality assessment, by enabling the analysis to be performed in clearly defined stages. The approach to the quality assessment of safety specifications presented in this chapter either provides evidence that the safety specifications cannot achieve the quality that is required, or increases confidence that the required quality has been attained.

The general methodology has been shown to be feasible and practical by applying it to a set of case studies, such as a train set /de Lemos 92b/, a chemical batch processing plant

/Anderson 93/, and a nuclear power reactor /Saeed 93/. In the next chapter the case study of the train set will be discussed in detail.

Chapter 6

Case Study: Train Set Example

6.1. Introduction

In this chapter, the aim is to exemplify some of the concepts of the methodology for the requirements analysis of critical real–time systems, by means of a case study. The adopted case study is that of a train set crossing, which raises safety–critical issues that are similar to those found at the traditional level crossing (i.e. road–rail) /Gorski 86, Leveson 87/. The main reason for choosing this case study is the availability in the Department of Computing Science of a large train set /Conner 90/ which permits the implementation of the requirements specifications produced from the analysis conducted. Another obvious benefit from using a train set instead of a real railway system is that safety specifications can be analysed and implemented without endangering the travelling public. Apart from the train set crossing, the methodology has also been applied to the requirements analysis of other case studies. In /Anderson 93/ we presented two case studies which showed how the methodology can applied to hybrid systems: the theoretical "cat and mouse" example /Maler 92/, and a chemical batch processing plant /Saeed 90/. In another paper, we discuss the usefulness of applying formal methods in the requirements analysis of a nuclear reactor /Saeed 93/.

In this case study we intend to illustrate the following features of the methodology. In the next four sections, we present, in terms of the train set crossing, the activities that have to be performed in the different phases of the framework, together with the qualitative analysis; whenever temporal analysis is required we employed the concepts of the E/A model. In section 6, the requirements analysis is summed up by presenting the safety specification hierarchy of the case study. In section 7, the quantitative assessment of the safety strategy of the crossing section is presented. Finally, in section 8, we present concluding remarks.

6.2. Conceptual Analysis

The physical process of the example case study consists of two track circuits *Cp* and *Cs*, and two types of trains – primary (*Trp*) and secondary (*Trs*). The circuits are divided into sections (numbered in a clockwise direction) and there are two separate crossing sections *CC(c,a)* and *CC(c,b)* at which the circuits intersect (i.e. a section that is part of both circuits). Trains of type *Trp* travel around circuit *Cp* and trains of type *Trs* travel around circuit *Cs*. Both types of train travel in one direction (clockwise) only, hence trains cannot reverse around the circuits. The longest train is shorter than the smallest section. A crossing section is that part of the track which consists of the sections (one from each circuit) at which the two circuits intersect. The circuits *Cp*, *Cs* and the crossing sections *CC(c,a)* and *CC(c,b)* are illustrated in figure 6.1.

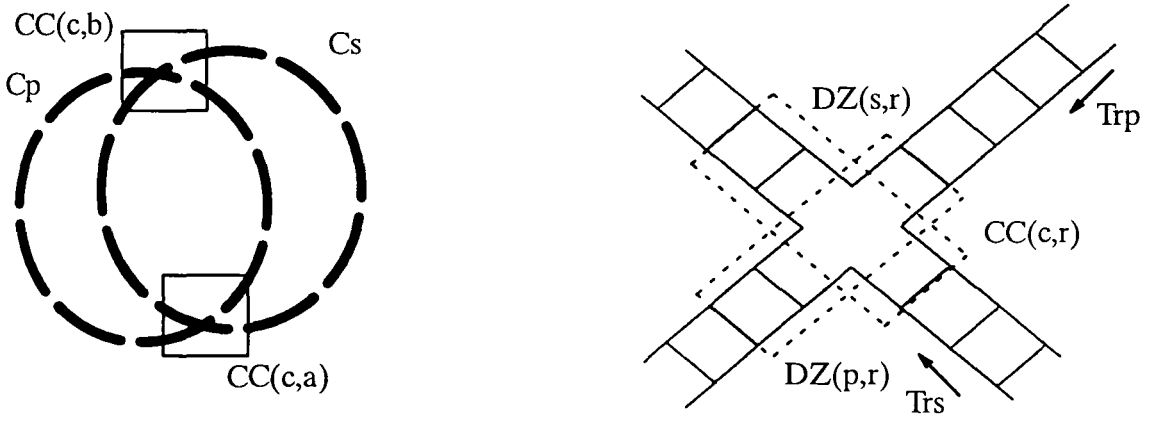


Figure 6.1. The train set circuits and crossing section.

As far as the safety requirements are concerned, several accidents are possible with this system, but we only consider the following two accidents:

a. trains of the same type collide;

b. trains of different type collide.

The mission requirements for the train set have to ensure that all trains complete their journeys, while allowing all (legitimate) concurrent movements of the trains. This mission is in fact a special case of the Merlin–Randell problem /Merlin 78/, which is

concerned with train journeys on arbitrary tracks. This problem has been heavily studied, and involves a complex analysis of synchronization strategies /Koutny 86/. A special case of synchronization is that the primary trains must always take priority over the secondary trains at the crossing sections – that is, a primary train must not be made to wait for a secondary train (c.f. priority of trains over road vehicles at a level crossing). Although in this work we are primarily concerned with the analysis of the safety requirements, the above priority mission requirement for priority of trains will be considered in detail in order to show how the analysis of timeliness requirements is performed in the proposed framework.

For the train set, one of the main advantages in separating the mission requirements from the safety requirements is that synchronization issues can be ignored in the analysis of the safety controller. As a result issues involving problems of optimal use of the sections of the tracks, routes of the trains, and to start moving stopped trains can be ignored during the analysis of the safety requirements. By focusing on the safety issues only, a thorough verification of the safety controller specification becomes practical. However, as will be shown later, the mission requirements must still be considered in order to ensure that any safety specifications do not lead to inconsistencies with the mission requirements.

6.3. Safety Plant Analysis

The Safety Plant Analysis is concerned with the identification of hazards, safety constraints, and the definition of safety strategies. We consider each potential disaster separately.

General Model

The type of circuit is denoted by $c \in L, L = \{p, s\}$, the crossing section by $r \in R, R = \{a, b\}$, the trains which run on Cc are denoted by $x, y \in Trc = \{1, \ldots, Ntc\}$ (Ntc is the number of trains in a circuit), the sections of Cc are denoted by $i, j, k, m, n \in Sc, Sc = \{0, \ldots, Nsc\}$ (Nsc is the number of sections in a circuit), and the velocity of a train is denoted by u, v

$\in V, V = \{u \in \mathbb{R} \mid 0 \leq u \leq Vm\}$ (Vm is the maximum velocity of a train). Addition $\oplus$ and subtraction $\ominus$ on circuit section numbers are performed modulo the number of sections of the circuit. The danger zone on circuit Cc for $CC(c,r)$ is defined as: $DZ(c,r) = \{CC(c,r), CC(c,r) \oplus 1\}$. The danger zones $DZ(p,r)$ and $DZ(s,r)$ for a crossing section $CC(c,r)$ are illustrated in figure 6.1. We define two functions $dmax(c,i,j)$ and $dmin(c,i,j)$ which give the maximum and minimum distance that a train in section i must travel to enter section j on circuit Cc. That is, $dmax(c,i,j)$ gives the distance to section j when a train is at the start of section i, and $dmin(c,i,j)$ the distance when a train is at the end of section i. The behaviour of the system is captured by three state variables *Ptrain*, *Rtrain* and *Vtrain* (a continuous variable) described below.

No.	Name	Range	Comments
p_1	*Ptrain*	$Sp^{Ntp} \times Ss^{Nts}$	The position of each train expressed as a section number, that is, the section containing the front of a train.
p_2	*Rtrain*	$Rp^{Ntp} \times Rs^{Nts}$	The reservation sets of the trains, where $Rc = \mathcal{P}(Sc)$.
p_3	*Vtrain*	$V^{(Ntp+Nts)}$	The velocity of the trains.

The behaviour of trains is captured by the state variable $Ptrain(c,x)$ which denotes the position of train x on circuit Cc. In the general model of the train set it is assumed that the position of the front of the trains is known at any time point during the system lifetime (captured by *Ptrain*). Hence, at the start of the system lifetime the fronts of the trains are positioned in specified known sections.

At this stage we introduce the notion of a "reserved section", which is a section that is reserved by a train, implying that other trains cannot either enter or reserve it. The notion of reserved sections of a train is captured by the state variable $Rtrain(c,x)$ which denotes the reservation set of train x on circuit Cc. The velocity of a train is captured by the state variable $Vtrain(c,x)$ which denotes the velocity of train x on circuit Cc. $\Gamma\mathbf{H}$ is the set of all functions H: $T \rightarrow Sp^{Ntp} \times Ss^{Nts} \times Rp^{Ntp} \times Rs^{Nts} \times V^{(Ntp+Nts)}$; these functions define an history, that is a mapping from each time point in the system lifetime to a system state. A history from $\Gamma\mathbf{H}$ satisfies a safety specification, if and only if the system (respectively

history) predicates that describe what are invariant (respectively history) relations for that history.

For the train set we identify two rules of operation:

a. trains of type *Trp* travel around circuit *Cp* and trains of type *Trs* travel around circuit *Cs*;

b. both types of trains travel in one direction (clockwise) and only advance one section at a time.

As an example, we present the formalization of rule of operation *b*, as a history predicate. For any interval $[T_0, T_1]$ if the position of a train x at time point T_1 does not equal the position of the train at time point T_0 there must be a time point t in $[T_0, T_1]$ such that the position of train x at t is equal to the section that immediately follows the position at time point T_0.

$$\mathrm{Hr}_1.\ \forall c \in L\text{: } \forall x \in Trc\text{:}[\, Ptrain(c,x)(T_0) \neq Ptrain(c,x)(T_1) \Rightarrow$$
$$\exists t \in [T_0, T_1]\text{: } Ptrain(c,x)(t) = Ptrain(c,x)(T_0) \oplus 1].$$

The following assumptions are made regarding the behaviour of the trains on the circuits:

a. a train will stop within a section (that is when a train is requested to stop at the beginning of a section, the train stops before it reaches the end of the section);

b. the longest train is shorter than the smallest section;

c. the average velocity of a moving train x that travels through a section on circuit *Cc* has a lower bound of $Vmin(c,x)$ and an upper bound of $Vmax(c,x)$.

The hazardous states for the collision of trains of the same type are identified by considering the relative positions of the trains on a circuit; a collision can only occur if some part of two trains are in the same section. Hence, since the longest train is smaller than the smallest section, a state is deemed to be hazardous if the front of one train is in the same, or adjacent, section as the front of another train.

The hazardous states for the collision of trains of different type are identified by considering the relative positions of the primary and secondary trains at a crossing section. A collision involving a primary train and a secondary train can occur only when both trains are in the same crossing section. Since the length of a train is less than a section, a train can only be in a crossing section if the front of that train is in the crossing section or the section that follows the crossing section. To express the hazard associated with a crossing section, for each circuit we introduce the notion of danger zones (one for each crossing section) as the set of sections in which the front of a train can be while that train is in a crossing section. We conclude that the hazardous states for the collision of trains of different type are those in which the front of a primary train and the front of a secondary train are in the danger zones of the same crossing section. The danger zones $DZ(p,r)$ and $DZ(s,r)$ of a crossing section $CC(c,r)$ are illustrated in figure 6.1.

6.3.1. Collision of Trains of Same Type

Circuit Safety Constraint

The hazardous states ($HZ_{1,1}$) for the collision of trains of the same type on a circuit Cc can be expressed, in the general model, by the system predicate:

$$\forall c \in L: \exists x,y \in Trc: [x \neq y \wedge Ptrain(c,x) \in \{Ptrain(c,y) \ominus 1, Ptrain(c,y)\}].$$

Thus we deduce that a safety constraint of the form "for any two trains there must be at least one section between the sections containing the fronts of the trains" will, if maintained on circuit Cc, prevent the occurrence of these hazardous states on Cc. This safety constraint $SC_{1,1}$ is expressed in terms of a system predicate:

$$\forall c \in L: \forall x,y \in Trc: [x \neq y \Rightarrow Ptrain(c,x) \notin \{Ptrain(c,y) \ominus 1, Ptrain(c,y), Ptrain(c,y) \oplus 1\}].$$

If each train has to reserve at most three sections, as a consequence we can deduce a limit on the number of trains in terms of the sections: $Nsc > 3Trc$ (for each circuit).

Circuit Safety Strategy

Once the safety constraint $SC_{1,1}$ is specified, we have to devise a safety strategy which maintains $SC_{1,1}$. The safety strategy $SS_{1,1,1}$ which we propose for a circuit *Cc* is based on a reservation scheme, and can be formalized as two system predicates:

ssa. for any train, the current section (i.e. the position of the front of the train) and the section behind the current section must always be reserved for that train;

$$\forall c \in L: \forall x \in Trc: [\{Ptrain(c,x) \ominus 1, Ptrain(c,x)\} \subseteq Rtrain(c,x)]; \text{ and}$$

ssb. no section can be reserved by more than one train.

$$\forall c \in L: \forall x, y \in Trc: [x \neq y \Rightarrow Rtrain(c,x) \cap Rtrain(c,y) = \emptyset].$$

We say that a history H from $\Gamma\mathbf{H}$ satisfies safety strategy $SS_{1,1,1}$ if and only if rules *ssa* and *ssb* are invariant relations (i.e. they are satisfied at all time points in T) for that history.

Lemma 6.1.

A history H that satisfies the safety strategy $SS_{1,1,1}$ must satisfy safety constraint $SC_{1,1}$.

Proof. (By contradiction.)

Assume $\exists H \in \Gamma\mathbf{H}$: H **sat** $SS_{1,1,1} \wedge \exists t \in T$: [$H$ **sat** ($\forall c \in L$: $\exists x, y \in Trc$:

$$x \neq y \wedge Ptrain(c,x) \in \{Ptrain(c,y) \ominus 1, Ptrain(c,y), Ptrain(c,y) \oplus 1\})@t].$$

Then from rule *ssa* of $SS_{1,1,1}$:

$$H \text{ sat } (\{Ptrain(c,x) \ominus 1, Ptrain(c,x)\} \subseteq Rtrain(c,x) \wedge$$

$$\{Ptrain(c,y) \ominus 1, Ptrain(c,y)\} \subseteq Rtrain(c,y)\ )@t,$$

hence H **sat** $(Rtrain(c,x) \cap Rtrain(c,y) \neq \emptyset)@t$. But this contradicts rule *ssb* of $SS_{1,1,1}$.

Therefore $\forall H \in \Gamma\mathbf{H}$: H **sat** $SS_{1,1,1} \Rightarrow H$ **sat** $SC_{1,1}$.

Corollary 6.2.

A history H that satisfies the safety strategy $SS_{1,1,1}$ for both circuits must satisfy the safety constraints $SC_{1,1}$ of the respective circuits.

Proof. Immediately from lemma 6.1.

Circuit Safety Strategy and Mission Requirements

The aim of the framework is to restrict the requirements analysis of safety–critical software to the safety issues of the system. However, it is usually impossible to maintain a complete dichotomy between the mission and safety requirements, because it would be futile to impose safety requirements which were so stringent that the system could not satisfy its mission. Here we consider the impact of safety strategy $SS_{1,1,1}$ on the mission requirements of the train set. More specifically, our aim is to check that $SS_{1,1,1}$ does not lead to the definition of an over restrictive safety controller strategy; the safety controller should not impose restrictions on the movement of trains in addition to those imposed by the mission controller (except when the system enters an unsafe state). The restrictions imposed on the movement of trains can be stated in terms of the reserved sections; the upper bound in the number of sections that have to be reserved by the safety controller for a train should be less than the lower bound in the number of sections that have to be reserved by the mission controller.

To determine the upper bound of the reservation set of the safety controller we consider how the reservation set of a train x is modified as the train travels around the circuit Cc. Firstly, while a train is travelling in a section, the current section and the previous section are reserved. Secondly, we make the observation that before train x enters a new section it must reserve that section, i.e. the sections $Ptrain(c,x) \ominus 1$, $Ptrain(c,x)$ and $Ptrain(c,x) \oplus 1$ must be reserved. Thirdly, immediately after the front of train x enters a new section the position of the train is updated, hence sections $Ptrain(c,x) \ominus 2$, $Ptrain(c,x) \ominus 1$ and $Ptrain(c,x)$ are reserved. Therefore, in order for the safety controller not to affect the mission controller, an upper bound for the reservation set is: $\{Ptrain(c,x) \ominus 2$, $Ptrain(c,x) \ominus 1$, $Ptrain(c,x)$, $Ptrain(c,x) \oplus 1\}$.

From the above analysis we can conclude that the safety strategy is not too restrictive on the movement of trains if the lower bound of the reservation set of the mission controller,

is at least $\{Ptrain(c,x)\ominus 2, Ptrain(c,x)\ominus 1, Ptrain(c,x), Ptrain(c,x)\oplus 1\}$. If this condition could not be satisfied then the mission requirements would have to be modified or a new safety strategy devised.

Here we construct a modified version of $SS_{1,1,1}$, denoted by $SS_{1,1,1}{}^*$, that includes the upper bound on the size of the reservation set; the two rules for $SS_{1,1,1}{}^*$ are given below:

$$ssa^*.\ \forall c \in L:\ \forall x \in Trc:\ [\{Ptrain(c,x)\ominus 1, Ptrain(c,x)\} \subseteq Rtrain(c,x) \wedge$$
$$Rtrain(c,x) \subseteq \{Ptrain(c,x)\ominus 2, Ptrain(c,x)\ominus 1, Ptrain(c,x), Ptrain(c,x)\oplus 1\}];$$

$$ssb^*.\ \forall c \in L:\ \forall x,y \in Trc:\ [x \neq y \Rightarrow Rtrain(c,x) \cap Rtrain(c,y) = \emptyset].$$

Lemma 6.3.

A history that satisfies the safety strategy $SS_{1,1,1}{}^*$ must satisfy safety constraint $SC_{1,1}$.

Proof. Follows directly from lemma 6.1, and the fact that $SS_{1,1,1}$ is a consequent of $SS_{1,1,1}{}^*$.

Corollary 6.4.

A history that satisfies the safety strategy $SS_{1,1,1}{}^*$ for both circuits must satisfy safety constraints $SC_{1,1}$ for the respective circuits.

Proof. Immediately from lemma 6.3.

6.3.2. Collision of Trains of Different Type

The following analysis can be applied to both crossing sections provided they are sufficiently far apart. That is, on both circuits there must be at least one section between the two crossing sections (which will ensure that the danger zones do not overlap).

Crossing Section Safety Constraint

Let $CC(p,r)$ (resp. $CC(s,r)$) denote the number of the section of Cp (resp. Cs) that is part of $CC(c,r)$. The danger zone for $CC(c,r)$ is defined as: $DZ(c,r)=\{CC(c,r), CC(c,r)\oplus 1\}$.

The hazardous states $HZ_{2,1}$ for the crossing section $CC(c,r)$ says that some part of a primary or secondary train are in the danger zone $DZ(c,r)$, which can be expressed by the system predicate:

$$\forall c \in L: \exists x \in Trc: [Ptrain(c,x) \in DZ(c,r)].$$

Thus we deduce that a safety constraint of the form "either the front of no primary train is in the danger zone $DZ(p,r)$ or the front of no secondary train is in the danger zone $DZ(s,r)$" will, if maintained, prevent the occurrence of any hazardous state. This safety constraint $SC_{2,1}$ can be expressed in terms of a system predicate:

$$\exists c \in L: \forall x \in Trc: [Ptrain(c,x) \notin DZ(c,r)].$$

Assuming that we wish to have at least one train on each circuit we can deduce that $Nsp \geq 3 \wedge Nss \geq 3$.

Crossing Section Safety Strategy

The safety strategy $SS_{2,1,1}$ we propose for the crossing sections is based on a modification of the reservation scheme. The two basic rules are formalized as two system predicates:

ssa. if any train x on circuit c is in a danger zone then the crossing section contained within that danger zone is reserved for that train;

$$\forall c \in L: \forall x \in Trc: [Ptrain(c,x) \in DZ(c,r) \Rightarrow CC(c,r) \in Rtrain(c,x)];$$

ssb. section $CC(p,r)$ and section $CC(s,r)$ cannot both be reserved.

$$\exists c \in L: [\forall x \in Trc: CC(c,r) \notin Rtrain(c,x)].$$

We will say that a history H from $\Gamma\mathbf{H}$ satisfies this safety strategy if and only if system predicates *ssa* and *ssb* are invariant relations for that history.

Lemma 6.5.

A history that satisfies the safety strategy $SS_{2,1,1}$ for the crossing section must satisfy safety constraint $SC_{2,1}$.

Proof. (By contradiction.)

Assume $\exists H \in \Gamma\mathbf{H}$: H **sat** $SS_{2,1,1} \wedge \exists t \in \mathbf{T}$:

$$[H \textbf{ sat } (\exists x \in Trp: Ptrain(p,x) \in DZ(p,r) \wedge \exists y \in Trs: Ptrain(s,y) \in DZ(s,r))@t].$$

Then from rule *ssa* of $SS_{2,1,1}$:

$$H \textbf{ sat } (CC(p,r) \in Rtrain(p,x) \wedge CC(s,r) \in Rtrain(s,y))@t.$$

But this contradicts rule *ssb* of $SS_{2,1,1}$. Therefore $\forall H \in \Gamma\mathbf{H}: H$ **sat** $SS_{2,1,1} \Rightarrow H$ **sat** $SC_{2,1}$.

Corollary 6.6.

A history that satisfies the safety strategy $SS_{2,1,1}$ for both crossing sections must satisfy safety constraint $SC_{2,1}$ for each crossing section.

Proof. Immediately from lemma 6.5.

Combination of Safety Strategies

In terms of the two circuits and the two crossing sections the safety controller must implement the safety strategies for circuits *Cp* and *Cs* and for the crossing sections *CC*(*c*,*a*) and *CC*(*c*,*b*). Here we analyse the relationship between these safety strategies and the restrictions imposed on the size of the circuits and the number of trains.

By inspecting the rules of the safety strategies we can deduce that the following three rules are equivalent to the conjunction of the safety strategies for the two circuits and the two crossing sections:

$$a.\ \forall c \in L: \forall x \in Trc: [\{Ptrain(c,x) \ominus 1, Ptrain(c,x)\} \subseteq Rtrain(c,x) \wedge$$
$$Rtrain(c,x) \subseteq \{Ptrain(c,x) \ominus 2, Ptrain(c,x) \ominus 1, Ptrain(c,x), Ptrain(c,x) \oplus 1\}];$$

$$b.\ \forall c \in L: \forall x, y \in Trc: [x \neq y \Rightarrow Rtrain(c,x) \cap Rtrain(c,y) = \emptyset];$$

$$c.\ \forall r \in R: \exists c \in L: [\forall x \in Trc: CC(c,r) \notin Rtrain(c,x)].$$

Rule *a* is concerned with the reservation of sections (i.e. maintaining the reservation sets) and rules *b* and *c* impose mutual exclusion conditions over the reservation sets. We must

confirm that the above three rules are not in conflict. The set of states which satisfy the three rules are characterized by the following conditions over the relative position of the trains:

$$\forall c \in L: \forall x, y \in Trc: [Ptrain(c,x) \in \{Ptrain(c,y) \ominus 1, Ptrain(c,y)\} \wedge$$
$$\forall r \in R: \exists c \in L: \forall x \in Trc: Ptrain(c,x) \notin DZ(c,r)].$$

These states are those that satisfy the circuit and crossing section safety constraints, hence the restrictions on the circuit size are obtained by examining the restrictions for the safety constraints: $Nsp \geq \max(2Trp, 3) \wedge Nss \geq \max(2Trs, 3)$.

The above restrictions are minimal constraints on the circuit size. For practical purposes, issues others than the satisfaction of the safety strategies must be considered, such as the ability of the safety controller to implement the safety strategies in the identified states and properties of the physical process. For example, the conditions given above do not preclude the possibility of a deadlock in the physical process. To prevent this situation, the conditions on the circuit sizes should be strengthened to strict inequalities.

Inspection of the three rules above also gives insight into the implementation of the safety strategies. More specifically, the reservation sets must be maintained for each circuit, and mutual exclusion must be achieved for reserved sections for the sections of each circuit and the sections $CC(p,r)$ and $CC(s,r)$ (i.e. the sections that form $CC(c,r)$).

Crossing Section Safety Strategy and Mission Requirements

Although safety strategy $SS_{2,1,1}$ is sufficient to prevent collisions involving primary and secondary trains, it takes no account of the mission requirement that trains on the primary circuit have priority over secondary trains; there is no mechanism which establishes that primary trains should pass first. The priority requirement between the primary and secondary trains is similar to the requirement of maintaining a safe distance (minimal separation) between trains where the notion of sections of constant length is inappropriate. A typical example is the control of underground trains where the utilization of tracks must be optimized. The approach taken to satisfy the priority

requirement will be based on a comparison of the relative velocities of the primary and secondary trains as they approach and cross the crossing section. In the following we consider how the E/A model can be used to analyse this priority requirement. To perform the requirements analysis the physical laws involving the velocity of the trains must be captured in terms of the THL model.

Physical Laws

In the following we present the laws (as history predicates) which capture the relevant relationships involving the velocity of the trains. Hr_1 captures the relationship between intervals, the velocity and position of trains. Hr_2 captures the relationship between the velocity of a train and the functions $Vmax(c,x)$ and $Vmin(c,x)$ (both of which are non−zero) that give the maximum and minimum average velocity of train x on circuit Cc.

Hr_1 − For any train, if during any interval $[T_0, T_1]$ the train travels more than the maximum distance between the section it occupies at T_0 and section i, then the train must occupy section i at some time point in $[T_0, T_1]$; and if the train travels less than the minimum distance between the section it occupies at T_0 and section i, then the train is never in section i during $[T_0, T_1]$.

$$\forall c \in L: \forall x \in Trc: \forall i \in Sc:$$

$$\int_{T_0}^{T_1} Vtrain(c,x)(t)dt \geq dmax(c,Ptrain(c,x)(T_0),i) \Rightarrow$$

$$\exists t \in [T_0, T_1]: Ptrain(c,x)(t) = i \wedge$$

$$\int_{T_0}^{T_1} Vtrain(c,x)(t)dt < dmin(c,Ptrain(c,x)(T_0),i) \Rightarrow$$

$$\forall t \in [T_0, T_1]: Ptrain(c,x)(t) \neq i.$$

Hr_2 – If any train x on circuit Cc travels a distance of ΔL (the length of the smallest section) during an interval $[T_0, T_1]$ without stopping then the average velocity $(\Delta L/(T_1 - T_0))$ during $[T_0, T_1]$ is bounded by the functions $Vmax(c,x)$ and $Vmin(c,x)$.

$$\forall c \in L: \forall x \in Trc: \int_{T_0}^{T_1} Vtrain(c,x)(t)dt = \Delta L \wedge$$

$$\forall t \in [T_0, T_1]: Vtrain(c,x)(t) \neq 0 \Rightarrow Vmax(c,x) \leq \Delta L/(T_1 - T_0) \leq Vmin(c,x).$$

The above laws must be validated as properties of the system, in this case by comparison with the behaviour of the train set; Hr_1 follows from the physics of the system (and the definitions of *dmax* and *dmin*) and Hr_2 depends on the observed minimum and maximum velocity of the trains. During the subsequent analysis these laws are treated as axioms of the model of the system. More precisely, henceforth we consider only those histories of **ΓH** for which Hr_1 and Hr_2 are history relations.

E/A Model in the Safety Plant Analysis

By inspection of rules *ssa* and *ssb* of $SS_{2,1,1}$, the priority requirement can be stated in terms of the reservation scheme as: the section $CC(s,r)$ cannot be reserved by a secondary train if any primary train may need to reserve section $CC(p,r)$ before the secondary train releases section $CC(s,r)$.

The actions for the secondary trains and primary trains are identified by examining the relationship between the reservation of $CC(c,r)$ and the position of the trains. We identify the following actions:

$APp(t,i)$ – a primary train approaches $DZ(p,r)$ after a secondary train has entered $CC(s,r) \ominus 1$ (i.e. the primary train may attempt to reserve $CC(p,r)$ before the secondary train will release $CC(s,r)$).

$APs(t,i)$ – a secondary train approaches $DZ(s,r)$ after it has entered $CC(s,r) \ominus 1$ (i.e. $CC(s,r)$ may need to be reserved by the train).

$STc(t,i)$ – a train of type Trc has stopped in section $CC(c,r)\ominus 1$ (i.e. the train was not allowed to reserve $CC(c,r)$).

$CZc(t,i)$ – a train of type Trc is crossing the danger zone $DZ(c,r)$ (i.e. the train must have reserved $CC(c,r)$).

The timing constraints must ensure that a secondary train cannot perform the action $CZs(t,i)$ if a primary train will need to perform the action $CZp(t,i)$ before completion of the action $CZs(t,i)$. To analyse this situation in the E/A model, we describe the utility functions $U_{Trp}(t,i)$ for actions $APp(t,i)$ and $CZp(t,i)$, and $U_{Trs}(t,i)$ for actions $APs(t,i)$, $STs(t,i)$ and $CZs(t,i)$, as illustrated by the graphs in figure 6.2. These graphs depict

a. the case when a secondary train does not stop when approaching $DZ(s,r)$ (i.e. the train is allowed to reserve $CC(s,r)$), and

b. the case when a secondary train must stop (i.e. the train is not allowed to reserve $CC(s,r)$) immediately after entering $CC(s,r)\ominus 1$.

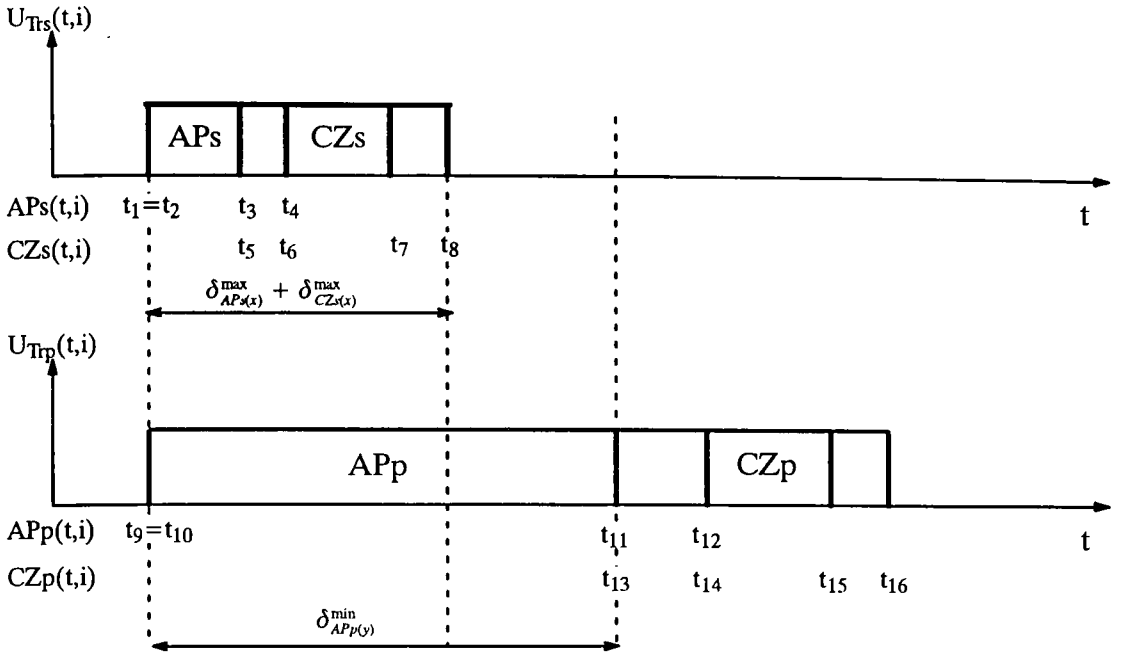
The duration of the utility function of action $ACc(t,i)$ for train x on circuit Cc is abbreviated to $\delta_{ACc(x)}$. For example, $\delta_{APs(x)}$ denotes the duration of action $APs(t,i)$ for train x on circuit Cs.

From Hr_1 and Hr_2 we can infer the following bounds on the duration of action $APs(t,i)$:

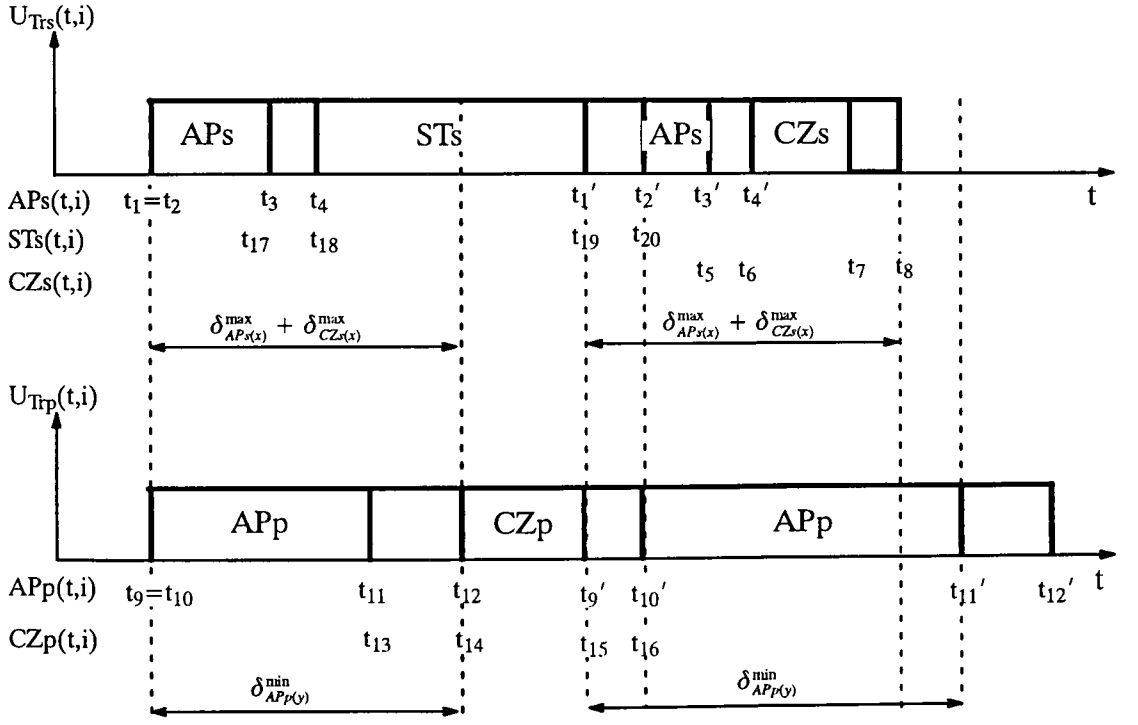
$$\delta^{\min}_{APs(x)}=0 \text{ and } \delta^{\max}_{APs(x)}= dmax(s,CC(s,r)\ominus 1,CC(s,r))/Vmin(s,x).$$

The duration for the action $APs(t,i)$ has an upper and a lower bound since the action may be aborted: the minimum duration is zero since $CC(s,r)\ominus 1$ immediately precedes $DZ(s,r)$, and the maximum duration occurs in the case when a train crosses the entire section $CC(s,r)\ominus 1$ at minimum velocity.

Bounds on the duration for action $APp(t,i)$ (assuming the train does not stop) starting at time point t, can be derived from Hr_1 and Hr_2:



a. A secondary train does not have to stop at a crossing section.



b. A secondary train must stop at a crossing section.

Figure 6.2. The E/A model of the priority requirement.

$\delta^{min}_{APp(y)}(t) = dmin(p,Ptrain(p,y)(t),CC(p,r))/Vmax(p,y)$; and

$$\delta^{max}_{APp(y)}(t) = dmax(p,Ptrain(p,y)(t),CC(p,r))/Vmin(p,y).$$

The duration for the action $APp(t,i)$ has an upper and a lower bound since the controller knows that the train is somewhere in section $Ptrain(p,y)$, but does not know exactly where and the velocity of the train is unknown.

Similarly, we can infer the following bounds on the duration of action $CZc(t,i)$ (under the assumption that a train does not stop in the danger zone):

$$\delta^{min}_{CZc(x)} = dmax(c,CC(c,r),CC(c,r)\oplus 2)/Vmax(c,x); \text{ and}$$

$$\delta^{max}_{CZc(x)} = dmax(c,CC(c,r),CC(c,r)\oplus 2)/Vmin(c,x).$$

For the action $CZc(t,i)$, since a train always travels a fixed distance related to the size of the sections, uncertainties in the duration of the action are entirely due to the uncertainties in the velocity of the train.

If we consider the behaviour of a secondary train x which is approaching the danger zone $DZ(s,r)$ with respect to a sequence of primary trains approaching the danger zone $DZ(p,r)$, we observe that train x may cross the danger zone $DZ(s,r)$ if it will cross $DZ(s,r)$ before the nearest primary train y (i.e. the train that must travel the shortest distance) reaches $DZ(p,r)$, which can be expressed in terms of the durations of the actions:

$$\delta^{max}_{APs(x)} + \delta^{max}_{CZs(x)} < \delta^{min}_{APp(y)}.$$

In order to use the E/A model to analyse the timing constraints associated with the three actions of the equation above, we must analyse the parallel composition of the utility functions for the secondary and primary trains (i.e. $U_{Tis}(t,i) \,||\, U_{Tip}(t,i)$). In the following we identify the PEA notation for $U_{Tis}(t,i)$ and $U_{Tip}(t,i)$, as depicted in figure 6.2.

a. A secondary train does not have to stop at a crossing section:

$$U_{Tis}(i)@\langle\,\rangle \Leftrightarrow U_{APs}(i)@\langle\,\rangle \prec U_{CZs}(i)@\langle\,\rangle;$$

$$U_{Trp}(i)@\langle\,\rangle \Leftrightarrow U_{APp}(i)@\langle\,\rangle \prec U_{CZp}(i)@\langle\,\rangle.$$

b. A secondary must stop at a crossing section:

$$U_{Trs}(i)@\langle\,\rangle \Leftrightarrow U_{APs}(i)@\langle\,\rangle \prec U_{STs}(i)@\langle\,\rangle \prec U_{Aps}(i+1)@\langle\,\rangle \prec U_{CZs}(i)@\langle\,\rangle;$$

$$U_{Trp}(i)@\langle\,\rangle \Leftrightarrow U_{APp}(i)@\langle\,\rangle \prec U_{CZp}(i)@\langle\,\rangle.$$

By examination of the graphs we conclude that the action $CZs(t,i)$ for any train x follows the action $APs(t,i)$ for train x only if the earliest finishing time of $APp(t,i)$ for all primary trains is greater than the latest finishing time of $CZs(t,i)$.

We specify the priority constraint as a history predicate that describes the conditions in which a secondary train cannot reserve the section $CC(s,r)$; we say that a history satisfies the priority constraint if the history predicate that describes it is a history relation for that history. To complete the safety strategy $SS_{2,1,1}$ for the crossing section we therefore add the priority constraint (rule *ssc*) to the rules *ssa* and *ssb*:

ssc. For any interval $[T_0, T_1]$, for any secondary train x, if $CC(s,r)$ is not reserved by train x at T_0 and during $[T_0, T_1]$ the maximum duration for all parts of train x to approach and pass the danger zone $DZ(s,r)$ from section $CC(s,r)\ominus 1$ (when it is moving) is at least the minimum duration for any primary train y to approach $DZ(p,r)$ or the current section is not $CC(s,r)\ominus 1$, then the section $CC(s,r)$ cannot be reserved by train x during $[T_0, T_1]$.

$$\begin{aligned}
&\forall r \in R: \forall x \in Trs: [\, CC(s,r) \notin Rtrain(s,x)(T_0) \;\wedge \\
&\quad \forall t \in [T_0, T_1]: \exists y \in Trp: \delta^{\max}_{APs(x)} + \delta^{\max}_{CZs(x)} \geq \delta^{\min}_{APp(y)} \;\vee \\
&\quad Ptrain(s,x)(t) \neq \quad CC(s,r)\ominus 1 \Rightarrow \forall t \in [T_0, T_1]: CC(s,r) \notin Rtrain(s,x)(t)].
\end{aligned}$$

Alternatively, this rule could be expressed in terms of the PEA notation. The timing constraints are then captured by the following formula, which represents a third rule *ssc* for $SS_{2,1,1}$:

$$\forall H \in \Gamma\mathbf{H}: \forall x \in Trs: \forall i \in \mathrm{N}: \forall t_1,.., t_{20} \in T:$$

$$[H \textbf{ sat } U_{APs}(i)@\langle t_1,t_2,t_3,t_4\rangle < U_{CZs}(i)@\langle t_5,t_6,t_7,t_8\rangle \Rightarrow$$

$$\exists j \in \mathrm{N}: \forall y \in Trp: H \textbf{ sat } U_{APp}(j)@\langle t_9,t_{10},t_{11},t_{12}\rangle \wedge t_8 < t_{11}].$$

To confirm that rule *ssc* captures the priority requirement, we state and sketch the proof of the following lemma.

Lemma 6.7

Provided *ssc* is satisfied, even if the primary trains never stop, the safety constraint $SC_{2,1}$ is maintained. This can be formally stated as: $ssc \wedge \forall y \in Trp: Vtrain(c,y) \neq 0 \Rightarrow SC_{2,1}$.

Proof. (By contradiction).

Assuming that there exists a time point t in which $SC_{2,1}$ is violated, this implies that the actions $CZs(t,i)$ and $CZp(t,i)$ are superimposed, which can be stated in terms of the utility functions as follows:

$$\exists t_5, t_6, t_7, t_8, t_{13}, t_{14}, t_{15}, t_{16} \in T: \exists x \in Trs: \exists y \in Trp: \exists i, j \in \mathrm{N}:$$

$$[U_{CZs}(i)@\langle t_5,t_6,t_7,t_8\rangle \wedge U_{CZp}(j)@\langle t_{13},t_{14},t_{15},t_{16}\rangle \wedge$$

$$((t_5 \leq t_{13} \wedge t_8 \geq t_{13}) \vee (t_{13} \leq t_8 \wedge t_{16} \geq t_8))].$$

But this contradicts rule *ssc*. The only condition assumed over the velocity of the primary trains in *ssc* is that they are bounded by $Vmax(p,y)$ (i.e. the earliest finishing time for $APp(t,i)$ is bounded), which is consistent with the constraint that primary trains never stop.

Combination of the Safety Strategy Rules

To implement the safety strategy $SS_{2,1,1}$ the safety controller must establish the rules *ssa*, *ssb* and *ssc*. We have already shown that the rules *ssa* and *ssb* are sufficient to maintain $SC_{2,1}$, and are not in conflict, provided the number of sections of the the circuits satisfy certain constraints. Now we have formulated the rule *ssc* to maintain the priority requirement, therefore the conjunction of the three rules is sufficient for both $SC_{2,1}$ and

the priority requirement provided the three rules do not conflict. Rule *ssc* imposes constraints only on secondary trains; specifically it identifies those intervals during which a secondary train cannot reserve a crossing section. For a secondary train x in section $CC(s,r)\ominus 1$ to reserve the crossing section $CC(s,r)$ under rule *ssc*, the minimum duration of $APp(t,i)$ for any primary train y must be greater than the sum of the maximum durations of $APs(t,i)$ and $CZs(t,i)$ for the train x. The definitions of actions $APp(t,i)$, $APs(t,i)$ and $CZs(t,i)$ can be used to derive conditions under which a secondary train x is allowed to reserve $CC(s,r)$ when the nearest primary train y is in section i:

$$dmin(p,i,CC(p,r))/Vmax(p,y) > dmax(s,CC(s,r)\ominus 1,CC(s,r)\oplus 2)/Vmin(s,x).$$

In general, we may have any secondary train x and primary train y, therefore for the above condition to be satisfied for any pair of trains we identify the following general condition over the train set which says that between any pair of primary and secondary trains, the secondary train is allowed to reserve the $CC(s,r)$ when from their relative positions, the maximum time it takes for a secondary train to approach and cross its danger zone, is less than the minimum time it takes for a primary train to approach its danger zone:

$$\exists i \in Sp: \forall x \in Trs: \forall y \in Trp:$$
$$dmin(p,i,CC(p,r))/Vmax(p,y) > dmax(s,CC(s,r)\ominus 1,CC(s,r)\oplus 2)/Vmin(s,x).$$

By imposing the above general condition on the train set, in addition to the constraints on the number of sections established for rules *ssa* and *ssb*, it is possible to satisfy all three rules. These would be minimal constraints on the physical process of the train set. For practical purposes, issues other than the satisfaction of safety strategy $SS_{2,1,1}$ must be considered, such as the ability of the safety controller to implement the safety strategy in the allowed states and the impact of safety strategy $SS_{1,1,1}$ derived to avoid collisions on the circuits, for which the number of trains on the circuits must be taken into consideration.

6.4. Safety Interface Analysis

After establishing the safety strategies during the Safety Plant Analysis, the Safety Interface Analysis phase investigates how these are to be implemented by the components of the interface as *robust safety strategies*. In this section, we will restrict the analysis to the crossing section because of the similarity in the type of analysis that has to be performed in order to obtain the other robust safety strategies.

In the physical model train set, at the beginning of every section there is a sensor that detects the presence of a train, and for each train there is an actuator that can stop the train within any section. A first step in the analysis is to identify the initiating event related with $SS_{2,1,1}$ (the analysis for $SS_{1,1,1}$ is similar to the analysis performed in the following).The initiating event occurs for a train x on circuit Cc for crossing section $CC(c,r)$ when the train is in section $CC(c,r)\ominus 1$ and section $CC(c,r)$ is not reserved by the train.

$$\forall r \in R: \forall c \in L: \forall x \in Trc: [IE_{2,1,1}(c,x,r) \Leftrightarrow$$
$$Ptrain(c,x) = CC(c,r)\ominus 1 \wedge CC(c,r) \notin Rtrain(c,x)].$$

The behaviour of the sensors and actuators is represented by two state variables *Sens* and *Act*, the observations recorded by the safety controller are held in the variable *Pos*, and the set of sections reserved by each train is recorded in the variable *Res*. The universal history set of the physical process must be extended to include these state variables.

No.	Name	Range	Comments
p_4	*Sens*	$B^{Nsp+1} \times B^{Nss+1}$	The state of each sensor expressed as a truth value.
p_5	*Act*	$B^{Nsp+1} \times B^{Nss+1}$	The state of each actuator expressed as a truth value.
p_6	*Pos*	$Sp^{Ntp} \times Ss^{Nts}$	The position of each train as recorded by the safety controller.
p_7	*Res*	$Rp^{Ntp} \times Rs^{Nts}$	The reservation sets of each of the trains as recorded by the safety controller.

In the following we present a set of assumptions, expressed in THL, over the properties of the sensors and actuators and the behaviour of the physical process.

Sensor and Actuator Relations

The sensor $Sens(c,i)$ returns the value true iff a train is in section i on circuit Cc.

$$\mathrm{Ir}_1.\ \forall c \in L: \forall i \in Sc: [\ Sens(c,i) \Leftrightarrow \exists x \in Trc: Ptrain(c,x) = i\ ].$$

If $Act(c,x)$ is set throughout any interval $[T_0, T_1]$, then for every time point in $[T_0, T_1]$ the position of the front of train x is the same as its position at time point T_0.

$$\mathrm{Hr}_1.\ \forall c \in L: \forall x \in Trc: [\ (\forall t \in [T_0, T_1]: Act(c,x)(t)) \Rightarrow \forall t \in [T_0, T_1]: Ptrain(c,x)(t) = Ptrain(c,x)(T_0)].$$

Trains on the circuits are modelled as moving in steps of one section in a clockwise direction.

$$\mathrm{Hr}_2.\ \forall c \in L: \forall x \in Trc: \exists k \in Sc: [\ Ptrain(c,x)(T_1) = Ptrain(c,x)(T_0) \oplus k\ \wedge\ \forall i \in \{0, ..., k\}: \exists t \in [T_0, T_1]: Ptrain(c,x)(t) = Ptrain(c,x)(T_0) \oplus i\ ].$$

Assuming perfect sensors (Ir_1), and the above restriction on the movement of trains (Hr_2) the position of trains as recorded by the safety controller is identical to the actual position of the trains.

$$\mathrm{Ir}_2.\ \forall c \in L: [\ \forall x \in Trc: Pos(c,x) = Ptrain(c,x)\ ].$$

$$\mathrm{Ir}_3.\ \forall c \in L: [\ \forall x \in Trc: Res(c,x) = Rtrain(c,x)\ ].$$

In this ideal case the rules of the safety strategies and the initiating events can be rewritten in terms of *Pos* and *Res* by simply substituting $Pos(c,x)$ for $Ptrain(c,x)$, and $Res(c,x)$ for $Rtrain(c,x)$, respectively.

Connection between Initiating Events and Actuators

While a train x on circuit Cc is approaching section $CC(c,r)$ and $CC(c,r)$ is not reserved the actuator on train x must remain set.

Ir_4. $\forall r \in R$: $\forall c \in L$: $\forall x \in Trc$: [$IE_{2,1,1}(c,x,r) \Rightarrow Act(c,x)$].

Properties of the Reservation Scheme and Physical Process

If $CC(c,r)$ is reserved for train x on circuit Cc and the front of train x is in section $CC(c,r)\ominus 1$ then $CC(c,r)$ must remain reserved for the train until it leaves danger zone $DZ(c,r)$.

$$\begin{aligned} Hr_3.\ \forall r \in R: \forall c \in L: \forall x \in Trc: \forall t \in [T_0, T_1]: [\ & CC(c,r) \in Rtrain(c,x)(T_0) \wedge \\ & Ptrain(c,x)(T_0) = CC(c,r)\ominus 1 \wedge \\ & Ptrain(c,x)(t) \in \{CC(c,r)\ominus 1\} \cup DZ(c,r) \Rightarrow \\ & \forall t \in [T_0, T_1]: CC(c,r) \in Rtrain(c,x)(t)]. \end{aligned}$$

In the initial state of the system there is no train in a danger zone or a section that immediately precedes a danger zone.

$$\begin{aligned} Hr_4.\ T_0 = S(T) \Rightarrow \forall r \in R: \forall c \in L: \forall x \in Trc: \\ [\ Ptrain(c,x)(T_0) \notin \{CC(c,r)\ominus 1 \cup DZ(c,r)\}]. \end{aligned}$$

The lemma below shows that the initial state of the system is a safe state, by proving that safety constraint $SC_{2,1}$ cannot be violated subsequent to the initial state if the initiating event does not occur.

Lemma 6.8

If Hr_1, Hr_2, Hr_3 and Hr_4 are history relations for a history which satisfies rule *ssb* of safety strategy $SS_{2,1,1}$ and during that history $IE_{2,1,1}(c,x,r)$ is never satisfied, then $SC_{2,1}$ is satisfied during that history.

Proof. (By contradiction)

Assume $\exists t \in T$: $\exists r \in R$: $\exists x \in Trp$: $\exists y \in Trs$:

$$[H \textbf{ sat } (Ptrain(p,x) \in DZ(p,r) \wedge Ptrain(s,y) \in DZ(s,r))@t].$$

From rule *ssb* of $SS_{2,1,1}$ we conclude: $\exists t \in T$: $\exists r \in R$: $\exists c \in L$: $\exists x \in Trc$:

$[H \textbf{ sat } (Ptrain(c,x) \in DZ(c,r) \wedge CC(c,r) \notin Rtrain(c,x))@t]$.

From Hr_2, Hr_3 and Hr_4 we conclude: $\exists t, t' \in T: \exists r \in R: \exists c \in L: \exists x \in Trc$:

$[t' < t \wedge H \textbf{ sat } (Ptrain(c,x) = CC(c,r) \ominus 1 \wedge CC(c,r) \notin Rtrain(c,x))@t']$.

But this contradicts the assumption that $\forall r \in R: \forall c \in L: \forall x \in Trc$:

$[H \textbf{ sat } \neg IE_{2,1,1}(c,x,r)]$.

The lemma below shows that if the system enters an unsafe state, corrective action by the safety controller (specifically setting an actuator) will maintain safety constraint $SC_{2,1}$.

Lemma 6.9

If Hr_1 is a history relation for a history and during that history $Act(c,x)$ is set whenever $IE_{2,1,1}(c,x,r)$ holds, then $SC_{2,1}$ will be maintained.

Proof. From Hr_1 we conclude: $\forall r \in R: \forall c \in L: \forall x \in Trc$:

$[H \textbf{ sat } IE_{2,1,1}(c,x,r)@T_0 \wedge H \textbf{ sat } Act(c,x)@[T_0, T_1] \Rightarrow$

$H \textbf{ sat } Ptrain(c,x) \notin DZ(c,r)@[T_0, T_1]]$.

Sensor and Actuator Failures

A sensor may fail permanently and not detect the presence of a train – "miss a train", or detect the presence of a train that does not exist – a "ghost train", or fail intermittently. An actuator may fail permanently and never stop a train when required, or stop a train inadvertently, or fail to stop the train within the required distance, or fail intermittently.

We consider only the case of sensor failures that "miss a train"; specifically we suppose that there can be at most m sensor failures in a sequence of sensors. To capture this failure assumption the relationship between the sensors and the position of the trains must be modified. For any $m \oplus 1$ consecutive sections, i, ..., $i \oplus m$, at least one of the sensors will still be working in the way described by Ir_1 and if a sensor detects a train there must be

a train in that section, in other words, we assume the nonexistence of sensor failures that detect "ghost trains".

$$\mathrm{Ir}_5.\ \forall c \in L: \forall i \in Sc: [\ (\exists j \in \mathrm{N}: j \leq m\ \wedge\ Sens(c,i \oplus j) \Leftrightarrow$$
$$\exists x \in Trc: Ptrain(c,x) = i \oplus j)\ \wedge\ Sens(c,i) \Rightarrow Ptrain(c,x) = i].$$

The relationship between *Pos* and *Ptrain* must also be modified to reflect the limitations of the sensors. If m consecutive sensors "miss a train" *Pos* will be m sections behind *Ptrain*. Hence, to allow for faulty sensors (Ir_5), the relationship between *Pos* and *Ptrain* must incorporate the uncertainty in position of the trains.

$$\mathrm{Ir}_6.\ \forall c \in L: \forall x \in Trc: [\ Ptrain(c,x) \in \{Pos(c,x), \ldots, Pos(c,x) \oplus m\}\ ].$$

Moving on to consider actuator failures, note that we cannot handle the situation when an actuator fails to stop a train without introducing redundant actuators. Instead, we examine the case of actuator failures that result in a train requiring more than one section to stop. More specifically, we assume that if an actuator on a train fails, then the train will still be stopped but may need $n \oplus 1$ sections (at most) to do so ($n \geq 0$).

$$\mathrm{Hr}_5.\ \forall c \in L: \forall x \in Trc: [\forall t \in [T_0, T_1]: Act(c,x)(t) \Rightarrow$$
$$Ptrain(c,x)(t) \in \{Ptrain(c,x)(T_0), \ldots, Ptrain(c,x)(T_0) \oplus n\}].$$

The initiating event for $\mathrm{RSS}_{2,1,1}$ ($\mathrm{IE}_{2,1,1}{}^{\mathrm{FT}}$) is a modification of $\mathrm{IE}_{2,1,1}$ that takes into account the limitations of the sensors and actuators.

$$\forall r \in R: \forall c \in L: \forall x \in Trc: [\mathrm{IE}_{2,1,1}(c,x,r)^{\mathrm{FT}} \Leftrightarrow$$
$$Pos(c,x) \in \{CC(c,r) \ominus (m+n+1), \ldots, CC(c,r) \oplus 1\}\ \wedge\ CC(c,r) \notin Res(c,x)].$$

Properties of the Reservation Scheme and Physical Process

The reservation scheme and the condition over the initial state of the physical process must be strengthened to allow the safety controller to tolerate failures in the sensors and actuators.

If $CC(c,r)$ is reserved for train x on circuit Cc and the front of train x is in the sections $\{CC(c,r)\ominus(m+n+1), ..., CC(c,r)\oplus 1\}$ then $CC(c,r)$ must remain reserved for the train until it leaves danger zone $DZ(c,r)$.

$$\begin{aligned}\mathrm{Hr}_6.\ \forall r \in R: \forall c \in L: \forall x \in Trc: [\ & CC(c,r) \in Res(c,x)(T_0) \wedge \\ & Pos(c,x)(T_0) \in \{CC(c,r)\ominus(m+n+1), ..., CC(c,r)\oplus 1\} \wedge \\ & \forall t \in [T_0, T_1]: Pos(c,x) \in \{CC(c,r)\ominus(m+n+1), ..., CC(c,r)\oplus 1\} \cup \\ & DZ(c,r) \Rightarrow \forall t \in [T_0, T_1]: CC(c,r) \in Res(c,x)(t)].\end{aligned}$$

The initial state of the system must be such that there is no train in an extended danger zone or the $n\oplus 1$ sections that immediately precede a danger zone.

$$\begin{aligned}\mathrm{Hr}_7.\ & T_0 = S(T) \Rightarrow \forall r \in R: \forall c \in L: \forall x \in Trc: \\ & [Pos(c,x)(T_0) \notin \{CC(c,r)\ominus(m+n+1), ..., CC(c,r)\oplus 1\} \cup DZ(c,r)].\end{aligned}$$

We define a robust safety strategy $\mathrm{RSS}_{2,1,1}$ which compensates for failures of the sensors and actuators, by extension of the danger zones (i.e. by employing spatial redundancy) to ensure that whenever any part of a train is in a crossing section, the crossing section is reserved for that train despite sensor/actuator failures. $\mathrm{RSS}_{2,1,1}$ is described by the following two rules:

rssa. if the recorded position of train x on circuit Cc is in an extended danger zone then the crossing section contained within that danger zone is reserved (on circuit Cc);

$$\begin{aligned}\forall r \in R: \forall c \in L: \forall x \in Trc: & \\ & [\ Pos(c,x) \in DZ(c,r)^{\mathrm{FT}} \Rightarrow CC(c,r) \in Res(c,x)\ ]; \\ & \text{where } DZ(c,r)^{\mathrm{FT}} = \{CC(c,r)\ominus(m+n+1), \ldots, CC(c,r)\oplus 1\};\end{aligned}$$

rssb. sections $CC(p,r)$ and section $CC(s,r)$ cannot both be reserved;

$$\forall r \in R: \exists c \in L: [\ \forall x \in Trc: CC(c,r) \notin Res(c,x)\ ].$$

Lemma 6.10

A history for which Ir_5 is an invariant relation, and which satisfies the robust safety strategy $RSS_{2,1,1}$, must satisfy the safety strategy $SS_{2,1,1}$.

Proof. This proof follows directly from Ir_4 and the definitions of $DZ(c,r)^{FT}$ and $DZ(c,r)$. The two lemmas below show that the initial state is safe and the safety controller can take corrective action (c.f. lemma 6.8 and lemma 6.10).

Lemma 6.11

If Ir_6 and Hr_2, Hr_3, Hr_4, Hr_6 are relations for a history which satisfies rule *ssb* of the robust safety strategy $RSS_{2,1,1}$ and $IE_{2,1,1}(c,x,r)^{FT}$ is never satisfied,then $SC_{2,1}$ is satisfied during that history.

Proof. (By contradiction)

Assume $\exists t \in T$: $\exists r \in R$: $\exists x \in Trp$: $\exists y \in Trs$:

$$[H \textbf{ sat } (Ptrain(p,x) \in DZ(p,r) \wedge Ptrain(s,y) \in DZ(s,r))@t].$$

From lemma 6.8 we have: $\exists t, t' \in T$: $\exists c \in L$: $\exists x \in Trc$: $\exists r \in R$:

$$[t' < t \wedge H \textbf{ sat } IE_{2,1,1}(c,x,r)@t'].$$

From $IE_{2,1,1}(c,x,r) \wedge Ir_5$ we conclude:

$$[H \textbf{ sat } (Pos(c,x) \in \{CC(c,r) \ominus (m+1), ..., CC(c,r) \oplus 1\})@t'].$$

Together with Hr_6 this contradicts the assumption that $\forall c \in L$: $\forall x \in Trc$: $\forall r \in R$:

$$H \textbf{ sat } \neg IE_{2,1,1}(c,x,r)^{FT}.$$

Lemma 6.12

If Ir_6 and Hr_7 are relations for a history and during that history $Act(c,x)$ is set whenever $IE_{2,1,1}(c,x,r)^{FT}$ holds, then $SC_{2,1}$ will be maintained during that history.

Proof.

From $IE_{2,1,1}(c,x,r)^{FT}$, Ir_6 and Hr_6 we conclude that at the instant at which $IE_{2,1,1}(c,x,r)^{FT}$ occurs

$$Ptrain(c,x) \in \{CC(c,r) \ominus (m+n+1), ..., CC(c,r) \oplus (n+1)\}.$$

Hence, from Hr_5 we conclude: $\forall c \in L$: $\forall x \in Trc$: $\forall r \in R$:

H **sat** $\mathrm{IE}_{2,1,1}(c,x,r)^{\mathrm{FT}}@T_0 \wedge H$ **sat** $Act(c,x)@[T_0, T_1] \Rightarrow$

H **sat** $Ptrain(c,x) \notin DZ(c,r)@[T_0, T_1]$.

From the analysis we conclude that the basic effect of the sensor and actuator failures considered here is that a block of sections must be treated as if they were a single section. The size of this block depends on the failure assumptions (i.e. values of m and n) and on the extent to which tolerance is to be provided for sensor and actuator failures.

6.5. Safety Controller Analysis

After establishing the robust safety strategies during the Safety Plant Analysis, the safety controller analysis phase investigates how these strategies are to be implemented by the safety controller as *safety controller strategies*; these are formed by mapping the robust safety strategies onto the system components. However, in presenting this case study, in order to simplify the analysis, the derivation of the safety controller strategies will ignore the results of the Safety Interface Analysis presented in the previous section; instead we will work directly from the safety specifications obtained in section 6.3 from the Safety Plant Analysis. In /de Lemos 92/, for the same example, the Safety Controller Analysis is conducted using the robust safety strategies obtained from Safety Interface Analysis.

The analysis to be performed entails modelling the relationship between the components of the safety controller, the physical process, and the interface between them (which includes sensors and actuators). The general approach followed in modelling the system is to maintain a clear separation between models of the physical process and the safety controller, even though they must cooperate whenever an action is to be performed. The advantage of adopting this approach is that both models can be independently developed and modified, and the state of the physical process can be seen to correspond to the sequence of control commands issued by the safety controller /Combacau 90/.

The analysis will be divided into two parts, corresponding to the safety strategies established during the Safety Plant Analysis. First, we model the safety strategy which

prevents the collision of trains of the same type, and second, we model the safety strategy which prevents the collision of trains of different types.

6.5.1. Collision of Trains of the Same Type

To illustrate modelling the behaviour of trains in a circuit, we assume that each circuit contains seven sections ($Ns=6$) and two trains ($Nt=2$). For simplicity, the modelling and analysis is performed for just one circuit. The PrT model is shown in figure 6.3 (because of the flexibility of the model it is possible to modify the number of sections and trains without affecting the structure of the model of the safety controller). An outline definition of PrT nets is presented in Appendix B.

The predicates of the PrT net model of a train set circuit are the following:

S0x to *S6x* – train *x* occupies a section of the circuit;
ICPxj – train *x* is allowed to enter section *j*;
IPCxj – train *x* has entered section *j*;
FSn – section *n* is not reserved by the safety controller;
RS1xm – train *x* has reserved section *m*;
RS2xk+xj – train *x* has (temporarily) reserved sections *k* and *j*.

To obtain the structural properties of the PrT net model of the circuit, we derive the S–invariants of the net; these are integer equation invariants which are obtained from the projection of the predicates /Genrich 87/. In order to simplify the calculation of the S–invariants of the PrT net model of figure 6.3, the predicates representing the circuit sections of the model of the physical process were folded into one predicate (*Sxj* – train *x* occupies section *j*).

The S–invariants of the PrT net model of the safety controller strategies of the circuit are:

$$|RS2|_2 + 2|RS1|_2 = 8; \qquad (si1)$$

$$|IPC| + |ICP| + |FS| = 3; \qquad (si2)$$

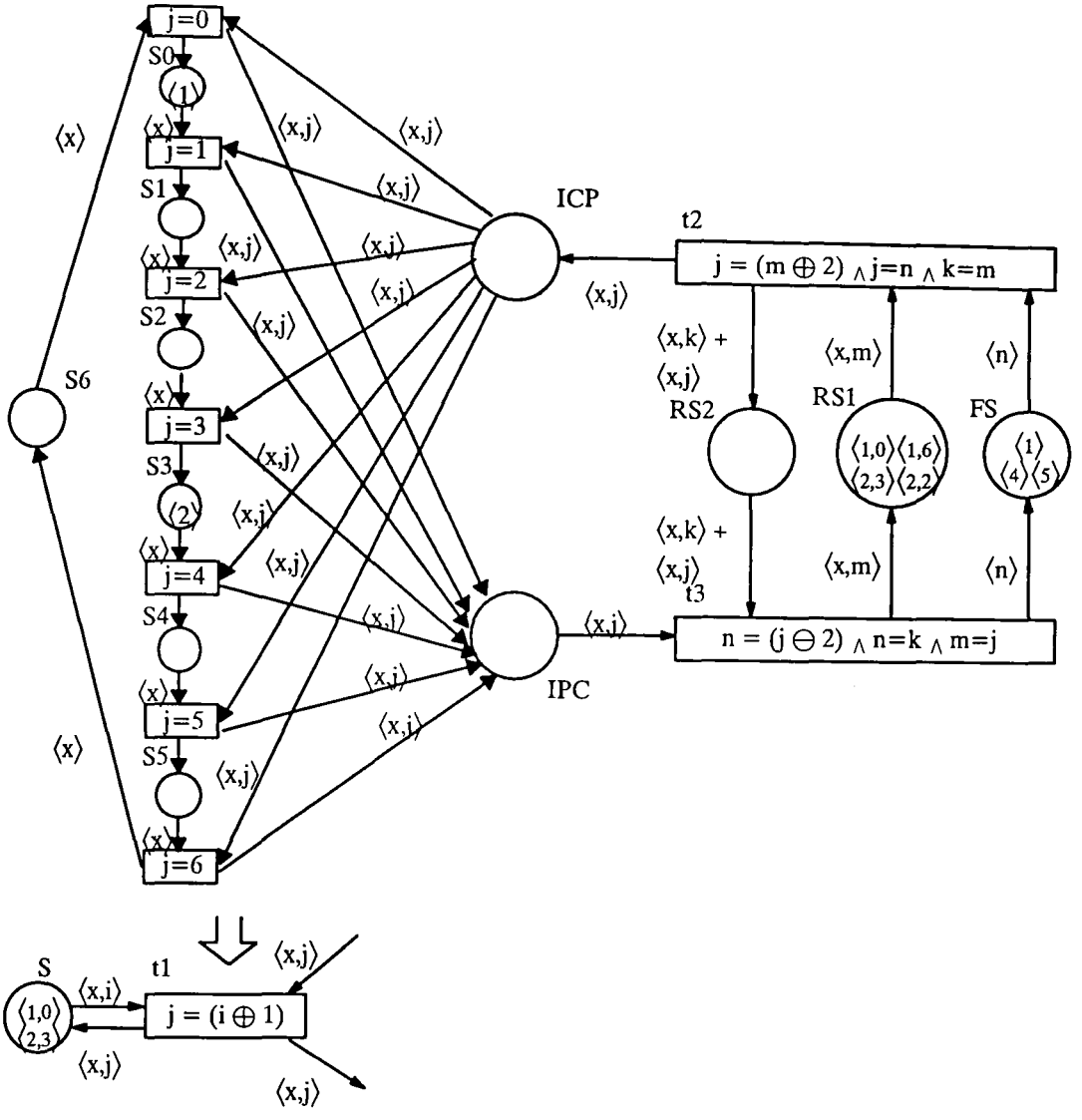


Figure 6.3. The PrT net model of the train set circuit.

$$|RS1| - |FS| = 1; \qquad (si3)$$

$$|RS2| + 2|FS| = 6. \qquad (si4)$$

The number of tuples in the predicate *Sxj* is constant and is established by the initial marking.

Lemma 6.13

The PrT net of the train set circuit shown in figure 6.3 is contact–free and deadlock–free. (Contact–freeness means that even under the weak transition rule allowing multi–sets on places there is no marking reachable that puts the same tuple on

a place more than once. Deadlock−freeness means that there is no forward reachable marking at which no transition is enabled.)

Proof:

To confirm that a PrT net is contact−free, we must show that there is no marking that puts the same tuple at the same predicate more than once. For the PrT net of figure 6.3, this follows immediately from equations (*si1*) and (*si2*).

To confirm that a PrT net is deadlock−free, we must show that there is no forward reachable marking at which all transitions are disabled. By inspecting the PrT net of figure 6.3, we show that if both *t2* and *t3* are disabled then *t1* must be enabled. Firstly, we consider the case when *t2* is disabled, and secondly when *t3* is disabled.

a. If *t2* is disabled, this implies that the tuples $\langle x,m\rangle$ in *RS1* and $\langle n\rangle$ in FS are not sufficient to satisfy the transition selector of *t2*. This condition can be confirmed from (*si3*), from which we obtain $|RS1|=2$ and $|FS|=1$. Hence, from (*si1*) we have $|RS2|_2=4$ and from (*si2*) we have $|IPC|+|ICP|=2$, which leads to two possible cases: if $|IPC|\neq 0$ then *t3* would be enabled, however $|IPC|=0$ implies that *t1* is enabled.

b. If *t3* is disabled, this implies that there are no tuples in the predicates *RS2* and *IPC*, that is, there is neither $\langle x, j\rangle$ in *IPC* nor $\langle x,k\rangle + \langle x\, j\rangle$ in *RS2*. If there are no tuples in *RS2* and *IPC* then it follows from (*si1*) and (*si3*) that the number of tuples in *RS1* and *FS* are identical to those in the initial marking, and it can be easily shown that *t2* is enabled. If there are no tuples in *RS2* then it follows from (*si2*) and (*si4*) that $|IPC|+|ICP|=0$ implying that there are also no tuples in *IPC*, in which case the number of tuples in *RS1* and *FS* are identical to the previous case. If there are no tuples in *IPC* then it follows from (*si2*) that $|ICP|+|FS|=3$ which leads to two possible cases: if $|FS|=3$ then $|ICP|=|RS2|=0$ in which case the number of tuples in *RS1* and *FS* are identical to the initial marking, but if $|FS|\neq 3$ then $|ICP|\neq 0$ implies that *t1* must be enabled.

Verification of Circuit Safety Controller Strategy

The safety controller strategy defined by the PrT net model of figure 6.3 is verified by proving that the safety strategy $SS_{1,1,1}$ is a property of the PrT net model. To verify this, a link must be identified between the THL and PrT net models of the train set system. This link can be established in terms of system predicates of the THL model and the predicates of the PrT net model, as follows:

$$\forall c \in L: \forall x \in Trc: \forall i \in Sc: [Ptrain(c,x) = i \Leftrightarrow Sxi];$$

$$\forall c \in L: \forall x \in Trc: \forall i \in Sc: [i \in Rtrain(c,x) \Leftrightarrow RS1xi \vee RS2xi].$$

The two rules of $SS_{1,1,1}$ can be expressed in terms of the PrT net model, as logical formulae (*lf1* and *lf2*) over the predicates of the PrT net model, by substituting the equivalent predicates of the PrT net model for the THL predicates:

lf1. If any train x is in section i, then sections i and $i \ominus 1$ are reserved by train x.

$$\forall x \in Trc: \forall i \in Sc: [Sxi \Rightarrow (RS1xi \vee RS2xi) \wedge (RS1xi \ominus 1 \vee RS2xi \ominus 1)].$$

lf2. Any section i is reserved by at most one train.

$$\forall x,y \in Trc: \forall i \in Sc: [x \neq y \Rightarrow (\neg RS1xi \wedge \neg RS2xi) \vee (\neg RS1yi \wedge \neg RS2yi)].$$

Lemma 6.14.

The formulae *lf1* and *lf2* are logical invariants (i.e. they hold on all reachable markings) of the PrT net model.

Proof: A sketch of a proof is given below by analysing the transitions of the PrT net model. In the initial marking *lf1* holds then both trains have reserved the current and previous sections:

$$\langle 1,0\rangle \in S \Rightarrow \{\langle 1,0\rangle, \langle 1,6\rangle\} \subseteq RS1 \cup RS2;$$

$$\langle 2,3\rangle \in S \Rightarrow \{\langle 2,3\rangle, \langle 2,2\rangle\} \subseteq RS1 \cup RS2.$$

We show that the firing of the three transitions cannot violate *lf1*. Transition *t1* can fire with $\langle x,i\rangle$ only when $\langle x,j\rangle$ is in *ICP* ($j = i \oplus 1$), therefore this can only happen after *t2* fires,

producing $\langle x,j\rangle$ in *ICP* and *RS2*. Prior to the firing of *t1*, $\{\langle x,i\rangle, \langle x,i\ominus 1\rangle\} \subseteq RS1 \cup RS2$; after *t1* fires, $\{\langle x,j\rangle, \langle x,i\rangle\} \subseteq RS1 \cup RS2$ (since $\langle x,j\rangle$ must remain in *RS2* until *IPC* fires producing $\langle x,j\rangle$). Firing *t2* adds $\langle x,j\rangle$ to $RS1 \cup RS2$, therefore *t2* cannot violate *lf1*. Firing *t3* removes $\langle x,k\rangle$ from $RS1 \cup RS2$ only when there is $\langle x,j\rangle$ in *S*, this is obtained from the transition selector of *t3* ($k = j\ominus 2$), therefore *t3* cannot violate *lf1*.

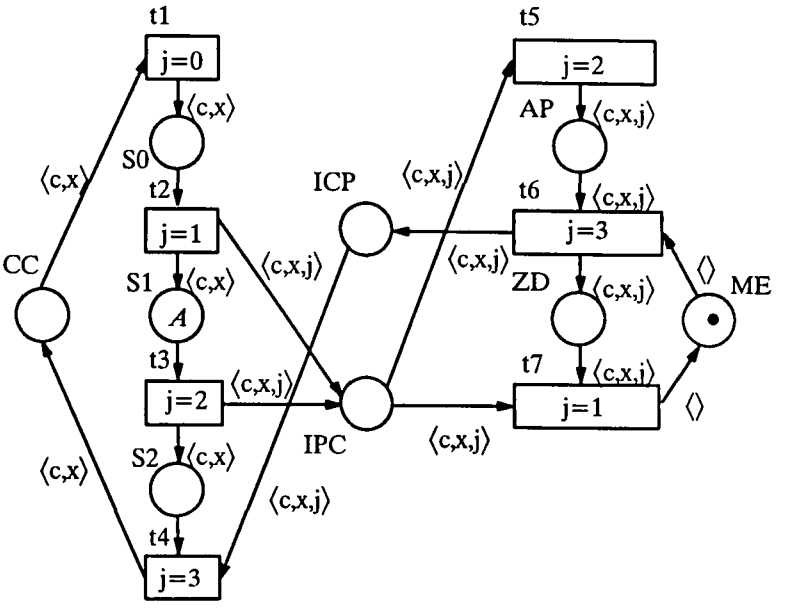
In the initial marking *lf2* holds; no section is reserved by more than one train: $RS1 \cup RS2 = \{\langle 1,0\rangle, \langle 1,6\rangle, \langle 2,3\rangle, \langle 2,2\rangle\}$. Firing *t1* does not change the markings on *RS1* or *RS2*. Firing *t2* removes $\langle x,m\rangle$ from *RS1* and adds $\langle x,k\rangle + \langle x,j\rangle$ to *RS2*, where $k=m$ and $j=m\oplus 2$. The only change in $RS1 \cup RS2$ is the addition of $\langle x,j\rangle$. From the transition selector of *t2*, j ($=n$) is a free section, hence prior to *t2* being fired $\neg(\exists y)\ \langle y,j\rangle \in (RS1 \cup RS2)$. Therefore *t2* cannot violate *lf2*. Firing *t3* removes $\langle x,k\rangle$ from $RS1 \cup RS2$, and therefore *t3* cannot violate *lf2*.

6.5.2. Collision of Trains of Different Type

For simplicity, the modelling and analysis of the behaviour of trains in a crossing section is performed for just one crossing section, and we assume that each circuit contains four sections ($Ns=3$) and one train ($Nt=1$), as shown in figure 6.4. In the PrT net model of the physical process we have folded the models of the primary and the secondary circuits into one net; the danger zone (*DZ*) of each circuit is represented by the predicates *CCcx* (crossing section) and *S0cx*, and the predicate *S2cx* represents the section which immediately preceds *CCcx*.

The predicates of the PrT net model of the crossing section are the following:

S0cx, *S1cx*, *S2cx* – a section of the circuit *c* is occupied by train *x*;
CCcx – the crossing section of circuit *c* is occupied by train *x*;
ICPcxj – train *x* in circuit *c* is allowed to enter section *j*;
IPCcxj – train *x* in circuit *c* has entered section *j*;
APcxj – train *x* in circuit *c* is in section $j=2$ (i.e. *x* is about to enter *DZ*);



$A=\{\langle p,1\rangle; \langle s,1\rangle\}$

Figure 6.4. The PrT net model of the crossing section.

$ZDcxj$ – train x in circuit c has access to section j (= 3, the crossing section);

ME – either primary or secondary trains allowed to enter the crossing section.

To obtain the structural properties of the PrT net model of the crossing section, we derive the S–invariants of the net, which are:

$$S0 + S1 + S2 + CC = 2; \qquad (si1)$$

$$ME + |ZD| = 1; \qquad (si2)$$

$$S1 + 2S2 - 2|ICP|_3 - |IPC|_3 - |AP|_3 + |ZD|_3 = 2; \qquad (si3)$$

$$S0 - S2 + CC + 2|ICP|_3 + |IPC|_3 + |AP|_3 - |ZD|_3 = 0; \qquad (si4)$$

$$2S0 + S1 + 2CC + 2|ICP|_3 + |IPC|_3 + |AP|_3 - |ZD|_3 = 2. \qquad (si5)$$

Lemma 6.14.

The PrT net model of the train set crossing section shown in figure 6.4 is contact–free and deadlock–free.

Proof:

The contact–freeness of the PrT net model of the crossing section is shown through inspection of (*si1*), (*si2*), (*si4*) and (*si5*). From (*si1*) we know that the number of tuples

in the net model of the physical process remains constant and equal to the initial marking. From (*si2*) we know that the number of tokens in the predicates *ME* and *ZD* do not increase. Finally, from (*si4*) and (*si5*) we know that *ICP*, *IPC* and *AP* are contact–free, implying that the number of tuples in the whole PrT net model remains constant.

To confirm that the PrT net model of the crossing section (in figure 6.4) is deadlock–free we show that if *t1*, *t2*, *t3*, *t5*, *t6* and *t7* are disabled then *t4* must be enabled. The approach followed here is to consider each of the transitions which we assume to be disabled individually and show that *t4* is enabled, implying deadlock–freeness.

If *t1* is disabled, it means that there are no tuples in *CC*. From (*si1*) it follows that if *S0* or *S1* have tuples, then transitions *t2* or *t3* must be enabled. However, if the two tuples are in *S2* then (*si4*) is reduced to $2|ICP|_3 + |IPC|_3 + |AP|_3 - |ZD|_3 = 2$. From this equation, if $|ZD| = 0$ then there are no tuples in *ICP* (reasoning from the PrT net model) and the two tuples must either be in *IPC* or *AP*, implying that at least *t5* or *t6* are enabled; if $|ZD| \neq 0$ then *t6* is disabled, one of the tuples is in either *IPC* or *AP*, implying that *t5* or *t6*, respectively, are enabled, and the other tuple is in *ICP*, implying that *t4* is enabled.

Similar reasoning can be applied for each of the other transitions that we assume to be disabled, and in each case we can show that *t4* will be enabled.

Verification of Crossing Section Safety Controller Strategy

The safety controller strategy defined by the PrT net model of figure 6.4 is verified by proving that the safety strategy $SS_{2,1,1}$ is a property of the PrT net model.

The THL predicates and the predicates of the PrT net model correspond as follows:

$$\forall c \in L: \forall x \in Trc: [Ptrain(c,x) \in DZ(c,r) \Leftrightarrow CCcx \vee S0cx];$$

$$\forall c \in L: \forall x \in Trc: \forall j \in Sc: [CC(c,r) \in Rtrain(c,x) \Leftrightarrow ZDcxj].$$

The two rules of $SS_{2,1,1}$ can be expressed in terms of the PrT net model as logical formulae (*lf3* and *lf4*) over the predicates of the PrT net model, by substituting the equivalent predicates of the PrT net model for the THL predicates.

lf3. If any train is in a danger zone then the crossing section is reserved by that train.

$$\forall x \in Trc: \forall i \in Sc: \forall r \in R: [CCcx \vee S0cx \Rightarrow ZDcxj].$$

lf4. A crossing section cannot be reserved for both the primary circuit and the secondary circuit.

$$\forall x \in Trc: \forall i \in Sc: \forall r \in R: [\neg ZDcxj].$$

Lemma 6.15

The formulae *lf3* and *lf4* are logical invariants (i.e. they hold on all reachable markings) of the PrT net model.

Proof: A sketch of a proof is given by analysing the transitions and S−invariants of the PrT net model.

In the initial marking *lf3* holds, since the sections *CC* and *S0* are empty: $(CC \cup S0) = \emptyset$. Firstly, we make the observation that tuples can be added to the set $CC \cup S0$ only by transition *t4* and removed only by transition *t2*. In the following we argue that $\langle c,x,j\rangle \in ZD$ before *t4* adds $\langle c,x\rangle$ to $CC \cup S0$ and until *t2* removes $\langle c,x\rangle$ from $CC \cup S0$. Transition *t4* can fire with $\langle c,x\rangle$ only if $\langle c,x,j\rangle$ is in *ICP*, therefore after *t6* fires with $\langle c,x,j\rangle$. Hence, before *t4* fires we have $\langle c,x,j\rangle \in$ ZD. Now $\langle c,x,j\rangle$ can be removed from *ZD* only when *t7* fires with $\langle c,x,j\rangle$; this transition can fire only when $\langle c,x,j\rangle \in ICP$. Therefore *t7* can fire with $\langle c,x,j\rangle$ only after *t2* has fired with $\langle c,x,j\rangle$. Hence $\langle c,x,j\rangle \in ZD$, at least until *t2* fires.

The fact that *lf3* is a logical invariant for the PrT net model follows from S−invariant (*si2*).

Crossing Section Safety Controller Strategy and Mission Requirements

Similar to the analysis conducted in Section 6.3, in the following we extend the safety controller strategy $SCS_{2,1,1,1}$ in order to consider the mission requirements established for the train set model which says that trains on the primary circuit have priority over trains on the secondary circuit. To illustrate the modelling of trains in the crossing section

of two circuits, we assume that: the two circuits cross only once, each circuit contains five sections (*Nsc=5*), and one train runs on each circuit (*Ntp=1*). To simplify the analysis we ignore the safety strategy for avoiding collisions of trains of the same type; thus we implicitly assume that each circuit has a safety controller strategy as developed previously.

The safety controller strategy must maintain the three rules of the safety strategy: rules *ssa* and *ssb* which say that no part of a primary or a secondary train can be in their respective danger zones at the same time, and rule *ssc* which says that a primary train must not be made to wait for a secondary train at a crossing section. In terms of the analysis of timeliness requirements, this can be represented by constructing the E/A model of the timing thresholds obtained from the Safety Plant Analysis, such as the example shown in figure 6.5. The names adopted for the actions and the events are equivalent to those adopted in the Safety Plant Analysis. This E/A model will be used as a reference for the construction of the PrT net model of the safety controller strategy. For every subsequent refinement of the PrT net model we should first construct the corresponding E/A model.

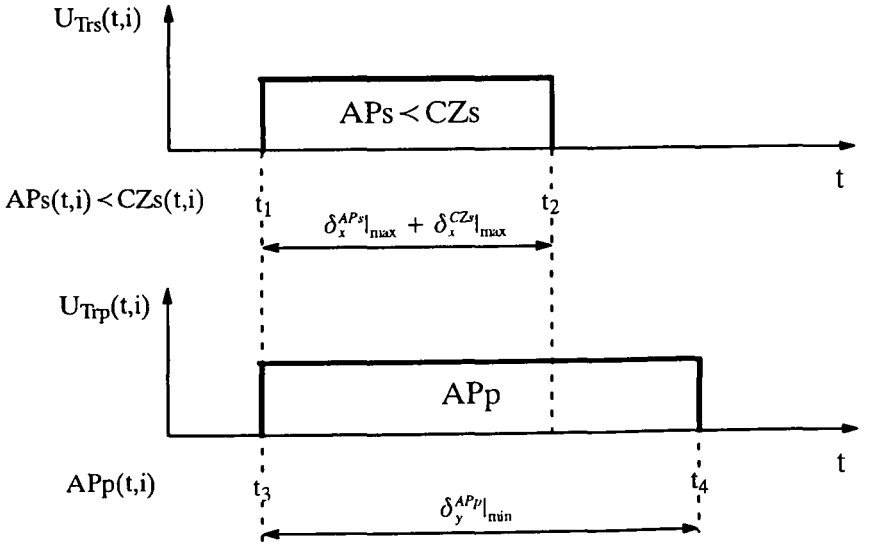
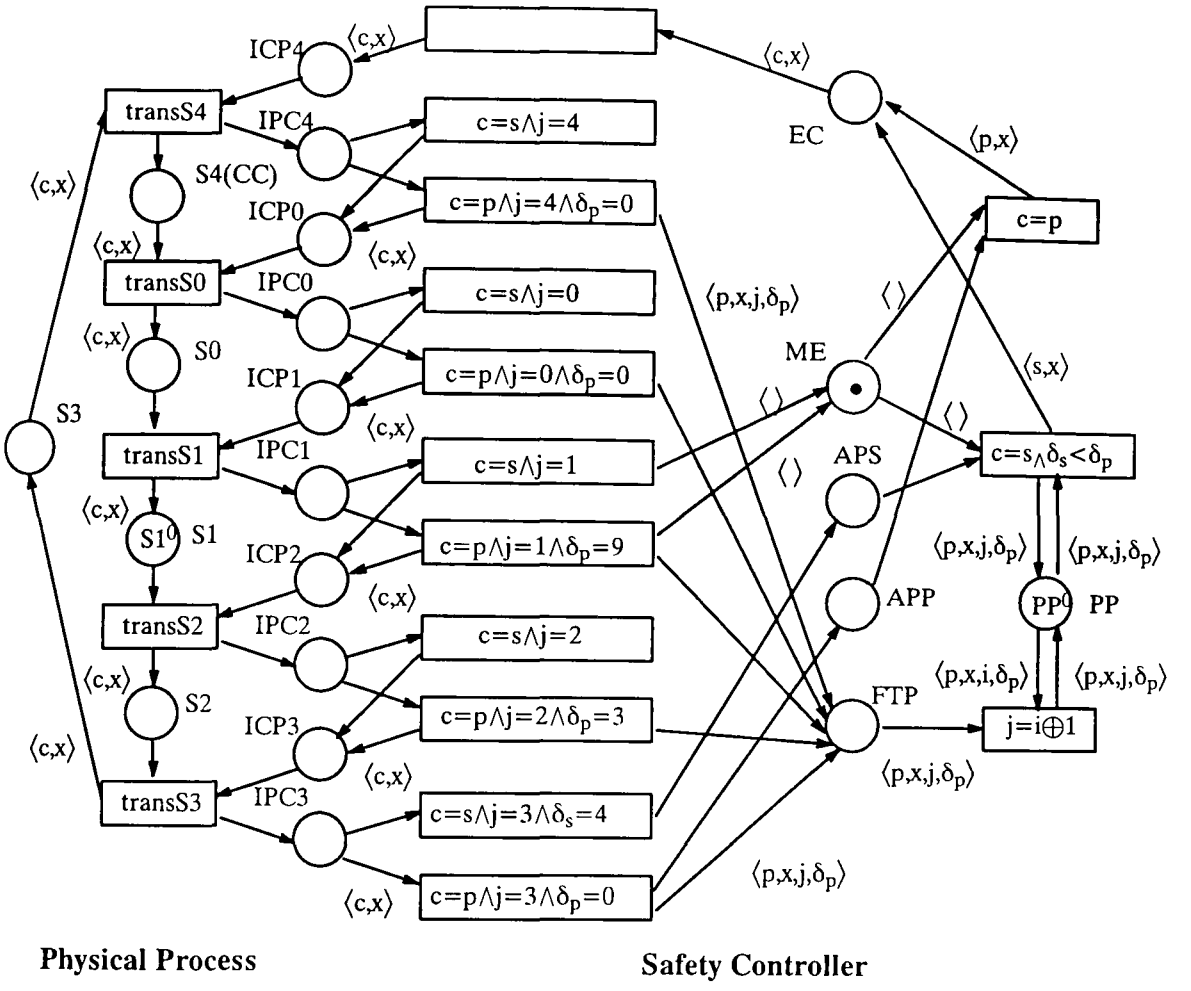


Figure 6.5. The E/A model for the priority requirement.

The PrT net model of the physical process and the safety controller which implements these rules is shown in figure 6.6. (In order to simplify the representation of the relational expressions associated with the transitions, we adopt the following notation for the duration of the actions of the safety strategy: $\delta_s = \delta_x^{APs}|_{\max} + \delta_x^{CZs}|_{\max}$ and $\delta_p = \delta_y^{APp}(t)|_{\min}$.) From the net model we notice that a primary train is never made to wait for a secondary train if the duration of key actions remains within known bounds: minimum time for a primary train to approach the crossing section, the maximum time for a secondary train to approach the crossing section, and the maximum time it takes for a secondary train to cross the danger zone. Whenever the duration of one of these key actions is no longer within the stipulated bounds, access to the crossing section is controlled by the mutual exclusion technique, which ignores the priority constraint.

The predicates of the PrT net model of the crossing section in figure 6.6 are the following (the names of the predicates follow the ones already defined in Section 6.3 during the Safety Plant Analysis):

S0cx, *S1cx*, *S2cx*, *S3cx* – a section of circuit *c* is occupied by train *x*;
S4cx (*CCcx*) – the crossing section of circuit *c* is occupied by train *x*;
ICP(j)cx – train *x* in circuit *c* is allowed to enter section *j* (= 4, the crossing section);
IPC(j)cx – train *x* in circuit *c* has entered section *j*;
$APPxj\delta_p$ – a primary train *x* is in section *j* =3 (i.e. *x* is about to enter *DZ(p,r)*), and the duration of the approach action *APp(t,i)* will take δ_p;
$APSxj\delta_s$ – a secondary train *x* is in section *j* =3 (i.e. *x* is about to enter *DZ(s,r)*), and the duration of the approach action *APs(t,i)* and crossing danger zone action *CZs(t,i)* will take δ_s;
$FPTpxi\delta_p$ – the next primary train *x* approaching the danger zone is in section *j*, and will take δ_p to approach it;
ME – either primary or secondary trains allowed to enter the crossing section;
ECcx – safety controller allows train *x* in circuit *c* to enter *DZ(c,r)*;
$PPpxi\delta_p$ – train *x* in circuit *p* is in section *i* and the approach action *APp(t,i)* will take δ_p.



transSj: $(dmax(c,j\ominus 1,j)/Vmax(c,x)) \leq tstamp(Sj) \leq (dmax(c,j\ominus 1,j)/Vmin(c,x))$, for j = 0 to 4

$S1^0=\{\langle p,1,1\rangle; \langle s,1,1\rangle\}$; $PP^0=\{\langle p,1,1\rangle\}$

Figure 6.6. The PrT net model of the crossing section.

The time intervals associated with the transitions of the model of the physical process are intended to model the movement of a train, in accordance with the physical laws established during the Safety Plant Analysis. Further refinements in the PrT net model of the safety controller will consist of introducing the timing constraints imposed on the exchange of information between the physical process and the safety controller; for instance, the time delays associated with the sensors and actuators, and the estimated execution time of the safety controller actions that form part of the control loop.

6.6. Safety Specification Hierarchy

Here we present some of the accidents, hazards, safety constraints, robust safety strategies and safety strategies associated with the train set case study. From the various accidents possible on the train set, we consider only two. The (extremely simple) safety specification hierarchy for the train set example is shown in figure 6.7.

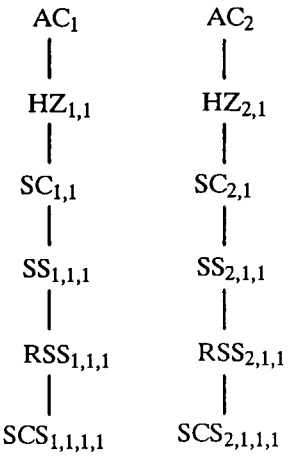


Figure 6.7. Safety specification hierarchy of the train set example.

Accidents

AC_1 – trains of the same type collide;

AC_2 – trains of different type collide.

Hazards

$HZ_{1,1}$ – some part of any two trains are in the same section.

$HZ_{2,1}$ – some part of a primary train and a secondary train are in the danger zone.

Safety Constraint

$SC_{1,1}$ – for any two trains on a circuit there must be at least one section between the sections containing the fronts of the trains:

$$\forall c \in L\colon \forall x, y \in Trc\colon [x \neq y \Rightarrow Ptrain(c,x) \notin \{Ptrain(c,y) \ominus 1, Ptrain(c,y), Ptrain(c,y) \oplus 1\}].$$

$SC_{2,1}$ – either the front of no primary train is in a danger zone $DZ(p,r)$ or the front of no secondary train is in the danger zone $DZ(s,r)$:

$$\forall r \in R: \exists c \in L: \forall x \in Trc: [\, Ptrain(c,x) \notin DZ(c,r) \,].$$

Safety Strategy

$SS_{1,1,1}$ – the safety strategy is based on a reservation scheme. The two essential rules which impose the safety strategy are as follows:

ssa. Reservation Constraint

If any train x on circuit c is in a danger zone then the crossing section contained within that danger zone is reserved for that train.

$$\forall c \in L: \forall x \in Trc: [\{Ptrain(c,x) \ominus 1, Ptrain(c,x)\} \subseteq Rtrain(c,x) \wedge$$
$$Rtrain(c,x) \subseteq \{Ptrain(c,x) \ominus 2, Ptrain(c,x) \ominus 1, Ptrain(c,x), Ptrain(c,x) \oplus 1\}];$$

ssb. Exclusion Constraint

No section can be reserved by more than one train;

$$\forall c \in L: \forall x, y \in Trc: [\, x \neq y \Rightarrow Rtrain(c,x) \cap Rtrain(c,y) = \emptyset \,].$$

$SS_{2,1,1}$ – the safety strategy is based on a reservation scheme. The two essential rules which impose the safety strategy are as follows:

ssa. Reservation Constraint

For any train the current section (i.e. the position of the train) and the section behind the current section must always be reserved:

$$\forall r \in R: \forall c \in L: \forall x \in Trc: [\, Ptrain(c,x) \in DZ(c,r) \Rightarrow$$
$$CC(c,r) \in Rtrain(c,x) \,];$$

ssb. Exclusion Constraint

Section $CC(p,r)$ and section $CC(s,r)$ cannot both be reserved at the same time:

$\forall r \in R$: $\exists c \in L$: [$\forall x \in Trc$: $CC(c,r) \notin Rtrain(c,x)$].

ssc. Priority Constraint

For any interval $[T_0, T_1]$, for any secondary train x, if $CC(s,r)$ is not reserved by train x at T_0 and during $[T_0, T_1]$ the maximum duration for all parts of train x to approach and pass the danger zone $DZ(s,r)$ from section $CC(s,r) \ominus 1$ (when it is moving) is at least the minimum duration for any primary train y to approach $DZ(p,r)$ or the current section is not $CC(s,r) \ominus 1$, then the section $CC(s,r)$ cannot be reserved by train x during $[T_0, T_1]$.

$$\forall r \in R: \forall x \in Trs: [\, CC(s,r) \notin Rtrain(s,x)(T_0) \wedge$$
$$\forall t \in [T_0, T_1]: \exists y \in Trp: \delta_x^{APs}|_{\max} + \delta_x^{CZs}|_{\max} \geq \delta_y^{APp}(t)|_{\min} \vee$$
$$Ptrain(s,x)(t) \neq CC(s,r) \ominus 1 \Rightarrow \quad \forall t \in [T_0, T_1]: CC(s,r) \notin Rtrain(s,x)(t) \,].$$

Robust Safety Strategy

$RSS_{1,1,1}$ – the robust safety strategy compensates for (consecutive) failures of the sensors (m) and actuators (n) by extending the number of sections that have to be reserved by a train. The two essential rules which impose the robust safety strategy for the circuit are as follows:

rssa. For any train, the current section as observed by the controller and the $m+n+1$ sections behind this current section must always be reserved;

$$\forall c \in L: \forall x \in Trc: [\{Pos(c,x) \ominus (m+n+1), ..., Pos(c,x)\} \subseteq Res(c,x)];$$

rssb. No section can be reserved by more than one train;

$$\forall c \in L: \forall x, y \in Trc: [\, x \neq y \Rightarrow Res(c,x) \cap Res(c,y) = \emptyset \,].$$

$RSS_{2,1,1}$ – the robust safety strategy compensates for failures of the sensors and actuators by extension of the danger zone. The two essential rules which impose the robust safety strategy for the crossing section are as follows:

rssa. If the recorded position of train x on circuit Cc is in an extended danger zone then the crossing section contained within that danger zone is reserved (on circuit Cc);

$\forall r \in R$: $\forall c \in L$: $\forall x \in Trc$:

$[\, Pos(c,x) \in DZ(c,r)^{FT} \Rightarrow CC(c,r) \in Res(c,x) \,]$;

where $DZ(c,r)^{FT} = \{CC(c,r) \ominus (m+n+1), \ldots, CC(c,r) \oplus 1\}$;

rssb. Sections $CC(p,r)$ and section $CC(s,r)$ cannot both be reserved at the same time;

$\forall r \in R$: $\exists c \in L$: $[\, \forall x \in Trc$: $CC(c,r) \notin Res(c,x) \,]$.

Safety Controller Strategy

The safety controller strategies for a circuit ($SCS_{1,1,1,1}$) and a crossing section ($SCS_{2,1,1,1}$) are the PrT net models of the safety controllers of figures 6.3 and 6.6, respectively.

6.7. Quantitative Risk Analysis of Safety Specifications

In this section, we exemplify how quantitative evaluation of a safety specification should be performed. As already mentioned, this assessment completes the qualitative analysis performed in the previous sections, in the sense that it provides more confidence that the level of risk for a certain safety specification is acceptable. Although the quantitative evaluation performed in this section is restricted to a safety strategy, a similar analysis, using different criteria, could be performed for other safety specifications. For example, the quantitative evaluation of a robust safety strategy would consider the failure rates of sensors and actuators.

To perform the quantitative evaluation of the crossing section safety strategies we employ fault tree techniques. For the fault tree analysis of safety strategy $SS_{2,1,1}$, presented in figure 6.8, we only consider those faults that can affect the behaviour of primary trains (*Trp*). Note that the refinement of rectangles I and F is symmetric to that

of rectangles H and G, respectively. The undesired state of the fault tree of $SS_{2,1,1}$ is the negation of $SC_{2,1}$. For this undesired state to occur, one of the rules of $SS_{2,1,1}$ must be violated: either at least one train does not reserve the crossing section while it is in its danger zone, or the crossing section is reserved by both trains. (In figure 6.8, RS stands for Reservation Scheme, and includes both rules of the safety strategy $SS_{2,1,1}$.)

The quantitative evaluation of the fault tree is obtained from the minimal cut sets by assigning probabilities to the primary events of the tree. Because the probability of each individual minimal cut set should be low, we use the rare event approximation equation /Vesely 81/ to obtain the minimal cut set expression for the fault tree of $SS_{2,1,1}$:

$$A = J \bullet H7 \bullet M + H8 \bullet H7 \bullet M + J \bullet H4 \bullet H5 + L \bullet I7 \bullet M + I8 \bullet I7 \bullet M + L \bullet I4 \bullet I5 + J \bullet K \bullet L \bullet M + J \bullet G1 \bullet H7 + L \bullet F1 \bullet I7.$$

In the following, in order to perform the quantitative evaluation of the safety strategy $SS_{2,1,1}$, we present estimates of the probabilities for the primary events. Some of these probabilities are related to the physical construction of the train set (P[*H7*], P[*I7*], P[*K*], P[*M*], P[*H5*], P[*I5*], P[*H4*], P[*I4*], P[*G1*] and P[*H1*]); these are the ratio of the length of a section, or set of sections, to the total length of the circuit *Cc* (for example, P[*H7*] is the ratio of the length of section $CCc \ominus 1$ (*CCc* is the short form for *CC(c,r)*) to the total length of the circuit *Cc*). Other probabilities are related to experiments conducted on the physical process (P[*H8*] and P[*I8*]), and to assumptions about expected failure rates of the components of the controller (P[*J*] and P[*L*]).

Primary Events	Probabilities
P[*J*], P[*L*]	0.001
P[*H8*], P[*I8*]	0.1
P[*H7*], P[*I7*]	0.128
P[*K*], P[*M*]	0.144
P[*H5*], P[*I5*]	0.016
P[*H4*], P[*I4*]	0.128
P[*G1*], P[*H1*]	0.144

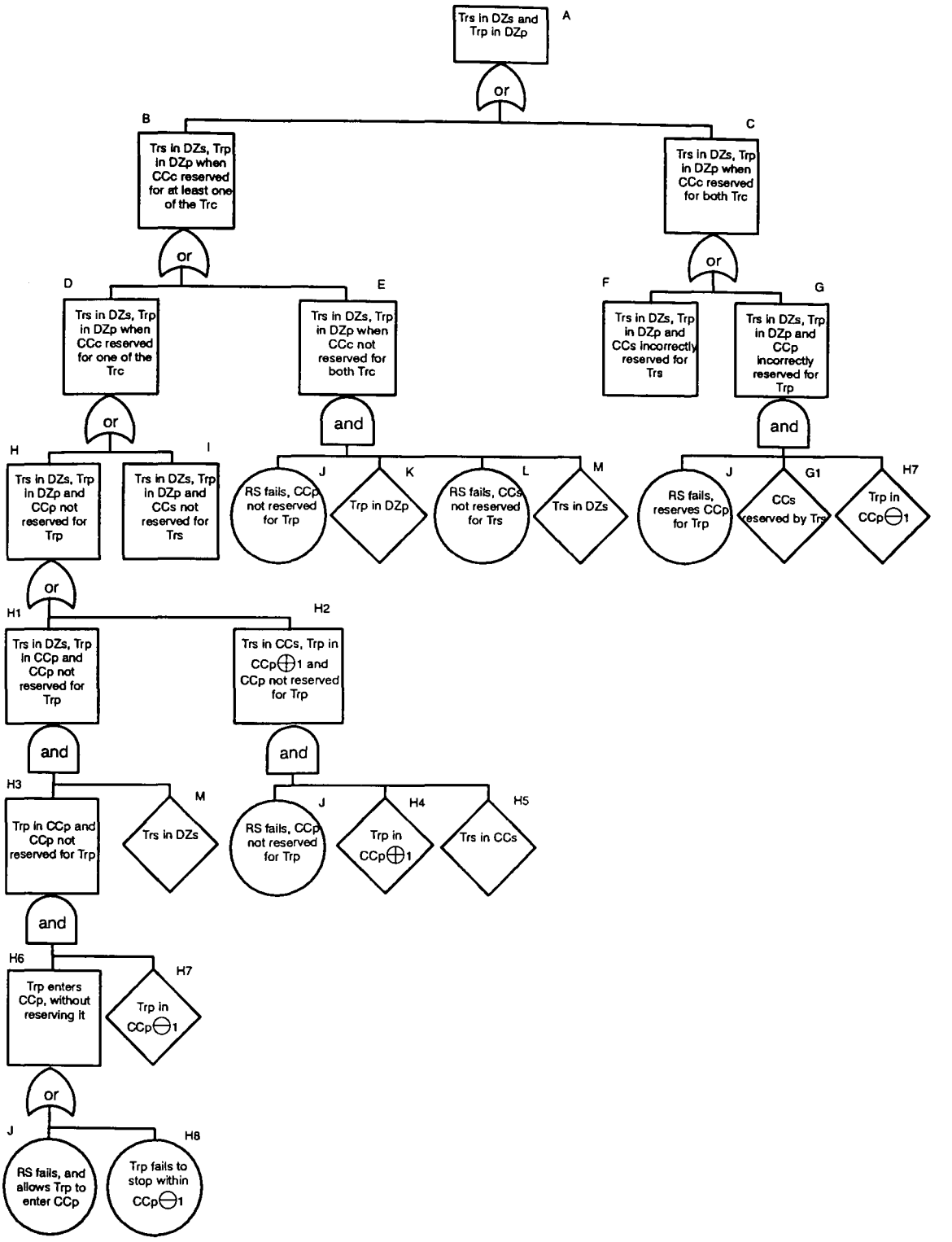


Figure 6.8. Fault Tree of $SS_{2,1,1}$.

The probabilities associated with the nine minimal cut sets are calculated as the product of the appropriate primary event failure probabilities (assuming failures to be independent). In the table below, we also take into account the relative quantitative

importance of the various minimal cut sets. This is done by calculating the ratio of the minimal cut set probability to the total undesired state probability.

Minimal Cut Set	Probability	Importance (%)
P[*J•H7•M*]	$1.84\text{x}10^{-5}$	0.49
P[*H8•H7•M*]	$1.84\text{x}10^{-3}$	48.9
P[*J•H4•H5*]	$2.05\text{x}10^{-6}$	0.05
P[*L•I7•M*]	$1.84\text{x}10^{-5}$	0.49
P[*I8•I7•M*]	$1.84\text{x}10^{-3}$	48.9
P[*L•I4•I5*]	$2.05\text{x}10^{-6}$	0.05
P[*J•K•L•M*]	$2.07\text{x}10^{-7}$	$5.5\text{x}10^{-4}$
P[*J•G1•H7*]	$1.84\text{x}10^{-5}$	0.49
P[*L•F1•I7*]	$1.84\text{x}10^{-5}$	0.49

The probability of the undesired state (negation of $SC_{2,1}$) occurring is the sum of all of the minimal cut set probabilities:

$$P[A] \approx 3.76\text{x}10^{-3}.$$

From the relative quantitative importance of each minimal cut set, we are able to analyse how the violation of an assumption may contribute to a hazard occurring. For instance, dubious assumptions (those with a high probability of being violated) should not be part of those minimal cut sets which have a high relative importance. This analysis allows us to select, from a set of safety strategies, the most adequate strategy in terms of the trade off between the safety requirements and the constraints imposed by the physical process. This can be seen in the example, where the minimal cut sets which have the assumption that a train is able to stop within one section as a primary event, have a higher relative quantitative importance. This leads us to conclude that we should not have high confidence in this safety strategy, unless we can modify it in order to guarantee that, when requested, a train always stops (within one section) before entering the crossing section.

6.8. Concluding Remarks

In this chapter, we showed how the requirements methodology could be applied to the analysis of a train set crossing. One of the benefits in applying the proposed methodology

was the identification of small domains of analysis in which a systematic approach to the analysis could be performed, in terms of the specific issues of the system, and employing the most appropriate formal method. However, we noticed that there are still some issues that have to be improved. One of these issues is the need to find an operational formalism which would enables the representation and analysis of systems which contain both discrete and continuous variables. Another issue is to find a formal linkage between the different formal methods employed in the analysis, which would facilitate the verification of the specifications obtained at different levels of abstraction.

Chapter 7

Conclusions

From both structural and behavioural perspectives, software within a system is just another component of the system. As the role of software has become more and more crucial in the struggle to obtaining more efficient and less costly systems, the complexity of software has increased without the (necessary) provision of means which would provide confidence that required levels of software quality can be attained. In respect of critical real–time systems, this is reflected by a lack of methods and techniques for effectively facilitating software development, or to adequately assess software quality while it is being developed.

In this thesis, we were particularly concerned with the phase of requirements analysis, which lately has received more emphasis due to increased recognition of its importance within the whole software development life cycle. Instead of proposing a general approach for requirements analysis which could be applied across a wide range of applications, we have restricted our approach to the class of systems known as process control systems. The advantages of having approaches for requirements analysis specific for a particular application domain have already been discussed in /Garlan 93/. One of the characteristics of process control systems is that high levels of dependability are associated with many of the application domains for this class of systems – for instance, transport systems, chemical plants and nuclear reactors. With the replacement of traditional technologies, such as hydraulic and analogue electronics, by computer based systems, the challenge has been for computer based systems, especially software, to achieve the same levels of dependability as the traditional techniques.

Essentially, the aim of the thesis is to propose an approach which systematizes the process of requirements analysis and provides confidence that the required quality of the

resulting requirements specifications can be attained. In the following, we present the achievements reported in this thesis by summarising its contents, and discuss how the proposed approach could be extended.

7.1. Achievements and Limitations

In this thesis we propose a methodology for the requirements analysis of critical real–time systems. The methodology deals with a range of issues which aim to provide confidence that the level of risk, in terms of safety, associated with the requirements specifications is acceptable. The issues that are dealt with in the methodology include: a framework with distinct phases of analysis, a graph that depicts the relationship between a set of safety specifications produced during the analysis, a set of formal techniques appropriate for the issues to be analysed at each phase, and a set of procedures for the risk (or safety) analysis of the safety specifications.

Instead of dealing with the whole range of system requirements, the analysis presented here was restricted to the safety requirements of the system; however, the methodology could be generalised to cover mission requirements. For the mission requirements, instead of adopting a formal approach, widely used rigorous methods, such as SADT and PSL/PSA, could be employed during the analysis. Separating and distinguishing the mission and the safety requirements (which is essentially a logical distinction) has the major benefit of allowing the system analyst to concentrate on what the system should *not* do.

The basis of the whole approach is the hypothetical system structure discussed in chapter 2. This structure captures the three fundamental components that are present in any system that has to be controlled: the plant, the operator and the controller. As already pointed out, each of these components has its own specific role in a system. However, depending on the level of abstraction being considered and the viewpoint adopted in the analysis of a system, other system structures could be found where the components might assume different roles.

Modelling of the passage of time, in terms of time structures, was discussed in chapter 3. Although the result of the study might look trivial, the study was nevertheless important because it provided a better understanding of time structures. However, more work needs to be done, mainly on the relationships between different time structures, perhaps in the direction of the work by Corsetti et al /Corsetti 91/. This would be of considerable benefit if analysis had to be performed for different time granularities. Another issue discussed in chapter 3 was the effects that uncertainties in the time domain can have upon the value domain, in particular on those systems which are based on synchronised clocks.

For the class of systems under consideration, process control systems, the techniques to be employed in requirements analysis need to have certain characteristics that are unique to the type of application involved. One of these characteristics is that the techniques employed have to handle both continuous and discrete variables – consequently, we have to combine algebraic and logical analysis within the same approach /Nerode 93/. In the approach presented in chapter 4, we adopted a slightly more conservative approach by making discrete the behaviour of continuous variables, and performing all of the analysis by using discrete mathematics. This is the role of the E/A model: to describe the behaviour of systems by employing a restricted set of primitive concepts, with first order logic as the underlying semantics. This set of concepts can be incorporated in existing formal techniques, such as THL and PrT nets, or can form core of new techniques, such as the PEA notation (which is not yet fully developed). Another issue that has to be tackled is the need to provide a proper and unified semantics for the techniques employed in the analysis, in order to facilitate the verification process of the formal specifications. However, this is not enough; because of the high degree of complexity of the requirements specifications, we need mechanical assistance (ideally theorem provers) to perform the verification of the specifications. Neither of these issues were discussed in the thesis.

A methodology for the analysis of safety requirements was presented in chapter 5. In the following, we summarise the four key issues on which the methodology is based. The first is the *framework for requirements analysis*, which provides a systematic approach to conduct analysis. This framework partitions the requirements analysis into phases, and defines dependencies between the phases. Each phase focuses on a specific domain of analysis, which is obtained from the system structure, providing partial knowledge of the overall system. During the analysis, a domain is viewed from different perspectives; each perspective captures a specific description of the domain. The second issue is the *directed graph* which records the safety specifications obtained from the requirements analysis; nodes represent the specifications and edges denote the relationships between the specifications. The graph provides a succinct representation which is amenable to formal analysis. The third issue is the *set of techniques* to be employed for a specific domain of analysis; from this set a technique is selected according to the properties of interest of that domain (and the perspective adopted). This leads to the utilization of a range of specialized informal and formal techniques. The problem of linking specifications obtained from different techniques is tackled by providing a primitive set of modelling concepts that are appropriate for the properties under analysis. The fourth issue is a *framework for the safety analysis* of the safety specifications which provides the confidence that the risk associated with the requirements specifications is acceptable. The quality of the requirements specifications is confirmed by firstly, applying *verification* and *validation* techniques to increase confidence that the specifications are consistent and maintain safe behaviour, and secondly, conducting a *safety analysis* (or risk analysis) of the requirements specifications. The approach enables the integration of formal techniques with traditional safety techniques to yield a coherent method for risk analysis.

The feasibility of the whole approach was demonstrated by applying the methodology to the train set case study, presented in chapter 6. Although the train set can be considered a "toy" example, the experience obtained from the application of the proposed methodology in other case studies /Anderson 93, de Lemos 93, Saeed 93a/, some of them

real systems, has shown that the complexity of the analysis depends on the assumptions made about the behaviour of the system. Usually, in order to reduce the complexity, the tendency has been to make strong assumptions about the behaviour of systems. In the train set case study, we have attempted to reduce the number of assumptions to those that are essential.

At the current stage of the proposed approach, a question that could be asked is whether the provision of tools to implement the different issues of the methodology is envisaged? It is still too soon to go this way, and one reason is that there are plenty of requirements tools available on the market that fall short of their objectives. Our aim is still to obtain a complete understanding of the process of requirements analysis – essentially, to search for improved abstraction concepts that can be used in the effective modelling and analysis of software requirements.

7.2. The Current State of the Work

In the following, we propose an approach which makes systematic the analysis of software requirements for application–specific domains, for the class of process control systems. From a generic system structure, common to an application domain, we obtain frameworks that guide analysts in the production and assessment of requirements specifications. The general approach is illustrated by presenting a framework for the analysis of safety requirements. By repeating some of the contents of the previous chapters, we seek to give a unified view of the future direction of the methodology for the requirements analysis of critical real–time systems proposed in this thesis.

Within the class of process control systems, the various types of applications can be grouped into application domains, such as chemical, aerospace, nuclear and railways, whose system structures, defined in terms of system components and interactions between components, have a great degree of similarity. In terms of requirements analysis, perhaps the main advantage, amongst several others, of grouping related systems into application domains is that to perform the requirements analysis for a

specific system should be easier, since the analysis will be based on a pre-defined set of simple abstractions /Garlan 93/.

The proposed approach systematizes the requirements analysis of process control systems that are within the same application domain. The approach relies on obtaining application-specific frameworks for requirements analysis that enhance the visibility of the requirements process, and guide the analysts in the production and assessment of the requirements specifications. The phases of these frameworks and their ordering, correspond, respectively, to the domains of analysis and their interactions, which are obtained by performing successive refinements on a generic system structure. The components identified at each refinement step provide the domains of analysis from which the requirements analysis is performed through multiple perspectives /Easterbrook 93, Finkelstein 92, Leite 91/. In other words, while a domain of analysis determines the scope of the system to be analysed, a perspective of analysis determines the scope of analysis to be performed on the domain. Both the domain and perspective of analysis have the aim of dividing up the task of requirements analysis of the whole system into manageable parts.

Although the tendency in system development has been the re-utilization of specifications from other similar systems, in the approach presented we propose the re-utilization of the *same framework* for the requirements analysis of systems within the same application domain. In order to exemplify the feasibility of the general approach for requirements analysis being proposed, we discuss in the following a framework for the analysis of safety requirements in critical real-time systems /de Lemos 93, Saeed 93b/.

7.2.1. System Structure and Refinement Process

The grouping of systems into application domains enables systems with common structures to be identified. In order to obtain a structure for a system, successive refinements are performed to the system and its components (recursively, a component

can be considered to be another system). These refinements are stopped when the components are considered as being atomic /Lee 90/. With each step in the refinement process we associate a level of abstraction which represents a particular view of the system structure, suppressing detail that is irrelevant to that view, but enabling the components of the system and their interactions to be identified. The components of the system are what make the system do what it does, and the interactions between the components prescribe the ways in which the components are connected. A detailed description of the interaction between components consists of the definition of the interface between the components (structural part) and the behaviour observed at that interface (dynamic part). To model the behaviour of a system, and its components, the following variables are introduced: *input variables* (v_u) and *output variables* (v_y) which describe the external behaviour of the system, and *state variables* (v_x) which describe the state of the system. A variable is represented by a function which maps time (T) into a set of values (V_{v_i}) of the variable, that is, $v_i(t)$: $T \rightarrow V_{v_i}$.

7.2.2. Perspectives and Domains of Analysis

In our approach, the process of requirements analysis is partitioned in terms of perspectives and domains of analysis. A *perspective of analysis* is a set of facts observed and modelled according to a particular modelling aspect. Perspectives capture a partial description of the system (or domain) without prescribing the components of those descriptions. The integration of a set of perspectives is called a *view*, which is obtained by a view–constructing process /Leite 91/. A *domain of analysis* is established from the components, or set of components, obtained from the structure of the system. The analysis performed in a domain, identifies the entities of that domain, the properties associated with these entities in terms of their standard, exceptional and failure behaviour, and the interactions between the entities. The entities identified for a domain of analysis, related to a particular level of abstraction, are considered to be the components of the next refinement level.

The different perspectives, from which a system is viewed, can establish different domains of analysis. Depending on the perspective of analysis adopted a system will present a different partial structure. However, by combining all the perspectives from which the analysis of the system is performed we obtain the complete structure of the system.

7.2.3. Frameworks

Fundamental for the definition of a framework for the requirements analysis, for a specific application domain, are the components and their interactions which describe a common system structure. The components define the domain of analysis for a system, and the interactions between the components define the dependencies that exist between the domains of analysis. A framework for requirements analysis is defined in terms of a set of phases of analysis and an ordering relation between these phases. Each phase corresponds to a domain of analysis and the ordering relations follow from the dependencies that exist between the domains of analysis. Each phase of analysis consists of the following notions.

- *Input specifications* – the specifications produced from previous phases of the framework and the requirements imposed on the entities of the domain that are used as the basis for analysing those entities.

- *Support models* – the models that are constructed to support the analysis in order to identify the entities of the domain, capture their assumed behaviour and describe their interactions.

- *Output specifications* – specifications produced during this phase that describe the required behaviour of the entities of the domain.

A framework systematizes the requirements analysis of an application domain, by organizing the analysis of the requirements into smaller domains of analysis with clearly delineated relationships. This enables analysts to systematically consider the

requirements imposed over the different domains of analysis, and guides the analysis of interactions between the requirements of related domains. Although the phases of a framework are obtained from the system structure, it is also valid to say that the structure of the system is refined as a result of the analysis performed at each phase of the framework.

As part of the methodology, also there exists another framework to conduct the risk (in terms of safety) analysis in parallel with each phase of requirements analysis, and this has the aim of ensuring that the risk associated with the safety specifications produced during each phase is acceptable. In the case where the risk associated with a safety specification is not deemed acceptable the safety specification must be modified. The overall approach to safety analysis considers both qualitative analysis and quantitative analysis /Saeed 93b/. The safety analysis to be performed for each phase of the process of requirements analysis is restricted to the information scoped by the level of abstraction of that phase, thereby limiting the effort required during in the safety analysis. The safety analysis for safety specifications, apart from analysing the normal behaviour must examine the failure behaviours of the safety specifications to determine their impact on safe behaviour. Our concern is with devising a systematic approach to examine failure behaviours of safety specifications, based on the principles that underpin traditional safety analysis techniques.

7.2.4. Techniques for the Analysis

The different activities performed during requirements analysis demand the utilization of a set of techniques whose features and expressive power match the characteristics of the activities. Selecting an appropriate technique for a specific activity allows emphasis to be placed on pertinent characteristics of the domain and enables the technique to work to its own strengths. The phases of analysis defined by the frameworks facilitate the selection of the most appropriate techniques. By examining the characteristics of the support models and output specifications, the properties that appropriate techniques must provide can be determined. As an example of identifying appropriate techniques,

we consider the role of formal support during requirements analysis. Formal support consists of formal techniques (or notations) that are used to represent and reason about the behaviour of the entities of the domain. In the context of our work, two classes of formalisms are identified: property–oriented and operational. *Property–oriented formalisms* specify behaviour in terms of the properties that are exhibited by an entity. *Operational formalisms* specify behaviour by constructing an executable model of the domain in terms of mathematical structures such as tuples, relations, functions, sets and sequences. For each phase of analysis, the properties of the entities and the interactions between entities must be determined. This suggests, respectively, the use of both property–oriented and operational formalisms. In general, the degree of utilization of one class of formalism rather than the other is related to the level of abstraction being considered. At higher levels of abstraction there is a great tendency to use property–oriented formalisms, whereas for lower levels the tendency is reversed.

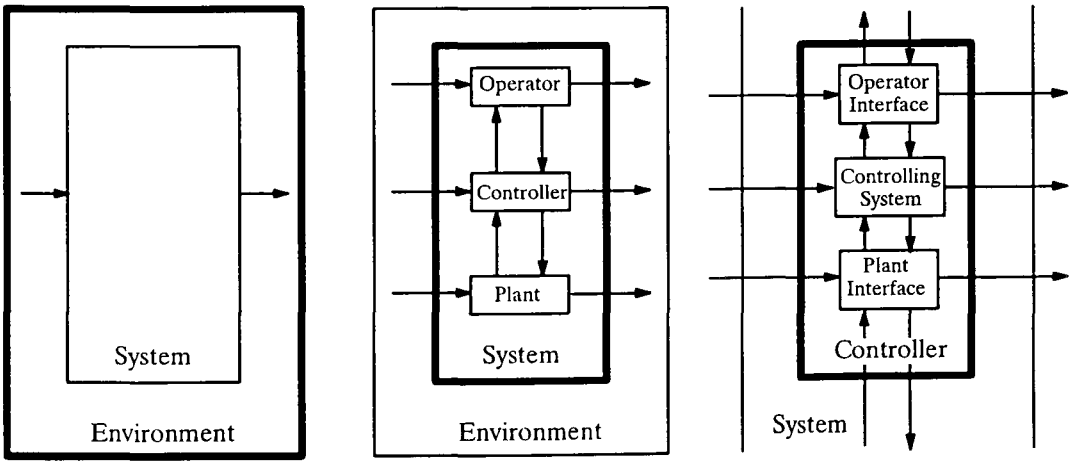
7.2.5. Analysis of Safety Requirements for Process Control Systems

In the following, we discuss the concepts presented in the previous sections in the context of an approach for the analysis of safety requirements for process control systems. To facilitate the presentation we adopt a system structure that is typically used for the class of systems under consideration. First, we identify the levels of abstraction associated with the adopted system structure, then we discuss the relationship between the domains of analysis and the structure of a system, and finally, we discuss a framework obtained from the adopted system structure.

The initial step is to enumerate the levels of abstraction identified for the adopted system structure, shown in figure 7.1. At the highest level of abstraction – the *environment level*, the system is identified within its environment, that is, the interactions between the system and its environment are stipulated. At the next level of abstraction – the *system level*, the components of the system and their interactions are identified. The adopted system structure defines three basic components: the physical process or plant, the controller, and the operator. At the next level of abstraction – the *components level*, the

following levels of abstraction are obtained from the components previously defined: the *plant level*, the *controller level*, and the *operator level*. For each of these levels of abstraction, other components are identified and recursively refined. For example, at the controller level, apart from the *controlling system*, which can be a computer based system, we identify those components of the controller that support the interfaces of the controller with the plant and the operator, which are referred to, respectively, as the plant interface and the operator interface. The levels of abstraction associated with these components are the *plant interface level* and the *operator interface level*. Figure 7.1 shows a partial system structure obtained through successive refinements. The boxes represent the components of the system, and the arrows represent the interactions between the components (that is, the inputs and outputs of the components).

At the level of abstraction of the environment the entity of analysis is the system, the behaviour of the system is described in terms of its input/output variables which are the state variables of the environment. At the level of abstraction of the system the entities of analysis are the plant, operator and the controller, and the input/output variables of these components (or sub–systems) are the state variables of the system. The same principle of refinement is applied to the other levels of abstraction; once the entities of a particular domain of analysis are defined these become domains of analysis for the next



a. Environment level b. System level c. Component (Controller) level

Figure 7.1. Levels of abstraction in process control systems.

level of abstraction. For instance, from figure 7.1 we notice that the components of the controller: the plant interface, operator interface and the controlling system – are to be considered the domains of analysis in the next level of abstraction.

Figure 7.2 illustrates a framework for the requirements analysis of process control systems, obtained by adopting the perspective of safety. The boxes represent the phases of the framework, and the arrows represent the information flows and ordering relation between the phases. The aim of the framework is to obtain the safety requirements specifications of the controlling system (a component of the controller), hence the need to perform the controller analysis from the perspectives of the plant and operator. This is stipulated in figure 7.2, by having as input specifications to the phase of controller analysis, the specifications output from the phases of plant analysis and operator analysis.

The specifications produced at the different phases of the requirements analysis are organized into a *Safety Specification Graph* (SSG). The structure embodied in modes of operation can be reflected in the organization of the requirements specifications by constructing a separate SSG for each mode. An SSG is a directed acyclic graph, in which the *vertices* represent the safety specifications and the *edges* denotes relationships between the specifications. For a mode of operation with p accidents, the SSG consists of p component graphs.

A safety specification is composed of the following elements: a safety strategy, a set of assumptions (hypotheses on the behaviour of the system at the level of abstraction being considered), and system states deemed unsafe for the level of abstraction being considered. The safety strategies are justified in terms of the assumptions which should hold for the strategy to be valid, and the system states in which the strategy is expected to be satisfied. A safety integrity level is associated with each safety specification, which must be inherited by related safety specifications at a lower level of abstraction /MoD 91/. Relative weights can be associated with the safety specifications of the same layer in order to provide a notion of relative risk associated with alternative safety

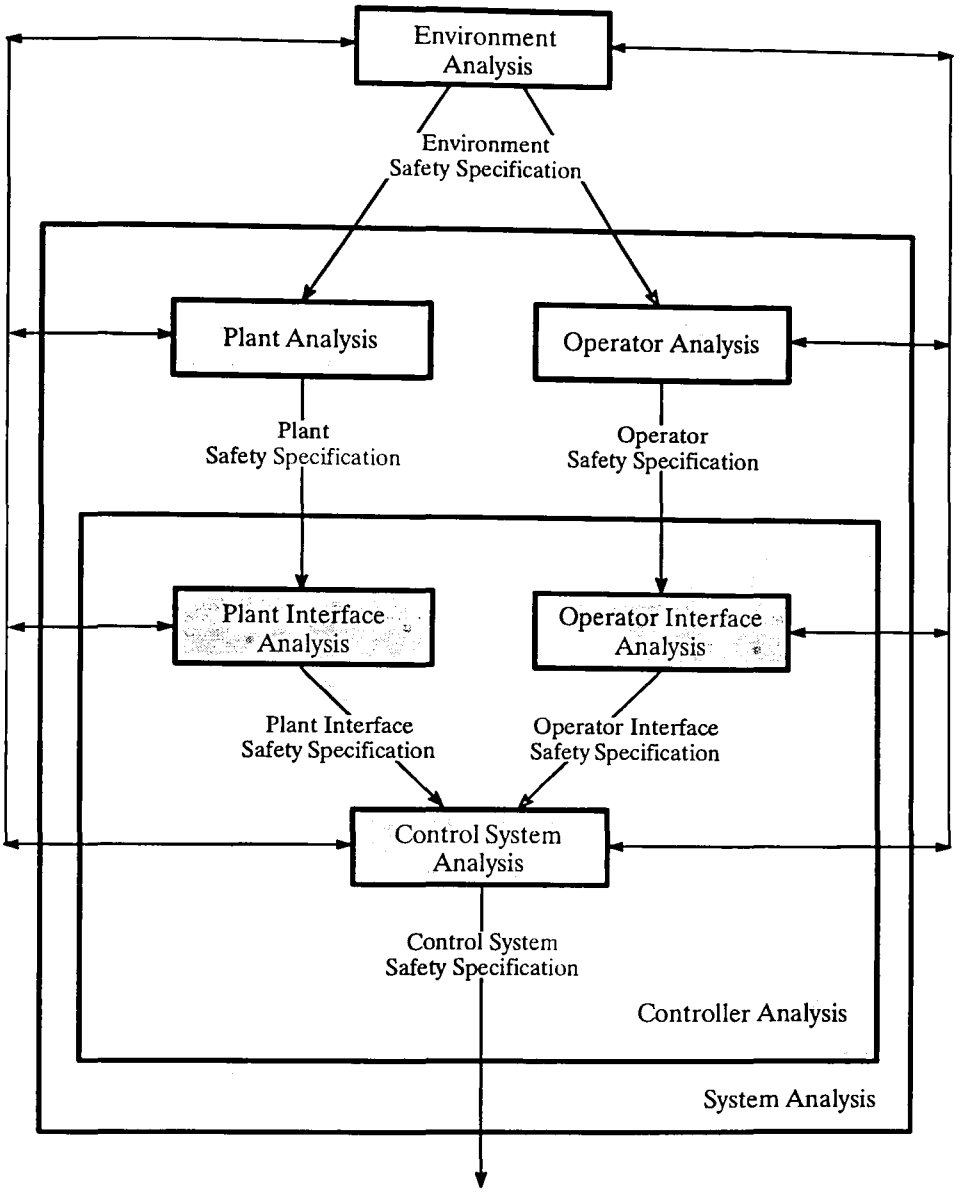


Figure 7.2. A framework for the analysis of the safety requirements.

specifications. However, the relative weights do not give the necessary assurance that a particular safety specification, or set of safety specifications, are able to attain the required safety integrity level.

The SSG encodes the relationships between safety specifications at consecutive layers of component graphs that must be confirmed to ensure that the specifications maintain safe behaviour. A relationship consists of the following elements: constraints (restrictions imposed on the safety specifications from guide-lines and standards, i.e. requirements that are not negotiable), and evidence (claim limits that are associated

with the techniques employed to perform the refinement of the safety specifications) to show that the techniques are effective in preserving the safety integrity levels associated with the safety specifications. This evidence is obtained by applying, for instance, formal proofs or rigorous analysis depending on the safety integrity levels demanded. The formal refinement process should not be used as the only evidence that a safety specification is able to preserve the safety integrity levels dictated from the safety specifications above. Additional analysis must be conducted in order to show that a safety specification does not violate the safe behaviour of the system by influencing hazards from which it is not derived.

When more than one safety specification is related to a specification in a previous layer, then either the safety specifications are exclusive alternatives and a choice will have to be made in later phases of development to select and refine a single strategy, or the strategies complement each other and all are needed to attain sufficient confidence that the risks are acceptable. These situations are distinguished by annotating the edges with a "$\oplus$" in the exclusive case, and a "$\odot$" in the complementary case. The SSG of a system provides support in conducting the risk analysis of safety specifications, provides the basis for a systematic approach to modifying the specifications, and facilitates the analysis of dependencies of the specifications on the assumptions.

7.2.6. Overview of the Approach

In the following we summarise how a framework for the requirements analysis can be obtained from the system structure while performing the requirement analysis of the system. Once a framework is obtained, it can be employed in the requirements analysis of process control systems that are in the same application domain.

Within a generic system structure, for a specific application domain, the levels of abstraction provide the means from which the system components and their interactions are identified. From the system components, domains are established from which the analysis is performed from different perspectives (e.g. mission and safety), and from the

interactions between the components the interactions between the domains of analysis are identified. Depending on the perspective adopted during the analysis of a domain, different system structures can be obtained (e.g. mission and safety controller). From the domains of analysis and their interactions, we define, respectively, the phases of a framework and their ordering. A framework enables a more systematic approach to be followed for the requirements analysis of other systems in the same application domain. For each phase of the framework the entities of analysis are identified from the analysis performed on the specifications obtained from previous phases of the framework and the requirements imposed on the domain. As a result of the analysis we specify the properties of the entities and the interactions between the entities.

7.3. Future Work

In the short and medium term, requirements analysis for critical real–time systems will require currently available solutions (in terms of methods and techniques) that are already tailored for a particular subject and type of analysis to be conducted; an example of such a solution is the approach proposed in this thesis, which exploits a set of different methods and techniques. However, in the long term, alternative solutions should be envisaged in order to provide a common approach that is independent of the subject and type of analysis to be conducted, in a similar way to the pivotal role that differential and integral calculus has in conducting different kinds of analysis for different systems across a range of fields of engineering. Although this aim might not be achieved for a wide spectrum of problems, we believe that solutions can be found for specific topics, such as the methods and techniques for conducting safety analysis in both application and software domains. In the following we describe, in the context of our work, the activities that will be conducted in the future, in order for software development (and in particular, requirements analysis) to become more like an engineering discipline.

Our methodology for requirements analysis of critical real–time systems aimed to provide an approach in which requirements analysis could be conducted in a more systematic and effective way. The purpose was not so much to present a solution to the

whole problem, but to concentrate on a set of issues that appeared to be most relevant for producing of the requirements specifications for process control systems. For this reason, the work described in this thesis could be expanded in several distinct ways, depending on the focus to be given. However, instead of describing future work in terms of issues that are needed to obtain a complete approach for requirements analysis, we have chosen to concentrate on those issues that can be tailored for the type of analysis to be conducted and for the class of systems under consideration.

The issues discussed in this thesis can be summarized as the following list of topics.

- The *modelling* of the system and its environment.
- An approach to conducting the *requirements analysis* which consists of: a method for conducting the analysis through different domains, a process of analysis that establishes the steps that should be followed at each domain, and a set of techniques which are employed (depending on the domain and the type of analysis to be performed).
- An approach for conducting *safety analysis* which is conducted in parallel with the requirements analysis and which aims to increase assurance that the level of risk associated with a requirements specification is acceptable.
- A *documentation* scheme which provides a concise and organized means for documenting the product of the requirements analysis, and provides the means by which the safety analysis is conducted.

In the following we present the future work that will be conducted in each of these topics.

The activity of domain modelling is concerned with the description of the structure and behaviour of a particular domain (a scope of analysis). One of the important issues in domain modelling is the provision of appropriate modelling abstractions that support the analysis of both behaviour and structure of a domain. In the proposed approach, the modelling abstractions being used depend on the methods and techniques being

employed. This causes difficulties, fundamentally, when there is a need to share information between the different analyses which are conducted when employing methods and techniques which are based on different modelling abstractions. A solution that we currently envisage is the utilization of a modelling abstraction based on objects. However, the principles involving the utilization of objects at the requirements analysis phase should be distinct from those applicable in the design (object–oriented design) and implementation (e.g. C++) phases. A reason for this distinction is that while during design and implementation the concern is with computational objects and the activities associated with each object, during requirements the concern is essentially with the physical objects and the identification of properties associated with those objects. The work to be conducted in this topic should initially consider how objects, as a modelling abstraction, can be exploited in domain modelling for the purpose for conducting requirements analysis. Furthermore, the work should also consider the provision of means which can facilitate the linkages, in terms of domain modelling, between the requirements analysis and the other phases of development.

In terms of requirements analysis for critical real–time systems, there are several topics for future research. In order to control and manage the complexity of the requirements analysis process, we can make use of the viewpoint analysis mechanism to determine the scope of the system to be analysed, and partition the analysis of the system into manageable parts. The mechanism of viewpoint analysis has been shown to be useful in general information processing systems /Easterbrook 93, Finkelstein 92, Leite 91/, however, nothing very concrete has been done in process control systems to demonstrate its effectiveness.

A major problem that we currently encounter when applying the proposed process for the requirements analysis is the identification of the appropriate levels of abstraction from which the analysis should be conducted. These levels of abstraction are established from an analysis of the system structure in terms of its components. In order to assist the analyst in the task of identifying the appropriate levels of abstraction, we envisage the

possibility of using knowledge based systems to provide support in identifying the relevant components of a particular level of abstraction of the system. Such knowledge based systems would need to be adequately embedded in the process of conducting requirements analysis.

In the approach proposed in this thesis, the method for conducting the requirements analysis, which leads to production of software for the controller of the system, has only considered the plant and the interface between the plant and the controller. However, this approach needs to be extended in order also to consider the operator and the operator interface, and will need to embrace an understanding of human error and the formulation of human error tolerance requirements, so as to ensure that safety is preserved in the presence of human errors. A challenging issue is how to incorporate established methods and techniques employed in the analysis of human factors within the methodology presented here. Recent work has already been performed on this topic, with some promising results.

During the presentation of our approach to conduct requirements analysis, we have only identified the phases in which the analysis should be conducted, whereas the individual stages in which the analysis is to be conducted during each phase were not fully described. Although the entities of analysis are distinct from phase to phase, there are certain activities that are common to every phase. For the future, the aim should be to identify and model (by using, for example, SADT diagrams /Ross 80/) the process of analysis to be associated with every phase of analysis, which should consist of: the input and output specifications, the activities to be conducted, and the interactions between the activities. The intent behind the modelling of the method and process for conducting the requirements is to obtain a systematic approach from which requirements analysis can be conducted. Instead of having a "product–oriented" approach for conducting the requirements analysis, we aim to establish a "process–oriented" approach which can be employed for many different systems within the same application domain. The ultimate objective is twofold; apart from being able to re–use the proposed approach (more

specifically the framework) to conduct the analysis of safety requirements across different systems within the same application domain, we also aim to be able to reuse some of the safety specifications that are produced during requirements analysis. In our understanding, whether these aims can be achieved will depend on the degree of modularity that can be obtained when establishing the levels of abstraction which are employed during the analysis.

The migration of traditional safety analysis techniques into software development has the potential to provide a good basis for an integrated approach to the overall safety analysis of both the application domain and the software domain. However, the application of traditional techniques tends to be less effective, and more complex, when applied to software because of the necessary parameterization or customization of the techniques. A more fundamental approach would be first to understand the basic (semantic) notions of the safety analysis techniques, and then investigate how such notions can be combined with the formal methods currently employed in the software domain. A more radical theme is to devise safety analysis techniques based on novel abstraction notions, such as object–oriented methods, that are appropriate for both the application and software domains.

Concerning documentation of the requirements specifications, although we have proposed a safety specification graph in which the specifications are recorded in a structured form, not much has been discussed about how such structure could be fully exploited in order to achieve a high degree of traceability between the specifications. The fact that all the safety specifications are formally recorded can be exploited in order to obtain a degree of semantic traceability for the specifications – a major advance on the more commonly found syntactic traceability. (Syntactic traceability is based on links established between the labels used to identify the elements of the specifications, while semantic traceability is based on the relationships – or transformation functions – established between the information contained within the specifications.) However, it will be an almost impossible task to achieve full semantic traceability between the safety

specifications produced at different levels of abstraction and perhaps written in different notations. A more pragmatic approach is to aim for a "quasi–semantic" traceability which would entail establishing some form of semantic relationship between a subset of the information contained within the specifications.

An important aspect that should be emphasized in future reviews of the proposed methodology is the provision of means that would allow the requirements for software to be expressed and analysed while the system in which it is to be embedded is still being designed. This reflects the fact that software has become ubiquitous in process control systems, which implies that the design of such systems should embody the software perspective. This requires a more integrated approach to system design, with the necessity of sharing information generated from different engineering domains. In order to achieve this aim, information models should be provided to allow the exchange of information produced by different methods and techniques that are specific for each of the engineering domains. However, mechanisms should also be provided to restrict the access to the information that is relevant for the analysis to be conducted. The ultimate aim of having an integrated approach for requirements analysis of critical real–time systems might only be achieved with the availability of tools that would guide the process of conducting the analysis.

In our judgement, some of the issues within the approach proposed here for the analysis of the safety requirement of a critical real–time system, could equally be applied to the broader activities of requirements analysis that were not discussed in this thesis, or even applied to later phases of software development.

References

/Allen 83/ J. F Allen. "Maintaining Knowledge about Temporal Intervals". *Communications of the ACM* Vol. 26 (11). November 1983. pp 832–843.

/Allen 84/ J. F Allen, P. J. Hayes. "A Common–Sense Theory of Time". *Procedings of the 9th International Joint Conference on Artificial Intelligence*. Los Angels, CA. August 1985. pp 528–531.

/Anderson 93/ T. Anderson, R. de Lemos, J. Fitzgerald, A. Saeed. "On Formal Support for Industrial–Scale Requirements Analysis". *Hybrid Systems*. Lectures Notes in Computer Science Vol. 736. Eds. R. L. Grossman et al. Springer–Verlag. 1993. pp 426–451.

/Åström 85/ K. J. Åström. "Process Control – Past, Present and Future". *IEEE Control Systems Magazine* Vol. 5 (3). August 1985. pp 54–60.

/Avizienis 87/ A. Avizienis, D. E. Ball. "On the Achievement of a Highly Dependable and Fault–Tolerant Air Traffic Control System". *IEEE Computer* Vol. 20 (2). February 1987. pp 84–90.

/Beyaert 81/ B. Beyaert, G. Florin, P. Lonc, S. Natkin. "Evaluation of Computer Systems Dependability using Stochastic Petri Nets". *Proceedings of the 11th IEEE International Symposium on Fault–Tolerant Computing*. Portland, Maine. June 1981. pp. 79–81.

/Barroca 92/ L. Barroca. "Using Real Time Logic to Prove Properties about Timed Statecharts and Transition Systems". *Proceedings of the 2nd International Workshop on Responsive Computer Systems*. Saitama, Japan. October 1992. pp 28–38.

/Bishop 1986/ P. Bishop. "Invariants as an Alternative to Petri Nets for Safety Design". *EWICS TC7, WP 498*. March 1986.

/Boehm 81/ B. W. Boehm. *Software Engineering Economics*. Prentice–Hall. 1981.

/Cassidy 85/ J. Cassidy, et al. "Research Needs in Manufacturing Systems". *IEEE Control Systems Magazine* Vol. 5 (3). August 1985. pp 11–13.

/Chandra 91/ V. Chandra, M. R. Verma. "A Fail Safe Interlocking System for Railways". *IEEE Design & Test of Computers*, Vol. 8 (1). March 1991. pp 58–66.

/Chintamaneni 88/ P. R. Chintamaneni, P. Jalote, Y.–B. Shieh, S. K. Tripathi. "On Fault Tolerance in Manufacturing Systems". *IEEE Network* Vol. 2 (3). May 1988. pp 32–39.

/CIGRÉ 83/ "*Interim Report on Computer Based Protection and Digital Techniques in Substations*". WG 34.02. CIGRÉ Sc 34. Tokyo. November 1983.

/Combacau 90/ M. Combacau, M. Courvoisier. "A Hierarchical and Modular Structure for F.M.S. Control and Monitoring". *Proceedings of the 1st International Conference on AI, Simulation and Planning in High Autonomy Systems*. Tucson, AZ. March 1990.

/Conner 90/ K. N. R. Conner, et at. "*A Train Control System for the Newcastle University Computing Laboratory Model Railway*". Documentation of the Group Project for the MSc in Computing Software and Systems Design. Computing Laboratory. University of Newcastle upon Tyne. April 1990.

/Corsetti 91/ E. Corsetti, et al. "Dealing with Different Time Scales in Formal Specifications". *Proceedings of the 6th International Workshop on Software Specification and Design*. Como, Italy. October 1991. pp 92–101.

/Cristian 89/ F. Cristian. "Exception Handling". In *Dependability of Resilient Computing Systems*. Ed. T. Anderson. Blackwell Scientific Publications. 1989. pp 365–397.

/Cunningham 85/ R. Cunningham, A. C. W. Finkelstein, S. J. Goldsack, T. Maibaum, C.Potts. "Formal Requirements Specification – The Forest Project". *IEEE Workshop on Specification and Design*. 1985. pp 186–191.

/Dasarathy 85/ B. Dasarathy. "Timing Constraints of Real-Time Systems: Constructs for Expressing Them, Methods of Validating Them". *IEEE Transactions on Software Engineering* Vol. SE-11 (1). January 1985. pp 80-86.

/Dauchy 91/ P. Dauchy, B. Marre. "*Test Data Selection for the Algebraic Specification of a Module of an Automatic Subway*". Laboratoire de Recherche en Informatique Research Report 638. Universite de Paris-Sud, Orsay. January 1991.

/de Lemos 90/ R. de Lemos, P. D. Ezhilchelvan. "Agreement on the Group Membership in Synchronous Distributed Systems". *Proceeding of the 4th International Workshop on Distributed Algorithms*. Lectures Notes in Computer Science Vol. 486. Eds. J. van Leeuwen, N. Santoro. Springer-Verlag. Bari, Italy. September 1990. pp 353-372.

/de Lemos 91/ R. de Lemos, A. Saeed, T. Anderson. "Value Inconsistencies due to Time Uncertainties". *IFAC Workshop on Distributed Computer Control Systems.* Semmering, Austria. September 1991. pp 1-6.

/de Lemos 92a/ R. de Lemos, A. Saeed, T. Anderson. "Analysis of Timeliness Requirements in Safety-Critical Systems". *Proceedings of the Symposium in Formal Techniques in Real-Time and Fault-Tolerant Systems*. Lectures Notes in Computer Science Vol. 571. Ed. J. Vytopil. Springer-Verlag. Nijmegen, Netherlands. January 1992. pp 171-192.

/de Lemos 92b/ R. de Lemos, A. Saeed, T. Anderson. "A Train Set as a Case Study for the Requirements Analysis of Safety-Critical Systems". *The Computer Journal.* Vol. 35 (1). February 1992. pp 30-40.

/de Lemos 92c/ R. de Lemos, A. Saeed, A. Waterworth. "Exception Handling in Real-Time Software from Specification to Design". *Proceedings of the 2nd International Workshop on Responsive Computer Systems*. Saitama, Japan. October 1992. pp 108-121.

/de Lemos 93/ R. de Lemos, A. Saeed, T. Anderson. "*Requirements Analysis for Safety–Critical Systems: A Chemical Batch Processing Example*". Department of Computing Science TR 469. University of Newcastle upon Tyne, UK. December 1993.

/de Roever 91/ W.–P. de Roever. "Foundations of Computer Science: Leaving the Ivory Tower". *Bulletin of the European Association for Theoretical Computer Science*. Number 44. June 1991. pp 455–492.

/de Silva 89/ C. W. de Silva. *Control Sensors and Actuators*. Prentice–Hall. 1989.

/Ericsson 81/ C. A. Ericsson. "Software and System Safety". *Proceedings of the 5th International System Safety Conference*. Vol. 1, part 1, Denver, 1981. pp III–B–1 to III–B–11.

/Easterbrook 93/ S. Easterbrook. "Domain Modelling with Hierarchies of Alternative Viewpoints". *Proceedings of the IEEE International Symposium on Requirements Engineering*. San Diego, CA. January 1993. pp. 65–72.

/Ezhilchelvan 90/ P. D. Ezhilchelvan, R. de Lemos. "A Robust Group Membership Algorithm for Distributed Real–Time Systems". *Proceedings of the 11th Real–Time Systems Symposium*. Lake Buena Vista, FL. December 1990. pp 173–179.

/Fidge 92/ C. J. Fidge. "Specification and Verification of Real–Time Behaviour using Z and RTL". *Proceedings of the Symposium in Formal Techniques in Real–Time and Fault–Tolerant Systems*. Lectures Notes in Computer Science Vol. 571. Ed. J. Vytopil. Springer–Verlag. Nijmegen, Netherlands. January 1992. pp 393–409.

/Finkelstein 92/ A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, M. Goedicke. "Viewpoints: A Framework for Integrating Multiple Perspectives in System Development". *International Journal of Software Engineering and Knowledge Engineering* Vol. 2 (1). March 1992. pp.31–57.

/Franklin 86/ G. F. Franklin, J. D. Powell, A. Emami–Naeini. "*Feedback Control of Dynamic Systems*". Addison–Wesley. 1986

/Garlan 93/ D. Garlan. "Domain Specification Require First Class Connectors". *Proceedings of the IEEE International Symposium on Requirements Engineering*. San Diego, CA. January 1993. pp. 78.

/Genrich 87/ H. Genrich. "Predicate/Transition Nets". *Petri Nets: Central Models and their Properties*. Lectures Notes in Computer Science Vol. 254. Eds. W. Brauer, W. Reisig, G. Rozemberg. Springer–Verlag. 1987. pp 206–247.

/Ghezzi 90/ C. Ghezzi, D. Mandrioli, A. Morzenti. "TRIO: A Logic Language for Executable Specifications of Real–Time Systems". *Journal of Systems and Software* Vol 12. June 1990. pp 107–123.

/Ghezzi 91/ C. Ghezzi, D. Mandrioli, S. Morasca, M. Pezzè. "A Unified High–Level Petri Net Formalism for Time–Critical Systems". *IEEE Transactions on Software Engineering* Vol. SE–17 (2). February 1991. pp 160–172.

/Goldsack 91/ S. J. Goldsack, A. C. W. Finkelstein. "Requirements Engineering for Real–Time Systems". *IEE Software Engineering Journal*. May 1991. pp 101–115.

/Gordon 88/ M. J. C. Gordon. "*Mechanizing Programming Logics in Higher Order Logic*". Computer Laboratory, University of Cambridge Technical Report 145. Cambridge, UK. September 1988.

/Gorski 86/ J. Gorski. "Design for Safety using Temporal Logic". *SAFECOMP'86*. Sarlat, France. October 1986. pp 149–155.

/Goswami 92/ A. Goswami, M. Bell, M. Joseph. "ISL: An Interval Logic for the Specifications of Real–Time Programs". *Proceedings of the Symposium in Formal Techniques in Real–Time and Fault–Tolerant Systems*. Lectures Notes in Computer Science Vol. 571. Ed. J. Vytopil. Springer–Verlag. Nijmegen, Netherlands. January 1992. pp 1–20.

/Hachiga 92/ A. Hachiga. "The Concepts and Technologies of Dependable and Real–Time Computer Systems for Shinkasem Train Control". *Proceedings of the 2nd*

International Workshop on Responsive Computer Systems. Dependable Computing and Fault–Tolerant Systems. Eds. Y. Kakuda, H. Kopetz. Springer–Verlag. Saitama, Japan. October 1992. (to appear.)

/Harel 87/ D. Harel. "Statecharts: A Visual Formalism for Complex Systems". *Science of Computer Programming,* Vol 8. 1987. pp 231–274.

/Harel 90/ D. Harel, et. al. "Statemate: A Working Environment for the Development of Complex Reactive Systems". *IEEE Transactions on Software Engineering* Vol. SE–16 (4). April 1990. pp 403–414.

/He 90/ X. He, J. A. N. Lee. "Integrating Predicate Transition Nets with First Order Temporal Logic in the Specification and Verification of Concurrent Systems". *Journal on Formal Aspects of Computing*. Vol. 2 (3). 1990. pp 226–246.

/Heninger 80/ K. L. Heninger. "Specifying Software Requirements for Complex Systems: New Techniques and Their Application". *IEEE Transaction on Software Engineering* Vol. SE–6 (1). January 1990. pp 2–13.

/Henley 92/ E. J. Henley, H. Kumamoto. "*Probabilistic Risk Assessment*". IEEE Press. 1992.

/Ho 89/ Y. C. Ho. "Dynamics of Discrete Event Systems". *Proceedings of the IEEE* Vol. 77 (1). January 1989. pp 3–6.

/Hoare 85/ C. A. R. Hoare. "*Communicating Sequential Processes*". Prentice–Hall International. 1985.

/ICE 85/ The Institution of Chemical Engineers. "Hazard Identification Procedures". *Risk Analysis in the Process Industries*. England. 1985. pp 7–19.

/Jaffe 91/ M. S. Jaffe, N. G. Leveson, M. P. E. Hiemdahl, B. E. Melhart. "Software Requirements Analysis for Real–Time Process–Control Systems". *IEEE Transactions on Software Engineering* Vol. SE–17 (3). March 1991. pp 241–258.

/Jahanian 86/ F. Jahanian, A. Mok. "Safety Analysis of Timing Properties in Real-Time Systems". *IEEE Transaction on Software Engineering* Vol. SE-12 (9). September 1986. pp 890-904.

/Jahanian 88a/ F. Jahanian, A. K. Mok, D. A. Stuart. "*Formal Specifications of Real-Time Systems*". Department of Computer Sciences TR-88-25. University of Texas at Austin, TX. June 1988.

/Jahanian 88b/ F. Jahanian, D. A. Stuart. "A Method for Verifying Properties of Modechart Specifications". *Proceedings of the 9th Real-Time Systems Symposium.* Huntsville, AL. December 1988. pp 12-21.

/Jensen 85/ E. Jensen, D. Locke, H. Tokuda. "A Time-Driven Scheduling Model for Real-Time Operating Systems". *Proceedings of the 6th Real-Time Systems Symposium.* San Diego, CA. December 1985. pp 112-122.

/Jones 86/ C. B. Jones. "Software Development: A Rigorous Approach". Prentice-Hall. Hemel Hempstead, UK. 1986.

/Kirrmann 87/ H. Kirrmann. "*A Method for the Design of Embedded Faul-Tolerant Computers for Process Control and a Design Example*". Technical Report CRB 87-70 C. Brown-Boveri. Baden, Switzerland. April 1987.

/Kopetz 87/ H. Kopetz, W. Ochsenreiter. "Clock Synchronization in Distributed Real-Time Systems". *IEEE Transactions on Computers* Vol. C-36 (8). August 1987. pp 933-940.

/Kopetz 89/ H. Kopetz, et al. "Distributed Fault-Tolerant Real-Time Systems". *IEEE Micro* 9 (1). February 1989. pp 25-40.

/Kopetz 89/ H. Kopetz, G. Grunsteidl, J. Reisinger. "Fault-Tolerant Membership Service in a Distributed Real-Time System". *Proceedings of the International Conference on Dependable Computing for Critical Applications.* Santa Barbara, CA. August 1989. pp 167-174.

/Kopetz 90/ H. Kopetz, K. H. Kim. "Temporal Uncertainties in Interactions among Real-Time Objects". *Proceedings of the 9th Symposium on Reliable Distributed Systems.* Huntsville, AL. October 1990. pp 165-174.

/Koutny 86/ M. Koutny. "The Merlin-Randell Problem of Train Journeys". *Acta Informatica,* Vol. 23 (4). July 1986. pp 429-463.

/Koymans 88/ R. Koymans, R. Kuiper, E. Zijlstra. "Paradigms for Real-Time Systems". *Proceedings of the Symposium in Formal Techniques in Real-Time and Fault-Tolerant Systems.* Lectures Notes in Computer Science Vol. 331. Ed. M. Joseph. Springer-Verlag. Warwick, UK. September 1988. pp 159-174.

/Laprie 89/ J.C. Laprie. "Dependability: A Unifying Concept for Reliable Computing and Fault-Tolerance". In *Dependability of Resilient Computing Systems.* Ed. T. Anderson. Blackwell Scientific Publications. 1989. pp 1-28.

/Laprie 91/ J.-C. Laprie, B. Littlewood. "Quantitative Assessment of Safety-Critical Software: Why and How?". *Probabilistic Safety Assessment and Management Conference.* Beverly Hills, CA. February 1991.

/Laprie 92/ J.-C. Laprie. "For a Product-in-a-Process Approach to Software Reliability Evaluation". *Proceedings of the 3rd International Symposium on Software Reliability Engineering.* Research Park Triangle, NC. October 1992. pp 134-139.

/Lee 90/ P. A. Lee, T. Anderson. *"Fault Tolerance: Principles and Practices" (Second, revised edition*). Springer-Verlag. Wien. 1990.

/Leite 91/ J. C. S. P. Leite, P. A. Freeman. "Requirements Validation through Viewpoint Resolution". *IEEE Transactions on Software Engineering* Vol. SE-17(2). December 1991. pp. 1253-1269.

/Leveson 86/ N. G. Leveson. "Software Safety: Why, What and How". *ACM Computing Surveys* Vol. 18 (2). June 1986. pp 125-163.

/Leveson 87/ N. G. Leveson, J. Stolzy. "Safety Analysis Using Petri Nets". *IEEE Transactions on Software Engineering* Vol. SE–13 (3). March 1987. pp 386–397.

/Leveson 91a/ N. G. Leveson, S. S. Cha, T. J. Shimeall. "Safety Verification of Ada Programs using Software Fault Trees". *IEEE Software*. July 1991. pp 48–59.

/Leveson 91b/ N. G. Leveson. "Software Safety in Embedded Computer Systems". *Communications of the ACM* Vol. 34 (2). February 1991. pp 34–46.

/Leveson 93/ N. G. Leveson. "An Investigation of the Therac–25 Accidents". *Computer* Vol. 26 (7). July 1993. pp 18–41.

/Lowe 71/ E. I, Lowe. "*Computer Control in Process Industries*". Peregrinus. London. 1971.

/Maler 92/ O. Maler, Z. Manna, A Pnuelli. "From Timed to Hybrid Systems". *Proceedings of the REX Workshop Real–Time: Theory and Practice*. Lectures Notes in Computer Science Vol. 600. Ed. J. W. de Bakker, C. Huizing, and G. Rozenberg. Springer–Verlag. 1992

/Marshall 71/ C. W. Marshall. "*Applied Graph Theory*". Wiley–Interscience. 1971.

/Merlin 76/ P. Merlin. "Methodology for the Design and Implementation of Communication Protocols". *IEEE Transactions on Communications* Vol. COM–24 (6). June 1976. pp. 614–621.

/Merlin 78/ P. Merlin, B. Randell, "Notes on Deadlock Avoidance on the Train Set". *Report MRM/144.* Comp. Lab. University of Newcastle upon Tyne. 1978.

/Middleburg 89/ C. A. Middleburg. "VVSL: A Language for Structured VDM Specifications". *Formal Aspects of Computing* Vol. 1. January–March 1989. pp 115–135.

/Miller 91/ D G Miller. "The Role of Statistical Modeling and Inference in Software Quality Assurance". *Software Certification.* Ed. B de Neumann. Elsevier Applied Science, 1990, pp 135–152

/Milner 83/ R. Milner, "Calculi for Synchrony and Asynchrony". *Theoretical Computer Science* Vol. 25. 1983. pp 267–310.

/MoD 91a/ Draft Interim Defence Standard 00–55. *The Procurement of Safety Critical Software in Defence Equipment*. UK Ministry of Defence. London. April 1991.

/MoD 91b/ Draft Interim Defence Standard 00–56. *Hazards Analysis and Safety Classification of the Computer and Programmable Eletronic System Elements of Defence Equipment*. UK Ministry of Defence. London. April 1991.

/Morasca 89/ S. Morasca, M. Pezzè. "A Rationale of an Environment for Real–Time Software". *Proceedings Euromicro Workshop on Real Time*. Como, Italy. June 1989. pp 37–42.

/Moser 90/ L. E. Moser, P. M. Melliar–Smith. "Formal Verification of Safety–Critical Systems". *Software Practice and Experience* Vol. 20 (8). August 1990. pp 799–821.

/Murata 89/ T. Murata. "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE* Vol 77 (4). April 1989. pp 541––580.

/Musa 90/ J. D. Musa, W. W. Everett. "Software–Reliability Engineering: Technology for 1990s". *IEEE Software 7(6)*. November 1990. pp 36–43.

/Nerode 93/ A. Nerode, W. Kohn. "Models for Hybrid Systems: Automata, Topologies, Contrabillity, Observability". *Hybrid Systems*. Lectures Notes in Computer Science Vol. 736. Eds. R. L. Grossman et al. Springer–Verlag. 1993. pp 317–356.

/Newton–Smith 81/ W. Newton–Smith. *The Structure of Time*. Oxford University Press. Oxford, UK. 1981

/Olderog 91/ E.–R. Olderog. *"Nets, Terms and Formulas"*. Cambridge Tracts in Theoretical Computer Science 23. 1991.

/Ostroff 86/ J. S. Ostroff. "*Real–Time Computer Control of Discrete Systems Modelled by Extended State Machines: A Temporal Logic Approach*". Report 8618. University of Toronto. System Control Group. September 1986.

/Ostroff 87/ J. S. Ostroff, W. M. Wonham. "Modelling, Specifying and Verifying Real–Time Embedded Computer Systems". *Proceedings of the 8th Real–Time Systems Symposium*. San Jose, CA. December 1987. pp 124–132.

/Ostroff 90/ J. S. Ostroff. "Deciding Properties of Timed Transition Models". *IEEE Transactions on Parallel and Distributed Systems* Vol. PDS 1 (2). April, 1990. pp 170–183.

/Parnas 91/ D. L. Parnas, J. Marley. "*Functional Documentation for Computer Systems Engineering*". CRL Report No. 237. McMaster University. Hamilton, Canada. September 1991.

/PDCS 90/ "Real–Time Systems (Specific Closed Workshop)". *ESPRIT PDCS Workshop Report W6*. London, UK. September 1990.

/Pnueli 86/ A. Pnueli. "Specification and Development of Reactive Systems". *Information Processing 86*. Ed. H.J. Kugler. 1986. pp 845–858.

/Pnueli 88/ A. Pnueli, E. Harel, "Applications of Temporal Logic to the Specification of Real Time Systems". *Proceedings of the Symposium in Formal Techniques in Real–Time and Fault–Tolerant Systems.* Lectures Notes in Computer Science Vol. 331. Ed. M. Joseph. Springer–Verlag. Warwick, UK. September 1988. pp. 84–97.

/Powell 92/ D. Powell. "Failure Mode Assumptions and Assumption Coverage". *Proceedings of the 22nd International Symposium on Fault–Tolerant Computing.* Boston, Massachusetts. July, 1992. pp 386–395.

/Ramamoorthy 85/ C. V. Ramamoorthy, N.–T. Tsai, T. Yamura, A. Bhide. "Metrics Guided Methodology". *Proceedings 9th International Computer Software and Applications Conference – COMPSAC'85.* Chicago, IL. October 1985. pp 111–120.

/Ramchandani 74/ C. Ramchandani. *"Analysis of Asynchronous Concurrent Systems by Petri Nets"*. MIT MAC TR–120. February 1974.

/Ravn 93/ A. P. Ravn, H. Rischel, K. M. Hansen. "Specifying and Verifying Requirements of Real–Time Systems". *IEEE Transactions on Software Engineering* Vol. SE–19 (1). January 1993. pp 41–55.

/Reed 86/ G. M. Reed, A.W. Roscoe, "A Timed Model for Communicating Sequential Processes". *Proceedings of 13th International Colloquium on Automata, Languages and Programming*. Lectures Notes in Computer Science Vol. 226. Ed. L. Kott. Springer–Verlag. Rennes, France. July 1986. pp 314–323.

/Ross 80/ D. Ross. "Structured Analysis (SA): A Language for Communicating Ideas". *Tutorial on Design Techniques*. P. Freeman and M. Wasserman (Eds.). IEEE Computer Society Press. 1980. pp. 107–125.

/Rouquet 86/ J. C. Rouquet, P. J. Traverse. "Safe and Reliable Computing on Board the Airbus and ATR Aircraft". *Proceedings of the 5th IFAC Workshop on Safety of Computer Control Systems.* Ed. W. J. Quirk. 1986. pp 93–97.

/Rushby 91/ J. Rushby, F. von Henke, A. Owre. *"An Introduction to Formal Specification and Verification using EHDM"*. Computer Science Laboratory, SRI International Technical Report SRI–CSL–91–2. Menlo Park, CA. February 1991.

/Saeed 90a/ A. Saeed, T. Anderson, M. Koutny. "A Formal Model for Safety–Critical Computing Systems". *SAFECOMP'90*. London, UK. October 1990. pp 1–6.

/Saeed 90b/ A. Saeed. *"A Framework for the Requirements Analysis of Safety–Critical Computing Systems"*. Ph.D. Thesis. Computing Laboratory. University of Newcastle upon Tyne. 1990.

/Saeed 91a/ A. Saeed, R. de Lemos, T. Anderson. "The Role of Formal Methods in the Requirements Analysis of Safety–Critical Systems: a Train Set Example". *Proceedings*

of the 21st Symposium on Fault–Tolerant Computing. Montreal, Canada. June 1991. pp 478–485.

/Saeed 91b/ A. Saeed, R. de Lemos, T. Anderson. "The Role of Formal Methods in the Requirements Analysis of Safety–Critical Systems: a Train Set Example". *2nd Year Report ESPRIT BRA Predictably Dependable Computing Systems* Vol. 3. May 1991.

/Saeed 92/ A. Saeed, R. de Lemos, T. Anderson. *"An Approach to the Assessment of Requirements Specifications for Safety–Critical Systems"*. Computing Laboratory TR 381. University of Newcastle upon Tyne, UK. April 1992.

/Saeed 93a/ A. Saeed, R. de Lemos, T. Anderson. "Formal Techniques for Requirements Analysis for Safe Reactor Control". *The Nuclear Engineer* Vol 34 (4). July/August 1993. pp 108–115.

/Saeed 93b/ A. Saeed, R. de Lemos, T. Anderson. "Robust Requirements Specifications for Safety–Critical Systems". *Proceedings of the 12th International Conference on Computer Safety, Reliability and Security* (*SAFECOMP'93*). Springer–Verlag. J. Górski (Ed.). Poznan–Kiekrz, Poland. October 1993. pp 219–229.

/Sayet 90/ C. Sayet, E. Pilaud. "An Experience of a Critical Software Development". *Proceedings of the 20th Symposium on Fault–Tolerant Computing*. Newcastle upon Tyne, UK. June 1990. pp 36–45.

/Scholefield 92/ D. J. Scholefield, H. S. M. Zedan. "TAM: A Formal Framework for the Development of Distributed Real–Time Systems". *Proceedings of the Symposium in Formal Techniques in Real–Time and Fault–Tolerant Systems*. Lectures Notes in Computer Science Vol. 571. Ed. J. Vytopil. Springer–Verlag. Nijmegen, Netherlands. January 1992. pp 411–428.

/Shrivastava 90/ S. K. Shrivastava, L. V. Mancini, B. Randell. *"The Duality of Fault–Tolerant System Structures"*. Computing Laboratory Technical Report No. 305. Newcastle upon Tyne, UK. February 1990

/Shrivastva 91/ S. K. Shrivastava, A. Waterworth."Using Objects and Actions tp provide Fault−Tolerance in Distributed, Real−Time Applications". *Proceedings of the 12th Real−Time Systems Synposium*. San Antonio, Texas. December 1991. pp 276−285.

/Sibertin 85/ C. Sibertin−Blanc. "High−Level Petri Nets with Data Structure". *Proceedings of the 6th European Workshop on Application and Theory of Petri Nets*. Espoo, Finland. June 1985.

/Stankovic 89/ J. Stankovic, W. Halang. M. Tokoro. Editorial. *The Journal of Real−Time Systems* Vol. 1 (1). June 1989. pp 5−6.

/Traverse 89/ P. Traverse. "Dependability of Digital Computers on Board Airplanes". *Proceedings of the International Conference on Dependable Computing for Critical Applications*. Santa Barbara, CA. August 1989. pp 53−60.

/Tsai 92/ J. J. P. Tsai, T. Weigert, H.−C. Jang. "A Hybrid Knowledge Representation as a Basis of Requirements Specification and Specification Analysis". *IEEE Transactions on Software Engineering* Vol. 18(12). December 1992. pp 1076−1100.

/van Benthem 90/ J. van Benthem. "*The Logic of Time*". Kluwer. 1991.

/Vesely 81/ W. E. Vesely, F. F. Goldberg, N. H. Roberts, D. F. Haasl. "*Fault Tree Handbook*". US Nuclear Regulatory Commission NUREG−0492. Washington, DC. January 1981.

/Wells 80/ G. L. Wells. "Formal Safety Studies". *Safety in Process Plant Design*. John Woley & Sons. New York. 1980. pp 101−120.

/Whitehead 29/ A. N. Whitehead. "*Process and Reality*". Cambridge University Press. Cambridge, UK. 1929.

/Wilkie 90/ J.D.F.Wilkie. "Batch Process Control". *Computer Control of Real−Time Processes*. Eds. S. Bennett and G.S.Virk. 1990. pp. 274−285.

/Williams 84/ T. J. Williams. "The Development of Reliability in Industrial Control Systems". *IEEE Micro* Vol. 4 (6). December 1984. pp 66–80.

/Wing 90/ J. M. Wing. "A Specifier's Introduction to Formal Methods". *Computer* Vol. 23 (9). September 1990. pp 8–24.

/Wohlin 92/ C. Wohlin, P. Runeson. "A Method Proposal for Early Software Reliability Estimation". *Proceeding of the Third International Symposium on Software Reliability Engineering*. Research Triangle Park, NC. October 1992. pp. 156–163.

Appendix A

Timed History Logic (THL)

THL is a formalism based on a dense time structure and consists of three main concepts: *histories* which are the underlying semantics of the model, *relations* which are constructs used to capture properties that are invariant over histories, and *modes* which are used to structure specifications expressed by the model. Here we present an overview of histories and relations; a more detailed description of the model is given elsewhere /Saeed 90/.

For a system with n state variables we have the state vector: $SV = \langle v_1, v_2, ..., v_n \rangle$ The set of possible values of a state variable (say, v_i) is defined by its variable range (V_{v_i}). The state space of a system (Γ) is defined as the cross product of the variable ranges in the state vector SV. The *system lifetime* ($\boldsymbol{T}$) is an interval which represents the operational lifetime of the system. A history H of a system is a function of the form H: $\boldsymbol{T} \rightarrow \Gamma$. The set of all "possible" histories of a system are defined as the universal history set $\Gamma\mathbf{H}$. For a history H the sequence of values taken by a variable v_i of SV is denoted by the function $H.v_i$: $\boldsymbol{T} \rightarrow V_{v_i}$. Over any interval the following functions are defined: *start point s(***Int***)* – the earliest time point in **Int**; *end point e(***Int***)* – the latest time point in **Int**; and *interval set SI(***Int***)* – the set of all intervals contained within **Int**.

Specifications are expressed by restricting the set of histories by two sorts of relations: invariant and history relations.

Invariant relations are used to express relationships over the state variables which hold at every time point within $\boldsymbol{T}$; these are formulated as system predicates. A *system predicate* is a predicate built using n free value variables $v_1, ..., v_n$ of types $V_{v_1}, ..., V_{v_n}$. No other free variables may be used. A tuple of values $V = \langle x_1, ..., x_n \rangle$, where x_i is of type V_{v_i} satisfies a system predicate P if and only if substitution of each x_i for v_i within $P(x_1, ..., x_n)$ evaluates to **true**. This is denoted by: $P(V)$. We will denote $P(H.v_1(t), ..., H.v_n(t))$ by H **sat** $P@t$. A history H satisfies an invariant relation P *iff*: $\forall\, t \in \boldsymbol{T}$: H **sat** $P@t$.

History relations are used to express relationships over the state variables which hold during every interval included within T; these are formulated as history predicates. A *history predicate* is a predicate built using two free time variables T_0, T_1 (T_0 and T_1 should be interpreted as being universally quantified over T) and n free function variables v_1, ..., v_n. No other free variables may be used. A history H satisfies a history predicate HP for an interval **Int** if and only if the expression resulting from substituting: (i) $s(Int)$ for T_0, (ii) $e(Int)$ for T_1, and (iii) $H.v_i$ for v_i for all i, evaluates to **true**. This is denoted by: H **sat** HP@**Int**. A history H satisfies an history relation HP *iff*: $\forall$ **Int** $\in SI(T)$: H **sat** HP@**Int**.

Appendix B

Petri Nets

B.1. Introduction

B.2. Petri Net Model

Let S, T, F be finite sets. The triple $N=(S, T, F, W)$ is called a *net structure* if and only if the following conditions hold:

$S \cap T = \emptyset$;

$S \cup T \neq \emptyset$;

$F \subseteq (S \times T) \cup (T \times S)$;

$W: F \rightarrow \mathrm{N}$, and

domain $(F) \cup$ codomain $(F) = S \cup T$.

For a given net $N=(S, T, F, W)$, S is the set of *places* of N, T is the set of *transitions* of N, F is the flow relation containing the *arcs* of N, and W the weight function.

For all $x \in S \cup T$ the *preset* $I(x)$ and the *postset* $O(x)$ are defined as follows:

$I(x) = \{y \in S \cup T \mid (y, x) \in F\}$, and

$O(x) = \{y \in S \cup T \mid (x, y) \in F\}$.

A *Petri net* is a net structure with a given initial marking (M_0), and is denoted as $PN=(N, M_0)$. A *marking* M of a Petri net is a mapping of the set of places S in to N $(M: S \rightarrow \mathrm{N})$, the set of natural numbers. With $|S|=m$, a marking M is represented by vector $M \in \mathrm{N}^m$; its *ith* component is denoted by $M(s_i)$.

A transition $t \in T$ is enabled in marking M if and only if $\forall s \in I(t)$: $M(s) \geq w(s, t)$, where $w(s, t)$ is the weight of arc from s to t. A firing occurrence produces a new marking M' from

M by choosing an enabled transition t in M. The new marking M' is defined as follows: $\forall s \in I(t): M'(s)=M(s)-w(s,t)$, $\forall s \in O(t): M'(s)=M(s)+w(t,s)$, and $M'(s)=M(s)$ for all other places. A finite sequence of transitions is called a *firing sequence* if transition t_i, $1 \leq i \leq n$, is enabled in marking M_{i-1} and its firing produces the marking M_i, starting from M_0.

B.3. Predicate–Transition Nets (PrT Nets)

Predicate–Transition nets (PrT nets) /Genrich 87/ is a form of high–level Petri net which is mainly used for the modelling and analysis of discrete–event systems which are concurrent, asynchronous, and non–deterministic.

A PrT net consists of the following constituents:

(1) a net structure (S, T, F) where, S is the set of predicates, and T is the set of transitions;

(2) predicates are variable relations amongst individuals ("first–order" places);

(3) the transitions are schemes of elementary changes of markings representing the actions carried out by the system;

(4) an arc label specifies a variable extension of a predicate to which the arc is connected;

(5) a marking is a mapping that assigns to each predicate formal sums of n–tuples of individual symbols, also called tuples.

The graphical representation of a PrT net is obtained by representing a predicate by a circle, a transition by a box, an element of $F \cap (S \times T)$ by a directed arc from a circle to a box, and an element of $F \cap (T \times S)$ by a directed arc from a box to a circle.

For the analysis of the PrT net model we have employed the S–invariant method /Genrich 87/. The S–invariants are obtained from the projection of the entries of the incidence

matrix C of the net. The projection along the j−th position of the tuple ($|\ |_j$) introduces a kind of partial cardinal number, by ignoring information at position j in the tuple. As a consequence every solution of $|C|^T\ |i|=0$ whose entries do not contain individual variables determines a family of S−invariants.

B.4. Petri Nets and the Modelling of Time

The concept of time is not explicitly given in the original definition of Petri nets. However, for performance evaluation and scheduling problems of dynamic systems, it is (at present) necessary and useful to introduce time delays associated with transitions and/or places in the net models. Such a Petri net model is known as a (deterministic) *time(d) net* if the timing constraints are deterministically given, or as a *stochastic net* if the delays are probabilistically specified. Only the former one is discussed in this appendix.

B.4.1. Timed Petri Nets

A Timed Petri Net /Ramchandani 74/ is a couple (PN, τ) where PN is a Petri net and τ is a function assigning a non−negative real number to each $t_i \in T$ called the firing time of transition t_i. The firing of a transition is carried out in three steps. In the first step, the transition starts firing as soon as it is enabled, removing the enabling tokens from its input places: $\forall s \in S: M'(s) = M(s) - w(s, t_i)$. In the second step, for a duration given by the firing time associated with the transition, the firing is in progress. In the third step, the firing terminates by producing tokens in the output places of the transitions: $\forall s \in S: M''(s) = M'(s) + w(t_i, s)$. $M''(s)$ is the marking of place s obtained after the firing of t_i. When a transition is enabled a firing is initiated unless the firing of another transition disables the first one.

B.4.2. Time Petri Nets

A Time Petri Net /Merlin 76/ is a couple (PN, θ) where PN is a Petri net and θ is a function assigning a closed interval $[d_{min}, d_{max}]$ to each transition $t_i \in T$, called the static firing

interval of transition t_i, which is interpreted as the interval of time in which the transition must fire. The first value gives the minimal time the transition, after being enabled, must wait before it can fire. The second value gives the maximal time before which the transition must fire if it is still enabled. Thus, for a transition to fire between d_{min} and d_{max}, it must stay enabled between d_{min} and d_{max}, and if the transition has not been fired before d_{max} and if it is still enabled then it fires at d_{max}. The firing of a transition is instantaneous as in Petri nets. No restrictions to the lower and upper bound of the interval is implied, except that d_{min} must be greater or equal than d_{max}.

B.5. High–Level Nets and the Modelling of Time

As far as the modelling of time in Petri nets is concerned, the comparison of using Time(d) Petri nets or high–level nets was already thoroughly discussed elsewhere /Ghezzi 91/. The basic difference between the two types of nets is related to the fact that in higher level nets tokens may have an identity. The property of tokens having an unique identity allows timestamps to be associated with tokens, representing the time point at which a token has been produced. When using timestamps the concern is not just with the ordering of events (firing of transitions) in time, but also with the explicit time reference in which the event has occurred. However, timestamps can only be employed if the "flow of time" is correctly modelled by the net.

B.5.1. Time ER Nets

ER nets /Ghezzi 91/ are high–level Petri nets where tokens are environments, i.e. functions associating values to variables. Furthermore, an action is associated with each transition, describing which input tokens can participate in a firing and which possible tokens are produced by the firing.

Time is introduced in ER nets by adding to each environment a variable *chronos*, whose value is of numerical type, representing the timestamp of the token, in other words, the timestamp represents the time point at which the token was produced. The intuitive concept

of time is captured by three axioms which can be satisfied by constraining the relationships associated with the transitions.

Axiom 1: Local monotonocity. For any firing, the value of chronos in the environments produced by the firing cannot be less than the value of chronos in any environment removed by the firing.

Axiom 2: Constraints on timestamps. For any firing, the value of the chronos in the environments produced by the firing are the same, called the time of the firing.

Axiom 3: Firing sequence monotonicity. For any firing sequence, the times of the firings should be monotonically nondecreasing with respect to their occurrence in firing sequence.

It is shown that, the three axioms are mutually independent, and once Axiom 1 and 2 are satisfied by an ER net, every firing sequence is equivalent to one that satisfies Axiom 3, which allows all observable behaviours of the net to match the intuitive notion of time.

Time ER net (abbreviated TER net) is defined as an ER net where all environments contain a variable chronos and where Axioms 1 and 2 are satisfied.

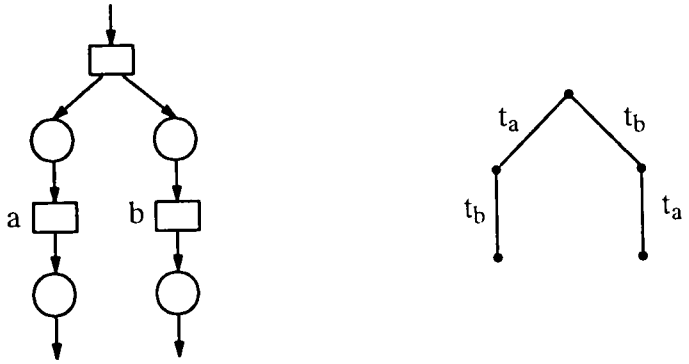
B.5.2. Time PrT Nets

When the flow of time is being modelled in nets, for a transition to fire it is not sufficient for the transition to be enabled; the timing relationships associated with the transition, which are the timing constraints to be modelled, must also be satisfied. A *timed firing rule* must capture the conditions necessary for a transition to fire, together with the model of the flow of time associated with the net. The flow of time can either be explicitly modelled, imposing timing relationships to the transitions of a net, or implicitly, by associating with the firing of a transition an underling clock.

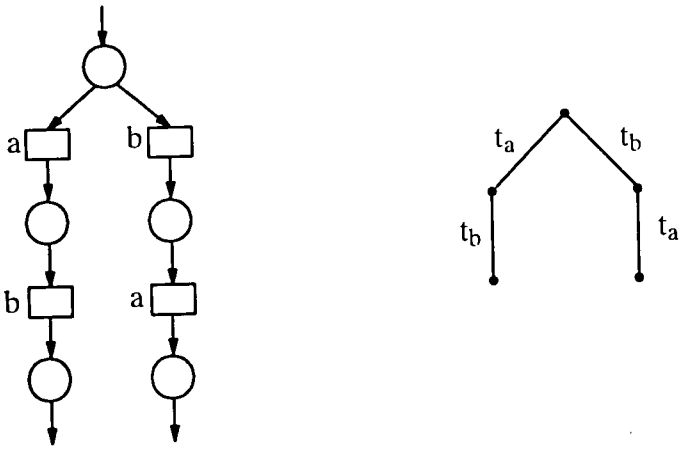
Definition 4.1: In the *explicit* timed firing rule the concept of time is captured by the relationships associated with the transitions.

Definition 4.2: In the *implicit* timed firing rule the firing of a transition is restricted by the properties of the underlying time structure.

The concept associated with the explicit timed firing rule is similar to the one employed in TER nets /Ghezzi 91/. To each tuple we associate a timestamp which represents the time when the tuple was produced, which depends on the relational expression associated to a transition. However, a disadvantage of this approach is that to capture the flow of time the relationships associated with any transition must include the basic conditions of the flow of time, in addition to the timing constraints of the system being modelled. Another disadvantage of the time model of TER nets is that we are not able to distinguish concurrency from non–determinism if we describe the behaviour of a net model as a transition sequence, as it is shown in figure 2.



a. Timestamping concurrent events.



b. Timestamping sequential events after a choice

Figure 2. Discriminating concurrency and non–determinism.

In the implicit timed firing rule the modelling of time in high level nets becomes transparent; we do not need to express explicitly, in the relationships of the transitions, the

basic conditions associated with the flow of time. This is achieved by associating with the net two types of clocks: a *reference clock*, and a *net clock* which has a different granularity for each level of abstraction being considered for the system. We assume that at each tick of the reference clock at most one transition fires, producing a token whose timestamp is given by the net clock. With this scheme for modelling the flow of time, the relationships associated with the transitions will only need to express the timing constraints of the system being modelled. Also, we are able to distinguish concurrency from non–determinism for the case in which the behaviour of a net model is described as a transition sequence: the timestamps of the tokens produced by the firing of two concurrent transitions may in this scheme assume the same value.

The timing constraints to be modelled are expressed as relational expressions which must be satisfied for a transition to become enabled. We assume a Weak Time Semantics (WTS) rather than a Strong Time Semantics (STS), which means that a transition does not have to fire when its maximum firing time has been reached; if it does fire it does so within the time interval specified by the time condition /Ghezzi 91/.

For the verification of a PrT net model which incorporates the implicit timed firing rule, the main analysis method consists of executing specifications, as suggested for TER nets /Ghezzi 91/. However, if we ignore the timestamps of the tokens, and the timing relations associated with the transitions, known techniques for the analysis of PrT nets, such as the S–invariant method /Genrich 87/, may be also employed to check the properties of a PrT net model, such as the liveness property.

Appendix C

Fault Tree Analysis (FTA)

Fault tree analysis (FTA) is a "top down" analysis which delineates the logical relationships between component malfunctions and the designated undesired event. During system analysis, there are often a number of undesired events identified by techniques such as Preliminary Hazard Analysis, which are considered by separate fault trees.

In a fault tree the undesired event appears as the top event, and this is linked to more basic fault events by event statements and logic events. The mains advantage of FTA comparing with other techniques such as FMEA is that the analysis is restricted only to the identification of the system elements and events that lead to one particular undesired failure or accident. In the following, we describe the basic building blocks of fault trees /Henley 92, Veseley 81/.

There are two types of building blocks: *gate symbols* and *event symbols*. Gate symbols connect events according to their causal relations. A gate may have one or more input events but only one output event. The OR gate is used to show that the output event occurs only if one or more of the input events occur. Causality never passes through a OR gate, or alternatively, the inputs to an OR gate are never the causes to an output event. Inputs to an OR gate are considered a refinement of the output. On the other hand, the AND gate specifies a causal relationship between inputs and outputs, that is the input events collectively represent the cause of the output event. The more important event gates are the rectangle, diamond and circle. A *rectangle* indicates an event which must be analysed further. A *diamond* indicates a specific event that is not further developed either because it is of insufficient consequence or because information is unavailable. A *circle* indicates a basic fault requiring no further development.

Both the qualitative and quantitative evaluation of a fault tree can be performed in straight forward manner by using *minimal cut sets*. Minimal cut sets define the "failure modes" of

the top event and are usually obtained when a fault tree is evaluated. A minimal cut set is the smallest combination of component failures which, if all occur, will cause the top event to occur, or alternatively, is a combination of primary events sufficient for the top event to occur. The minimal cut set (M_i) expression for the top event (T) is the following:

$$T = M_1 + ... + M_i + ... + M_k.$$

Appendix D

Operators for the PEA notation

D.1. Standard Logical Operators

In the following the standard logical operators are defined over the point and interval predicates, in the value and time domains. The operators are defined in terms of the functions of the E/A model.

D.1.1. Value domain – Point predicate

Negation ($\neg$)

$$\forall i \in I^{+}: \neg E(i) \stackrel{def}{=} \forall t \in T: \neg E_V(t,i)$$

Conjunction ($\wedge$)

$$\forall i,j \in I^{+}: E(i) \wedge F(j) \stackrel{def}{=} \exists t \in T: E_V(t,i) \wedge F_V(t,j)$$

D.1.2. Time domain – Point predicate

Negation ($\neg$)

$$\forall t \in T: \forall i \in I^{+}: \neg E(i)@t \stackrel{def}{=} \neg E_T(t,i)$$

Conjunction ($\wedge$)

$$\forall t \in T: \forall i,j \in I^{+}: E(i)@t \wedge F(j)@t \stackrel{def}{=} E_T(t,i) \wedge F_T(t,j)$$

D.1.3. Value domain – Interval predicate

Negation ($\neg$)

$$\forall i \in I^+: \neg A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \stackrel{def}{=} \forall t \in T: \neg A_V(t, i)$$

Conjunction ($\wedge$)

$$\forall i, j \in I^+: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=} \exists t \in T: A_V(t,i) \wedge B_V(t,j).$$

D.1.4. Time domain – Interval predicate

Negation ($\neg$)

$$\forall t_{\uparrow A}, t_{\downarrow A} \in T: \forall i \in I^+: \neg A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \stackrel{def}{=}$$
$$\exists t \in T: (t_{\uparrow A} \le t < t_{\downarrow A} \wedge \neg A_T(t,i)) \vee ((t_{\uparrow A} > t \ge t_{\downarrow A} \wedge A_T(t,i))$$

Conjunction ($\wedge$)

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i, j \in I^+: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$
$$\forall t \in T: (t_{\uparrow A} \le t < t_{\downarrow A} \Leftrightarrow A_T(t,i)) \wedge \forall t \in T: (t_{\uparrow B} \le t < t_{\downarrow B} \Leftrightarrow B_T(t,j)) \wedge$$
$$max(t_{\uparrow A}, t_{\uparrow B}) < min(t_{\downarrow A}, t_{\downarrow B})$$

D.2. Interval Operators

Four additional logical operators are introduced over the interval predicates of the PEA notation.

D.2.1. Value domain

Choice Operator (+)

$$\forall i, j \in I^+: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle + B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$
$$(A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge \neg B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle) \vee$$
$$(\neg A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle)$$

Meet Operator ($\prec$)

$$\forall i,j \in I^+: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \prec B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$
$$A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \wedge \downarrow A(i) \Leftrightarrow \uparrow B(j).$$

Overlap Operator ($\|$)

$$\forall i,j \in I^+: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \| B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$
$$A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \wedge B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \wedge$$
$$\uparrow A(i) \Rightarrow B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \vee \uparrow B(j) \Rightarrow A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle.$$

Disjoint Operator (∇)

$$\forall i,j \in I^+: A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \nabla B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \stackrel{def}{=}$$
$$\neg (A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \| B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle) \wedge$$
$$\neg (A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle \prec B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle) \wedge$$
$$\neg (B(j)\langle\uparrow B, B_{INV}, \downarrow B\rangle \prec A(i)\langle\uparrow A, A_{INV}, \downarrow A\rangle).$$

D.2.2. Time domain

Choice Operator ($+$)

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i,j \in I^+: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle + B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$
$$(A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge \neg B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle) \vee (\neg A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle).$$

Meet Operator ($\prec$)

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i,j \in I^+: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \prec B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$
$$A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \wedge (t_{\downarrow A} = t_{\uparrow B}).$$

Overlap Operator ($\|$)

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i,j \in I^+: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \| B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$
$$(A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \wedge B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle) \wedge ((t_{\uparrow B} \le t_{\uparrow A} < t_{\downarrow B}) \vee (t_{\uparrow A} \le t_{\uparrow B} < t_{\downarrow A})).$$

Disjoint Operator ($-$)

$$\forall t_{\uparrow A}, t_{\downarrow A}, t_{\uparrow B}, t_{\downarrow B} \in T: \forall i, j \in I^{+}: A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \nabla B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \stackrel{def}{=}$$

$$\neg (A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \| B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle) \wedge$$

$$\neg (A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle \prec B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle) \wedge$$

$$\neg (B(j)@\langle t_{\uparrow B}, t_{\downarrow B}\rangle \prec A(i)@\langle t_{\uparrow A}, t_{\downarrow A}\rangle).$$