9 10 10.95 12

A Principled Approach To The Integration Of Human Factors And Systems Engineering For Interactive Control System Design

Christopher William Johnson

Submitted for the degree of Doctor of Philosophy

The University of York

The Human Computer Interaction Group,
The Department of Computer Science.

April 1992

Abstract

This thesis argues that principles provide a framework which helps to ensure that human factors and systems engineering are integrated at all stages of development. The argument ranges from the fundamental problems of control, to the problems of a development architecture, to the problems of designing a particular interface.

Part I of this thesis justifies an attempt to use principles as a means of integrating human factors and systems engineering.

Part II argues that principles provide common objectives for the human factors and systems engineering of interactive control systems. They provide criteria against which to assess the utility of potential solutions for problems that are common to many different interfaces: dynamism; complexity and openness. Interval temporal logic is proposed as an appropriate notation in which to represent these solutions.

Part III argues that principles provide criteria against which to assess the utility of architectures for human factors and systems engineering. In particular, it is argued that a strength of object orientation is that it supports the development of consistent and predictable interfaces. A weakness of this architecture is that display objects can provide inaccurate views of the physical components which they represent. Abstract analyses of design principles and architectures are frequently conducted at a level that is inappropriate for the development of particular interfaces to particular control systems. PRELOG, a tool for the Presentation and REndering of LOGic specifications, has been implemented to avoid this limitation. It is argued that human factors and systems engineers might use prototypes to determine whether particular principles have any relevance for system operators.

Part IV argues that the principled approach advocated in this thesis is methodologically inadequate. It does not support the integration of human factors and systems engineering during all stages of development. Further work is proposed to rectify this limitation.

Contents

List of Figures

Acknowledgements

I would like to thank the members of the Department of Computer Science at the University of York who have encouraged the work that is presented in this thesis. Professor Ian Wand provided the initial impetus to continue my studies. Dr Colin Runciman provided valuable guidance at every stage in this research. Professor Michael Harrison originally proposed the application of interval temporal logic to support the design of interactive systems. Without his friendship and advice this thesis would not have been written.

The research described in this thesis has benefited from the help of researchers in a number of different disciplines. Dr Andrew Monk and Dr John McCarthy from the Department of Psychology, University of York provided useful correctives to preconceptions about their discipline. Dr Henning Anderson of the Cognitive Systems Group, National Research Laboratories, Risø, Denmark arranged for my participation in an experimental analysis of direct perception displays and provided valuable insights into the potential of interval temporal logic as a tool for cognitive modelling. Professor Jim Crowley from the Laboratoire d' Informatique Fondamentale et d' Intelligence Artificielle and Professor Joëlle Coutaz from the Laboratoire de Génie Informatique, Grenoble, France arranged for my participation in the Mithra robotics project.

The work presented in this thesis has benefited from the support of a number of commercial companies. British Telecom provided funding. British Aerospace provided information about commercial and military avionics. The McGibbon Consultancy and Agie Industrial Electronics supplied information about interactive machine tools.

This work has been supported by a CASE award funded by British Telecom, by SERC grant 88503497 and by the 1990 Gibbs-Plessey travel award.

Declaration

Professor Michael Harrison first suggested that interval temporal logic might be applied to support the design of interactive systems. With this exception, the work presented in this thesis is entirely that of the author.

Some of the material presented in Chapters 1 and 4 is based upon a joint paper presented with Victoria Miles, Dr John McCarthy and Professor Michael Harrison at HCI'91 [210]. Chapter 4 is also based upon a paper presented by the author at HCI'91 [162]. The implementation of the PRELOG prototyping tool, described in Chapter 7, was presented in a joint paper with Professor Michael Harrison at EUROGRAPHICS'90 [163]. The incorporation of an interval temporal logic interpreter into this system was discussed in a paper, again with Professor Michael Harrison, at EUROGRAPHICS'91 [164]. We also discuss the application of PRELOG to support the development of interactive control systems in a forthcoming article to appear in the International Journal Of Man-Machine Studies [165]. This thesis only exploits those parts of collaborative papers that are directly attributable to the author.

Part I

Introduction

Introduction To Part I

This part of the thesis argues that human factors and systems engineering must be integrated in order to improve the usability of interactive control systems.

Chapter 1 argues that the task-artifact cycle does not support the integration of human factors and systems engineering. This approach assesses an interface late in the development cycle when the costs of making changes are likely to be prohibitive. Alternatively, designers might use properties of previous systems as heuristics to guide subsequent development. It is argued that these heuristics, or principles, provide criteria against which to assess the usability of existing interfaces. In order to exploit this approach designers must be provided with a precise and concise means of representing techniques that could be used to achieve these principles. It is argued that formal, mathematically based, notations can be used to satisfy this requirement.

Chapter 1

The Justification For Integration

> "The purpose here is to show what has gone wrong in the past and to suggest how similar incidents might be prevented in the future. Unfortunately, the history of the process industries shows that many incidents are repeated after a lapse of a few years. People move and the lessons are forgotten" (Kletz, [179]).

1.1 Introduction

Accidents at Bhopal, Chernobyl, Flixborough, Seveso, Three Mile Island and Windscale have raised questions about the safety and reliability of many production processes. This is illustrated by the quotation that opens this chapter. Public anxiety increasingly focuses upon the role of the operator within control systems. The Commission of the European Community [158], the Japanese Fifth Generation Initiative [316] and United States' Presidential Task Forces [243] have all cited human intervention as a primary factor in the cause and exacerbation of accidents in the process industries. This thesis argues that the role of human error extends from the operation of control systems to include their design. Previous accidents have occurred because systems engineers have paid insufficient attention to the demands posed by operating interactive control systems. This observation is far from novel. Both the Zeebruge [283] and King's Cross [98] enquiries concluded that operators were not the main instigators of the disasters, rather that they inherited defective systems created by poor design and management. Such findings have also been a central inspiration for recent research in the field of human factors. Much of this work has not been accompanied by the provision of tools that would enable designers to carry human factors research beyond the 'laboratory bench' and onto the 'shop floor'. This thesis argues that principles might provide such a tool. Properties of previous interfaces can be used as heuristics to guide the human factors and systems engineering of interactive control systems.

The first question to be answered in any attempt to enhance the usability of interactive control systems is: what are they? For the purposes of this thesis, 'interactive control systems' are defined to be systems in which operators intervene to regulate the behaviour of application processes. They include chemical and en-

ergy production control systems. They include avionics, as well as maritime and rail transportation control systems. They include dependent and semi-autonomous robotics control systems. Within such systems there can be applications that do not directly control process behaviour, such as databases of production information. These are included in our definition because they are essential components of the control system. This thesis focusses upon the interaction between a system and its user, the interested reader is directed to [64, 29] for background information on the more general causes of system failure.

The second question to be answered is: precisely what do we mean by enhancing usability? The Concise Oxford Dictionary defines 'usability' as "the ability to be used" [12]. This hardly clarifies matters. In the context of this thesis, usability refers to the users' ability to effectively operate control systems and, through them, the processes of underlying applications. Human factors engineering has been concerned to improve usability through the application of physiological and social ergonomics, perceptual and cognitive psychology [261]. Systems engineering has been concerned to improve usability through automation and sensing techniques, decision support tools, dialogue design and system modelling [49]. This thesis intends to show how designers, who can be human factors or system engineers, might avoid the limitations and exploit the benefits of both disciplines. In order to do this we must first identify the strengths and weaknesses of human factors and systems engineering.

1.2 An Assessment Of Systems Engineering

This thesis defines 'systems engineering' to be the application of the physical sciences and of computer science to support systems development.

1.2.1 Automation And Sensing

Advances in computer science and software engineering have provided designers with means of monitoring and sensing effects that were previously only predicted by physical sciences, such as chemistry and physics. For example, the development of high-speed processors and real-time programming languages has supported the implementation of distributed control systems that can simultaneously sense and respond to changes in many different parts of a production process [288]. These advances have also enabled designers to exploit automation, the application of machines to reduce manual and mental labour, as a means of supporting the operation of interactive control systems. For instance, Covey, Mascetti, Roessler and Bowles describe automated avionics which can ensure that aircraft follow fuel-efficient ascents and descents [78]. Sensors within the airframe and engines enable these systems to achieve a twelve percent fuel saving for commercial aircraft. Pilots need not assume the mental burden of calculating these flight patterns.

The application of automation and sensing techniques has not enjoyed universal success. For example, the Habsheim crash occurred when the pilot of a fly-by-wire Airbus Industries' A320 judged that they had sufficient power to pull their aircraft out of a descent. Fly-by-wire systems are controlled by computer programs that receive sensor readings as input. They output the signals that are necessary in order to control the hydraulics which move the aircraft's control surfaces. The crew of the A320 later complained that the engines had been slow to respond once the

pilot had opened the throttle. This can be explained by the fact that avionics have been designed to reduce engine wear by gradually applying power irrespective of immediate commands from the crew. This incident illustrates the point that the introduction of advanced systems engineering can create human factors problems. Similar causes have been cited for the Air India A320 crash [244]. The consequences of calculating an inefficient ascent are insignificant in comparison to the consequences of a pilot misunderstanding the performance parameters of their aircraft. Following the Habsheim and Air India crashes, Airbus Industries' A320s now provide audible alarms to warn pilots if they have insufficient flying energy to pull out of slow, nose high, approaches [219]. In order to reduce the likelihood of such accidents occuring again it is vital that human factors engineering should play some part in the development of automated control systems.

1.2.2 Decision Support

Decision support tools, or expert systems, provide operators with advice about optimal courses of interaction [198]. They exploit advances in computer science to infer as much information as possible about application processes. For instance, Chan has used neural networks to diagnose the causes of supply failures in electricity distribution networks [62]. Expert systems are capable of monitoring and assessing amounts of information that would stretch the cognitive and perceptual resources of human operators. No user can monitor the power loading on every one of the thousands of nodes in power distribution networks. These loadings can, however, be fed as input into Chan's system. It can be taught to recognise loading patterns that will lead to supply problems. It can direct operator intervention towards nodes that are likely to have caused system failures.

A number of problems limit the utility of decision support tools. Their advice is frequently incorrect. Van Daele found that over sixty-seven percent of automated diagnosis and planning systems in casting control applications made "suboptimal decisions" [82]. Bainbridge describes how one group of users switched off their advisory system rather than be distracted by its incorrect predictions [22]. A more fundamental problem for systems designers is that decision support tools provide low bandwidth communication [13]. By this we mean that advice is based upon a mass of information which is accessible to the system but not to its operator. For example, avionics sample a range of sensors many times a second in order to determine aircraft attitude [293]. Pilots cannot sample at this rate; they must trust the information which is displayed to them. This increases the potential for operator error because pilots must act upon system recommendations without fully understanding the context in which advice is offered. Taylor and Selcon argue that designers can reduce the frequency of such errors by displaying justifications for the decisions recommended by expert systems [300]. In other words, designers must not only consider systems engineering but also the human factors of control when developing decision support tools.

1.2.3 Dialogue Design

Systems engineering has developed a number of interactive dialogues that enable users to rapidly issue commands and view process information [97]. These dia-

logues often apply computer science techniques, such as problem decomposition, to support the presentation of application processes. For instance, hierarchical presentation systems provide facilities for focusing process displays so that operators can access information about individual components. They can select the level of detail that is appropriate for particular control tasks. These dialogue designs have been exploited in a number of commercial products, including the current range of Agiecraft machine tools [4] and Transmitton's Flexible Energy Management System [208]. Agiecraft claim that these "dialogue capabilities substantially cut idle times" [4].

Improving dialogue design has been a primary concern of the European Strategic Programme for Research and development in Information Technology (ESPRIT) GRAphical DIalogue environmENT (GRADIENT) project [151]. In spite of such initiatives, the design of display structures and dialogue patterns remains a non-trivial problem. For instance, "cognitive lockup" has been observed when operators can decompose control system displays [144]. This occurs when users become preoccupied with one level of detail and fail to observe errors which manifest themselves at another level. Dialogue designs, such as those proposed by Transmitton and Agiecraft, again illustrate the need to consider human factors requirements when applying advanced systems engineering techniques.

1.2.4 System Modelling

The physical sciences, such as chemistry and astronomy, have developed a range of modelling techniques that can be used to describe the behaviour of complex natural objects [154]. Designers have exploited these techniques to support the engineering of interactive control systems. For instance, a series of accidents involving commercial radiation therapy systems inspired a Washington hospital to analyse the design of their clinical cyclotron [167]. The operator of this system controlled a nine hundred ampere electromagnet and a thirty ton rotating gantry. The cyclotron control system handled over one thousand input signals. Jacky describes how mathematical modelling techniques considerably simplified the task of re-designing this system which included the analysis of over sixty thousand lines of Fortran source code [160].

There are a number of problems which limit the utility of system modelling techniques for the development of interactive control applications. In particular, they fail to capture what De Montmollin and De Keyser call the "knowledge of utilisation" [211]. In other words, system modelling techniques frequently fail to capture the demands that an interface places upon its operators. Instead, they capture a designer's "knowledge of functioning" that is chiefly concerned with system development rather than operation. Human factors engineering provides techniques that can be used to avoid this bias of systems engineering.

1.3 An Assessment Of Human Factors Engineering

This thesis defines 'human factors engineering' to be the application of physiology, the social sciences and psychology to support the development of interactive systems.

1.3.1 Physiological Ergonomics

Physiology is "the science of the functions of living organisms" [12]. Ergonomics is "the study of the efficiency of persons in their working environment" [12]. Physiological ergonomics applies the experimental techniques of physiology to assess the impact that an operator's physique can have upon their performance at work. For instance, Junge and Giacomo have analysed an operator's posture in order to improve work-station layout in the American Space Shuttle [169]. Galer and Yap conducted a range of experiments to resolve the physiological problems of keyboard data-entry in the cramped environment of intensive care units [113]. Plath and Kolesnik exploited a similar approach when developing thumb-wheel switches to enter navigational information in aircraft cockpits [237].

Functional requirements help to determine the layout of many working environments [113]. The optimal position for a patient monitoring unit in an intensive care ward need not be appropriate for departments that offer a lower level of clinical automation. An optimal position for a thumb-wheel switch in one cockpit is often inappropriate for aircraft with different avionics. The introduction of new computer systems can change workstation layout and this, in turn, can affect an operator's posture [169]. The findings of physiological ergonomics can, therefore, only be applied if human factors engineers are aware of the constraints imposed by systems engineering.

1.3.2 Social Ergonomics

Social ergonomics is concerned to assess the impact that interaction between system operators can have upon their working performance. Designers can exploit techniques from social sciences, such as linguistics [107], sociology [114] and management studies [289], to support this task. For instance, the United States' Nuclear Regulatory Commission are currently evaluating the Management Analysis Concept developed by Haber, Metlay and Crouch [127]. This methodology is intended to detect instances in which safety-critical decisions are not passed through a chain of command but are usurped by committees.

Brouwers and Pots observe that "policies regarding social organisation and social policies (have) played an insignificant role in the design process" [45]. Competitive advantage is seen purely in terms of the improvements in quality and productivity that can be derived from systems engineering. Organisational issues are typically only considered after new technology has been introduced. Clegg and Wall argue that this lack of integration jeopardises the success of innovative systems engineering [73]. Operators will obstruct the use of new technology if they are unsure of their role after automation is introduced. New technology frequently requires increased cooperation and communication between fewer users and this, in turn, can increase the social pressures on those workers. In order to asses the impact of such factors upon a potential implementation there must be some means of integrating the findings of human factors and systems engineering.

1.3.3 Perceptual Psychology

Perceptual psychology is concerned to identify the ways in which humans exploit information from their senses. Designers have exploited findings about human percep-

tion in order to guide the development of interactive control systems. For instance, Boeing applied perceptual psychology to reduce the number of accidents involving their 727 aircraft. One of their engineers observed that many of these accidents occurred during night-time landings over 'dark hole' approaches; above water or unilluminated terrain. They used a range of experiments to establish a link between the sensory information available to pilots under such circumstances and incorrect estimates of aircraft altitude. The training of all commercial pilots now involves warnings about false altitude estimates during such 'dark hole' approaches [194].

A number of problems restrict the utility of the findings of perceptual psychology for the development of interactive control systems. Experimental results cannot always be used to directly guide systems engineering. For instance, Braune and Wickens distinguish between functional aging, the deterioration of perceptual performance above that expected for an operator's age group, and chronological aging, the expected deterioration of performance with age [41]. Designers could use these findings to help them select a potential workforce; candidates with a high level of functional aging might be rejected. These findings could also be used to inform dialogue and display design; operators suffering from a high level of chronological or functional aging can be supported by the presentation of additional sources of information. Braune and Wickens stress the difficulty of measuring the deterioration of perceptual performance for particular operators. It is, therefore, difficult for designers to develop displays that account for aging processes. This illustrates the problems that can frustrate the application of perceptual psychology to inform system engineering.

1.3.4 Cognitive Psychology

Cognitive psychology is concerned to identify the ways in which humans store and manipulate information. Singleton [287], Reason [251] and Woods [329] have applied research in this area to explain the causes of operator error. Rasmussen has built upon the findings of cognitive psychology in order to develop a skill, rule and knowledge based classification of operator performance [246]. Skill based behaviour involves the unconscious control of a process rather than the conscious operation of an application through a control system. Rule based behaviour is characterised by rules for action which are fired by the observation of system attributes. Knowledge based behaviour involves the inference that must be used in order to determine the state of a process from individual monitor readings. Designers can apply cognitive psychology to determine whether operators are likely to learn, retrieve and process the information that is necessary in order to control application processes. For instance, Pew, Miller and Feehrer have used Rasmussen's skill, rule and knowledge framework to analyse problems in the presentation of the Westinghouse Electric Corporation's nuclear power production control systems [236]. Poorly designed displays inhibited skill based behaviour and were characterised by knowledge based interaction.

A number of questions must be answered before cognitive psychology provides designers with adequate tools for control system development. What are the differences between the cognitive processes required during routine work and during rare events? What are the differences between experts and novices? Rasmussen concludes that "very little has, so far, been done in order to identify the overall

cognitive competence required for the entire job of a person" [247]. The perceived weaknesses of physiological and social ergonomics, perceptual and cognitive psychology have led to an interest in what Norman calls cognitive engineering [227]. The integration of research within these different domains is currently being addressed by the United Kingdom's Science and Engineering Research Council's Task Oriented Modelling (TOMS) project and the ESPRIT MOdels of Human Actions in Work Contexts (MOHAWC) project. Cognitive engineering seeks to integrate the different strands of human factors engineering [152]. Rasmussen terms this the "new profession" of design [247]. The following chapters take such integration one step further by introducing systems engineering into the design techniques proposed by human factors research.

1.4 Integration Through The Task-Artifact Cycle

The iterative refinement advocated by Carroll and Kellog's task-artifact cycle provides designers with a framework for the integration of human factors and systems engineering [57]. This cycle can be decomposed into the three phases shown as boxes in Figure 1.1. Systems engineers implement or modify a control system. The intro-

Figure 1.1: The task-artifact cycle

duction and use of this system creates new tasks which can be identified by human factors engineering. Observations of use can be interpreted to assess the strengths and weaknesses of the new system [56]. This process of interpretation is termed hermeneutics; designers must identify and justify the human factors claims that are embodied within an interactive system. For instance, the Cathode Ray Tube (CRT) monitors of aircraft, such as Boeing's 757 or Airbus Industries' A310, embody the claim that colour displays support the perception of avionics information. Design progresses as the veracity of these claims is confirmed or denied. United States' Air Force and Navy planes, commissioned since 1970, reject the claims embodied in these commercial aircraft by restricting the use of colour displays [321]. Lewis' agenda for research in human computer interaction cites hermeneutics as one of the five most powerful techniques currently available to human factors practitioners [195]. The following sections argue that there are practical limitations to the

application of this approach as a means of integrating human factors and systems engineering.

1.4.1 Hermeneutics And Openness

Many previous studies of human computer interaction have focused upon single users operating single applications [210]. These systems can be described as closed because they exclude input from other operators and application processes. Designers and users can assume that their systems will be immune from external interference. In contrast, control systems are usually open to simultaneous input from more than one source. This openness impairs the successful application of hermeneutics because designers frequently make inappropriate claims about the interaction between multiple users and processes. For instance, the Flixborough Nypro disaster occurred because nitrate discharges contaminated coolant supplied to a process by another production-line [179]. The designers of the Nypro system claimed that the operators of the supply source monitored their discharges. Unfortunately, this interpretation was unjustified and the final artifact was a flawed control system.

Hermeneutics provides inadequate support for the integration of human factors and systems engineering because it can be extremely difficult to interpret the usability of systems that are simultaneously operated by a number of users. For example, if an aircraft engine is damaged then the crew must rotate its propeller blades in line with the direction of flight [320]. Accident analysis has revealed that confusion amongst the crew will often result in the wrong set of blades being rotated. As a result auto-feathering systems have been developed to automatically perform this rotation. Operating procedures have been devised to ensure that the crew agree before securing the engine after its blades have been rotated. Auto-feathering systems, therefore, embody the claim that pilots will follow operating procedures. Such assumptions have proven to be unjustified. The claims which guided the development of many auto-feathering systems did not adequately account for the group dynamics of multiple users operating open control systems. In 1979 a Swift Aire Lines Nord 262 crashed because the automated equipment correctly auto-feathered the right propeller while the crew agreed to shut down the left [320].

1.4.2 Hermeneutics And Complexity

Designers must selectively present relevant and timely information from the large amounts of data that are available to a control system. Hermeneutics provides inadequate support for control system development because it can be extremely difficult to interpret the success or failure of complex artifacts during each iteration of the design cycle, illustrated in Figure 1.1. The crash of a Lauda Air Boeing 767-300 provides an example of this. According to the Austrian Transport Ministry this accident occurred when one of two computer-controlled PW-4060 engines went into reverse in mid-flight. The pilots tried to solve this “totally unforeseen problem with the aid of the flight manual but were unable to do so” [309]. The complexity of modern avionics prevented designers from identifying the three independent system failures required in order to reverse the thrust. They failed to accurately assess the usability of their interface because they did not identify the claims that were embodied in their artifact. In other words, they failed to recognise the implicit

assumption that operators must resolve such unexpected problems with minimal support from their avionics.

1.4.3 Hermeneutics And Dynamism

Carroll and Kellog argue that the task-artifact cycle encourages designers to treat "situations, users and artifacts as unique instances" [57]. This complicates the design of systems that must operate in situations which change over time. For instance, a Liquid Crystal Display (LCD) engine monitoring system was initially introduced into the Boeing 737-400 series. The interface to this application is very different to those previously available on Boeing 747-200s (electromechanical dials or trend monitors) and 747-400s (CRT displays) [191]. The introduction of LCD engine monitors altered the situation of use for avionics in the B-737-400 series. In such circumstances, Carroll and Kellog argue that the cockpit must be reinterpreted in order to assess the claims which it now makes about the human factors of control. This approach can be justified for many safety-critical applications. The expense of re-evaluation after such alterations is a considerable disincentive to the commercial exploitation of the task-artifact cycle. There are also ethical and regulatory problems with this approach. For instance, is it acceptable to test new components in situations that can lead to loss of life? If artifacts must be evaluated in their working environment then hermeneutics consigns designers to post-hoc analysis. It traps them within what can be termed the task-artifact-accident cycle. The United Kingdom's Department of Transport's Air Accidents Investigation Branch cited LCD engine monitoring systems as a possible cause of the pilot error that contributed to the Kegworth disaster [191]. Many airlines reverted to the use of electromechanical dials and CRT displays following the Kegworth accident. This was clearly a costly lesson in interface design. The disaster demonstrates that the task-artifact cycle is unacceptable when design iterations sacrifice the safety of a system and its operators. There is a pressing need for development techniques that can guide human factors and systems engineering prior to full implementation.

1.5 Integration Through Principled Design

A 'principle' is defined to be "a fundamental truth or law as the basis of reasoning or action" [12]. Thimbleby argues that principles can be exploited to support the development of interactive systems [301]. They are to be distinguished from guidelines that inform particular decisions during the design of particular interfaces [216]. For instance, experimental evidence suggests that unanticipated cursor positioning is faster using a mouse. Thimbleby argues that this can only be a guideline because other evidence suggests that key selection is faster than mouse selection on a known menu [302]. Before investigating whether principles can support the integration of human factors and systems engineering it is important to ask: where do principles come from? There is no simple answer to this question and it is a research issue in its own right [304]. Principles can be derived from widely accepted design objectives. These principles are the "fundamental" truths. For instance, the 'What You See Is What You Get' principle is a common, if rarely achieved, objective for interface development [304]. Principles can also be derived from objectives that have proven useful in one domain and which could have a wider application. These principles

are laws that provide "the basis of ... action". For instance, the 'What You See Is What I See' principle has been applied to multi-author text editors but it might also be used to guide the development of multi-user control systems [290]. Principles can be derived from risky objectives that have paid-off in previous systems. For instance, there is little agreement about the value of consistency in the presentation of interactive systems [125]. Consistency does, however, provide a useful principle if designers can reason about the costs and benefits that are to be gained from it. These principles are laws that provide "the basis of reasoning".

Principles can be used as criteria against which to assess the strengths and weaknesses of existing designs. For instance, the principle of accountability is embodied within the Notification Of Installations Handling Hazardous Substances Regulations [140] and the Control of Major Industrial Accident Hazard Regulations [141]. Plant management is accountable to the Health and Safety Executive (HSE), the HSE is accountable to Parliament. A design can be criticised if this chain of accountability is broken. Principles can also guide development prior to implementation. For example, the HSE advocates the principle of localised risk. Potential accidents in a production process should not threaten the safety of large population centres. It follows that certain forms of chemical reactors must not be constructed up-wind of large towns or cities. Companies need not go to the costs of designing and building such artifacts to know that they will be judged as unsafe!

1.5.1 Principles And Openness

Principles can support the development of control systems that are open to input from multiple users and application processes. For instance, the United States' Electric Power Research Institute's human factors review of nuclear power plant control rooms found meters that could not be read from their control panels which were located thirty feet away [280]. Critical displays were presented on the reverse of panels whose primary displays were devoted to non-critical information. They discovered two monitor scales which were identical except that the left hand device was read at ten times the value of the right. These design problems arose because control panels, displays and meters were frequently sub-contracted to a number of different companies. Each of these manufacturers adopted different standards for the presentation of application processes. Principles provide human factors and systems engineers with a means of tackling this lack of standardisation. For instance, designers might adopt the principle of allocating display resources in proportion to the importance of the data to be presented. This principle could be imposed as a requirement that must be accepted by any company tendering for the instrumentation of a production process.

1.5.2 Principles And Complexity

Designers must determine how best to allocate finite development resources to support the design of interactive control systems. Principles can be used to guide this task. For instance, the MANpower and PeRsonnel INTegration (MANPRINT) project has developed a set of objectives which must be used to evaluate any interface procured by the United States' Army [202]. These objectives can be viewed as design principles; they have been derived from analyses of previous successes and

failures in military procurements. They can help to focus the application of development resources. For instance, MANPRINT recommends that all manufacturers conduct "front-end analyses" to ensure that display resources are allocated in proportion to the importance of the data which they present [137]. These objectives have been selected to encourage the integration of human factors and systems engineering. Contractors are not only required to test the performance of automated systems, they are also required to evaluate the cognitive and perceptual workloads that systems impose on their operators. American defence suppliers, such as Horizons Technology, claim that principles direct the development of complex control systems and avoid the "problems of piecemeal and uncoordinated action" [137].

1.5.3 Principles And Dynamism

As a consequence of the Presidential investigation into the Three Mile Island accident, the United States' Nuclear Regulatory Commission (NRC) adopted the principle of minimal intervention [236]. Whenever possible operators should not be required to intervene in order to preserve the safety of their system. This principle was embodied in the terms and conditions of operation that were imposed upon plant management. For instance, it was stipulated that automatic systems must run high-head charging pumps, part of the emergency cooling equipment, for at least twenty minutes after reactor scrams. A scram occurs when neutron absorbers are inserted into a reactor in order to slow the reaction process. It was assumed that such legislation would ensure minimal operator intervention. The 1979 North Anna incident illustrates some of the problems which can occur when principles are embodied in the regulations that govern the operation of dynamic systems. Changes in the generation process employed by the North Anna reactor led to dangerous temperature profiles following a scram. The operators were faced with a difficult choice. If they obeyed NRC regulations then the safety of the plant would be threatened; they would no longer be able to predict its behaviour. If they disobeyed the regulations then the plant could be saved but they would break the NRC conditions of operation. Fortunately, plant management chose to disregard the principle of minimal intervention. A pump was taken off the coolant circuit and the emergency was resolved. Duncan observes that this incident underlines the dangers of trying "to prescribe regulations, procedures or algorithms, especially when these prescriptions are backed by legal sanctions" [91]. This emphasises the point that designers should not impose principles as inflexibly as the rules proposed by the NRC. Principles should not be viewed as axioms that must hold throughout interaction; they provide constraints that should only be violated if the consequences are understood and accepted by designers and operators.

1.6 A Principle, A Notation And A Tool

The North Anna incident, described in the previous section, illustrates two important requirements that must be satisfied by techniques which support principled design. Firstly, they should help designers to identify potential conflicts between principles. Minimal intervention might threaten other objectives, such as safety or efficiency. In order to resolve such conflicts, designers must be able to identify the costs and benefits of particular principles. Secondly, development techniques must

help designers to ensure that principles have some relevance for system operators. The NRC regulation was ignored because reactor control staff viewed the ability to predict the consequences of their actions as more important to the operation of their system. In other words they chose to adopt their own principle of predictability rather than the regulatory authorities' principle of minimal intervention. This thesis argues that formal notations and prototyping tools provide means of satisfying these requirements for principled design techniques.

1.6.1 The Principle: Predictability

Predictability provides an exemplar with which to assess the utility of formal notations and prototyping tools. This principle requires that the effect of any operation which can be invoked from an interface can be deduced unambiguously from a knowledge of its function combined with the data displayed at the time of invocation [87]. Predictability has been chosen because human factors research has identified it as an important property of many interactive control systems. De Keyser argues that anticipation is vital for decision making in complex environments [173]. Umbers suggests that a users' control strategy is determined by the predicted effects of their actions [310]. Bainbridge argues that operators rely upon predictions about the impact of their commands when responding to unexpected or novel situations [21]. A further justification for adopting this principle is that human factors engineers must recruit systems engineering in order to support predictable interaction. For example, pigment production plants frequently rely upon varying air pressures to regulate magnetic flow meters [47]. If the air supply fails then production is stopped. In consequence, operators cannot predict that their commands will be successful unless they can guarantee a correct level of air pressure. Predictability has been preserved in these applications by using the automation of systems engineering to detect and resolve air supply failures [36].

1.6.2 The Notation: Interval Temporal Logic

Human factors and systems engineering might use everyday language to represent design principles. A number of problems limit the utility of this approach. Brooks notes that natural language is "not a precision instrument" [44]. Its semantics are so rich and its linguistic structures are so varied that it is difficult for designers to define principles unambiguously. For instance, it is unclear what is meant by "data displayed" in the previous definition of predictability. Does this refer only to control system displays or does it also include manuals and paper documentation? Does this refer to all the information displayed or just to that portion which can be sampled by the finite perceptual resources of system operators at a particular point during interaction? These problems can be resolved by elaborating the definition. Predictability requires that the effect of any operation which can be invoked from an interface can be deduced unambiguously from a knowledge of its function combined with the data that a user can perceive from the display, excluding paper documentation, which is presented to them at the time of invocation. This does not avoid ambiguity. What is meant by "the effect"? What is meant by "any operation"? These ambiguities could, of course, be resolved by further elaboration. Designers must then determine whether these definitions require additional clarifi-

cation. Further problems are created because principles can be expressed in a large number of syntactic forms. For instance, one design team could draft their principles using brief rules of thumb that are similar to our definitions of predictability. Others might require that principles are described using detailed and numerous case studies. These differences can make it difficult to ensure that development teams abide by the same design principles [150]. Human factors and systems engineers could overcome these problems by exploiting languages that offer a more restricted syntax and semantics [77].

Production Systems

Production systems constrain the syntax that designers can use to represent the requirements of principled design. Chapter 2 will describe their application in more detail. For now it is sufficient to realise that these systems consist of rules which specify that **IF** an antecedent is true **THEN** a consequent holds:

$$\textbf{CONDITION}(< \textbf{rule\_name} >, \textbf{IF} < \textbf{antecedent} >, \textbf{THEN} < \textbf{consequent} >)$$

The angled brackets ($<$ and $>$) denote terms that must be instantiated by designers in order to specify a particular production. The following rule illustrates how they can be used to represent systems engineering requirements; the **SYSTEM** must handle input to start production. It also represents a human factors requirement because the operator must issue **INPUT** to turn the system **on**:

$$\begin{aligned}&\textbf{CONDITION}(\textbf{effect\_rule}, \textbf{IF FACT}(\textbf{INPUT}, \textbf{user}, \textbf{on})\textbf{THEN}\\ &\quad\textbf{ADD FACT}(\textbf{SYSTEM}, \textbf{on}, \textbf{on\_display}))\end{aligned}$$

Production systems are not ideally suited to the design of interactive control systems. Many principles cannot be guaranteed throughout interaction; it is frequently impossible to achieve the predicted effect of operator input if components fail. Such changes can be represented by adding and removing production rules from a database of design constraints. This imposes considerable burdens upon development resources. Designers must not only select and represent relevant principles but they must also consider the way in which those representations will be maintained within a database of rules.

Z

Human factors and systems engineers might use the Z specification language to represent an interactive system at a high level of abstraction [296]. For instance, they could use Z to represent the **PiE** model [88]. This describes an interactive system in terms of a closed world; the behaviour of the system is completely determined by previous operator input. Section 4.1.2 will argue that such assumptions are inappropriate for control systems that are open to interaction with multiple users and application processes. A set, **Programs**, can contain all the possible command sequences that an operator could enter into a system. The elements of another set can describe the **Effects** of **Programs** on a system:

$$[\textbf{Programs}, \textbf{Effects}]$$

Schemas are syntactic units which are used to declare, type and relate variables. They can be used to specify predictability requirements in terms of properties that hold between elements of sets. In the following schema, $\mathbf{p}_2\ \mathbf{p}$ denotes the input sequence $\mathbf{p}_2$ followed by sequence $\mathbf{p}$. It describes a predictability requirement because the effects of command sequences, $\mathbf{p}$, are dependent only upon the effects achieved by previous sequences, $\mathbf{p}_1$ and $\mathbf{p}_2$. In other words, if operators can determine the effects of previous interaction then they might predict the effects of future commands. This can be viewed as a requirement to be satisfied by systems engineering; an implementation of the interpretation function $\mathbf{i}$ must satisfy the constraints described by the schema. It can also be viewed as a human factors requirement; operators must be able to determine the effects of previous interaction:

PredictableSystem

$$\mathbf{i} : \mathbf{Programs} \rightarrow \mathbf{Effects}$$

$$\forall \mathbf{p}_1, \mathbf{p}_2 : \mathbf{Programs}$$
$$\bullet\ \ \mathbf{i}(\mathbf{p}_1) = \mathbf{i}(\mathbf{p}_2) \Rightarrow$$
$$\forall \mathbf{p} : \mathbf{Programs}$$
$$\mathbf{i}(\mathbf{p}_1\ \mathbf{p}) = \mathbf{i}(\mathbf{p}_2\ \mathbf{p})$$

Z cannot easily be applied to describe interaction with the multiple processes of open control systems. Designers must explicitly construct a process algebra on top of the schema notation and its associated calculus [297]. In other words, it does not possess a language that is specifically intended to represent and reason about the problems of concurrent interaction. In order to avoid this limitation Abowd [3] has integrated Hoare's Communicating Sequential Processes (CSP) notation [148] into the Z language.

CSP

Designers might use CSP to describe interaction between a user and their system. For instance, an **Operator** provides input, then reads the display then continues to behave like an **Operator** or ceases to use the system:

$$\mathbf{Operator} =$$
$$(\mathbf{c}!\mathbf{input} \rightarrow (\mathbf{d}?\mathbf{read} \rightarrow (\mathbf{Operator} \sqcap \mathbf{Skip})))$$

A **System** receives input, provides output and continues to behave like a **System** or non-deterministically terminates:

$$\mathbf{System} =$$
$$(\mathbf{c}?\mathbf{input} \rightarrow (\mathbf{d}!\mathbf{output} \rightarrow (\mathbf{System} \sqcap \mathbf{Skip})))$$

The **Operator** and **System** interact concurrently and this is denoted as follows:

$$\mathbf{Operator} \parallel \mathbf{System}$$

It is possible to develop this approach to describe interaction between multiple operators by increasing the number of channels, **c** and **d**, over which communication

can occur. Designers might use this CSP description to identify predictability requirements. For instance, a **System** violates the previous requirement and behaves unpredictably if it does not present any output in response to **Operator** input.

Not only can CSP be used to analyse design principles but it can also be used to build prototypes that embody those principles. The experimental evaluation of partial implementations can ensure that the principles which guide human factors and systems engineering also have some utility for system operators. Alexander has developed a range of prototyping tools to implement CSP specifications of interactive systems [7]. Designers must perform non-trivial translations from CSP to eventCSP and from eventCSP to eventISL (Interaction Specification Language) before prototypes can be implemented using the `me too` programming language [143]. Alexander describes `me too` as "an executable subset of VDM (Vienna Development Methodology)" [6]. It seems appropriate, therefore, to examine whether the notation of VDM is better suited to support a principled approach to the integration of human factors and systems engineering.

VDM

Designers might use VDM operations to describe the effect of user input upon the state of a control system:

```
OP(x1:T1,...,xn:Tn) r1:Tn+1,...,rm:Tn+m
ext rd readvars wr writevars
pre pre
post post
```

The operation `OP` has inputs $\mathbf{x}_1$ of type $\mathbf{T}_1$ to $\mathbf{x_n}$ of type $\mathbf{T_n}$. `OP` yields the results $\mathbf{r}_1$ of type $\mathbf{T_{n+1}}$ up to $\mathbf{r_m}$ of type $\mathbf{T_{n+m}}$. After the keyword `rd` a number of state variables are given for reading and writing during this operation. The `pre` state defines conditions between the *readvars* and *writevars* that must hold before the operation can be applied. The `post` state describes the conditions that hold for those variables after the invocation of `OP`. This notation can be used to represent predictability requirements. Designers could require that operators can predict the effect, `post`, of all operations which can be invoked from an interface given some `pre` conditions.

Human factors and systems engineers cannot easily exploit VDM to represent concurrent interaction with complex and dynamic control systems. Describing an interactive system in terms of `pre` and `post` conditions implies a very sequential view of interaction. Users call an operation, wait for the effect and then call another. The state of a process is visible through the sequential evaluation of individual `OP`s. This is a significant limitation. Human factors and systems engineers are frequently concerned that certain conditions do or do not hold during an `OP`. For instance, a system should not enter an error state before it reaches a `post` condition. Kooij resolves this deficiency by introducing intermediate conditions, `inter`, which hold between `pre` and `post` conditions [180]. These are specified using interval temporal logic.

Interval Temporal Logic

Human factors and systems engineering have recognised that temporal properties frequently determine the success or failure of interactive control systems. Hysteresis effects can reduce the accuracy and reliability of sensing technology [36]. Delays in system responses lead to operator frustration and error [186]. The notation selected to demonstrate the feasibility of principled design should, therefore, be able to capture such properties. Interval temporal logic fulfills this requirement by providing operators such as $\bigcirc$ (read as 'next'), $\Box$ (read as 'always'), $\Diamond$ (read as 'eventually') and $\mathcal{U}$ (read as 'until') [205]. Appendix A presents the syntax and semantics of this language. It is important to note that "... all effective proof systems for temporal logic are incomplete for the standard semantics, in the sense that some formulas hold in every intended model but cannot be proved" [1]. Appendix B describes research that is intended to resolve such limitations. In contrast, this thesis intends to show how temporal operators might be used to specify requirements that must be satisfied by the human factors and systems engineering of interactive control systems. For instance, it is important that operators can view the effects, $\mathbf{s}'$, of their commands, $\mathbf{c}$, through a display, $\mathbf{d}$, if they are to learn to predict the consequences of their interaction:

$$\begin{aligned}&\forall\,\mathbf{c}\in\mathbf{C},\forall\,\mathbf{s}\in\mathbf{S},\exists\,\mathbf{d}\in\mathbf{D}:\\&\mathbf{visible\_effect}(\mathbf{c},\mathbf{s},\mathbf{d})\Leftarrow\exists\,\mathbf{s}'\in\mathbf{S}(\mathbf{effect}(\mathbf{c},\mathbf{s},\mathbf{s}')\wedge\Diamond\mathbf{view}(\mathbf{s}',\mathbf{d}))\end{aligned}\quad(1.1)$$

This might be interpreted as a human factors requirement; operators must eventually deploy their perceptual resources in order to view the effect of their commands. Systems engineers could interpret this as a requirement to eventually display the state transitions, from $\mathbf{s}$ to $\mathbf{s}'$, that are caused by commands.

1.6.3 The Tool: PRELOG

Thimbleby argues that Generative User Engineering Principles (GUEPs) might be expressed in two different forms [304]. One could be used by designers, the other might be exploited by system operators. He argues that formal notations, such as those described in the previous section, provide the precision and clarity required by designers. Vernacular, everyday language is proposed as a means of expressing GUEPs in a form that is accessible to system operators. Neither vernacular nor formal expressions help designers to determine whether operators will actually be able to predict the effects of their commands. Providing users with an informal definition of predictability does not guarantee that this principle will help them during the operation of a particular control system. Formal requirements and their natural language counterparts provide users with little impression of the 'look and feel' of potential implementations. In contrast, this thesis argues that prototypes can embody GUEPs in a form that is accessible to designers and operators. PRELOG, a system for Presenting and REndering LOGic specifications of interactive systems, has been developed to support this argument. The term 'rendering' is used here to describe the introduction of device and presentation details into abstract requirements. This additional information is necessary in order to implement prototypes that can be used to validate the products of a formal analysis of design principles.

Unlike Alexander's tools, mentioned in Section 1.6.2, PRELOG minimises the transformation necessary to move from principles to executable prototypes of interactive control systems.

1.7 Structure Of This Thesis

The central argument of this thesis is that designers can exploit principles to integrate the human factors and systems engineering of interactive control systems. Figure 1.2 illustrates how each chapter of the thesis supports this argument. The

Figure 1.2: The structure of this thesis

following paragraphs itemize the contributions of these chapters.

Chapter 1 has argued that designers must be able to integrate human factors and systems engineering in order to enhance the usability of many interactive control systems. The task-artifact cycle does not support this integration because an interface is assessed after it has been implemented. Principled design has been advocated as an alternative. Designers can use important properties of interactive control systems as heuristics to guide subsequent development. Principles also provide criteria against which to assess the usability of existing interfaces. Predictability has been chosen as an exemplar principle because it has been an important factor in the successful design and operation of many control systems. In order to exploit this approach designers must be able to represent the requirements imposed by design principles in a form that avoids the limitations which restrict the utility of vernacular specifications. It has been argued that formal, mathematically based, notations can be used to achieve this.

Chapter 2 focuses the search for a notation which designers might use to represent requirements imposed by principles, such as predictability. It is argued that production systems are intractable as a means of representing information which changes over time. The lack of explicit sequencing in first order predicate logic limits its utility for the principled design of dynamic systems. Interval temporal logic is advocated as an alternative that avoids these limitations. Human factors and systems engineers might use this notation to represent techniques that support principles, such as predictability.

Chapter 3 argues that complexity threatens the utility of principles as a means of integrating human factors and systems engineering. It is extremely difficult to represent and reason about design principles in terms of thousands of sensor readings and command options. Formal, mathematically based notations ease these burdens by providing abstractions which represent common components of many different control systems. Systems engineers might use these formalisms to identify high level objectives for automation without committing themselves to particular implementations. Human factors engineers could use them to identify techniques that ease the burdens which complexity places upon the finite cognitive and perceptual resources of system operators.

Chapter 4 argues that design principles might guide the development of open control systems. They can be used to identify the concurrency requirements that frequently determine the success or failure of such applications. Formal notations can be used to clarify the costs and benefits of systems engineering techniques, such as automated state locking, as means of supporting predictable interaction with open systems. They might also support analyses of human factors alternatives, such as the organisational changes recommended by social ergonomics.

Chapter 5 argues that principles provide designers with a means of informing their choice of particular development architectures. It is argued that this decision can have important consequences for both the human factors and systems engineering of a control system. In particular, recent claims that object orientation supports the development of consistent and predictable interfaces are assessed.

Chapter 6 argues that designers might use principles to highlight the potential weaknesses of development architectures. These weaknesses can frequently only be resolved by integrating techniques drawn from human factors and systems engineering. In particular, it is argued that object oriented control systems must provide their operators with additional sources of information when display objects fail to provide accurate representations of process components. Human factors design techniques, drawn from cognitive and perceptual psychology, must be used to ensure that operators can exploit the additional sources of information provided by systems engineering.

Chapter 7 argues that design bias limits the successful application of principled design. Design bias occurs when either experimental or formal techniques dominate development. Prototyping is proposed as a means of avoiding this bias. Partial implementations are amenable to the experimental analyses of human factors engineering. They can be used to assess the physiological ergonomics of potential implementations. Prototypes can also be used to identify some of the cognitive and perceptual demands that a control system places upon its operators. They can be shown to systems engineers to determine whether principles, such as predictability, could be achieved within the constraints of control technology. Finally and most importantly, prototypes can be shown to potential users in order to determine whether principles, such as predictability, have any relevance for system operators.

Chapter 8 argues that the approach, proposed in this thesis, is methodologically inadequate. It does not support the integration of human factors and systems engineering during all stages of design. Better development techniques must be devised to support the elicitation, verification, refinement and validation of principles. Notational improvements must be made so that designers might represent real-time requirements, risks and operator expertise. Better tool support must be provided

for the generation of detailed designs from abstract principles, for the generation of prototype displays and for the transition between partial and full implementations.

Chapter 9 summarises the conclusions that can be drawn from this thesis.

1.8 Insights Provided By This Thesis

In addition to the central argument, advocating a principled approach to design, it is hoped that this thesis will also provide a number of insights into techniques which can be used to support the development of interactive control systems.

1.8.1 Dialogue Cycles And Optimised Display Design

Chapter 2 builds upon work by Harrison, Roast and Wright [134] and argues that dialogue cycles provide designers with a powerful means of optimising the presentation of interactive control systems. These cycles use elements of a display, called gates, to indicate the beginning and end of command dialogues. A gate is presented when a user first issues a command and it is not presented again until that command has taken effect. It is argued that formal notations provide a means of representing the requirements that must be satisfied in order to exploit dialogue cycles, without considering low-level presentation details. This helps designers to develop display optimisation techniques early in development when it is often impossible to determine the presentation of a final implementation.

1.8.2 Transparency, Focusing And Restriction

Chapter 3 identifies state, display and command-view correspondence as problems that complicate the human factors and systems engineering of interactive control systems. State correspondence occurs when different states of an application process are represented by the same state of a control system. Display correspondence occurs when different states of a control system are presented by the same display. Command-view correspondence occurs when different sources, such as CRT displays and paper-based documentation, present information about the same command. Transparency, focusing and restriction are advocated as techniques which designers could exploit in order to reduce the problems that correspondence creates. Transparency requires that operators can differentiate between process states, control system states or command options. Focusing ensures that users can detect differences between states which they can affect. Restriction ensures that if a number of different sources of information are available for particular states or commands then those sources are not all presented at the same time.

1.8.3 Input And Output Protocols

Chapter 4 identifies input and output contention as problems facing the human factors and systems engineering of open control systems. Input contention can arise when a number of operators simultaneously attempt to access a shared resource. Output contention occurs when operators must simultaneously allocate finite cognitive and perceptual resources to monitor different application processes. It is argued that these problems can lead to unpredictability. Interval temporal logic is used to

specify input and output protocols which designers might use to reduce the impact of input and output contention.

1.8.4 Object Orientation And Consistency

Chapter 5 assesses recent claims that human factors and systems engineers can exploit object orientation to achieve consistency [190]. Previous authors have argued that it is extremely difficult to define the term 'interface consistency' [125]. It is, therefore, a non-trivial task to justify the claimed benefits for object oriented design. Interval temporal logic is used to demonstrate the basis for these assertions and to represent techniques that designers might exploit to support consistency and predictability. If objects are consistent then operators might predict that the consequences of issuing a command to one instance will be the same as issuing that command to any other instance of the same class. It is argued that this form consistency can be achieved through type instantiation.

1.8.5 Break-Down And Indirect Presentation

Chapter 6 agues that since the early 1960s designers have attempted to make the graphical representation of control objects conform as much as possible to the real-world appearance of process components [53]. Some human factors researchers have recently abandoned this pictorial realism [59]. Using interval temporal logic it is possible to explain this as an attempt to avoid the problem of break-down. Break-down occurs when operators must interact with physical components that appear to be complex and unpredictable in contrast to the control system objects which represent them. Many of the displays proposed by human factors research provide insufficient information for complex control tasks. Indirect presentation techniques are proposed as alternatives that avoid this limitation.

1.8.6 Input Events And Structured Graphics

Chapter 7 argues that, although partial implementations provide a means of validating the products of formal analyses, specification and prototyping are usually treated as alternatives. Integrating these techniques involves the introduction of device and presentation details, such as mouse handling and screen updates, into high-level designs. These details do not normally form any part of mathematical specifications and can threaten the tractability of a design. It is demonstrated that device abstractions and structured graphics systems provide means of avoiding this problem. In order to support these assertions we have implemented PRELOG, a tool for Presenting and REndering LOGic specifications of interactive systems.

The contribution of this thesis can be summarised by re-iterating the quotation that introduced this chapter. The purpose is to "show what has gone wrong in the past and to show how similar incidents might be prevented in the future". The integration of human factors and systems engineering through principled design is intended to ensure that "the lessons are (**not**) forgotten".

Part II

Principles And The Fundamental Problems Of Control

Introduction To Part II

This part of the thesis argues that designers might recruit principles to help them resolve the fundamental problems of control: dynamism; complexity and openness.

Chapter 2 identifies a notation which designers can use to represent the requirements that principles impose upon dynamic systems. It is argued that production systems are intractable as a means of representing information that changes over time. The lack of explicit sequencing in first order predicate logic poses problems for the representation of interactive dialogues. Interval temporal logic is advocated as an alternative that avoids these limitations. This notation supports integration because it can be used to represent the trade-offs that human factors and systems engineers must frequently make in order to achieve design principles.

Chapter 3 argues that complexity hinders both the design and operation of interactive control systems. Users must monitor and detect deviations in large amounts of interconnected application data. Users must select the most appropriate form of intervention from all the command options provided by an interactive control system. Designers must develop systems that help operators to perform these tasks. Principles can be recruited to support the design of complex systems because they establish high-level objectives for development. These objectives encourage integration because human factors and systems engineering must frequently be used in conjunction in order to achieve principles, such as predictability.

Chapter 4 argues that designers can exploit principles to guide the development of control systems that are open to interaction with multiple operators and processes. Principles provide criteria against which to assess human factors and systems engineering techniques that are intended to avoid input contention between concurrent users. They can also be used to assess techniques that are intended to support the concurrent presentation of application processes. It is argued that principles support the integration of human factors and systems engineering because they provide common standards against which to assess the products of these complementary approaches to design.

Chapter 2

Dynamism

> "When a world is dynamic ... there can be time pressures, tasks can overlap, sustained performance is required, the nature of the problem to be solved can change and monitoring requirements can be continuous or semi-continuous and change over time" (Woods, [329]).

2.1 Introduction

This chapter argues that designers might use principles to guide the development of dynamic control systems. The dictionary defines the adjective 'dynamic' to be "energetic, active, potent" [12]. In terms of this thesis, a dynamic system is one that requires its operators to monitor and control process variables that change at a rate which is likely to stretch their cognitive and perceptual resources. The problems of developing interfaces to energetic, active and potent applications are illustrated by the quotation that opens this chapter. The following pages argue that:

- formal notations provide means of representing techniques that can be used to achieve principles in dynamic control systems;
- first order predicate logic and production systems are inadequate for this task;
- interval temporal logic avoids the limitations of first order predicate logic and production systems.

It is concluded that interval temporal logic provides a notation which designers can exploit to support the integration of human factors and systems engineering.

2.1.1 Consequences: Stress And Unpredictability

The demands of controlling dynamic applications can cause considerable stress for system operators. For instance, Kuhmann, Boucsein, Schaefer and Alexander found that the incidence of high blood pressure, headaches and eye pain increased with the rate at which the values of process variables were updated [187]. In contrast, Schleifer and Amick found that operators quickly become frustrated if their system responds too slowly to their commands [274].

The temporal properties of an application can also determine the success or failure of design principles. Schleifer and Okogbaa argue that the psychological

characteristics which "make slow system response time a particularly stressful event include unpredictability and lack of control" [275].

2.1.2 Causes: Process And Dialogue Dynamics

The problems of operating dynamic systems are partly caused by the temporal properties of underlying production processes. For instance, faults in oil-production platforms can develop in seconds. There are, therefore, certain operations that require constant monitoring from system operators. Lord Cullen's report found that the accommodation module of the Piper Alpha platform was engulfed in flames within fifteen seconds of the initial explosion [81].

Production processes do not always determine the temporal properties of interaction. The problems of operating dynamic control systems are also caused by the temporal properties of interactive dialogues. The crew's response to the Piper Alpha explosion was delayed by the problems of issuing commands to start fire-fighting systems. Deluge equipment could have been activated but dialogue interlocks, or safety checks, prevented them from being started during diving operations.

2.1.3 Solution: Formal Notations

Designers might exploit a number of techniques to reduce the problems of operating dynamic control systems. Human factors and systems engineering could optimise dialogue design; the tempo of interaction can be changed by altering the rate at which information is displayed. For example, improvements in the presentation of safety systems, such as the fire-fighting equipment installed on the new Piper Bravo platform, are intended to buy time for system operators [147]. Graphical displays can present process summaries rather than the details of every sensor reading. Alternatively, designers could increase the tempo of interaction by providing predictive displays about the anticipated state of application processes. A predictive display is defined to be any display that presents information about the possible future states of a system. Flexible, or mixed initiative, task allocations can change the tempo of interaction by altering the allocation of control tasks between a system and its users at run-time. For instance, deluge equipment could be activated automatically if operators fail to detect a rig-fire [119]. This flexible task allocation shares the burdens of predictability. Systems must anticipate the effects of its actions in order to select an appropriate course for intervention, just as users must predict the effects of their interaction.

Designers must make a number of trade-offs if they are to exploit techniques, such as display optimisation and flexible task allocation. There are consequences for human factors engineering if systems engineering allocates operator tasks to automated equipment at run-time. The cognitive and perceptual demands upon users can be increased if system intervention changes the effects of their commands from those that would otherwise have been predicted. There are similar problems with optimised dialogue design. For instance, Yeh and Wickens argue that predictive displays add to clutter and increase visual workload [332]. It is important, therefore, that human factors and systems engineers can represent and reason about the costs and benefits which such techniques have for the principled design of dynamic systems.

2.2 Production Systems

Production systems provide a suitable medium for integration through principled design because they have been used by human factors and systems engineering. Systems engineering has exploited production systems to support the development of automated control systems. For instance, the mission planner of the Mithra au-

Figure 2.1: The Mithra autonomous robot

tonomous robot, illustrated in Figure 2.1, was developed using a production system [61]. Post argued that mathematics can be viewed as a process of symbol manipulation and that any computable manipulation can be modelled using production systems [240]. Human factors researchers, such as Rasmussen and Pederson [250], Reisner [255] and Shneiderman [285], have used production systems to analyse the cognitive and perceptual demands that interfaces impose upon their operators.

2.2.1 Facts and Rules

There are two elements of most production systems: facts and rules [83]. Designers could use facts to represent properties of predictable control systems. For instance, operators might make accurate predictions about the effects of their commands upon the state of a control system if they can observe that state from the information displayed. It must, therefore, be a fact that displays provide accurate information about the state of a control system. Human factors engineering must develop presentation formats that enable operators to recognise that the **on_display** represents a fire-fighting system which is **on**. Systems engineering must deploy sufficient sensing equipment so that it is possible to detect that the system is in the **on** state.

$$\mathbf{FACT}(\mathbf{SYSTEM}, \mathbf{on}, \mathbf{on\_display})$$

This illustrates an important advantage of production systems. Facts can be used to specify objectives for principled design without specifying the images and sensor readings that must be used by a particular implementation. Designers can postpone such decisions until later stages in development when more information is available about the costs and benefits of hardware platforms and presentation devices [304]. Facts can also be used to represent input. This is important if production systems are to represent techniques that are intended to support predictions about the effects of operator intervention. For instance, designers could require that a **user** provides **INPUT** to **start** a system:

$$\mathbf{FACT}(\mathbf{INPUT}, \mathbf{user}, \mathbf{start})$$

Unfortunately, human factors and systems engineers cannot easily use the facts of production systems to represent the effects of operator **INPUT**. Rules, or productions, can be used to avoid this limitation. They consist of a pre-condition and a post-condition. The pre-condition describes constraints that must hold before a rule can be fired. The post-condition describes constraints that should hold after a rule has been fired. Computation can be modelled in terms of the changes caused to the facts known about a system by the successive selection and firing of rules. For instance, operator predictions could be supported by ensuring that if they **start** a fire-fighting system then it is **on**, not **off**, and the **on_display** is presented, not the **off_display**:

$$\begin{aligned}&\mathbf{CONDITION}(\mathbf{effect\_rule}, \mathbf{IF\ FACT}(\mathbf{INPUT}, \mathbf{user}, \mathbf{start})\mathbf{THEN}\\ &\quad \mathbf{REMOVE\ FACT}(\mathbf{SYSTEM}, \mathbf{off}, \mathbf{off\_display})\mathbf{AND}\\ &\quad \mathbf{ADD\ FACT}(\mathbf{SYSTEM}, \mathbf{on}, \mathbf{on\_display}))\end{aligned}$$

Designers might use facts and rules to assess the impact that techniques, such as display optimisation and flexible task allocation, have upon principles for the human factors and systems engineering of dynamic control systems.

2.2.2 Rule Competition And Task Allocation

Automated systems are, typically, allocated tasks that require rapid intervention. In contrast, operators are frequently required to provide a flexible response to unexpected conditions. There are circumstances when such a static allocation of tasks fails to guarantee the safety of dynamic control systems. Equipment failure can prevent a system from providing a timely response. Heavy workloads can prevent users from intervening to resolve unexpected errors. Lee and Moray [192] and Bainbridge [22] argue that systems must assume operator tasks when the dynamism of application processes stretches their perceptual and cognitive resources. Rouse suggests that designers can prepare for changes in the allocation of tasks by encouraging operators to routinely perform system activities and vice versa [261]. For instance, automated equipment on oil-rigs can intervene to perform some operator tasks, such as tightening threaded drill connections with hydraulic tongs [159]. If systems fail then operators can resume manual control. Designers can use rule-competition between two productions to represent the requirements that must be satisfied in order to exploit this flexible task allocation. Rule-competition occurs when two productions share the same antecedent but have different consequents. For instance,

designers might specify that if a **sensor** detects a pipe with a **thread_loose** then a **user** issues **INPUT** to secure it. It could also be specified that if a **sensor** detects a pipe with a **thread_loose** then the **system** intervenes:

CONDITION(**effect_rule**, **IF FACT**(**INPUT**, **sensor**, **thread_loose**)**THEN ADD FACT**(**INPUT**, **user**, **secure_thread**))

CONDITION(**effect_rule**, **IF FACT**(**INPUT**, **sensor**, **thread_loose**)**THEN ADD FACT**(**INPUT**, **system**, **secure_thread**))

These production rules provide designers with a basis upon which to assess the advantages and disadvantages of techniques, such as flexible task allocation, for predictable interaction. They capture the benefits of flexible task allocation because either the **system** or its **user** responds to **sensor INPUT**. This provides redundancy. If human factors engineering fails to ensure that users can deploy their perceptual and cognitive resources to respond to an error then the automation of system engineering can intervene. Alternatively, if automation fails then users might respond. These production rules also capture the costs which limit the utility of flexible task allocation; either rule can fire given the antecedent **INPUT**. The rules do not determine whether the operator or the control system will respond to a **thread_loose** warning. This provides an example of what Kowalski describes as "don't care non-determinism" [182]. It does not matter which of the rules is fired because both describe legal traces of interaction. This non-determinism has a profound consequence for usability. Operators cannot predict that the effects of their commands will be free from the interference that can occur if automated equipment responds to the **sensor INPUT**. The use of production rules helps to clarify the point that flexible task allocation provides redundancy at the cost of jeopardising predictability. In order to avoid such non-determinism designers must specify the order in which these rules are to be selected. Section 2.2.4 will explain how this can be done.

2.2.3 Rule Collision And Display Design

Designers might optimise display design in order to reduce the cognitive and perceptual demands that are placed upon the operators of dynamic control systems [249]. For instance, Schraagen describes how the Royal Netherlands Navy has developed systems that filter the presentation of damage control information [277]. Instead of presenting every change in process parameters, designers could display information about major trends in production. Rule-collision provides a means of representing the requirements that must be satisfied in order to exploit this form of optimised display design. Rule-collision occurs when two productions share the same consequent but have different antecedents. For instance, it might be specified that if **sensor INPUT** indicates a **drill_error** then the system is in the **error** state and the **error_display** is presented. Designers could also require that if **sensor INPUT** indicates a **tank_error** then the system is in the **error** state and the **error_display** is presented. These requirements describe an instance of optimised display design because the two sensor readings are hidden behind a single **error_display**:

CONDITION(**effect_rule**, **IF FACT**(**INPUT**, **sensor**, **drill_error**)**THEN**

ADD FACT(**SYSTEM**, **error**, **error_display**))

CONDITION(**effect_rule**, **IF FACT**(**INPUT**, **sensor**, **tank_error**)**THEN**
ADD FACT(**SYSTEM**, **error**, **error_display**))

The strength of this technique is that operators can focus their monitoring to detect the effect of their intervention at a relatively high level of detail. Optimised display design can, however, jeopardise principles, such as predictability. If operators focus their attention at a high level of abstraction then they are likely to miss detailed information about the state of their control system. For instance, users could not distinguish between a **tank_error** and a **drill_error** from the **error_display** alone. This need not be a significant limitation if systems engineering guarantees that all errors will be resolved. If this is not the case then human factors engineering must ensure that operators are provided with sufficient information for them to diagnose the causes of an **error**. This could seem a trivial requirement but a recent survey of the Canadian oil industry found interfaces in which drillers could not distinguish between many different power system failures from the information displayed [315].

2.2.4 The Problems Of Production Systems

Production systems provide inadequate support for the integration of human factors and systems engineering. In particular, they cannot easily be applied to represent and reason about the consequences of design principles for dynamic applications. Designers must maintain a number of different representations of an interactive control system. Changes in state or display are represented by rules that alter facts. Changes in the effect of input upon the state and display are represented by meta-rules that alter rules in the database [46]. For instance, equipment failure changes the effect of operator commands. This can prevent users from making accurate predictions about the effects of their intervention. Designers might, therefore, specify that if there is a system **error** then a meta-rule is fired modifying the rule describing the **effect** of **user** input so that a warning is displayed:

CONDITION(**meta_rule**, **IF FACT**(**INPUT**, **sensor**, **error**)**THEN**
MODIFY CONDITION(**effect_rule**, **IF FACT**(**INPUT**, **user**, **start**)**THEN**
REMOVE FACT(**SYSTEM**, **off**, **off_display**)**AND**
ADD FACT(**SYSTEM**, **off**, **error_display**)))

In order for designers to reason about interface behaviour they must know how meta-rules alter rules and how rules alter facts. The maintenance of these different representations imposes non-trivial burdens upon the development resources available to human factors and systems engineering. Designers are also constrained by the algorithm that is used to select which rules and meta-rules are to be fired. For instance, if a production system always selected a rule requiring operator input before one requiring system input then it would be difficult to represent techniques such as flexible task allocation. Designers could avoid this problem by explicitly specifying the selection algorithm [46]. Human factors and systems engineers would then have to represent and reason about the consequences of these meta-meta-rules upon a potential control system.

2.3 First Order Predicate Logic

First order logic provides a notation capable of supporting both human factors and systems engineering. Hammond and Sergot argue that first order logic is an ideal medium in which to represent system data and operator expertise [129]. Fox recruits this formalism for similar ends in the Oxford System of Medicine [105]. These observations motivate the use of this formalism for the principled design of interactive control systems.

2.3.1 Sets And Predicates

Section 1.5 argued that principles should provide a high degree of domain independence. Sections 2.2.2 and 2.2.3 showed how abstractions, such as **tank_error** and **drill_error**, might be used to represent a control system design. This level of detail is inappropriate if designers are to represent techniques that support the principled development of systems, such as avionics applications, that do not contain any drills. First order logic can be used to avoid such domain dependence. For instance, designers might introduce a set, **S**, to include all the states that a control system could possibly be in. The set **P** contains all input sequences that could be entered into a control system. The set **C** can be introduced to describe all operator commands. The set **D** contains all display images that could be presented to system operators. Figure 2.2 illustrates the relationships between the sets **P**, **C**, **S** and **D**. Operator input is interpreted as a command that affects the state of the system which is presented by a display. The resulting structure is similar to the **PiE** model, briefly described in Section 1.6.2. Both provide high-level models of interaction. The elements of sets **P**, **C**, **S** and **D** are terms. Terms in first order logic can be compared to the subjects of sentences in English. For example, 'Christopher is taller than Victoria' contains two terms, 'Christopher' and 'Victoria', and a predicate, 'is taller than'. Predicates, when combined with one or more terms, form sentences in first order logic. Quantification can also be introduced in order to specify whether a predicate holds for all the terms in a set, denoted by the $\forall$ symbol, or for some terms in a set, denoted by $\exists$, or for only one term in a set, denoted by $\exists!$. For instance, a predicate might be used to specify that a control system is in a state **s**:

$$\exists! \mathbf{s} \in \mathbf{S} : \mathbf{state}(\mathbf{s}) \tag{2.1}$$

A predicate could also be introduced between similar control system states. This illustrates an important benefit of a high-level of abstraction. In practice, some control system states are regarded as identical even though there can be subtle differences in the application information which they represent [36]. Designers might use first order predicate logic to represent similar states without defining the ranges of variable values that must be considered in order to determine whether two states of an implementation are the same. These details can be gradually introduced as development progresses and the state of an application, in terms of sensor reading and application information, becomes better defined:

$$\exists \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{same\_state}(\mathbf{s}, \mathbf{s}') \tag{2.2}$$

It is important to emphasise that the elements of **S**, control system states, are not the states of the application processes which a system interacts with. Sections 3.2

Figure 2.2: A high-level model of interaction

and 4.3 will argue that this distinction must be considered during the human factors and systems engineering of predictable control systems. For now it is sufficient to realise that in order to assess the impact of flexible task allocation upon principles, such as predictability, it must be possible to represent operator input. Designers might, therefore, introduce a predicate that is true for input sequences, $\mathbf{p}$, that are entered into the system:

$$\exists\, \mathbf{p} \in \mathbf{P} : \mathbf{input}(\mathbf{p}) \tag{2.3}$$

First order logic can also be used to specify more complex relationships between the elements of $\mathbf{P}$ and $\mathbf{S}$. For instance, designers could require that input sequences are interpreted as commands whatever the state of the system:

$$\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{c} \in \mathbf{C} : \mathbf{interpret}(\mathbf{p}, \mathbf{s}, \mathbf{c}) \tag{2.4}$$

In order to reason about the impact of presentation techniques, such as optimised display design, upon principles, such as predictability, it must be possible to represent the appearance of an interface at this level of abstraction. This can be achieved by introducing a predicate that is true for displays which are presented to system

operators:

$$\exists\, \mathbf{d} \in \mathbf{D} : \mathbf{display}(\mathbf{d}) \tag{2.5}$$

It is important to emphasise that the elements of **D** are the graphical images which are presented to system operators and not the devices or monitors that are used to present those images. Section 7.4 will present techniques for gradually introducing the details of display presentation formats and screen layouts into an abstract design. For now it is sufficient to observe that human factors and systems engineers might also introduce an identity relation between displays:

$$\forall\, \mathbf{d} \in \mathbf{D} : \mathbf{same\_display}(\mathbf{d}, \mathbf{d}) \tag{2.6}$$

Predicates can also be interpreted as high-level predictability requirement that must not be jeopardised by techniques, such as optimised display design. For instance, in order for operators to determine the effect of their commands it must be possible for them to view all states, **s**, through displays, **d**:

$$\forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{view}(\mathbf{s}, \mathbf{d}) \tag{2.7}$$

Designers can use first order predicate logic to describe the effect of commands, **c**, in terms of their pre-conditions, **s**, the states in which the commands are received by the system, and post-conditions, $\mathbf{s}'$, the states of the system after the commands are acted upon. Commands can affect many future states of a system. The post-condition can be viewed as the first in a sequence of state transitions that are determined not only by that command but also by subsequent interaction. For convenience this thesis will simply refer to $\mathbf{s}'$ as the effect of **c** on **s**. It should be noted that the following predicate describes the potential effect of **c** on **s**, it makes no commitment to the command actually being issued nor does it assume that $\mathbf{s}'$ will be achieved:

$$\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{effect}(\mathbf{c}, \mathbf{s}, \mathbf{s}') \tag{2.8}$$

Designers might use first order predicate logic to clarify vernacular descriptions of human factors and systems engineering principles. For instance, the informal description of predictability, given in Section 1.6.1, can be restated in terms of the elements of **C** and **S**. Operators must be able to predict the effect, $\mathbf{s}'$, of their command, **c**, issued in a state, **s**. This makes explicit the idea that operator predictions are likely to be based upon the state of their control system. The point is not that logic can be used to derive a single formalisation of predictability. Rather it is argued that logic provides a means of teasing out some of the assumptions and ideas that are implicit within vernacular descriptions.

In addition to the predicates, introduced above, human factors and systems engineers might exploit a number of logic connectives and operators to represent techniques that are intended to support design principles. In the following **p** and **q** are assumed to be predicates:

- a negation is represented as $\neg\, \mathbf{p}$. This denotes that **p** is not true;
- a conjunction is represented as $\mathbf{p} \wedge \mathbf{q}$. This denotes that **p** and **q** are both true;

- a disjunction is represented as $\mathbf{p} \vee \mathbf{q}$. This denotes that either $\mathbf{p}$ is true or $\mathbf{q}$ is true or both $\mathbf{p}$ and $\mathbf{q}$ are true;
- an implication is represented as $\mathbf{p} \Leftarrow \mathbf{q}$. This denotes that $\mathbf{p}$ is true if $\mathbf{q}$ is true;
- a bi-condition is represented as $\mathbf{p} \Leftrightarrow \mathbf{q}$. This denotes that $\mathbf{p}$ is true if and only if $\mathbf{q}$ is true.

Kowalski describes the syntax and semantics of first order logic in greater detail [182]. In contrast, the following sections concentrate on the application of this formalism as a means of integrating human factors and systems engineering to support the principled design of dynamic control systems.

2.3.2 Context Dependent Effects And Task Allocation

The conjunction and bi-condition operators, introduced in the previous section, can be used to represent further relationships between the elements of the high-level model of interaction, illustrated in Figure 2.2. For example, the effect of input $\mathbf{p}$ on $\mathbf{s}$ is interpreted to be $\mathbf{s}'$ if and only if $\mathbf{p}$ is interpreted as a command, $\mathbf{c}$, which has the effect of changing the state of the system from $\mathbf{s}$ to $\mathbf{s}'$:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \exists\, \mathbf{c} \in \mathbf{C}\ (\mathbf{interpret}(\mathbf{p}, \mathbf{s}, \mathbf{c}) \wedge \mathbf{effect}(\mathbf{c}, \mathbf{s}, \mathbf{s}')) \end{aligned} \tag{2.9}$$

Human factors and systems engineers might use predicates, such as **interpret_effect**, to help them assess the impact of techniques, such as flexible task allocation, upon principles, such as predictability. For instance, systems engineering has developed automatic deluge systems because oil-rig operators frequently fail to detect and respond to fires. There are states of the system which are unprotected by these additional safety features, for instance, during periodic maintenance. Operator input, $\mathbf{p}$, can have the effect of activating automated equipment, $\mathbf{s}''$, and can have the effect of raise an error condition, $\mathbf{s}'''$, depending on whether that equipment is functioning correctly, $\mathbf{s}$, or is faulty, $\mathbf{s}'$. In other words, the consequences of operator intervention depend upon the context of interaction:

$$\begin{aligned} \exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{context\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \exists\, \mathbf{s}'', \mathbf{s}''' \in \mathbf{S} \\ (\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}'') \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}', \mathbf{s}''') \wedge \\ \neg\, \mathbf{same\_state}(\mathbf{s}, \mathbf{s}') \wedge \neg\, \mathbf{same\_state}(\mathbf{s}', \mathbf{s}''')) \end{aligned} \tag{2.10}$$

This predicate illustrates how first order predicate logic can be used to clarify the ways in which flexible task allocation jeopardises predictability. Systems engineers must anticipate that systems might fail to provide back-up protection. If this occurs then operators cannot predict that the safety of an application will be preserved should they, in turn, fail to provide appropriate input. The importance of this point is illustrated by the Piper Alpha disaster. The deluge systems were designed to provide automatic support should operators fail to respond to a fire. Diving operations meant that these systems had to be disabled. Operators assumed manual control of the fire-fighting equipment. They failed to predict that their commands could not activate deluge heads in the accommodation module without system support [81].

Designers might use logic abstractions to represent techniques which support the operation of dynamic systems but which do not sacrifice principles, such as predictability. Different displays, $\mathbf{d}$ and $\mathbf{d}'$, could have warned the Piper Alpha operators about the different effects of input, $\mathbf{p}$, to activate deluge equipment during normal operation, $\mathbf{s}$, and during diving operations, $\mathbf{s}'$. In other words, predictability is preserved if users can accurately view the context in which they issue their input:

$$\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{visible\_context\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \exists\, \mathbf{s}' \in \mathbf{S}$$
$$(\mathbf{context\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge$$
$$\mathbf{view}(\mathbf{s}', \mathbf{d}') \wedge \neg\, \mathbf{same\_display}(\mathbf{d}, \mathbf{d}')) \tag{2.11}$$

This predicate illustrates the point that human factors engineering must be recruited in order to preserve predictability if the automation of systems engineering fails to support flexible task allocations. Human factors considerations must inform the design of displays, $\mathbf{d}$ and $\mathbf{d}'$, so that the cognitive and perceptual resources of system operators can detect different contexts of interaction.

2.3.3 Visible Input Effects And Display Design

Displays can be optimised by hiding some of the state transitions that occur during interaction with a dynamic control system. For example, if operators are responding to an alarm then the maritime damage control system, mentioned in Section 2.2.3, hides information about any subsequent warnings that have a lower 'calamity' rating [277]. This additional state information can be accessed once the original error has been resolved. There are, however, a number of problems that limit the utility of this form of optimised display design. In particular, operators might not be able to determine whether their predictions have been confirmed if pre-conditions, $\mathbf{s}$, and post-conditions, $\mathbf{s}'$, are viewed through the same display, $\mathbf{d}$:

$$\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{invisible\_input\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{d}) \Leftrightarrow \exists\, \mathbf{s}' \in \mathbf{S}$$
$$(\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d})) \tag{2.12}$$

Designers could exploit first order predicate logic in order to represent techniques that are intended to support predictability. The pre-condition, $\mathbf{s}$, and the post-condition, $\mathbf{s}'$, of input $\mathbf{p}$ are viewed through different displays, $\mathbf{d}$ and $\mathbf{d}'$:

$$\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{visible\_input\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \exists\, \mathbf{s}' \in \mathbf{S}$$
$$(\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge$$
$$\mathbf{view}(\mathbf{s}', \mathbf{d}') \wedge \neg\, \mathbf{same\_display}(\mathbf{d}, \mathbf{d}')) \tag{2.13}$$

This predicate can be interpreted as a requirement that must be satisfied by predictive displays. The **interpret_effect** specifies the potential effect, $\mathbf{s}'$, of $\mathbf{p}$ on $\mathbf{s}$. The $\mathbf{d}'$ display, therefore, presents a prediction about the consequences of operator input. This prediction is confirmed if $\mathbf{state}(\mathbf{s}')$ is true. Human factors and systems engineering must be integrated in order to successfully exploit this technique. Systems engineering must ensure that sensors can detect the present state of a control system. It must also develop systems that can predict the consequences of operator intervention. Designers might recruit human factors engineering to inform the choice of displays, $\mathbf{d}$ and $\mathbf{d}'$, which are most likely to enable users to perceive and recognise

the future effects of their input. This is a non-trivial task. Predictive displays of control systems must not only present application data but they must also present information about the passage of future time. Stokes and Wickens review the many problems that must be overcome in order to estimate the future state of a control system based upon its present state [293]. These problems justify the decision not to concentrate the remainder of this thesis solely upon predictive presentation techniques. Instead, the intention is to demonstrate that principles can be applied to support the human factors and systems engineering of many different interfaces for many different control systems. The techniques that will be described are also intended to support the development of predictive displays. For instance, Section 4.2 will propose a number of protocols that can be used to support system predictions about the impact of input from operators and application processes. Section 7.4 will describe prototyping tools that human factors engineers can exploit to assess the utility of novel display formats for predictive displays.

2.3.4 The Problems Of First Order Predicate Logic

First order logic is monotonic; the number of requirements specified by predicates always increases. Designers must take care to ensure that the addition of a predicate does not lead to a contradiction. For instance, it could be specified that operator input, $\mathbf{p}$, to activate fire fighting equipment causes it to change from the off state, $\mathbf{s}$, to the on state, $\mathbf{s}'$. Intuitively, it makes little sense to require that a system is and is not in the off state, $\mathbf{s}$:

$$\begin{aligned}\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{effect\_contradiction}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftarrow \\ \mathbf{state}(\mathbf{s}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \neg\, \mathbf{state}(\mathbf{s})\end{aligned} \tag{2.14}$$

Designers might avoid this problem by using a database to record the facts that are known about a system during interaction. For instance, in order to specify a change in the state of a control system, designers must use the meta-predicate **retract** to remove information about the existing state from a database. A similar meta-predicate can be used to **assert** information about the new state:

$$\begin{aligned}\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{database\_change}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftarrow \\ \mathbf{state}(\mathbf{s}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \\ \mathbf{retract}(\mathbf{state}(\mathbf{s})) \wedge \mathbf{assert}(\mathbf{state}(\mathbf{s}'))\end{aligned} \tag{2.15}$$

Such techniques introduce an important distinction between the logic of a specification and the mechanisms which designers must use to describe dynamic interaction [217]. Designers must not only assess the impact of principles upon an interactive control system but they must also consider how best to maintain their database of assertions. The lack of explicit sequencing in first order predicate logic poses further problems for the representation of interactive dialogues. For instance, consider the following implication:

$$\begin{aligned}\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{no\_sequence}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftarrow \\ \mathbf{input}(\mathbf{p}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}')\end{aligned} \tag{2.16}$$

The logic does not determine the order in which predicates are to be satisfied. This implication would be true if input, $\mathbf{p}$, were received after it had taken effect.

This would have disastrous consequences for predictability if a deluge system was activated in response to operator commands that had yet to be issued! In order to implement a specification, designers must determine an order of evaluation. This can be achieved by introducing a distinction between declarative and procedural interpretations of first order predicate logic. Declarative statements of the form 'A if and only if B and C and D' can be read as procedure calls 'to do A, you must do B and C and D'. Kowalski observes that the declarative reading of rules makes the meaning of a specification clear but "it is the procedural interpretation of rules that makes them useful" [183]. It is important to notice that implementations would only exhibit the desired behaviour because control considerations influence the ordering of logic specifications. Changing the order of **interpret_effect** and **input** would have no effect on the specification of a dialogue but would radically affect the behaviour of an implementation. This introduces considerable complexity into the refinement necessary to realise high-level principles within an interactive control system. The ordering of clauses might have to be substantially revised in order to describe the control strategy of an implementation. Alternatives must be sought if logic is to provide a convenient bridge between human factors and systems engineering.

2.4 Logic And Time

The following pages argue that the application of temporal formalisms can be extended from artificial intelligence research [10] and concurrency theory [1] to support the principled design of interactive control systems. Sakuragawa has exploited temporal extensions to first order logic in order to support the systems engineering of production processes [270]. Goldsack and Finkelstein [122] and Hale [128] have exploited similar formalisms to reason about the scheduling properties of systems. Anderson and Thimbleby have used a temporal logic to model interaction with multi-user applications [15]. This work motivates an investigation into the utility of temporal logics for the integration of human factors and systems engineering.

2.4.1 Time Stamps

The lack of sequencing in first order predicate logic can be overcome by the introduction of temporal information into logic specifications of interactive systems. An order of evaluation can be described with respect to points in time. Choi, Yang and Chang have applied this approach to introduce temporal information into an expert system for fault diagnosis in nuclear power plants [67]. For instance, time points can be used to specify the conditions which must hold in order to diagnose that a fire spreads between points **A** and **B** at twenty seconds after midday:

$$\begin{aligned}&\mathbf{accident}(\mathbf{fire\_from\_A\_to\_B}, 120020) \Leftarrow \\ &\quad \mathbf{input}(\mathbf{fire\_sensor\_A}, 120010) \wedge \mathbf{input}(\mathbf{fire\_sensor\_A}, 120015) \wedge \\ &\quad \mathbf{input}(\mathbf{fire\_sensor\_B}, 120020) \wedge \neg\, \mathbf{input}(\mathbf{fire\_sensor\_A}, 120020) \quad (2.17)\end{aligned}$$

Choi, Yang and Chang's approach avoids some of the limitations of first order predicate calculus. Without time-stamps the previous implication would have included

a contradiction; fire is and is not detected by sensor A. Time stamps can also capture the sequencing information that was absent from **no_sequence** (2.16). The following implication illustrates how designers might exploit time-stamps to specify the predictability requirement that input is issued before it takes effect. The effect of input, $\mathbf{p}$, occurs five seconds after it has been issued. The following assumes that predicates, such as $\mathbf{input}(\mathbf{p}, 120000)$, have the same semantics as those introduced in Sections 2.3.1 and 2.3.2 except that a time-point is included as an additional parameter to specify the particular moment when they hold:

$$\begin{aligned}&\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{time\_stamp\_change}(\mathbf{p}, \mathbf{s}, \mathbf{s}', 120000) \Leftarrow \\ &\ \mathbf{input}(\mathbf{p}, 120000) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}', 120000) \wedge \\ &\mathbf{state}(\mathbf{s}', 120005) \wedge \neg\, \mathbf{state}(\mathbf{s}, 120005) \end{aligned} \tag{2.18}$$

An important advantage of introducing time-stamps into first order predicate logic is that designers can make explicit the real-time requirements which must be satisfied by systems engineering. In order to fulfill **time_stamp_change** (2.18) a user must input $\mathbf{p}$ at midday exactly. If this input is not provided then systems engineers could allocate some of the user's tasks to automated systems. The 120000 time-stamp provides a deadline for operator intervention. Designers might also exploit time-stamps to represent the temporal requirements that human factors engineering must satisfy. In order to predict the effect of their interaction, operators must be trained to anticipate that their input, $\mathbf{p}$, will have an effect $\mathbf{s}'$ on $\mathbf{s}$ five seconds after it is issued. Users should not issue commands that take ten seconds to complete if their system will shut-down in five.

2.4.2 Time-Variables

Introducing time-stamps into first order predicate logic does not provide designers with a panacea for the development of dynamic control systems. The previous implication, **time_stamp_change** (2.18), holds at midday. In order to specify the effect of input at other times, such as 120010 or 120015, designers would have to produce additional predicates. This imposes significant burdens upon the development of control systems that operate over long periods of time. This limitation can be avoided by introducing a set of time-variables, $\mathbf{T}$. The following predicates have the same semantics as those introduced in Sections 2.3.1 and 2.3.2 except that time-variables, $\mathbf{t}$ and $\mathbf{t}'$, are included as additional parameters in order to specify the moments when they hold. These time-variables could be instantiated at 120000, 120005, 120010 or 120015:

$$\begin{aligned}&\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S}, \exists\, \mathbf{t}, \mathbf{t}' \in \mathbf{T} : \mathbf{time\_variable\_order}(\mathbf{p}, \mathbf{s}, \mathbf{s}', \mathbf{t}, \mathbf{t}') \Leftrightarrow \\ &\mathbf{input}(\mathbf{p}, \mathbf{t}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}', \mathbf{t}) \wedge \mathbf{state}(\mathbf{s}', \mathbf{t}') \wedge \mathbf{t} < \mathbf{t}' \end{aligned} \tag{2.19}$$

The use of time-variables can still introduce considerable complexity into a design [111]. In particular, designers must maintain a clear semantics for relations, such as $<$, which describe an ordering over time-variables. It is possible to create circular models of time in which a temporal variable occurs both before and after itself. This is likely to have disastrous consequences for the human factors and systems engineering of any implementation that was derived from such a specification.

2.4.3 Modal Logic

Modal logics do not explicitly represent time. They extend first order logic by introducing the necessity operator, denoted by $\mathcal{L}$, and the possibility operator, denoted $\mathcal{M}$. These are defined in terms of a set of states of world knowledge, $\mathbf{St}$. The elements of this set capture all the information that might be known about a potential implementation and its environment. In the following, $\mid \mathbf{w} \mid_{\mathbf{st}}$ denotes the truth value of the formula $\mathbf{w}$ in state of world knowledge $\mathbf{st}$. A predicate is necessarily true in $\mathbf{st}$ if and only if it holds in all states of world knowledge, $\mathbf{st}'$, that are accessible from $\mathbf{st}$:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \mathcal{L}(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \forall\, \mathbf{st}' \in \mathbf{St}[\mathbf{accessible}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'}] \quad (2.20)$$

The $\mathcal{M}$ operator can be specified in a similar fashion. A predicate is possibly true in states of world knowledge $\mathbf{st}$ if and only if it is true in at least one state of world knowledge, $\mathbf{st}'$, that is accessible from $\mathbf{st}$:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \mathcal{M}(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \exists\, \mathbf{st}' \in \mathbf{St}[\mathbf{accessible}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'}] \quad (2.21)$$

It is important to note that a number of problems limit the utility of analyses conducted in terms of the elements of $\mathbf{St}$. For instance, designers cannot hope to refine these abstractions to capture complete knowledge about worlds that include many different users and application processes. Alternatively, human factors and systems engineers might use terms and predicates to focus an analysis upon particular aspects of world knowledge that are important for particular design principles. Techniques that are intended to support predictability could be represented in terms of control system states, rather than entire states of world knowledge. This does not abandon the modal model, elements of $\mathbf{S}$ can be extracted from elements of $\mathbf{St}$. Informally, it should be possible to determine the state, $\mathbf{s}$, of a control system given complete world knowledge, $\mathbf{st}$. Designers might use the $\mathcal{M}$ operator to specify that input $\mathbf{p}$ has the possible effect of changing the state of the system. The following assumes that predicates, such as $\mathbf{state}(\mathbf{s})$, have the same semantics as those introduced in Sections 2.3.1 and 2.3.2 except that they can be qualified by a modal operator that specifies the states of world knowledge in which they hold:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{modal\_state\_change}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{state}(\mathbf{s}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathcal{M} \neg\, \mathbf{state}(\mathbf{s}) \end{aligned} \quad (2.22)$$

This predicate illustrates how modal logic avoids the constraints of monotonicity; the system is in state $\mathbf{s}$ and it is possible for it not to be in state $\mathbf{s}$. Human factors and systems engineers could exploit modal operators to represent techniques that are intended to support predictability. For instance, Section 2.3.2 argued users operators cannot easily predict the consequences of their input if its effects change. The $\mathcal{L}$ operator provides designers with a means of explicitly stating that such unpredictability must be avoided. The effect of $\mathbf{p}$ on $\mathbf{s}$ is $\mathbf{s}'$ whatever changes occur to the model of world knowledge:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{modal\_persistent\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathcal{L}\ \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \end{aligned} \quad (2.23)$$

Not all the members of multi-disciplinary design teams can be expected to understand the theoretical foundations of modal logic [44]. Section 7.2.2 will argue that formal design requirements can be more easily communicated by developing prototype implementations that embody principles, such as predictability. For now it is sufficient to observe that this might be achieved by deriving natural language interpretations of the requirements that are intended to support principled design. For instance, the previous predicate can be interpreted as a requirement that the effect of input $\mathbf{p}$ on state $\mathbf{s}$ is $\mathbf{s}'$ in all worlds accessible from the present one. This example shows that it is not easy to derive a vernacular translation of modal requirements. The term 'accessible from the present one' is vague because it is difficult to translate from the formal underpinning of modal logic into natural language.

2.4.4 Interval Temporal Logic

Interval temporal logic 'interprets' the accessibility relation between the states of world knowledge in modal logic to be an ordering over time [205]. With this refinement the $\mathcal{L}$ operator can be re-interpreted as 'always', denoted $\Box$, and the $\mathcal{M}$ operator as 'eventually', denoted $\Diamond$. It is important to emphasise that the properties described using temporal and modal operators, such as $\mathcal{L}$ and $\Box$, could be specified in first order predicate logic by explicitly representing states of world knowledge, $\mathbf{st}$ and $\mathbf{st}'$. $\Box\mathbf{w}$ is true in $\mathbf{st}$ if and only if it is true in all states of world knowledge, $\mathbf{st}'$, that hold after $\mathbf{st}$:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \Box(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \forall\, \mathbf{st}' \in \mathbf{St}[\mathbf{after}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'}] \tag{2.24}$$

The $\Diamond$ operator can be specified in a similar way. A formula, $\mathbf{w}$, is eventually true in state of world knowledge $\mathbf{st}$ if and only if it is true in a state of world knowledge, $\mathbf{st}'$, which occurs after $\mathbf{st}$:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \Diamond(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \exists\, \mathbf{st}' \in \mathbf{St}[\mathbf{after}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'}] \tag{2.25}$$

These operators can be used to represent temporal requirements without explicitly constructing a temporal model like that built using $<$ in **time_variable_order** (2.19). The model is hidden within the definition of $\Box$ and $\Diamond$. For instance, it might be specified that input $\mathbf{p}$ has the effect, $\mathbf{s}'$, of changing the state, $\mathbf{s}$, of a system so that it is eventually not $\mathbf{s}$. The following assumes that predicates, such as $\mathbf{state}(\mathbf{s})$, have the same semantics as those introduced in Sections 2.3.1 and 2.3.2 except that they can be 'qualified' by temporal operators that specify the interval in which they are true:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{temporal\_logic\_state\_change}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{state}(\mathbf{s}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \Diamond\neg\, \mathbf{state}(\mathbf{s}) \end{aligned} \tag{2.26}$$

Additional temporal operators, such as $\bigcirc$ (read as 'next') and $\mathcal{U}$ (read as 'until'), can be specified by introducing an **immediate_after** relation between states of world knowledge. The following predicate states that $\bigcirc\mathbf{w}$ is true in states of world knowledge, $\mathbf{st}$, if and only if $\mathbf{w}$ is true in the unique state of world knowledge which holds immediately after $\mathbf{st}$. The universal quantifier of $\mathbf{st}$ provides a non-ending model of time; every state of world knowledge is immediately followed by another.

The unique existential quantifier of $\mathbf{st}'$ provides a linear model of time; there is only one immediate successor for each state of world knowledge. The **immediate_after** relation provides a discrete model of time; it specifies a base granularity that cannot be decomposed into shorter intervals:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \bigcirc(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \exists! \mathbf{st}' \in \mathbf{St}[\mathbf{immediate\_after}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'} \quad (2.27)$$

Designers could use the $\bigcirc$ operator to make explicit the sequencing between predicates that hold in successive states of world knowledge. For instance, the sequencing that was implicit in **visible_input_effect** (2.13) might be made explicit. The $\bigcirc$ operator specifies that the effect, $\mathbf{s}'$, of the input, $\mathbf{p}$, is visible through display, $\mathbf{d}'$, in the next state of world knowledge:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{temporal\_visible\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftarrow \exists\, \mathbf{s}' \in \mathbf{S} \\ (\mathbf{input}(\mathbf{p}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \\ \bigcirc (\mathbf{view}(\mathbf{s}', \mathbf{d}') \wedge \neg\, \mathbf{same\_display}(\mathbf{d}, \mathbf{d}'))) \end{aligned} \quad (2.28)$$

The **immediate_after** relation can also be used to specify other temporal operators. A formula $\mathbf{w}$ is true until a formula, $\mathbf{w}_1$, is true in state of world knowledge $\mathbf{st}$ if and only if $\mathbf{w}_1$ is true in state of world knowledge $\mathbf{st}$ or there exists a state of world knowledge $\mathbf{st}'$ which is immediately after $\mathbf{st}$ and $\mathbf{w}$ is true in $\mathbf{st}$ and in $\mathbf{st}'$ it is the case that $\mathbf{w}$ is true until $\mathbf{w}_1$:

$$\begin{aligned} \forall\, \mathbf{st} \in \mathbf{St} \mid \mathbf{w} \mathcal{U} \mathbf{w}_1 \mid_{\mathbf{st}} \Leftrightarrow [(\mid \mathbf{w}_1 \mid_{\mathbf{st}}) \vee \\ (\exists\, \mathbf{st}' \in \mathbf{St}(\mathbf{immediate\_after}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}} \wedge \mid (\mathbf{w} \mathcal{U} \mathbf{w}_1) \mid_{\mathbf{st}'}))] \end{aligned} \quad (2.29)$$

Before continuing with the central argument of this thesis, it is important to note that there is an apparent redundancy in using both temporal operators and state parameters. This point can be illustrated by the following two bi-conditions:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{temporal\_logic\_persistent\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \Box \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \end{aligned} \quad (2.30)$$

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{logic\_persistent\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \end{aligned} \quad (2.31)$$

The first predicate, **temporal_logic_persistent_effect**, specifies that the effect of $\mathbf{p}$ on $\mathbf{s}$ is $\mathbf{s}'$ at all times. The second predicate, **logic_persistent_effect**, specifies that the effect of $\mathbf{p}$ is $\mathbf{s}'$ for all states, $\mathbf{s}$, of a control system. These bi-conditions could, therefore, be interpreted as specifying the same requirement. This is not the case. As mentioned in the previous section, there is an important distinction between the states of world knowledge, elements of $\mathbf{St}$, and control system states, elements of $\mathbf{S}$. The first predicate should be interpreted as specifying that the effect of $\mathbf{p}$ on $\mathbf{s}$ is $\mathbf{s}'$ at all times, irrespective of the state of world knowledge. It is true only if $\mathbf{s}'$ is always the effect of $\mathbf{p}$ on $\mathbf{s}$. The second predicate should be interpreted as specifying that the effect of $\mathbf{p}$ on $\mathbf{s}$ is $\mathbf{s}'$ in the present interval, irrespective of the control system state. It can be true if eventually $\mathbf{s}'$ is not the effect of $\mathbf{p}$ on $\mathbf{s}$.

It is important to emphasise that this thesis does not attempt to develop a new interval temporal logic. Our notation is the same as that presented by Manna and

Pnueli [205]. Appendix A introduces the syntax and semantics of this language. Appendix B provides a brief overview of the axioms and theorems of the formalism. The interested reader is directed to Manna and Pnueli for a more complete introduction. The remainder of this thesis applies interval temporal logic in order to demonstrate that it can support a principled approach to the integration of human factors and systems engineering.

2.4.5 Bounded Effects And Task Allocation

One of the main findings of the inquiry conducted after the Piper Alpha disaster was that North Sea Oil production should come under the supervision of the Health and Safety Executive (HSE), rather than the Department of Energy [81]. A prime motivation for this change in responsibility was that the HSE has well established procedures for certifying the safety of production processes [27]. These procedures force companies to produce a Safety Case which states the objectives that they must fulfill in order to guarantee the safe operation of their production processes. Interval temporal logic provides a notation which designers might exploit for the formal safety assessment required when producing this document. For instance, the Piper Alpha inquiry recommended that systems engineering should safeguard against any operator intervention that could impair intervention by automated safety systems [81]. Designers might exploit such safeguards to ensure that if an operator issues input, $\mathbf{p}$, to disable a deluge system during a fire, $\mathbf{s}$, then the input does not affect its state, $\mathbf{s}$, until the fire is dealt with and a safe state, $\mathbf{s}'$, is regained:

$$\begin{aligned} \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{bounded\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ (\mathbf{input}(\mathbf{p}) \wedge (\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}) \mathcal{U} \mathbf{state}(\mathbf{s}'))) \end{aligned} \tag{2.32}$$

This requirement imposes constraints upon a designer's ability to exploit flexible task allocation. Operators cannot assume control tasks until the system is in state $\mathbf{s}'$ even if their intervention is necessary in order for the system to enter that state. The previous bi-condition also threatens predictability. Operator input, $\mathbf{p}$, could be handled in the next interval, the next next interval or the next next next interval and so on. Users cannot predict when it will take effect because the consequences of their intervention are postponed until a state transition occurs in the control system. Human factors and systems engineers might use predicates, such as **bounded_effect** (2.32), as a basis upon which to assess the costs and benefits of design techniques. Postponing the effects of operator input protects automated systems from human intervention but is likely to jeopardise predictability.

2.4.6 Dialogue Cycles And Display Design

Bainbridge argues that users optimise scarce perceptual and cognitive resources by recognising and responding to patterns of interaction [23]. Dialogue cycles provide a practical application of this observation [134]. They are characterised by:

- the presentation of an entry gate to mark the start of a dialogue;
- the redisplay of the entry gate to mark the termination of the dialogue;
- the effect of a command taking place once the dialogue terminates.

Dialogue cycles provide a means exploiting optimised display design to support flexible task allocation because operators can observe intervention by a system without monitoring every change in state. They might also be used to reduce the unpredictability caused by a **bounded_effect** (2.32); users could exploit cycles to monitor the delayed effects of their intervention. Figure 2.3 provides an illustration of a dialogue cycle. The operator wants to supply some water to deluge equipment.

Figure 2.3: A dialogue cycle

`Diagram a)` shows the image of the system prior to the user's input. `Diagram b)` shows the image of the system after the command has been issued. The appearance of the water inlet, represented by a circle inside a square, has been shaded to inform the user that their input is being handled by the system. `Diagram c)` shows that the command has successfully terminated. In this example the gate of the cycle is the image of the inlet in diagrams `a)` and `c)`. The user can be confident that their command has not been completely effective until the inlet regains its original appearance. This technique could be applied to support flexible task allocation by using the same gate to present the effects of both operator and system intervention. Users might then be confident that the automated systems had not finished interacting with an application until the inlet regained its original appearance.

Logic abstractions provide a means of representing and reasoning about dialogue cycles without considering the level of detail necessary for full implementation. In order to do this a further predicate is introduced:

$$\exists \mathbf{s} \in \mathbf{S}, \exists \mathbf{d}, \mathbf{g} \in \mathbf{D} : \mathbf{gate}(\mathbf{s}, \mathbf{d}, \mathbf{g}) \tag{2.33}$$

This relation can be thought of as a template which designers might place over a display, $\mathbf{d}$, in order to extract those elements, $\mathbf{g}$, which form the gate of a dialogue cycle in state $\mathbf{s}$. Sections 3.4.2 and 4.3.2 will exploit similar templates to represent a number of additional techniques that human factors and systems engineers might use to support predictability. Delays in the effect of operator input are visible through delays in the return of the gate that marks the successful termination of the cycle. In other words, an operator can predict that any command will be ineffective until the dialogue has been completed. Designers could, therefore, specify that a dialogue cycle presents the effect, $\mathbf{s}'$, of input, $\mathbf{p}$, on state $\mathbf{s}$ if and only if the gate is presented when the input is issued and it is not presented again until that effect has been achieved:

$$\begin{aligned} \forall \mathbf{p} \in \mathbf{P}, \forall \mathbf{s} \in \mathbf{S}, \exists \mathbf{s}' \in \mathbf{S}, \exists \mathbf{g} \in \mathbf{D} : \mathbf{cycle}(\mathbf{p}, \mathbf{s}, \mathbf{s}', \mathbf{g}) \Leftrightarrow \exists \mathbf{d}, \mathbf{d}' \in \mathbf{D} \\ (\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{gate}(\mathbf{s}, \mathbf{d}, \mathbf{g}) \wedge \mathbf{display}(\mathbf{g}) \wedge \\ \bigcirc (\neg \mathbf{display}(\mathbf{g}) \mathcal{U} (\mathbf{state}(\mathbf{s}') \wedge \mathbf{gate}(\mathbf{s}', \mathbf{d}', \mathbf{g})))) \end{aligned} \tag{2.34}$$

The success or failure of a cycle is determined by an operator's ability to notice the gate that marks the beginning and end of the dialogue. Inattention or distraction can prevent users from observing such elements of a display. Design principles do not prevent designers from making unwarranted assumptions about the human factors of operator performance. Wardell argues that fatigue is a frequent cause of many of the accidents on oil production platforms [315]. Empirical validation is necessary in order to determine whether operators can exploit a particular dialogue cycle. Chapter 7 will argue that designers could use prototypes to perform this validation.

2.5 Conclusions

Optimised display design can be used to ease the burdens of monitoring dynamic control systems. Flexible task allocation provides both the system and its operator with the means of intervening to re-distribute control tasks as interaction progresses. In order to successfully exploit these techniques, designers must be able to assess their impact upon the human factors and systems engineering of a potential interface. In particular they must be able to assess the consequences of techniques, such as flexible task allocation and optimised display design, for principles, such as predictability.

Production rules provide an intractable medium for representing techniques that are intended to support design principles. The maintenance of facts, rules, meta-rules and meta-meta-rules imposes non-trivial burdens upon the finite resources of human factors and systems engineering. First order logic is unsuitable for this task. It cannot easily be used to represent state changes nor can it easily be used to represent dialogue sequences.

Interval temporal logic avoids the limitations of production rules and first order predicate logic. It provides designers with a convenient means of specifying the objectives of principled design. This notation has been used to represent the objectives that must be satisfied in order to support predictability through bounded effects and dialogue cycles. It is, therefore, concluded that interval temporal logic provides a suitable notation for the integration of human factors and systems engineering.

The following chapters argue that human factors and systems engineers might exploit interval temporal logic and principled design to resolve the two remaining fundamental problems of control system development: openness and complexity.

Chapter 3

Complexity

> "We have seen a tendency for proposed soft displays to become a jungle of clutter, of ill-considered symbols and text, or a dazzling presentation of colours. Little wonder that pilots are now referring to the modern instrument panel as 'Pacman' and 'Ataris'. The attitude of too many computer experts is we can do it, so let's throw it up on the display - it can't hurt anything" (Wiener, [319]).

3.1 Introduction

The dictionary defines 'complexity' in terms of the verb 'to complicate'; "to make or become difficult, confused" [12]. In terms of this thesis, complexity refers to the difficulty and confusion that arise when operators must supervise an application through the many states, displays and commands of an interactive control system. The quotation which opens this chapter argues that complexity also increases the problems of design. This chapter agrees with the assertion and argues that:

- complexity arises because users must monitor large amounts of interconnected application data through the states and displays of interactive control systems;
- complexity arises because operators must select and issue appropriate commands from many options for intervention;
- principles provide designers with high-level objectives for the techniques that they must employ in order to support these control tasks.

It is concluded that human factors and systems engineering must be integrated in order to achieve principles, such as predictability, in the design of complex control systems.

3.1.1 Consequences: Poor Performance And Unpredictability

Human factors research has argued that the increasing complexity of control applications leads to degraded operator performance. For instance, Henneman and Rouse show that the simultaneous presentation of large amounts of information reduces the ability of users to respond to changes in the state of a process [145].

Moray, Lootstein and Pajak argue that an operator's ability to predict and diagnose process errors is impaired if the number of subsystems which they must control is increased [214]. Rasmussen and Lind argue that complexity hinders the development of expectations or predictions about the effect of interaction [249].

3.1.2 Causes: Command-View, Display And State Correspondence

Designers frequently require operators to consult different sources of information before interacting with a control system. This can lead to command-view correspondence if many of these sources provide data about the same command. Kirkpatrick and Mallory argue that substitution errors are more likely to occur if reactor operators have to reference a number of displays in order to identify their options for intervention [176]. Such errors are violations of predictability; a user issues one command believing that they have issued another whose effects can be quite different. Command-view correspondence jeopardises safety. For instance, the operators of the Unit Two reactor at Three Mile Island were forced to consult many volumes of special operating procedures. This delayed their response and the reactor entered an unstable state for which they had not been trained. As a consequence they made inaccurate predictions about the effects of the commands which they eventually issued [292].

The task of operating an application process is complicated by display correspondence. This occurs if a display presents identical information about two different states of a control system. This cause of complexity jeopardises safety because users cannot distinguish between some control system states from the information presented to them. For instance, the operators of the Unit Four reactor at Chernobyl could not distinguish between states in which their coolant pumps were operating efficiently and those in which they had insufficient water for heat transfer from the reactor core. During the accident they could not accurately view the state of their control system and failed to predict that their commands would damage the coolant pumps [242].

Complexity is caused by state correspondence. This occurs when a single control system state corresponds to two or more different states of an application. For instance, the state of the Unit Two reactor control system at Three Mile Island provided insufficient information for operators to determine whether or not the integrity of their containment was threatened [292]. The problems of avoiding state correspondence are illustrated by statistics that described the Unit Four reactor at Chernobyl [331]. The primary coolant circuit consisted of two parallel loops. Each loop connected two carbon steel steam-drums. Each drum had four hundred and ninety-four heat exchangers, known as risers and down-comers. Both drums were connected to a common pump inlet. The inlet in each loop had four electrically driven pumps, one of which was held in reserve. The Unit Four reactor also had an intermediate cooling system of forty-four distribution headers. Such applications pose a considerable challenge for systems engineering. Sensors must be deployed so that different states of an application process do not correspond to the same state of the control system. This is a prerequisite if a control system is, in turn, to present sufficient information for its operators to determine the state of their application.

3.1.3 Solution: Abstract Design Principles

Figure 3.1 illustrates the way in which different layers of correspondence separate an operator from their application. At the top level a user issues commands, `c`.

Figure 3.1: The layers of correspondence

Command-view correspondence occurs if an operator must consult displays, `d`, `d'` and `d''`, presented by different sources in order to inform their selection of commands. The lines showing the observation of display `d'` is omitted to simplify the diagram. Display correspondence occurs if a user must also detect different control system states, `s`, `s'` and `s''`, through a display, `d'`. State correspondence occurs if many different states of an application, `ps`, `ps'`, `ps''` are represented by a control system state, `s'`.

Complexity arises because operators must determine the most appropriate command to issue from many different sources of information. They must accurately determine the state of a control system through displays which represent many different states. They must determine the state of their application process from the many process states which are represented by each state of their control system. There is no panacea for these causes of complexity. It is, therefore, important that human factors and systems engineers can represent and reason about the strengths and the weakness of potential solutions. Part III will argue that designers can exploit interval temporal logic to capture the detailed trade-offs which must be assessed prior to the implementation of a particular control system. In contrast, the following

sections argue that interval temporal logic might be used to assess techniques for countering complexity without representing the thousands of sensor readings and command options that will be available in an implementation. Principles provide human factors and systems engineers with high-level objectives that must not be sacrificed by these solutions to state, display and command-view correspondence.

3.2 State Correspondence

Systems engineering must ensure that a control system can detect the state of a production process. It is seldom feasible to deploy sensors to monitor every aspect of an application. Economic constraints limit the use of expensive sensing devices, technological limitations prevent engineers from directly monitoring very high temperatures, pressures and radiation levels [36]. This complicates the task of controlling an application because operators cannot differentiate between all states of a process from the information available to their control system. In order for a designer to reason about solutions to this state correspondence it is important that its causes should not be obscured by the thousands of low-level sensor readings which can be received by a control system. The predicates, introduced in Sections 2.3.1 and 2.4.4, can be used to support an abstract analysis that does not explicitly represent these details. Chapter 4 will argue that designers could use interval temporal logic to represent and reason about concurrent interaction between multiple operators and processes. In contrast, the argument that is presented in this chapter is simplified by analysing complexity in terms of a single operator controlling a single application process. In order to do this we introduce a set, **Ps**, of application states. A relation is introduced between control system states and the application states that they represent in the present interval:

$$\forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{ps} \in \mathbf{Ps} : \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \tag{3.1}$$

Designers can also introduce a predicate that is true for elements of **Ps** which are considered to be identical in the present interval. It is important to realise that two process states, **ps** and $\mathbf{ps}'$, could be regarded as the same even though they differ in particular details, such as a one degree difference in temperature. The following pages will demonstrate that designers must take extreme care when determining which elements of **Ps** can be regarded as the same:

$$\exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}') \tag{3.2}$$

Using the previous relation it is possible to formalise the conditions that lead to state correspondence. This eventually arises when different control system states, **ps** and $\mathbf{ps}'$, are represented by the same control system state, **s**:

$$\begin{aligned} \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Leftrightarrow \\ \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \wedge \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \wedge \\ \neg\, \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}') \end{aligned} \tag{3.3}$$

For instance, the operators of the Unit Two reactor at Three Mile Island could not establish from the state, **s**, of their control system whether the containment was damaged by an increase in gas pressure, **ps**, or whether it was intact, $\mathbf{ps}'$. This

lack of information forced operators to use robots to inspect contaminated areas of the plant [292]. At Chernobyl, when similar robots failed they were forced to use volunteers from the Red Army [331].

3.2.1 State Correspondence And Predictability

Thimbleby observes that if a user does not have sufficient reason to doubt that things are different, when solving a problem, then they will treat them as if they were the same [304]. This application of Pólya's concept of non-sufficient reason accurately characterises the way in which users operate interactive control systems [238]. If they cannot detect differences between process states then they will treat them as if they were the same. For example, the Watt Committee's investigation into the Chernobyl accident reports that shortly after one o'clock in the morning the operators believed that they had stabilised the reactor. It was "in fact, in an extremely unstable condition" [9]. Not realising this the operators continued with the test that was eventually to release one hundred times the normal power output of the plant. There were insufficient reasons for them to distinguish between reactor states based upon evidence provided by the state of their control system. Users eventually issued input, $\mathbf{p}$, to take their control system from normal operation, $\mathbf{s}$, into a state, $\mathbf{s}'$, in which they could conduct their tests. The operators could not determine whether the reactor was stable, $\mathbf{ps}$, or unstable, $\mathbf{ps}'$, from the new state of the system, $\mathbf{s}'$. In consequence, they failed to predict the consequences of their actions during the test:

$$\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{unpredictable}(\mathbf{p}, \mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Leftarrow \exists\, \mathbf{s}' \in \mathbf{S} \\ \Diamond(\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{state\_correspondence}(\mathbf{s}', \mathbf{ps}, \mathbf{ps}')) \quad (3.4)$$

It is important to note that the previous implication has a temporal quantification. The $\Diamond$ operator specifies that predictability can eventually be violated at any point during interaction. During the Three Mile Island accident it became increasingly difficult for operators to predict the impact of their commands because sensor failure prevented the control system from detecting the true state of the reactor [292].

3.2.2 Resolving State Correspondence

Interval temporal logic has been applied to represent the way in which state correspondence jeopardises predictability. Designers might also use this formalism to represent potential solutions for this cause of complexity.

State Transparency

State correspondence is avoided if equipment failure does not eventually prevent a control system from distinguishing between the states of an application process. In other words, the state of a control system should provide a transparent view of the state of an application. Different process states, $\mathbf{ps}$ and $\mathbf{ps}'$, are never represented by the same state, $\mathbf{s}$, of a control system. For instance, designers might avoid the problems of state correspondence by ensuring that stable, $\mathbf{ps}$, and unstable, $\mathbf{ps}'$, states of a reactor are never represented by a single control system state, $\mathbf{s}$.

Human factors and systems engineers can use interval temporal logic to represent this requirement:

$$\begin{aligned}\forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{no\_state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Leftrightarrow \\ \Box\neg\, (\mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \wedge \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \wedge \\ \neg\, \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}'))\end{aligned} \tag{3.5}$$

Appendix B introduces a number of theorems and axioms that hold for the interval temporal logic used in this thesis. Designers can apply these to re-write predicates. This might clarify the requirements which must be satisfied by potential implementations. For example, the theorems and axioms of interval temporal logic can be applied to **no_state_correspondence** (3.5), see Appendix C.1, to derive a requirement that it is always the case that if process states, **ps** and $\mathbf{ps}'$, are represented by a control system state, **s**, then those process states are identical:

$$\begin{aligned}\forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{no\_state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Leftrightarrow \\ \Box(\mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \wedge \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \Rightarrow \\ \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}'))\end{aligned} \tag{3.6}$$

A number of practical limitations restrict the utility of this approach. In particular, it is not always possible to determine which application states will be represented by particular control system states. The United Kingdom's Health and Safety Executive recognise that it is impossible for designers to identify all the possible errors that can arise in nuclear installations [138]. Human factors and systems engineers cannot, therefore, enumerate all of the possible application states that might be represented by a control system state. This prevents them from ensuring that the previous bi-condition will always hold for a particular implementation. The limitations of this approach are illustrated by Poteralski and Vogel's observation that an atmospheric emission of iodine 131 from a light water nuclear reactor is likely to occur once every ten million years [241]. Over 30,000,000 curies were released from this form of reactor during the Chernobyl accident.

State Restriction

State restriction prevents an application process from entering a state that cannot be identified given the evidence provided by the state of its control system. For instance, human factors and system engineers could require that it is always the case that if the state, **s**, of a control system does not distinguish between a loss of coolant accident, $\mathbf{ps}'$, and normal operating conditions, **ps**, then designers must ensure that a loss of coolant does not occur. It is important to note that the success of this approach relies upon designers making an appropriate selection of process states, $\mathbf{ps}'$, that are to be restricted. In practice, this is unlikely to be as straightforward as the decision to avoid a loss of coolant. Chapter 7 will present techniques that human factors and systems engineers might use to inform such choices for particular control systems. For now it is sufficient to realise that interval temporal logic provides a means of representing the high level requirements that are imposed by such techniques:

$$\exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{state\_restriction}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Leftrightarrow$$

$$\Box(\mathbf{state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \Rightarrow \mathbf{process\_state}(\mathbf{ps}) \wedge \neg\, \mathbf{process\_state}(\mathbf{ps}')) \quad (3.7)$$

In the above, **process_state**(**ps**) is true for an element of the set of process states that is the actual state of an application in the present interval:

$$\exists\, \mathbf{ps} \in \mathbf{Ps} : \mathbf{process\_state}(\mathbf{ps}) \quad (3.8)$$

Predicates, such as **state_restriction** (3.7), can be imposed as requirements that must be satisfied in order to achieve principles, such as predictability. For instance, the systems engineering of the Chernobyl Unit Four reactor might have helped operators to determine the effects of their intervention if it had been prevented from entering an unstable state; $\neg$ **process_state**(**ps**′). There were few inter-locks to prevent the control rods, which damp a nuclear reaction, from being withdrawn [331]. Such mechanisms were not provided because of the limited availability of specialist electrical equipment in the former Soviet Union. Instead, the designers of the Unit Four reactor relied upon the human factors techniques of social ergonomics to ensure the safety of their system. Management hierarchies were established and operating procedures were drafted to ensure that the application did not enter an unstable state; $\neg$ **process_state**(**ps**′). Such techniques were favoured as low cost alternatives to the application of more advanced systems engineering. Unfortunately, the Power Ministry's strictures were insufficient to prevent operators from withdrawing the control rods too far. Human factors techniques must, therefore, be supported by systems engineering if state restriction is to guarantee the success and safety of production. This analysis is confirmed by the observation that two of the four Finnish nuclear reactors were constructed by Russian engineers, to a Soviet design. They are, however, fitted with Finnish safety features. These reactors have exemplary safety records. Improved systems engineering has helped to place them amongst the most efficient reactors in the world [178].

State Focusing

Principles, such as predictability, provide a justification for concentrating upon particular aspects of complexity, such as state correspondence. Interval temporal logic provides abstractions that can be used to specify design objectives without identifying the mechanisms which are to achieve them. This is intended to encourage designers to assess more fully the trade-offs which must be made when exploiting human factors and systems engineering. For instance, the low initial cost of using human factors techniques to enforce state restriction must be balanced against the high cost of potential disasters. Conversely, the high initial cost of systems engineering techniques must be balanced against the reduced likelihood of errors through state correspondence. A further benefit of this level of analysis is that designers might be encouraged to seek a number of alternative techniques that can be used to avoid the causes of complexity. For instance, state focusing requires that state correspondence does not impair operator intervention. This approach has been embodied within decision support tools, such as the nuclear Disturbance Analysis and Surveillance System [5]. These are intended to provide as much information as possible about those states of an application which require operator intervention. The intention is to reduce any ambiguity about the state of an application prior to a

command being issued; $\neg$ **state_correspondence**$(\mathbf{s}, \mathbf{ps}, \mathbf{ps}')$. State focusing might also be used to support predictability after a command has been issued. Operators must be able to establish that their intervention has been successful if they are to predict the continued success and safety of their application. Decision support tools should, therefore, provide as much information as possible about the state of an application after operator intervention. This would reduce any ambiguity about the effects of input; $\neg$ **state_correspondence**$(\mathbf{s}', \mathbf{ps}, \mathbf{ps}')$. Interval temporal logic can be used to represent the requirements that must be satisfied in order to exploit state focusing:

$$\begin{aligned}\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S}, \forall\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{state\_focusing}(\mathbf{p}, \mathbf{s}, \mathbf{s}', \mathbf{ps}, \mathbf{ps}') \Leftrightarrow \\ \Box(\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \Rightarrow \\ \neg\, \mathbf{state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}') \wedge \\ \neg\, \mathbf{state\_correspondence}(\mathbf{s}', \mathbf{ps}, \mathbf{ps}'))\end{aligned} \tag{3.9}$$

This predicate illustrates how interval temporal logic could be used to represent objectives that can only be achieved through the integration of human factors and systems engineering. Systems engineers must deploy sufficient sensing devices so that a control system can detect and represent an operator's impact upon an application. Human factors engineers must ensure that users can deploy their cognitive and perceptual resources in order to deduce the state of an application from the state of their control system.

3.3 Display Correspondence

Display correspondence complicates the task of operating application processes if users cannot observe every state of their control system from the information presented to them. If operators cannot determine the state of their control system then they are unlikely to be able to identify the state of an underlying application, even if designers can resolve the problems of state correspondence. Human factors and systems engineers might exploit interval temporal logic to represent this cause of complexity. Display correspondence occurs if and only if different states of a control system, $\mathbf{s}$ and $\mathbf{s}'$, are represented by the same display, $\mathbf{d}$ in the present interval:

$$\begin{aligned}\exists\, \mathbf{d} \in \mathbf{D}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d}) \wedge \neg\, \mathbf{same\_state}(\mathbf{s}, \mathbf{s}')\end{aligned} \tag{3.10}$$

For example, the International Atomic Energy Agency (IAEA) concluded that the alarm systems of the Chernobyl Unit Four reactor were inadequate [331]. Operators were not presented with warnings once the control system had detected a rapid increase in the number of coolant voids. These occur when water within the reactor coolant cycle turns to steam. They greatly increase the power of a reaction. In other words, the same information, $\mathbf{d}$, was displayed for a control system in a state, $\mathbf{s}$, which detected a normal coolant system and a state, $\mathbf{s}'$, in which a dangerous number of voids were forming. As a result of accidents, such as those at Chernobyl and Sosnovy Bor, the production of all new Reactor Bolshoi Moschnosti Kipyashiy reactors has been halted. Those still in operation are to be fitted with additional warning systems. The IAEA argue that displays must be designed so that operators

can accurately diagnose the state of their control system and, through it, the states of an underlying application [157].

3.3.1 Display Correspondence And Predictability

Bainbridge argues that operators must be able to determine their current situation in order to predict the course of future interaction [20]. In other words, predictability is sacrificed if users cannot view the consequences of their interaction. Norman confirms this analysis when he criticises the "inappropriate feedback" provided by many interactive control systems [228]. This prevents operators from determining whether their intervention has moved the control system towards an intended state. Human factors and systems engineers might use interval temporal logic to establish the relationship between such observations and the problem of display correspondence. Inappropriate feedback is provided and predictability is threatened if operators eventually cannot observe the effect, $\mathbf{s}'$, of their input, $\mathbf{p}$, through a change in the display, $\mathbf{d}$:

$$\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{d} \in \mathbf{D}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{unpredictable}(\mathbf{p}, \mathbf{d}, \mathbf{s}, \mathbf{s}') \Leftarrow \\ \Diamond(\mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}')) \quad (3.11)$$

For instance, during the Three Mile Island accident some areas of the reactor reached over two thousand degrees Fahrenheit. Control system states in which the maximum reading was eight hundred degrees and states in which the maximum was two thousand degrees were presented by the same display image [292]. Users were forced to take a large number of manual core readings to clarify the results presented by computer-based monitoring systems. Operators could not use the control system display, $\mathbf{d}$, to determine whether commands, $\mathbf{p}$, to reduce the core temperature, $\mathbf{s}$, were having the predicted effect, $\mathbf{s}'$.

3.3.2 Resolving Display Correspondence

Section 2.3.3 contained an initial discussion of techniques that human factors and systems engineers could exploit to present the effects of operator input. It was argued that different displays must be used to present the context and consequences of commands. The following sections extend this analysis.

Display Transparency

Display correspondence can be avoided by requiring that displays never provide views of different control system states:

$$\forall\, \mathbf{d} \in \mathbf{D}, \forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{no\_display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \Box\neg\, (\mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d}) \wedge \neg\, \mathbf{same\_state}(\mathbf{s}, \mathbf{s}')) \quad (3.12)$$

Designers might use the theorems and axioms of interval temporal logic, see Appendix C.2, to clarify the requirements that this bi-condition imposes upon potential implementations. It is always the case that if states of a control system, $\mathbf{s}$ and $\mathbf{s}'$, are represented by the same display then those states are identical. This is termed

display transparency because different states of a control system are not hidden behind a display:

$$\forall \mathbf{d} \in \mathbf{D}, \forall \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{no\_display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Leftrightarrow$$
$$\Box(\mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d}) \Rightarrow \mathbf{same\_state}(\mathbf{s}, \mathbf{s}')) \quad (3.13)$$

For instance, if the clinical or therapeutic exposure to radioactive materials exceeds the prescribed dosage by more than ten percent then any United States' Nuclear Regulatory Commission (NRC) licensee is guilty of "maladministration" [281]. There were four hundred and twenty-three reported cases of therapeutic and diagnostic maladministrations in the United States between 1981 and 1987. The NRC is, therefore, concerned that control systems enable their operators to monitor and observe as much information as possible about the dosage administered to their patients. If a display, $\mathbf{d}$, is presented for states, $\mathbf{s}$ and $\mathbf{s}'$, of a clinical therapy system then those states can be regarded as identical. The consequences of violating this requirement are illustrated by a recent case in which over one thousand North Staffordshire cancer patients were administered radiotherapy at thirty percent below the intended level [17]. This occurred because the system programmer made incorrect calculations about the dosage level that was administered by the radioactive source. In consequence, the same display, $\mathbf{d}$, was presented for very different states. Clinical staff could not detect the thirty percent difference between the actual dosage level, $\mathbf{s}$, and the intended level, $\mathbf{s}'$, from the information that was presented to them. The implementation details that are necessary in order to achieve **no_display_correspondence** (3.13) in a particular interface vary from application to application. The states of an external teletherapy control system are not the same as those of a control system for internal brachytherapy. The use of logic abstractions, such as $\mathbf{p}$, $\mathbf{s}$ and $\mathbf{d}$, provides designers with common design aims for many different applications. This is a significant advantage. A criticism of NRC guidelines for the design of nuclear control systems is that they are too specifically aimed at power generation and cannot easily be applied to support the development of medical applications [312].

Display Restriction

Display transparency is impracticable for many applications. Systems engineering can deploy sensors to collect large amounts of information about application processes. In order to present different displays for different states of a control system, designers would have to present much of this detail to system operators. This is impossible given the constraints of control room layout; there might not be enough room to provide sufficient monitors. There are also human factors constraints. An operator's ability to detect changes in process information, typically, decreases if designers increase the amount of data presented [318]. Alternatively, human factors and systems engineers could exploit display restriction to combat the problem of complexity. This requires that it is always the case that if different control system states, $\mathbf{s}$ and $\mathbf{s}'$, are presented by a display, $\mathbf{d}$, then an operator cannot cause a transition between those states. In other words, users are restricted to commands which have an effect that can be viewed through a change in the display:

$$\forall \mathbf{d} \in \mathbf{D}, \forall \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{display\_restriction}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Leftrightarrow$$

$$\Box(\mathbf{display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Rightarrow \neg\, \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \quad (3.14)$$

For example, the problems of presenting operators with information about radiotherapy have led to the development of automated dosage control systems [281]. The patient's treatment planning system encodes their requirements onto a magnetic card which is inserted into the therapy control system. The operator does not directly control the exposure, $\neg\, \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}')$, because they cannot monitor all of the subtle changes in dosage levels, $\mathbf{s}$ and $\mathbf{s}'$. This illustrates how systems engineering can be recruited to reduce the unpredictability caused by display correspondence. The same requirement can also be satisfied by human factors engineering. Operators can be trained not to interrupt clinical treatments if they cannot detect a change in the state of their control system. In other words, if users cannot determine the consequences of their interaction then they should do nothing.

Display Focusing

Serig argues that heavy workloads on hospital staff make it dangerous to rely upon human factors techniques as a means of achieving display restriction [281]. Tired operators make mistakes. Similarly, the automation of systems engineering does not entirely avoid the problems of display correspondence. Equipment failure can force operators to intervene given limited information about the state of their control system. Display focusing provides a further alternative to transparency and restriction. This requires that users can eventually view sufficient information in order to determine the state of their system. For instance, British Nuclear Fuels recently undertook a survey of human factors requirements for their Thermal Oxide Reprocessing Plant at Sellafield. This project identified techniques that could enable operators to recover from human or system failures. British Nuclear Fuels concluded that "the results tended to be the identification of requirements for additional information displays" [177]. These displays might provide access to a database. Operators could use this to eventually gain detailed data about the state of their control system. In other words, if operators cannot distinguish between states, $\mathbf{s}$ and $\mathbf{s}'$, of a system from a display, $\mathbf{d}$, then eventually additional displays, $\mathbf{d}'$, are available so that they can differentiate between them:

$$\begin{aligned} &\forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S}, \forall\, \mathbf{d} \in \mathbf{D}, \exists\, \mathbf{d}' \in \mathbf{D} : \mathbf{display\_focusing}(\mathbf{s}, \mathbf{s}', \mathbf{d}, \mathbf{d}') \Leftrightarrow \\ &\quad \mathbf{display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}') \Rightarrow \\ &\quad \Diamond(\mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d}') \wedge \neg\, \mathbf{same\_display}(\mathbf{d}, \mathbf{d}')) \end{aligned} \quad (3.15)$$

It is important to note that there is likely to be a delay before operators can view different states, $\mathbf{s}$ and $\mathbf{s}'$, through different displays, $\mathbf{d}$ and $\mathbf{d}'$. This is captured by the $\Diamond$ operator in the previous bi-condition. If this interval is prolonged then the state of a control system could change before an operator can determine whether it is $\mathbf{s}$ or $\mathbf{s}'$. Section 8.3.1 will argue that real-time operators can be introduced into interval interval temporal logic in order to specify time limits upon such delays. For now it is sufficient to observe that interval temporal logic predicates provide a basis upon which to assess the strengths and weaknesses of techniques, such as display transparency, restriction and focusing, for the human factors and systems engineering of predictable control systems. The utility of display transparency is limited by the human factors problems of observing many different items of information. It is also

limited by the constraints of systems engineering; there might not be enough space in a control room to provide sufficient monitors. The utility of display restriction is limited by the human factors constraints of human error; operators might interact with a system even though they cannot determine the consequences of their intervention. It is also limited by the systems engineering constraints of equipment failure; under certain circumstances operators must intervene with only limited information about the state of their control system. Display focusing provides an alternative that exploits both human factors and systems engineering. Systems engineering must provide a database of additional information; operators can eventually access this data in order to determine the state of their control system. Human factors engineering must determine whether users are likely to perceive and understand this information before state changes threaten safe and successful operation.

3.4 Command-View Correspondence

Control systems increasingly provide their users with many different commands. This leads to complexity because designers cannot, typically, present information about all possible options for intervention on a single display device. Presenting multiple sources of information about a command has profound consequences for the usability of interactive control systems [145]. Operators must not only remember which input sequences must be issued in order to invoke a command, they must also recall which sources are used to present information about that command. Users frequently have neither the time nor the opportunity to monitor primary and back-up displays under the pressures imposed by many working environments. It is important that human factors and systems engineers can reason about this cause of complexity without being forced to represent all the commands offered by a potential implementation. If a nuclear power control system were to be analysed at this level then its designers would have to consider the thousands of commands used to control the reactor, outgoing power-grid connections, boilers, turbo-generators and station electrical supplies. In contrast, human factors and systems engineers could recruit interval temporal logic to represent the causes of command-view correspondence without considering such details early in development. A predicate might be introduced that is true for a display which contains information about a command that can be issued to a control system in a particular state in the present interval. For instance, a display, $\mathbf{d}$, presenting data about boilers which are at full pressure, $\mathbf{s}$, could include information about a command, $\mathbf{c}$, to reduce that pressure:

$$\exists\, \mathbf{c} \in \mathbf{C}, \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{command\_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}) \tag{3.16}$$

A relation is also introduced between displays that are presented by the same presentation device in the present interval. For instance, a cathode ray tube (CRT) monitor might be a source of information, $\mathbf{d}$, about generator boilers, as well as data, $\mathbf{d}'$, about turbo-generators:

$$\exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{same\_source}(\mathbf{d}, \mathbf{d}') \tag{3.17}$$

It is possible to formalise the conditions that lead to command-view correspondence. This occurs if different sources present information about the same command in the

present interval:

$$\exists\, \mathbf{c} \in \mathbf{C}, \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{command\_view\_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \mathbf{command\_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}) \wedge \mathbf{command\_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}') \wedge \neg\, \mathbf{same\_source}(\mathbf{d}, \mathbf{d}') \quad (3.18)$$

For instance, the operators of the Chernobyl Unit Four reactor could access information about a command, **c**, to manually introduce neutron-absorbing control rods from their control panel, **d**. Written documentation, $\mathbf{d}'$, was also provided to explain the intended use of this command. Operators were required to consult these different sources in order to determine whether they should select this option for intervention. The difficulty of accessing this dispersed information prevented them from identifying the likelihood of a positive scram being caused by introducing the control rods. They failed to realise that their command would increase, rather than decrease, radioactivity once the rods had been withdrawn beyond a certain point. Information about the precise contents of the Chernobyl Unit Four displays and documentation has not been published [331]. It is known that the Soviet nuclear industry was aware of the risk of positive scrams [242]. It is, therefore, possible to speculate that such information could have been provided in paper-based documentation to system operators. What is certain is that the presentation of different sources of information about options for intervention delayed an effective response to the accident.

3.4.1 Command-View Correspondence And Predictability

Command-view correspondence jeopardises predictability. Data remembered from one display can be forgotten in the process of accessing another. The likelihood of this occuring increases as the amount of time available to system operators decreases [318]. Users, therefore, eventually make inappropriate assumptions about the effects of a command, **c**, if they must act on the basis of information, **d** and $\mathbf{d}'$, gleaned from different sources:

$$\exists\, \mathbf{c} \in \mathbf{C}, \exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{unpredictable}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftarrow \Diamond \mathbf{command\_view\_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \quad (3.19)$$

For instance, the main control panel of Unit Two at Three Mile Island was over forty feet long. Despite the physical size of this presentation resource, it did not provide all the information, **d**, that operators were expected to access before issuing a command, **c**. Valuable time was wasted while users scanned volumes of paper documentation to find the Emergency Procedures, $\mathbf{d}'$, that described the effects their actions would have upon the state of the reactor. Ainsworth notes that "one of the primary problems which was identified in the Three Mile Island incident was the problem of (users) identifying the appropriate procedure" [5]. The problems created by command-view correspondence were exacerbated by the lack of integration between systems and human factors engineering. Kirkpatrick and Mallory argue that this lack of integration prevented operators from predicting the effects of their commands upon the Unit Two reactor [176]. Systems engineers developed the format and selected the content of the information provided to operators. Control room displays were so cluttered that up to twenty-six percent of the meters could not be read by users during routine tasks [203]. Written documentation and control room displays

used different classification schemes to group command information. The difficulties that this created were compounded by the lack of any consistent page numbering scheme in the paper-based documentation [5]. These problems could have been avoided through the integration of human factors and systems engineering.

3.4.2 Resolving Command-View Correspondence

Interval temporal logic provides a means of representing solutions for command-view correspondence in a form that can be used to guide the design of interfaces to many different control systems.

Command-View Transparency

Command-view correspondence is avoided if and only if it is never the case that different sources display information, $\mathbf{d}$ and $\mathbf{d}'$, about a command, $\mathbf{c}$:

$$\begin{aligned}
&\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \\
&\quad \mathbf{no_command_view_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \\
&\quad\ \Box\neg\, (\mathbf{command_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}) \wedge \mathbf{command_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}') \wedge \\
&\quad\ \neg\, \mathbf{same_source}(\mathbf{d}, \mathbf{d}'))
\end{aligned} \tag{3.20}$$

This is termed command-view transparency because human factors and systems engineers must ensure that information about users' options for intervention is not eventually hidden by dispersing it amongst several different sources. The use of abstractions, such as $\mathbf{d}$ and $\mathbf{d}'$, enables designers to represent techniques that can be used to guide the development of many different control systems. The previous predicate might be used to guide the development of paper based documentation, mimic boards and CRT monitors. The importance of this flexibility cannot be over-emphasised. Whitfield reflects the attitude of the United Kingdom's Nuclear Installations Inspectorate (NII) when he argues that control systems will continue to provide computer generated displays, paper documentation and hard-wired instrumentation [317]. Designers could re-write **no_command_view_correspondence** (3.20), see Appendix C.3, to clarify the requirements imposed by this bi-condition. It is always the case that if a command, $\mathbf{c}$, is viewed through displays, $\mathbf{d}$ and $\mathbf{d}'$, then those displays are presented by the same source:

$$\begin{aligned}
&\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \\
&\quad \mathbf{no_command_view_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \\
&\quad\ \Box(\mathbf{command_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}) \wedge \mathbf{command_view}(\mathbf{c}, \mathbf{s}, \mathbf{d}') \Rightarrow \\
&\quad\ \mathbf{same_source}(\mathbf{d}, \mathbf{d}'))
\end{aligned} \tag{3.21}$$

A number of practical problems restrict the application of command-view transparency. For instance, the NII is extremely reluctant to use a single source to present all information about any aspect of a control system [322]. If this primary source fails then operators will be deprived of all the data presented about some command options. Under such circumstances back-up sources of information must be provided. This re-iterates the point, first made in Section 1.5.3, that design principles must not be viewed as axioms but as constraints that should only be violated

if designers understand and accept the consequences. A consequence of distributing command information between primary and back-up sources is that operators might fail to access sufficient information for them to predict the effects of their interaction.

Command-View Restriction

Designers can support predictability by ensuring that if command data is available through primary and back-up displays then it is not presented by both sources at the same time. This is termed command-view restriction because an operator's access to command information is limited; they are not simultaneously presented with competing sources of data. For instance, a number of different bodies governed the operation of nuclear reactors in the former Soviet Union [279]. The State Committee on the Supervision of Safe Operation in Industry and Mining issued objectives in terms of output levels. The State Engineering Inspectorate and the State Committee on Standards regulated reactor operating procedures. The State Nuclear Safety Inspectorate issued safety objectives for the generation process. The State Sanitary Inspectorate issued safety objectives for the handling of generation by-products. If the regulations issued by all of these bodies had been distributed through primary and back-up displays then operators would have been hard-pressed to access the documentation that they were required to consult before issuing their commands. Fortunately, designers always ensured that operators did not have to monitor simultaneously many different sources of data about control regulations:

$$\begin{aligned}\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{command\_restriction}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftrightarrow \\ \Box(\mathbf{command\_view\_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Rightarrow \\ \mathbf{display}(\mathbf{d}) \wedge \neg\, \mathbf{display}(\mathbf{d}'))\end{aligned} \tag{3.22}$$

In order to exploit command-view restriction human factors and systems engineers must determine which displays are to be presented and which displays are not. Predicates, such as **command_restriction** (3.22), provide a framework that can be used to support this task. Abstractions, such as $\mathbf{d}$ and $\mathbf{d}'$, might be instantiated to capture the details of a particular control system display. For example, if the system is transferring fission products, $\mathbf{s}$, then users are presented with information about regulations, $\mathbf{d}$, governing commands, $\mathbf{c}$, to move spent fuel [31]. The State Committee on the Supervision of Safe Operation in Industry and Mining's data about the output levels of those commands might not be presented, $\neg\, \mathbf{display}(\mathbf{d}')$. An important limitation of this approach is that control tasks frequently require the simultaneous presentation of many different items of information. Under such circumstances, designers might not be able to display all necessary command information without using different sources.

Command-View Focusing

Command-view focusing concentrates the presentation of data in order to support an operator's control tasks. This approach is in accordance with the design objectives recommended by regulatory and governmental authorities. The International Atomic Energy Agency recommends that displays must be focussed to provide "optimal support" for control tasks [157]. The Commission of the European Communities

stipulates that sufficient information must be presented so that operators can select appropriate commands for all necessary control tasks [76]. Pragmatically, this raises the question of how to identify an operator's current task. Designers could request that users explicitly inform the system of their current objectives. This approach is appropriate for well-defined tasks, such as those specified in the United States' Nuclear Regulatory Commission's Emergency Procedures [5]. It is inappropriate for more opportunistic interaction. Advanced pattern recognition techniques, such as those implemented using neural networks, offer the possibility that future control systems could infer an operator's current task from a trace of their previous interaction [100]. Designers might exploit interval temporal logic to represent the requirements that must be satisfied in order to achieve command-view focusing through these techniques. This formalisation could exploit Harrison, Roast and Wright's task templates [134]. These can be thought of as masks which might be placed over a display in order to hide any information that is irrelevant for an operator's current activity. They are similar to the templates, introduced in Section 2.4.6, that designers could use to extract the gates of a dialogue cycle. The **template** relation is introduced between an operator task, **ta**, and a display, **d**, that provides information about a command, **c**, which is relevant to that task in the present interval. The elements of the set **Ta** include all operator tasks:

$$\exists\, \mathbf{ta} \in \mathbf{Ta}, \exists\, \mathbf{c} \in \mathbf{C}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{template}(\mathbf{ta}, \mathbf{c}, \mathbf{d}) \tag{3.23}$$

Command-view focusing relaxes some of the constraints imposed by command-view restriction; multiple sources of information can be presented if they are necessary for an operator's control task. It is always the case that if different sources can be used to display information, **d** and $\mathbf{d}'$, about a command, **c**, and $\mathbf{d}'$ is irrelevant to an operator's current task, **ta**, then $\mathbf{d}'$ is not presented and **d** is displayed:

$$\begin{aligned}\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d}, \mathbf{d}' \in \mathbf{D}, \exists\, \mathbf{ta} \in \mathbf{Ta} : \mathbf{command\_focusing}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}', \mathbf{ta}) \Leftrightarrow \\ \Box(\mathbf{command\_view\_correspondence}(\mathbf{c}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \wedge \mathbf{template}(\mathbf{ta}, \mathbf{c}, \mathbf{d}) \wedge \\ \neg\, \mathbf{template}(\mathbf{ta}, \mathbf{c}, \mathbf{d}') \Rightarrow \mathbf{display}(\mathbf{d}) \wedge \neg\, \mathbf{display}(\mathbf{d}'))\end{aligned} \tag{3.24}$$

A number of limitations restrict the utility of command-view focusing. For instance, designers frequently make inappropriate assumptions about the information that users require in order to perform particular tasks. Operators have glued beer-keg handles and cardboard labels to control panels in order to remind themselves of possible input options [280]. A further problem is that designers frequently cannot prevent the presentation of buttons and switches that are irrelevant to an operator's task. Paper documentation cannot easily be removed from a control room every time an operator's task changes. These caveats lead to two important observations about the application of command-view focusing. Firstly, human factors engineers must recruit the support of systems engineering if they are to exploit this technique as a means of achieving predictability. Flexible sources, such as CRT monitors, must be provided in order to alter the presentation of command information as an operator's tasks change over time. This is already occuring as a result of the increasing automation employed by systems engineering. Sizewell B will be the United Kingdom's first nuclear power station in which computers will provide the majority of its safety systems [322]. Secondly, systems designers must recruit human factors engineering in order to identify the command information required during operator

tasks. This is also taking place. The Chernobyl and Three Mile Island accidents have led the United Kingdom's Nuclear Industry Inspectorate to recommend careful consideration of control tasks during the development of CRT displays [5]. It is hypothesised that principles, such as predictability, could support such analyses by establishing common objectives for the human factors and systems engineering of complex, interactive control systems.

3.5 Conclusions

Complexity is caused by state correspondence which occurs when a control system state represents different states of an application process. Complexity is caused by display correspondence which occurs when operators cannot accurately determine the state of a control system from the information displayed. Complexity is caused by command-view correspondence which occurs when users must monitor many different sources in order to access information about a command.

Complexity jeopardises predictability. If operators cannot determine the state of an application or a control system then they cannot predict the effect of their commands upon that state. If command information is dispersed amongst different sources then operators might not access enough data to determine the consequences of their intervention.

Principles support the design of complex systems because they establish high-level objectives for potential implementations. This encourages integration; techniques that support the operation of complex systems must frequently exploit both human factors and systems engineering in order to achieve objectives such as predictability. Interval temporal logic can be used to represent these techniques without specifying whether they are to be implemented using CRT or paper-based presentation systems. Examples from a range of different nuclear control systems have been used to illustrate these points. The intention has been to avoid Moray's criticism that unless they can be applied to real-life applications then "no amount of excellent logic or empirical research will have an impact" [212].

Section 3.2 made the simplifying assumption that control systems operate a single application. This is unrealistic because many systems are open to interaction with multiple users and processes. For instance, there are never less than three operators in United Kingdom nuclear control rooms whenever two reactors are running. The next chapter argues that human factors and systems engineers might also use interval temporal logic to support the principled design of such open control systems.

Chapter 4

Openness

> "Current automation...integrates parts of the system which were previously isolated... The operator rarely has access to all of the information coming from the processes and loses direct control of the system... The specialised working teams that stand as satellites around the process grow in number with the increase in automation... All these people interact with each other and the control room operator... He no longer controls the process but the global complex system" (Decortis, de Keyser, Cacciabue and Volta, [85]).

4.1 Introduction

An open system is defined to be one that supports simultaneous interaction between multiple users and processes. The quotation that starts this chapter describes the consequences that openness can have for the operation of a control system. This chapter argues that:

- openness can frustrate control tasks because the success of commands can be jeopardised by input from other users and because operators must simultaneously interact with more than one process;
- interval temporal logic provides a means of representing techniques that human factors and systems engineers might employ to resolve the problems created by openness;
- designers must be able to assess the impact that these techniques have upon principles, such as predictability.

It is concluded that principles which are intended to encourage the integration of human factors and systems engineering can seldom be achieved by techniques that ignore one of these complementary approaches to design.

4.1.1 Consequences: Operator Error And Unpredictability

Open control systems provide multiple operators with access to multiple processes. Schmidt reviews the benefits that openness provides: flexibility; integration; articulation and cooperation [276]. Rather than explore these benefits, this chapter

analyses the costs of openness. Human factors research has demonstrated that the demands of operating open applications frequently lead to operator error [273]. Duncan and Praetorius argue that if users must simultaneously control several processes then they are liable to make substitution errors, these were previously described in Section 3.1.2 [92]. Reason argues that interference errors occur when users maintain multiple plans for interacting with more than one process [252]. In other words, the cognitive demands of pursuing one plan can interfere with the successful pursuit of another. Tatar, Foster and Bobrow demonstrate that the presentation of inadequate information about the activities of other users leads to frustration and confusion [299].

Wagenaar and Groeneweg argue that "unpredictability is caused by ... the spread of information over the participants" in control [314]. Muir argues that "the more constrained (i.e., closed) a machine's behaviours are, the greater will be its predictability" [220].

4.1.2 Causes: Input Contention And Output Contention

Many of the problems that impair the usability of open applications are caused by input contention. This occurs when an operator's control task is frustrated by concurrent input from another user. For instance, each London Underground line is supervised from a Line Control Room by a Line Controller and some Line Assistants [142]. They can intervene in the running of the line by issuing commands to train drivers or station staff. Input contention would occur if a Line Controller requested that a train stop at the same time as a Line Assistant gave it clearance to continue its journey. Input contention has a powerful impact upon the predictability of open control systems. Operators are unlikely to make accurate predictions about the success or failure of their commands if they cannot determine whether other users have issued concurrent input. In the previous example, the Line Controller cannot predict that their command will not be frustrated by input from a Line Assistant.

The problems of openness are also caused by output contention. This occurs when operators must monitor state changes in concurrent processes. For instance, if a Line Controller diverts a train then they must monitor the knock-on effects that their actions have upon other trains and lines. In order to do this, operators must simultaneously interact with different aspects of control system functionality. They must monitor their own section of track as well as those controlled by other operators. If users fail to allocate sufficient attention to these tasks then they are likely to miss important changes in the state of application processes. If operators miss important changes in the state of application processes then they are likely to make incorrect predictions about the effects of their intervention. For instance, the Severn Tunnel crash occurred because Bristol signal-men were simultaneously required to interact with track-side watchmen, signal-men at the Welsh end of the tunnel and trains approaching their section of track [34]. As a result, they failed to observe that a Paddington to Cardiff InterCity 125 had stopped inside the tunnel. They also failed to predict the consequences of allowing a local Sprinter train to enter the tunnel.

4.1.3 Solution: Generic Design Principles

Figure 4.1 illustrates the structure of interaction between the many users and processes of open control systems. The arrows on the left hand side of the diagram

Figure 4.1: A model of interaction with an open control system

represent the input that users provide to a control system. They also represent the presentation of information by the control system to those operators. The arrows on the right hand side of the diagram represent sensor input from application processes to the control system. They also represent the transmission of control instructions from the system to those processes. Input contention occurs if users, `u`, `u'` and `u''`, issue input to a control system at the same time as one of their colleagues. Output contention occurs when operators must simultaneously observe information about a number of different application processes; `a`, `a'` and `a''`.

Just as these is no panacea for the problems created by complexity, there is also no panacea for those created by openness. The following sections argue that design principles provide criteria against which to assess techniques that can be used to resolve the problems of input and output contention. This supports integration; principles provide a common focus for the assessment of human factors and systems engineering solutions. It is argued that interval temporal logic is an appropriate medium for such assessments because it provides generic abstractions that can be used to support the development of interfaces to many different applications.

4.2 Input Contention

In order to identify solutions for input contention, designers must first have a clear idea of the causes of this problem. Interval temporal logic can support this task. The set **U** includes all the users of an interactive control system. A relation can be introduced between an input sequence, **p**, and the user, **u**, who issues it in the present interval:

$$\exists\, \mathbf{u} \in \mathbf{U}, \exists\, \mathbf{p} \in \mathbf{P} : \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \tag{4.1}$$

The following pages adopt the convention of describing input contention between two operators. This analysis can easily be extended to describe concurrent interaction between many more users by introducing requirements for **user_input** from other operators. An identity relation is also introduced between elements of **U**:

$$\forall\, \mathbf{u} \in \mathbf{U} : \mathbf{same\_user}(\mathbf{u}, \mathbf{u}) \tag{4.2}$$

Input contention occurs when different operators simultaneously issue commands to a control system. Temporal formalisms provide a means of representing vernacular descriptions that include terms such as 'simultaneously'. For instance, the following predicate specifies that input contention occurs if and only if different users, **u** and $\mathbf{u}'$, issue input, **p** and $\mathbf{p}'$, in the present interval:

$$\begin{aligned} \exists\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \exists\, \mathbf{p}, \mathbf{p}' \in \mathbf{P} : \mathbf{input\_contention}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}') \Leftrightarrow \\ \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \wedge \mathbf{user\_input}(\mathbf{u}', \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}') \end{aligned} \tag{4.3}$$

For example, the 1988 Paris-Luxembourg rail accident was caused because contradictory input was issued to a control system [33]. One signal-man, **u**, issued input, **p**, which indicated that the line was clear, whilst another, $\mathbf{u}'$, entered input, $\mathbf{p}'$, to indicate that it was unsafe to proceed.

4.2.1 Input Contention And Predictability

Input contention jeopardises predictability. For example, a London Underground Line Assistant, **u**, might issue input, **p**, that orders a train onto a new section of track, **s**. At the same time the Line Controller, $\mathbf{u}'$, issues input, $\mathbf{p}'$, that is intended to stop the train on a section of track, $\mathbf{s}'$. It is not possible to satisfy both commands. One operator will, therefore, make an inaccurate prediction if both anticipate the success of their interaction. Interval temporal logic provides human factors and systems engineers with a means of representing the relationship between input contention and violations of predictability. It is difficult for operators to predict the effect of their input if eventually a colleague simultaneously issues input with a potentially different effect:

$$\begin{aligned} &\exists\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \exists\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{unpredictable}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Leftarrow \exists\, \mathbf{s}'' \in \mathbf{S} \\ &\Diamond(\mathbf{input\_contention}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}') \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}'', \mathbf{s}) \wedge \\ &\mathbf{interpret\_effect}(\mathbf{p}', \mathbf{s}'', \mathbf{s}')) \end{aligned} \tag{4.4}$$

This bi-condition illustrates the benefits of genericity which are to be derived from the use of logic abstractions. Designers could exploit interval temporal logic to characterise usability problems that are common to many different multi-user control systems. For instance, input contention is a problem for the operation of railway networks and passenger aircraft. Foushee and Helmrich describe how a captain, **u**, issued input, **p**, to dump fuel so that their aircraft would reach its desired landing weight, **s** [104]. The flight engineer, $\mathbf{u}'$, issued input, $\mathbf{p}'$, to terminate the pumping procedure, $\mathbf{s}'$. The captain was amazed to find that the gross landing weight of the aircraft was over six hundred and forty thousand pounds rather than the intended five hundred and seventy thousand pounds. Subsequent analysis revealed that neither operator had accurately predicted the consequences of their commands because

neither had considered the possibility of input contention. Similar examples can also be identified in oil production, discussed in Chapter 2, and nuclear reactor control, discussed in Chapter 3. Doyle, Gaddy, Burgy and Topmiller review the communications problems that arise between nuclear power plant operators [90]. For instance, input, $\mathbf{p}$, from an operator, $\mathbf{u}$, to increase the power loading of a supply network, $\mathbf{s}$, might be issued at the same time as another operator, $\mathbf{u}'$, attempts to close down the reactor, $\mathbf{s}'$. Input contention also provides a possible explanation of the problems that delayed a response to the loss of Cormorant Alpha's Sea Puma helicopter. There is evidence of contradictory instructions, $\mathbf{p}$ and $\mathbf{p}'$, being issued by the rig management, $\mathbf{u}$, and the operators, $\mathbf{u}'$, of the field's search and rescue helicopter [221]. The former approved flights, $\mathbf{s}$, under poor weather conditions, $\mathbf{s}''$, whilst the later did not, $\mathbf{s}'$. It is alleged that the rig management failed to predict the problems that would arise if their helicopter crashed; they were unaware that the search and rescue team had ceased flying. The search and rescue team did not predict the consequences that their commands would have because they were unaware that the Sea Puma was still flying.

4.2.2 Resolving Input Contention

The following section argues that human factors and systems engineers might use interval temporal logic to represent potential solutions to the problems caused by input contention.

Voluntary Input Protocols

Voluntary input protocols exploit the flexibility of system operators to avoid contention. For instance, input contention is avoided if and only if different operators ensure that they never enter concurrent input into a system:

$$\forall \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall \mathbf{p}, \mathbf{p}' \in \mathbf{P} : \mathbf{no\_input\_contention}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}') \Leftrightarrow \\ \Box\neg \left(\mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \wedge \mathbf{user\_input}(\mathbf{u}', \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')\right) \quad (4.5)$$

For example, De Keyser observes that operating instructions are frequently drafted to ensure that only one user has control of an application at any point during interaction [172]. This is termed a primary user protocol because the input of one operator takes precedence over that of their colleagues. In order to exploit this approach designers must have a clear understanding of the social ergonomics of control, discussed in Section 1.3.2. For instance, they must ensure that disputes within the workforce do not lead to violations of primary user protocols. Designers could apply the axioms and theorems of interval temporal logic, see Appendix D.1, to clarify the constraints imposed by **no_input_contention** (4.5). It is always the case that if a London Underground Line Controller, $\mathbf{u}$, issues input, $\mathbf{p}$, then it is not the case that different operators, $\mathbf{u}'$, issue input, $\mathbf{p}'$, at the same time:

$$\forall \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall \mathbf{p}, \mathbf{p}' \in \mathbf{P} : \mathbf{no\_input\_contention}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}') \Leftrightarrow \\ \Box(\mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \Rightarrow \neg\, (\mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}'))) \quad (4.6)$$

A limitation of this approach is that it is not always possible to identify a primary user, especially in processes that are controlled by different operators working in

different locations [36]. A number of alternative voluntary input protocols avoid this limitation. For instance, Popitz, Bahrdt, Jüres and Kesting argue that operators can avoid contention through state synchronisation [239]. They demonstrate that users coordinate their activities through the medium of the process itself. Operators do not issue input until they observe particular changes in the state of a process. These changes trigger their intervention. Their actions cause further state changes which, in turn, trigger input from their colleagues. Contention is avoided because different users do not respond to the same state change. Interval temporal logic can be used to represent the requirements that must be satisfied in order to implement this solution for input contention. For instance, if one operator, $\mathbf{u}$, issues input, $\mathbf{p}$, then their colleagues, $\mathbf{u}'$, wait until that input takes effect before interacting with the control system:

$$\begin{aligned}\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S} : \mathbf{synchronisation}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}) \Leftrightarrow \exists\, \mathbf{s}' \in \mathbf{S} \\ (\mathbf{user\_effect}(\mathbf{u}, \mathbf{p}, \mathbf{s}, \mathbf{s}') \Rightarrow \\ (\neg\, (\mathbf{user\_input}(\mathbf{u}', \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}'))\ \mathcal{U}\ \mathbf{state}(\mathbf{s}')))\end{aligned} \tag{4.7}$$

Section 4.3.1 will introduce a predicate to describe the effects of operator input upon particular application processes. In contrast, the **user_effect** predicate in the previous bi-condition is true for the effect, $\mathbf{s}'$, of an input sequence, $\mathbf{p}$, from a user, $\mathbf{u}$, upon the state, $\mathbf{s}$, of a control system in the present interval:

$$\begin{aligned}\forall\, \mathbf{u} \in \mathbf{U}, \forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{s}' \in \mathbf{S} : \mathbf{user\_effect}(\mathbf{u}, \mathbf{p}, \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}')\end{aligned} \tag{4.8}$$

A number of limitations restrict the application of voluntary protocols to avoid input contention. Designers have no guarantee that users will abide by the operating procedures intended to avoid interference. There are many examples where voluntary protocols have broken-down with catastrophic consequences. In November 1990, confusion between line controllers led to two InterCity trains colliding outside Newcastle Upon Tyne central station. In August 1990, similar problems led to a packed commuter train being switched onto the wrong track outside Reading station; it collided with another passenger train [200].

System Imposed Input Protocols

System imposed input protocols exploit automation to avoid input contention. It is possible to automatically lock out input so that some users cannot affect the state of the system. For example, if a London Underground Line Controller, $\mathbf{u}$, issues input, $\mathbf{p}$, to close down the line, $\mathbf{s}$, and a Line Assistant, $\mathbf{u}'$, attempts to keep the track in an operating state, $\mathbf{s}'$, then in the next interval the Line Assistant's input does not have the predicted effect but the Line Controller's does:

$$\begin{aligned}\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{state\_lock}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Leftrightarrow \\ \mathbf{unpredictable}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Rightarrow \bigcirc(\mathbf{state}(\mathbf{s}) \wedge \neg\, \mathbf{state}(\mathbf{s}'))\end{aligned} \tag{4.9}$$

Technical limitations restrict the exploitation of such automated approaches to input contention. The Hidden Report into the Clapham crash accepted that the high costs of installing new signaling equipment prevented British Rail from adopting

this solution [200]. Human factors problems also limit the utility of this approach. Locking restricts access to shared resources. A Line Assistant's attempts to get a train into a station can be frustrated by their Line Controller closing the line. Fairness is not guaranteed [94]. In other words, some operators can be unfairly excluded from interaction if their commands are frequently prevented from affecting the state of the system. Input queues provide a means of avoiding the human factors and system engineering limitations of state locking. If users, $\mathbf{u}$ and $\mathbf{u}'$, issue concurrent input, $\mathbf{p}$ and $\mathbf{p}'$, then in the next interval the system handles $\mathbf{p}$ and $\mathbf{p}'$ is eventually serviced. Amendola, Bersini, Cacciabue and Mancini exploit this input queueing technique in their system response analyser [14]. Goals are assigned to operators on the basis of previous traces of interaction. If interference is detected between the goals of two users then the system attempts to satisfy the goal with the highest priority. This approach is fair because the goals of other users are eventually satisfied once the first goal has been achieved:

$$\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{fair\_input}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Leftrightarrow \mathbf{unpredictable}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Rightarrow \bigcirc(\mathbf{state}(\mathbf{s}) \wedge \Diamond \mathbf{state}(\mathbf{s}')) \quad (4.10)$$

Applying systems engineering to maintain queues of low-priority input can create a number of human factors problems. Delays in system response times lead to frustration and error [186]. Unpredictability is likely to occur when periods of quiescence allow the system to process a backlog of input [94]. Delayed commands can eventually take effect at inappropriate moments during interaction.

Mixed Approaches

System imposed input protocols exploit automation whilst voluntary input protocols exploit the flexibility of system operators to avoid contention. Mixed approaches exploit elements of both these techniques. Designers might combat input contention by isolating users from the activities of their colleagues [95]. Operator predictions can be confirmed if they are given the impression of sole access to shared resources. For instance, a London Underground Line Controller, $\mathbf{u}$, could issue input, $\mathbf{p}$, to close down the track, $\mathbf{s}$, at the same time as a Line Assistant, $\mathbf{u}'$, issues input, $\mathbf{p}'$, to keep the line open, $\mathbf{s}'$. A control system which provided independent views might confirm that both of these contradictory commands had been successful. Maintaining independent views abandons the 'What You See Is What I See' principle, introduced in Section 1.5. Each user has a different view of the state of their system. This requirement is represented in the following bi-condition by the absence of any explicit relation between the displays, $\mathbf{d}$ and $\mathbf{d}'$, that are used to confirm the effect of operator intervention:

$$\begin{aligned} &\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \forall\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \\ &\quad \mathbf{independent\_views}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{d}, \mathbf{d}') \Leftrightarrow \exists\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} \\ &\quad (\mathbf{unpredictable}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}, \mathbf{s}') \Rightarrow \\ &\quad \mathbf{user\_view}(\mathbf{s}, \mathbf{d}, \mathbf{u}) \wedge \mathbf{user\_view}(\mathbf{s}', \mathbf{d}', \mathbf{u}')) \end{aligned} \quad (4.11)$$

The **user_view** predicate is true for a display, $\mathbf{d}$, that presents information about the state, $\mathbf{s}$, of a control system to a user, $\mathbf{u}$, in the present interval:

$$\exists\, \mathbf{s} \in \mathbf{S}, \exists\, \mathbf{d} \in \mathbf{D}, \exists\, \mathbf{u} \in \mathbf{U} : \mathbf{user\_view}(\mathbf{s}, \mathbf{d}, \mathbf{u}) \quad (4.12)$$

This independent view technique is mixed in the sense that it relies upon the integration of human factors and systems engineering. Systems engineering must provide conflict resolution mechanisms [299]. In the previous example, conflict between the Line Controller and the Line Assistant might be resolved by delaying the effects of a command to close the line even though the Line Controller's view could have shown that their input took immediate effect. Human factors engineering must ensure that users can resolve instances in which these mechanisms fail. For instance, conflict resolution techniques might not be able to cope with two operators simultaneously ordering a train to travel in opposite directions. In such circumstances, users must cooperate in order to resolve potential conflicts between their commands. A number of mixed techniques have been developed to encourage this cooperation. For instance, voting mechanisms require mutual agreement between operators before their commands can take effect. Voting relies upon synchronisation points. The provision of input serves to synchronise the activities of multiple users. This democratic approach is the antithesis of **no_input_contention** (4.6); no primary user can dictate the course of interaction. Designers must integrate human factors and systems engineering if they are to exploit voting as a means of avoiding input contention. Systems engineers must provide the necessary communications support to enable voting to take place. Human factors engineers must assess the perceptual and cognitive burdens imposed upon operators who must suspend their activities in order to vote for and against the commands of their colleagues. For instance, input from the Bristol signal-man, $\mathbf{u}$, to the system described in Section 4.1.2 was not intended to enable a train to enter the Severn tunnel, $\mathbf{s}'$, until clearance, $\mathbf{p}'$, had also been given by a signal-man, $\mathbf{u}'$, at the Welsh end:

$$\begin{aligned}&\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P}, \forall\, \mathbf{s} \in \mathbf{S} : \mathbf{vote\_input}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}', \mathbf{s}) \Leftrightarrow \exists\, \mathbf{s}', \mathbf{s}'' \in \mathbf{S} \\ &\quad (\mathbf{user\_effect}(\mathbf{u}, \mathbf{p}, \mathbf{s}, \mathbf{s}') \Rightarrow (\neg\, \mathbf{state}(\mathbf{s}')\mathcal{U} \\ &\quad (\neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}') \wedge \mathbf{user\_effect}(\mathbf{u}, \mathbf{p}', \mathbf{s}'', \mathbf{s}'))))\end{aligned} \tag{4.13}$$

Section 1.5 argued that designers could exploit principles as the basis for reasoning about interactive control systems. The previous discussion provides an example of this; the analysis of predictability and openness has revealed trade-offs between contention, cooperation and control. Voluntary input protocols avoid contention through mutual cooperation between system operators. If this cooperation fails then it might no longer be possible to guarantee safety. Input contention can frustrate operator predictions about the effects of their commands. System imposed protocols avoid contention by placing locks upon operator input. This reduces the potential for cooperation and can jeopardise safety if users are prevented from accessing shared resources. Operators cannot predict that their commands will have the desired effect because they could be locked out by the system. Mixed approaches have been recommended as alternatives that integrate human factors and systems engineering. Independent views rely upon conflict resolution mechanisms to guarantee the safety of an application. Cooperation is required if these mechanisms fail. Voting mechanisms provide locks for security and rely upon mutual agreement before commands take effect. This ensures cooperation. The utility of this approach is limited if adequate communications equipment cannot be provided by systems engineering. The human factors techniques of social ergonomics must ensure that users will reach a consensus before application processes require operator intervention.

4.3 Output Contention

Control system operators are increasingly required to supervise many different application processes. For example, a London Underground Line Controller must ensure the safe and timely passage of trains over their sections of track. They must also coordinate the activities of cleaning and maintenance teams. Designers can represent application processes as elements of a set, $\mathbf{A}$. An identity relation is introduced between the elements of this set:

$$\forall\, \mathbf{a} \in \mathbf{A} : \mathbf{same\_process}(\mathbf{a}, \mathbf{a}) \tag{4.14}$$

The elements of the set $\mathbf{Ps}$, introduced in Section 3.2, represent the state of an application process. Displays, $\mathbf{d}$, present information about the states, $\mathbf{ps}$, of processes, $\mathbf{a}$, in the present interval:

$$\forall\, \mathbf{a} \in \mathbf{A}, \forall\, \mathbf{ps} \in \mathbf{Ps}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{process\_view}(\mathbf{a}, \mathbf{ps}, \mathbf{d}) \tag{4.15}$$

It is important to note that this predicate makes no assumptions about the sources or devices used to present displays, $\mathbf{d}$. The following pages describe the problems of operating two different processes. This analysis can easily be extended by requiring a **process_view** of other processes. Output contention occurs if and only if a display, $\mathbf{d}$, simultaneously presents information about the states, $\mathbf{ps}$ and $\mathbf{ps}'$, of different application processes, $\mathbf{a}$ and $\mathbf{a}'$:

$$\begin{aligned}&\exists\, \mathbf{a}, \mathbf{a}' \in \mathbf{A}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{output\_contention}(\mathbf{a}, \mathbf{a}', \mathbf{ps}, \mathbf{ps}', \mathbf{d}) \Leftrightarrow \\ &\mathbf{process\_view}(\mathbf{a}, \mathbf{ps}, \mathbf{d}) \wedge \mathbf{process\_view}(\mathbf{a}', \mathbf{ps}', \mathbf{d}) \wedge \neg\, \mathbf{same\_process}(\mathbf{a}, \mathbf{a}')\end{aligned} \tag{4.16}$$

Output contention frustrates the task of controlling many application processes. Human factors research has identified the costs incurred by users when they move the focus of their attention from one process to another [318]. For example, operators in the Waterloo control room for British Rail's Southern Division are frequently required to switch services between lines in their region. To help them do this they are presented with displays, $\mathbf{d}$, that simultaneously provide information about many different trains, $\mathbf{a}$ and $\mathbf{a}'$. If operators fail to notice that a train, $\mathbf{a}$, has been delayed, $\mathbf{ps}$, then the knock-on effects can be considerable. It is a non-trivial task to re-route the hundreds of services that are required to carry over one hundred and forty thousand commuters between Kent, Essex and London every weekday morning [161].

4.3.1 Output Contention And Predictability

De Keyser suggests that operators often fail to comprehend the consequences of their actions in multi-process applications [172]. Predictability is violated if the effects of input, $\mathbf{p}$, on different application processes, $\mathbf{a}$ and $\mathbf{a}'$, are presented in the present interval. It is important to note that operators must not simply anticipate the effects of input upon the state of a control system; they must predict the consequences of their intervention for individual processes:

$$\begin{aligned}\exists\, \mathbf{p} \in \mathbf{P}, \exists\, \mathbf{a}, \mathbf{a}' \in \mathbf{A}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{unpredictable}(\mathbf{p}, \mathbf{a}, \mathbf{a}', \mathbf{d}) \Leftarrow \\ \exists\, \mathbf{ps}, \mathbf{ps}', \mathbf{ps}'', \mathbf{ps}''' \in \mathbf{Ps}(\mathbf{output\_contention}(\mathbf{a}, \mathbf{a}', \mathbf{ps}', \mathbf{ps}''', \mathbf{d}) \wedge \\ \mathbf{process\_effect}(\mathbf{p}, \mathbf{a}, \mathbf{ps}, \mathbf{ps}') \wedge \mathbf{process\_effect}(\mathbf{p}, \mathbf{a}', \mathbf{ps}'', \mathbf{ps}'''))\end{aligned} \tag{4.17}$$

The **process_effect** relation in the previous implication describes the effect, $\mathbf{ps}'$, of input, $\mathbf{p}$, on the state, $\mathbf{ps}$, of an application process, $\mathbf{a}$, in the present interval:

$$\forall\, \mathbf{p} \in \mathbf{P}, \forall\, \mathbf{a} \in \mathbf{A}, \forall\, \mathbf{ps} \in \mathbf{Ps}, \exists\, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{process\_effect}(\mathbf{p}, \mathbf{a}, \mathbf{ps}, \mathbf{ps}') \quad (4.18)$$

Railway signaling systems provide examples of this form of unpredictability. Line controllers issue input, $\mathbf{p}$, to divert trains, $\mathbf{a}$, onto different tracks, $\mathbf{ps}'$. When they do this it is important to ensure that following trains, $\mathbf{a}'$, are slowed, $\mathbf{ps}'''$, to allow the first train to clear the junction. In order to make accurate predictions about the effects of their input, operators must able to monitor many different trains. In 1990 there were approximately one thousand incidents in which British Rail signalmen provided incorrect information about blocked junctions or diverted rolling-stock onto the wrong side of two-way tracks [34]. The consequences of such errors can be catastrophic. In December 1988, thirty-five people died when an InterCity express ploughed into the back of a rush-hour commuter train at Clapham Junction. The problems of simultaneously controlling many different trains were cited as a major cause of this accident [200].

4.3.2 Resolving Output Contention

The following section argues that designers might exploit interval temporal logic to represent potential solutions for the problem of output contention.

System Imposed Output Protocols

System imposed output protocols exploit automation and display design in order to reduce the cognitive and perceptual burdens that output contention places upon system operators. Zachary argues that designers must consider the limitations imposed by the "weak concurrency" of human information processing [333]. Users simultaneously maintain goals for many different processes; "only one of which may be actively pursued by the human operator at any one time". Human factors and systems engineers can exploit this observation by multiplexing the presentation of interactive control systems [24]. In other words, they might ensure that different processes are never presented by the same display. For instance, the presentation of information by cathode ray tube (CRT) monitors in London Underground Line Control Rooms could be filtered to ensure that data about different trains, $\mathbf{a}$ and $\mathbf{a}'$, is never simultaneously presented. It is hypothesised that this might reduce the substitution errors that occur when operators issue input to the wrong process on multi-process displays:

$$\begin{aligned}&\forall\, \mathbf{a}, \mathbf{a}' \in \mathbf{A}, \forall\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps}, \forall\, \mathbf{d} \in \mathbf{D} : \\ &\quad \mathbf{no\_output\_contention}(\mathbf{a}, \mathbf{a}', \mathbf{ps}, \mathbf{ps}', \mathbf{d}) \Leftrightarrow \Box\neg\, (\mathbf{process\_view}(\mathbf{a}, \mathbf{ps}, \mathbf{d}) \wedge \\ &\quad \mathbf{process\_view}(\mathbf{a}', \mathbf{ps}', \mathbf{d}) \wedge \neg\, \mathbf{same\_process}(\mathbf{a}, \mathbf{a}')) \end{aligned} \quad (4.19)$$

A number of problems limit the utility of this technique. Hiding information about the state of application processes leads to display correspondence; discussed in Section 3.3. If operators are unaware of the state of an application then they are unlikely to make accurate predictions about the effects of their interaction upon that state. This is illustrated by the operator 'errors' that compounded the King's Cross fire.

Five of the eight CRT monitors in the Line Control Room were either broken or were not switched on. In consequence, the Line Controller could not display the passage of trains, $\mathbf{a}$ and $\mathbf{a}'$, through their station. The Fennell investigation concluded that this information would have helped them to predict the consequences of ignoring police instructions to prevent Picadilly and Victoria line trains from stopping at King's Cross [98]. A further limitation is that **no_output_contention** (4.19) can be unfair. In other words, information about the state of a process might be hidden indefinitely. There is no guarantee that it will be presented to system operators. Designers could avoid this problem by recording information about the states of application processes. This information could eventually be displayed as soon as resources are released by the presentation of information about other processes [40]. This output protocol is fair because information about the state, $\mathbf{ps}$, of process, $\mathbf{a}$, is presented by a display, $\mathbf{d}$, in the present interval and in the next interval it is eventually the case that information about a different process, $\mathbf{a}'$, is presented:

$$\exists\, \mathbf{a}, \mathbf{a}' \in \mathbf{A}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps}, \exists\, \mathbf{d} \in \mathbf{D} : \mathbf{fair\_output}(\mathbf{a}, \mathbf{a}', \mathbf{ps}, \mathbf{ps}', \mathbf{d}) \Leftrightarrow$$
$$\mathbf{process\_view}(\mathbf{a}, \mathbf{ps}, \mathbf{d}) \wedge \bigcirc\Diamond(\mathbf{process\_view}(\mathbf{a}', \mathbf{ps}', \mathbf{d}) \wedge \neg\, \mathbf{same\_process}(\mathbf{a}, \mathbf{a}')) \quad (4.20)$$

The abstractions of interval temporal logic provide designers with a means of specifying generic requirements that can be applied to guide the development of many different control systems. For instance, this approach has been embodied within the systems engineering of Boeing 757 and 767 aircraft [282]. Information about a malfunctioning avionics process, $\mathbf{a}$, is immediately displayed by the Engine Indication and Crew-Alerting System (EICAS). Automated systems control other avionics applications during these critical events. Eventually the EICAS presents information about the other processes, $\mathbf{a}'$, so that the flight can continue. Sanquist and Fujita have used fair output protocols in a nuclear reactor control system [272]. The presentation of information about low-priority processes, $\mathbf{a}'$, is postponed during system errors. Similar alarm suppression systems could also be incorporated into the new generation of fire-fighting systems being developed for oil-rigs such as Piper Bravo [147]. The fact that it has been possible to represent generic requirements for **fair_output** (4.20) does not imply that this technique is appropriate for all applications. Operators must frequently have immediate access to data about many different processes. For instance, the Relief Station Inspector, the Line Controller and the Station Manager required constant information about trains, platforms and passengers during the King's Cross fire [98]. Such genericity does, however, encourage designers to explore techniques that have proven useful in one domain and which could have a wider application.

Voluntary Output Protocols

Voluntary output protocols exploit the flexibility of operator monitoring techniques to support display design. Liu and Wickens argue that designers might exploit these protocols by developing displays which do not present different application processes through the same channel [197]. A channel consists of a number of display elements which operators can observe in parallel. For example, production graphs and error messages need not be monitored in sequence if they are displayed close together. Rapid eye movements between many different areas of a CRT monitor indicate an inefficient grouping of process information. This can be assessed experimentally

using an Eye-mark recorder; infra-red light is projected onto an operator's retina to determine the object of their visual fixations [130]. In order to represent voluntary output protocols a relationship is introduced between a display, **d**, and an element, **ch**, of a set of channels, **Ch**, through which an operator observes a process, **a**, in the present interval. This relation is similar to **gate** (2.33) and **template** (3.23); it can be thought of as a template which designers could place over a display in order to extract those elements that can be monitored in parallel:

$$\exists\, \mathbf{a} \in \mathbf{A}, \exists\, \mathbf{d} \in \mathbf{D}, \exists\, \mathbf{ch} \in \mathbf{Ch} : \mathbf{observe}(\mathbf{a}, \mathbf{d}, \mathbf{ch}) \tag{4.21}$$

Human factors and systems engineers might use this predicate to represent the requirements that must be satisfied if operators are to exploit output channels when monitoring concurrent processes. For example, a London Underground Line Controller could focus their attention if and only if it is never the case that maintenance, **a**, and cleaning, $\mathbf{a}'$, processes are observed through the same channel, **ch**, of a display, **d**:

$$\begin{aligned}&\forall\, \mathbf{d} \in \mathbf{D}, \forall\, \mathbf{ch} \in \mathbf{Ch}, \forall\, \mathbf{a}, \mathbf{a}' \in \mathbf{A} : \mathbf{channel\_focus}(\mathbf{d}, \mathbf{ch}, \mathbf{a}, \mathbf{a}') \Leftrightarrow \\ &\quad \Box\neg\, (\mathbf{observe}(\mathbf{a}, \mathbf{d}, \mathbf{ch}) \wedge \mathbf{observe}(\mathbf{a}', \mathbf{d}, \mathbf{ch}) \wedge \neg\, \mathbf{same\_process}(\mathbf{a}, \mathbf{a}'))\end{aligned} \tag{4.22}$$

A number of limitations restrict the utility of devoting an output channel to the presentation of a single process. Line controllers must sequentially scan different output channels in order to schedule maintenance, **a**, and cleaning, $\mathbf{a}'$, processes. They must, therefore, remember the data presented by one channel whilst observing another. Broadbent argues that the sustained attention necessary to fixate an output channel and remember process information extracts a toll in fatigue [43]. In consequence, the longer an operator must monitor a channel the more likely they are to forget data about other processes. Fischer, Haines and Price argue that by fixating upon one process, users will forget to scan sequentially other process information which is displayed through different channels [101]. Critical changes in power loading will be overlooked if an operator becomes pre-occupied with monitoring the progress of maintenance activities. In order to represent a solution to these problems a relation is introduced between an operator's task, **ta**, and a process, **a**, which is involved in that task in the present interval:

$$\exists\, \mathbf{ta} \in \mathbf{Ta}, \exists\, \mathbf{a} \in \mathbf{A} : \mathbf{task\_process}(\mathbf{ta}, \mathbf{a}) \tag{4.23}$$

Designers might exploit task analysis techniques to determine which application processes, **a**, are involved in particular tasks, **ta**. Section 8.2.1 describes how task analysis techniques can be directly used to inform principled design. For now it is sufficient to realise that human factors and systems engineers might focus the presentation of process information to support particular tasks. Processes, **a** and $\mathbf{a}'$, that are relevant to a control task, **ta**, are displayed through the same channel, **ch**, in the present interval:

$$\begin{aligned}&\exists\, \mathbf{ta} \in \mathbf{Ta}, \exists\, \mathbf{ch} \in \mathbf{Ch}, \exists\, \mathbf{a}, \mathbf{a}' \in \mathbf{A} : \\ &\quad \mathbf{task\_focus}(\mathbf{ta}, \mathbf{ch}, \mathbf{a}, \mathbf{a}') \Leftrightarrow \exists\, \mathbf{d} \in \mathbf{D} \\ &\quad (\mathbf{task\_process}(\mathbf{ta}, \mathbf{a}) \wedge \mathbf{task\_process}(\mathbf{ta}, \mathbf{a}') \wedge \\ &\quad \mathbf{observe}(\mathbf{a}, \mathbf{d}, \mathbf{ch}) \wedge \mathbf{observe}(\mathbf{a}', \mathbf{d}, \mathbf{ch}))\end{aligned} \tag{4.24}$$

Human factors and systems engineering limitations restrict the utility of voluntary output protocols. Differences in the perceptual and cognitive resources available to individual operators can determine their ability to exploit techniques such as channel and task focusing [41]. It is frequently impossible for human factors engineers to predict which channels operators will use to monitor process information. Wickens observes that most systems engineers "... do not know precisely what configuration of warning indicators will signal the onset of an abnormal condition" [318]. Eighty trains pass through Northern Line Underground stations every hour. The many different control tasks that must be performed in order to synchronise the running of such processes makes it difficult to identify an optimal allocation of process information to output channels [121].

Mixed Approaches

Mixed approaches integrate the automation and display design of system imposed protocols with the operator monitoring techniques exploited by voluntary output protocols. For instance, a London Underground control system might ensure that information about maintenance, $\mathbf{a}$, and cleaning, $\mathbf{a}'$, processes is presented until it is observed by a Line Controller. Human factors engineering could ensure that this information can be viewed through the same channel, $\mathbf{ch}$. This observational focus reduces output contention. Information about the state of the maintenance and cleaning processes can be hidden once it has been observed:

$$\begin{aligned}
&\exists\, \mathbf{u} \in \mathbf{U}, \exists\, \mathbf{ch} \in \mathbf{Ch}, \exists\, \mathbf{a}, \mathbf{a}' \in \mathbf{A}, \exists\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \\
&\quad \mathbf{observe_focus}(\mathbf{u}, \mathbf{ch}, \mathbf{a}, \mathbf{a}', \mathbf{ps}, \mathbf{ps}') \Leftrightarrow \exists\, \mathbf{d} \in \mathbf{D} \\
&\quad (\mathbf{output_contention}(\mathbf{a}, \mathbf{a}', \mathbf{ps}, \mathbf{ps}', \mathbf{d}) \mathcal{U} \\
&\quad (\mathbf{observe}(\mathbf{a}, \mathbf{d}, \mathbf{ch}) \wedge \mathbf{observe}(\mathbf{a}', \mathbf{d}, \mathbf{ch}) \wedge \mathbf{fixates}(\mathbf{u}, \mathbf{ch})))
\end{aligned} \tag{4.25}$$

In the above $\mathbf{fixates}(\mathbf{u}, \mathbf{ch})$ is true when a user, $\mathbf{u}$, fixates upon an output channel, $\mathbf{ch}$. A system might use an Eye-mark recorder in order to detect the visual fixations of their operators:

$$\exists\, \mathbf{u} \in \mathbf{U}, \exists\, \mathbf{ch} \in \mathbf{Ch} : \mathbf{fixates}(\mathbf{u}, \mathbf{ch}) \tag{4.26}$$

A number of practical limitations restrict the utility of this technique. In particular, the real-time duration of the $\mathcal{U}$ operator is not specified. How long must a display be presented if it has not been observed? Section 8.3.1 will argue that real-time operators might be introduced into interval temporal logic in order to represent such deadlines. Such notational enhancements do not resolve the practical limitations that restrict the utility of observational focusing. In order to monitor fixations, designers must ensure that operators wear an Eye-mark recorder during their observation of control system displays. These recorders restrict head movements and must be calibrated at regular intervals [130]. Designers must seek alternative techniques if they are to avoid these limitations. For instance, handshaking protocols provide a means of encouraging operators to acknowledge the presentation of application data [304]. The control system displays some data then waits for the user to acknowledge it before presenting more information. Handshaking involves a synchronisation point, $\mathbf{input}(\mathbf{p})$, to coordinate the presentation and observation of process information. Human factors engineering must ensure that operators confirm their

observation of process information by issuing input, **p**. Human factors engineers must also ensure that users perceive and understand the information presented to them. This is a non-trivial requirement. Operators can confirm that they have observed and understood a message even though their attention is not focussed upon the display [318]. Systems engineering must ensure that information is presented until the user acknowledges the display by issuing input, **p**. Handshaking, therefore, requires the integration of human factors and systems engineering. Designers might exploit this technique to support predictability. Users are presented with sufficient information for them to predict the effects of their intervention upon a process, **a**, until they confirm that it has been observed. The requirements for handshaking can be formalised as follows:

$$\exists\,\mathbf{p} \in \mathbf{P}, \exists\,\mathbf{a} \in \mathbf{A}, \exists\,\mathbf{ps} \in \mathbf{Ps}, \exists\,\mathbf{ch} \in \mathbf{Ch} : \mathbf{hand\_shake}(\mathbf{p}, \mathbf{a}, \mathbf{ps}, \mathbf{ch}) \Leftrightarrow \exists\,\mathbf{d} \in \mathbf{D}$$
$$(\mathbf{process\_view}(\mathbf{a}, \mathbf{ps}, \mathbf{d})\mathcal{U}(\mathbf{observe}(\mathbf{a}, \mathbf{d}, \mathbf{ch}) \wedge \mathbf{input}(\mathbf{p}))) \quad (4.27)$$

It is important to re-iterate the point that interval temporal logic abstractions provide generic representations of design techniques which are intended to support principles, such as predictability. Human factors and systems engineers could apply the previous bi-condition to guide the development of many different control systems. Hess describes the use of handshaking in the Rotorcraft Digital Advanced Avionics System of Bell UH-1H helicopters [146]. When avionics detects that an application process, **a**, is in a critical state, **ps**, then a caution is presented until the pilot confirms it. Handshaking has also been proposed as a means of improving safety on roll-on roll-off ferries. The negative reporting procedures used on the Herald of Free Enterprise were heavily criticised by Mr Justice Sheen's inquiry into the Zeebruge disaster [283]. The master was instructed to assume that all was well unless he was explicitly told otherwise. The Sheen report argued that positive confirmation should have been used. In other words, the Captain should have issued input, **p**, to confirm their observation that the bow doors, **a**, were closed, **ps**. This genericity provides great benefits for the development of interactive control systems. Techniques which support principles in one application can be represented in a form that can be applied to guide the human factors and systems engineering of many different control systems.

4.4 Conclusions

Input contention frustrates the design and operation of open control systems because different users can simultaneously issue input with different effects. Output contention frustrates the design and operation of open control systems because users must observe concurrent changes in the state of different application processes.

Input and output contention jeopardise predictability. Input contention threatens predictability if operators cannot determine which of their commands will be effective. Output contention threatens predictability if users fail to observe the state of a process that is affected by their input.

Interval temporal logic provides abstractions that can be used to specify generic solutions for the problems of openness, such as handshaking and voting. It has been argued that the support which these techniques offer for principles, such as predictability, depends upon the integration of human factors and systems engineering.

Principles which are intended to encourage integration can seldom be achieved by techniques that exploit only one of these engineering disciplines.

This part of the thesis has argued that principles provide human factors and systems engineering with criteria against which to assess solutions for the fundamental problems of control: dynamism; openness and complexity. Interval temporal logic has been proposed as an abstract and generic notation that can be used to support such assessments. This analysis has, however, been pitched at a level of detail that is inappropriate for the principled design of particular interfaces to particular control systems. In contrast, Part III argues that principles provide a means of integrating human factors and systems engineering to support the detailed design of interactive control systems.

Part III

Principles And The Problems Of Detailed Design

Introduction To Part III

This part of the thesis argues that principles provide a means of integrating human factors and systems engineers to support the detailed design of interactive control systems.

Chapter 5 argues that principles provide designers with a means of informing their choice of particular development architectures. Interval temporal logic is used to assess claims that object orientation not only supports the systems engineering of modular software but also supports the human factors engineering of consistent and predictable interfaces. It is argued that if these claims can be justified then object orientation could provide considerable support for the integration of human factors and systems engineering.

Chapter 6 argues that principles provide criteria against which to assess the weaknesses of development architectures. In particular, it is argued that object orientation threatens the development of predictable control systems because models of application components frequently break down. It is argued that designers might exploit interval temporal logic to clarify the strengths and weaknesses of potential solutions to this problem. The intention is to demonstrate that human factors and systems engineers could exploit development architectures, such as object orientation, without sacrificing principles, such as predictability.

Chapter 7 argues that principles, expressed in interval temporal logic, provide the non-formalist with little idea of what it would be like to operate a potential implementation. Prototypes provide users with a much better impression of the 'look and feel' of an interactive control system. In practice, formal analyses and prototyping are usually treated as alternatives. It is argued that the integration of human factors and systems engineers is encouraged if designers can exploit both of these techniques. PRELOG, a system for Presenting and REndering LOGic specifications of interactive systems, has been implemented to demonstrate that prototypes can be derived from a formal analysis of design principles. Partial implementations, developed using this tool, are intended to serve as blue-prints for the detailed design of interactive control systems.

Chapter 5

Inconsistency

> "There are clear relationships between the quality of an interactive system and not only the software components of that system, but also the embracing structure for those components" (Cockton, [74]).

5.1 Introduction

Development architectures are defined to be frameworks that provide a structure for the design of software components. This chapter argues that principles provide a focus for an analysis of the strengths of development architectures. Integration is encouraged if these strengths extend to both human factors and systems engineering. The quotation which opens this chapter observes that the choice of such frameworks not only affects the way in which software and hardware are designed, but that it also affects usability. In particular, the following pages argue that:

- object orientation supports the development of consistent interfaces;
- consistency supports predictability;
- some object oriented techniques jeopardise these benefits.

It is concluded that if human factors and systems engineers avoid techniques, such as method overriding, then some forms of consistency and predictability can be achieved through object instantiation.

5.1.1 Consequences: Skill Dependency And Unpredictability

The adjective 'inconsistent' is defined as being "at variance with one's own principles or former conduct" [12]. In the context of this thesis, the noun inconsistency is interpreted as variations in the presentation of similar output or differences in the handling of similar input. Payne and Green argue that the degree of inconsistency in an interface is a major determinant of learnability [233]. Carroll argues that operators learn an interface more rapidly and find it subjectively "easier to use" if it is perceived to be designed in a rational and consistent manner [55]. Yeh and Wickens argue that if users are familiar with a particular presentation format then minimal re-training is required in order for them to use other systems which also exploit that format [332]. Barnard, Hammond, Morton and Long argue that if

similar dialogues are handled differently then operators cannot transfer skills gained in one task to support another [25]. These observations support the argument, first made in Section 1.5, that consistency is a principle which might guide interface development.

Consistency supports predictability. Smith and Ellis find that users exploit knowledge gained from one control system to predict the behaviour of other control systems [**?**]. They point out the dangers that arise when designers fail to support such strategies; inconsistency implies that even if commands appear to be similar they can have very different effects.

5.1.2 Causes: Different Designers And Different Requirements

Grudin argues that systems are inconsistent because few designers can dictate all aspects of interface development [125]. Managerial and proprietary reasons prevent the imposition of consistency criteria over different design teams. For instance, Gopher, Olin, Donchin and Bieski observe that input sequences to increase the speed of medical pumps vary from manufacturer to manufacturer [124]. They also vary within product lines sold by the same manufacturer.

Changes to system requirements can lead to interface inconsistency. These are frequently made in response to consumer complaints or accident reports. For instance, oil-rig deluge pumping equipment has been substantially modified following the Cullen report into the Piper Alpha accident [81]. Inconsistency occurs if such modifications do not meet the criteria adopted by previous designers.

5.1.3 Solution: Object Oriented Design

Inconsistency might be avoided if designers were provided with documents that detail the consistency criteria to be satisfied throughout the life-cycle of a control system. A number of limitations restrict the utility of this approach. Kellogg [170], Nielsen [226] and Reisner [255] all note the difficulty of explicitly representing consistency criteria. Formal notations, such as interval temporal logic, can be exploited to resolve this problem [256]. Unfortunately, the use of such notations does not directly provide human factors and systems engineers with a means of achieving consistency. In contrast, Cockton suggests that designers could exploit development architectures to achieve interface consistency [74]. These architectures might, in turn, support predictability. Not only could operators assume that input and output are handled in similar fashions, they might also assume that similar commands have similar effects. Hewlett Packard's New Wave architecture, Sun Microsystem's Openwindows and the Digital Equipment Company's Compound Document Architecture have all exploited object orientation to support consistency [223]. Designers might, therefore, attempt to achieve principles, such as consistency and predictability, by using an object oriented architecture rather than the Seeheim model [74] or Edmonds' data-flow architecture [93].

This chapter specifically intends to assess the support which object oriented development provides for consistency and predictability. It is not intended to provide solutions for all of the many different forms of inconsistency, identified by the authors cited in Section 5.1.1. Such caveats stem from the problems of attempting to formalise a nebulous concept or principle, such as consistency. Interval temporal

logic can make these concepts more precise but, in doing so, may lose some of the findings of informal analyses [87]. Chapter 8 will address this issue in more depth. Enhanced notations, tools and development techniques will be proposed as means of improving the range of requirements that can be captured using formal notations.

5.2 Inconsistency And Object Oriented Design

Other authors [209, 263] have identified four necessary features of the object oriented architecture: objects; message passing; classes and type instantiation. Objects provide designers with a means of hiding information. They possess states, some aspects of which can be hidden from other objects in the system. They also possess methods that operate on those states. For instance, the state of a coolant tank might be changed from full to empty by invoking a method to evacuate it. Message passing provides the computational model that is associated with object oriented design. An object sends a message to invoke a method which acts upon the state of another object. It is important to note that operator input can also be described in terms of messages passing. For instance, a user might send a message that invokes a method to start a coolant pump. Classes provide designers with templates for objects that can be used in many different interfaces. Classes can inherit attributes from other classes. For example, instances of the class of pumps might be re-used in industrial and domestic heating control systems. Type instantiation provides a means of ensuring that similar objects share certain attributes. All instances of the same class can provide common methods and states. For example, tank objects could offer the full, half-full and empty states.

In order to determine the influence of object orientation upon consistency there must be some means of reasoning about a design without representing the thousands of lines of code necessary to implement a particular control system. This is illustrated by part of a program initialising a gauge object within the Smalltalk implementation of Lazarev's [190] furnace control system:

```
setWeightColor
   "Set the colour of the weight gauge."
   "Return if weightPane is nil."
   weightPane isNil
      ifTrue:  [↑ self].
   (temp<tmp1)
      ifTrue:  [(weightPane sliderGaugePane) colour:  2].
   (temp >= tmp1 and:  [temp < tmp2])
      ifTrue:  [(weightPane sliderGaugePane) colour:  10].
   (temp >= tmp2 and:  [temp < tmp3])
      ifTrue:  [(weightPane sliderGaugePane) colour:  11].
   (temp >= tmp3 and:  [temp < tmp4])
      ifTrue:  [(weightPane sliderGaugePane) colour:  12].
   (temp >= tmp4)
      ifTrue:  [(weightPane sliderGaugePane) colour:  13].
```

In order to establish consistency between control objects, a designer would be forced to compare many of these segments of code. For instance, they must ensure that every object used `colour:12` to represent states in which its temperature lay between `tmp3` and `tmp4`. A further limitation is that consistency criteria can only be expressed at this level of detail late in the development of a control system. Brooks argues that the costs of correcting flaws, such as inconsistency, are often prohibitive during the later stages of design [44]. Mathematically based notations provide a means of avoiding this limitation; designers might use them to express consistency criteria before a system is implemented. For instance, Backus Naur Form (BNF) grammars have been applied to analyse inconsistency [234]. This approach has its limitations. Reisner argues that "consistency is not a property of a system language" [256]. Interval temporal logic provides an alternative to BNF as a notation for reasoning about consistency. A set of control system objects, **O**, might be introduced. The elements of this set should be thought of as abstract representations of the software components that would be developed during implementation in an object oriented programming language. Section 6.2 will introduce a distinction between such control system objects and the physical components of application processes. An identity relation is introduced between elements of **O**:

$$\forall\, \mathbf{o} \in \mathbf{O} : \mathbf{same\_object}(\mathbf{o}, \mathbf{o}) \tag{5.1}$$

The set **Os** includes all the states of these control system objects. A relationship is introduced between objects and their state in the present interval. For instance, tank objects, **o**, can be in the full state, **os**:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{os} \in \mathbf{Os} : \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \tag{5.2}$$

Om is the power-set, the set of sets, of methods that can be applied to the state of an object. A relationship is introduced between objects and the methods which can affect their state in the present interval. For instance, methods, **om**, might be available to fill and evacuate tanks, **o**:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{om} \in \mathbf{Om} : \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \tag{5.3}$$

Object orientation is here being applied to support the design of interactive control systems and so we introduce a set, **Od**, that contains the images which are used to present objects. A relation is introduced between an object and its image in the current interval. For instance, an error display, **od**, can be used to present information about tanks, **o**:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{od} \in \mathbf{Od} : \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \tag{5.4}$$

These predicates can be used to describe the attributes of control system objects, **o**. They are accessible via methods, **om**, have an internal state, **os**, and are displayed by an image, **od**:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{om} \in \mathbf{Om}, \exists\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{od} \in \mathbf{Od} : \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \tag{5.5}$$

Designers might use predicates, such as **object** (5.5), to represent the causes of inconsistency in object oriented control systems. Two objects, **o** and $\mathbf{o}'$, are inconsistent if and only if they do not always have the same display, **od**, the same state,

os, and the same methods, **om**. It should be noted that if the $\neg$ **same_object**$(\mathbf{o}, \mathbf{o}')$ requirement is dropped from the following predicate it would be possible for **o** and **o′** to unify. In other words, an object is inconsistent with itself if and only if it does not always possess the same attributes:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{inconsistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \neg\, \Box(\mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{object}(\mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}'))\end{aligned} \tag{5.6}$$

This bi-condition captures both external and internal inconsistency. Internal inconsistency occurs if input and output are treated differently by the same system. Malone, Kirkpatrick, Mallory, Eike, Johnson and Walker provide examples of this in their review of nuclear reactor control rooms [202]. For instance, two pumps were represented by monitor scales that were identical except that one was read at ten times the value of the other. The pumps, **o** and **o′**, were inconsistent because they were not represented by the same display, **od**. External inconsistency occurs if different systems handle input and output differently. For example, considerable instruction was required before control room staff could operate Kershner, Gebhard and Silverman's iconic displays for nuclear reactors [171]. Inconsistency complicated the task of operating this interface because coolant pumps, **o** and **o′**, were not represented by the textual displays, **od**, that were presented by existing systems.

5.2.1 Image Inconsistency

The **inconsistent** (5.6) predicate can be re-written, see Appendix E.1, to clarify particular forms of inconsistency:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{inconsistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \Diamond(\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o}', \mathbf{om}) \wedge \\ &\quad \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o}', \mathbf{os}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}') \Rightarrow \\ &\quad \neg\, (\mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o}', \mathbf{od}))\end{aligned} \tag{5.7}$$

This bi-condition can be expressed in a more tractable form by introducing a predicate that is true if and only if control system objects, **o** and **o′**, are in the same state, **os**, in the present interval:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{os} \in \mathbf{Os} : \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \Leftrightarrow \\ &\quad \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o}', \mathbf{os})\end{aligned} \tag{5.8}$$

The bi-conditions **same_object_view** (5.9) and **same_object_methods** (5.10) are introduced in a similar manner for elements of **Od** and **Om**. These predicates are incorporated into **inconsistent** (5.7) as follows:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{inconsistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \Diamond(\mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \wedge \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \wedge \\ &\quad \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}') \Rightarrow \neg\, \mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}))\end{aligned} \tag{5.11}$$

The $\Diamond$ operator provides human factors and systems engineers with a means of representing the observation, made in Section 5.1.2, that consistency can eventually be violated by modification and re-design. For instance, Kletz describes a chemical processing plant that presented pumps using serial numbers starting at J1000 [179]. After a period of time the unit reference number allocation was changed and equipment was numbered from JA1000. Considerable confusion resulted when operators were forced to distinguish between equipment assigned JA or J reference numbers. Inconsistency eventually occurred when the new numbering scheme was introduced. Pumps in the same state, *same_object_state(o,o′,os)* accessed by the same methods, *same_object_methods(o,o′,om)* were represented using different numbering schemes, $\neg$ *same_object_view(o,o′,od)*.

5.2.2 Resolving Image Inconsistency

The previous bi-condition, **inconsistent** (5.11), illustrates how designers could use interval temporal logic to represent particular forms of inconsistency. This formalism might also be used to represent solutions to such problems. Designers must ensure that if objects share the same state and methods then they also share the same image. If this constraint had been adopted as a design requirement then the problems observed by Kletz could have been avoided. J serial numbers, **od**, could always have been used to present pumps, **o** and **o′**:

$$\forall \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists \mathbf{od} \in \mathbf{Od} : \mathbf{image\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{od}) \Leftrightarrow \exists \mathbf{om} \in \mathbf{Om}, \exists \mathbf{os} \in \mathbf{Os}$$
$$\Box(\mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \wedge \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \Rightarrow$$
$$\mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od})) \tag{5.12}$$

It is important to note that there is a trade-off between consistency and discrimination in many control systems. For instance, designers might exploit the previous bi-condition to require that the same numbering scheme, **od**, is used to represent all pumps, **o** and **o′**, irrespective of their source of manufacture or age. This would hide information that is frequently vital if operators are to identify particular objects [36]. Alternatively, different numbering schemes could be used to reflect differences in **o** and **o′**. If this were done then the formal analysis suggests that designers might anticipate the consequences of inconsistency: skill dependency and unpredictability. Operators trained to use other systems might fail to predict the consequences of their interaction if they did not understand that JA numbers referred to new, more reliable, plant whilst J prefixes referred to older equipment.

5.2.3 State Inconsistency

State inconsistency occurs if objects share the same image and methods but are not in the same state. For instance, two pumps, **o** and **o′**, might eventually be presented using an icon showing normal operation, **od**. They could be operated through the same start and stop methods, **om**, but they might not both be operating normally, **os**. It would be extremely difficult for operators to predict the effects of their commands upon objects that eventually looked identical but which were not in the same state. Designers could represent this problem by applying the axioms and theorems of interval temporal logic, see Appendix E.2, to **inconsistent** (5.6):

$$\exists \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall \mathbf{om} \in \mathbf{Om}, \forall \mathbf{os} \in \mathbf{Os}, \forall \mathbf{od} \in \mathbf{Od} :$$

$$\begin{aligned}&\mathbf{inconsistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\Diamond(\mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \wedge \mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}) \wedge \\ &\neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}') \Rightarrow \neg\, \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}))\end{aligned} \tag{5.13}$$

The dangers of this form of inconsistency are illustrated by Pólya's [238] concept of non-sufficient reason, introduced in Section 3.2.1. If a user does not have sufficient reason to doubt that things are different when solving a problem then they will treat them as if they were the same. Reason argues that operators assume objects to be in the same state if they look identical and can be operated in a similar fashion [252]. Such assumptions threaten the safety of many interactive control systems. For example, the Bhopal accident can be interpreted as a consequence of incorrect assumptions about the state of control system objects. As with many of the accidents cited in this thesis, it is difficult to find an impartial account of the causes of the Bhopal disaster. Reports issued by the operators, Union Carbide [311], and by the Council on International and Public Affairs [215] embody a clear bias. There is, however, some agreement that an untrained plant superintendant was required to pump methyl-isocyanate into a number of tanks. The state of one tank eventually became inconsistent with those of similar components. It sprang a leak which prevented it from repressurising, $\neg$ *same_object_state(o,o′,os)*. The problem went undetected and warnings were not presented; the leaking tank appeared to be in the same state as all the others, *same_object_view(o,o′,od)*. Automatic cut-outs were not available and so the operator continued to apply the same pumping methods to all tanks, *same_object_methods(o,o′,om)*, causing more methyl-isocyanate to leak into the environment.

5.2.4 Resolving State Inconsistency

The formal analysis of the causes of inconsistency helps to clarify techniques that designers could exploit to support the development of consistent control systems. For instance, state inconsistency might be avoided if and only if it is always the case that if tanks, **o** and **o′**, have the same image, **od**, and can be operated by the same methods, **om**, then they are in the same state, **os**:

$$\begin{aligned}&\forall\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{os} \in \mathbf{Os} : \mathbf{state\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \Leftrightarrow \exists\, \mathbf{om} \in \mathbf{Om}, \exists\, \mathbf{od} \in \mathbf{Od} \\ &\quad \Box(\mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \wedge \mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}) \Rightarrow \\ &\quad \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}))\end{aligned} \tag{5.14}$$

Designers must integrate human factors and systems engineering in order to satisfy the requirements imposed by this bi-condition. For instance, systems engineering might have been used to provide the additional sensors necessary to detect leaking tanks in the Bhopal plant. The costs of doing this led process management to rely upon human factors solutions. Social ergonomics was applied in a very ad hoc way to draft operating instructions. These were intended to ensure the safety of the production process; it was assumed that periodic maintenance would check the integrity of the methyl-isocyanate tanks. Such assumptions are reminiscent of those made by the management of the Chernobyl Unit Four reactor, discussed in Section 3.2.2. At Bhopal, systems engineering might have provided maintenance logging systems as low-cost alternatives to automated sensing devices. These could

have recorded the results of previous inspections and might have scheduled maintenance activities. Such systems would have reduced the administrative burdens upon plant management and might have avoided the ad hoc application of human factors engineering.

5.2.5 Method Inconsistency

Method inconsistency occurs if eventually two objects have the same display and state but are not accessed through the same methods. Designers might apply the axioms and theorems of interval temporal logic to **inconsistent** (5.6), see Appendix E.3, in order to represent this form of inconsistency:

$$\begin{aligned}&\exists\,\mathbf{o},\mathbf{o}' \in \mathbf{O}, \forall\,\mathbf{om} \in \mathbf{Om}, \forall\,\mathbf{os} \in \mathbf{Os}, \forall\,\mathbf{od} \in \mathbf{Od}:\\ &\quad \mathbf{inconsistent}(\mathbf{o},\mathbf{o}',\mathbf{om},\mathbf{os},\mathbf{od}) \Leftrightarrow\\ &\quad \Diamond(\mathbf{same\_object\_view}(\mathbf{o},\mathbf{o}',\mathbf{od}) \wedge \mathbf{same\_object\_state}(\mathbf{o},\mathbf{o}',\mathbf{os}) \wedge\\ &\quad \neg\,\mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\,\mathbf{same\_object\_methods}(\mathbf{o},\mathbf{o}',\mathbf{om}))\end{aligned} \tag{5.15}$$

For instance, many pumps are protected by blow-back sensors which automatically halt operation if the pressure in their outlet increases over a predetermined level. Kletz describes how these safety features introduced new methods into the operation of a pump [179]. Users relied upon the new protection systems to stop pumping automatically while they performed other duties. The control system used the same image to present information about pumps whether or not they were protected by additional safety features; *same_object_view(o,o′,od)*. Failure in the sensing equipment meant that the protected pump entered the same state as all the other pumping equipment; *same_object_state(o,o′,os)*. Users did not operate this pump in the same manner as the other pumps, ¬ *same_object_methods(o,o′,om)*. They relied upon the safety features to stop it. Kletz describes how a tank was filled beyond its capacity because the protection systems failed to halt the pump.

5.2.6 Resolving Method Inconsistency

The provision of additional safety features led to the introduction of inconsistent operating methods in Kletz's system. Paradoxically these features threatened safety because users relied too much on blow-back protection. This provides an example of the flexible task allocation, described in Section 2.1.3. Operators optimised their finite cognitive and perceptual resources by allocating control tasks to automated systems. Problems arose because this allocation had not been intended by the engineers who had developed the automated equipment. In other words, systems engineering had provided additional features without considering the human factors of control. Such problems could be avoided if additional safety features are not introduced. Designers might always ensure that if pumps, **o** and **o′**, are in the same state, **os**, and have the same image, **od**, then they are operated by the same methods:

$$\begin{aligned}&\forall\,\mathbf{o},\mathbf{o}' \in \mathbf{O}, \exists\,\mathbf{om} \in \mathbf{Om} : \mathbf{method\_consistent}(\mathbf{o},\mathbf{o}',\mathbf{om}) \Leftrightarrow \exists\,\mathbf{os} \in \mathbf{Os}, \exists\,\mathbf{od} \in \mathbf{Od}\\ &\quad \Box(\mathbf{same\_object\_view}(\mathbf{o},\mathbf{o}',\mathbf{od}) \wedge \mathbf{same\_object\_state}(\mathbf{o},\mathbf{o}',\mathbf{os}) \Rightarrow\\ &\quad \mathbf{same\_object\_methods}(\mathbf{o},\mathbf{o}',\mathbf{om}))\end{aligned} \tag{5.16}$$

This discussion of consistency has demonstrated that human factors and systems engineers might exploit interval temporal logic to represent techniques which are intended to support principles other than predictability. In particular, this notation has been used to represent the way in which inconsistency can eventually occur at any point during interaction. This is apparent from the use of the $\Diamond$ operator in **inconsistent** (5.11, 5.13, 5.15). Interval temporal logic has also been used to represent the persistent nature of solutions for this problem. This is apparent from the use of the $\Box$ operator in **image_consistent** (5.12), **state_consistent** (5.14) and **method_consistent**(5.16). The logic has not, however, been used to assess the impact of inconsistency upon predictability. The following section makes good this omission.

5.3 Predictability And Message Passing

Message passing provides the computational model associated with the object oriented architecture. A message can be dispatched by one object and received by another which responds by invoking one of its methods. This, in turn, causes an internal state transition in the recipient. Designers could represent this computational model by using the elements of a set, $\mathbf{M}$, to describe all the messages that can be sent to the objects in a control system. It is important to re-emphasise the point made in Section 5.2 that operator input can also be viewed as messages sent to control system objects. For instance, the effect of a message, $\mathbf{m}$, to evacuate a tank object, $\mathbf{o}$, would transform it from the full state, $\mathbf{os}$, to the empty state, $\mathbf{os}'$:

$$\forall\, \mathbf{o} \in \mathbf{O}, \forall\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{os}' \in \mathbf{Os} : \mathbf{message\_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \quad (5.17)$$

This predicate can be used to represent the causes of unpredictability in object oriented control systems. It is difficult for operators to determine the consequences of their interaction, $\mathbf{os}$ or $\mathbf{os}'$, if at some time during interaction a message, $\mathbf{m}$, potentially affects different objects, $\mathbf{o}$, and $\mathbf{o}'$:

$$\begin{aligned}
&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{os}, \mathbf{os}' \in \mathbf{Os} : \\
&\quad \mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}, \mathbf{os}') \Leftarrow \exists\, \mathbf{os}'', \mathbf{os}''' \in \mathbf{Os} \\
&\quad \Diamond(\mathbf{message_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}'', \mathbf{os}) \wedge \mathbf{message_effect}(\mathbf{o}', \mathbf{m}, \mathbf{os}''', \mathbf{os}') \wedge \\
&\quad \neg\, \mathbf{same_object}(\mathbf{o}, \mathbf{o}')) \qquad (5.18)
\end{aligned}$$

For instance, Kirkpatrick and Mallory describe how input, $\mathbf{m}$, to increase the flow of coolant to the Three Mile Island Unit Two reactor could be applied to two different pumps, $\mathbf{o}$ and $\mathbf{o}'$ [176]. During the emergency one of these pumps had to be closed down. Substitution errors, described in Sections 3.1.2 and 4.1.1, occurred when operators attempted to increase the action of the disabled pump, $\mathbf{o}$, instead of the functioning pump, $\mathbf{o}$. Users failed to predict the consequences of their input being sent to the wrong object and, in consequence, the coolant system was damaged.

5.3.1 Image Inconsistency And Image Unpredictability

Image inconsistency exacerbates the problems of unpredictability if a message can affect objects that have the same state and methods but are presented by different images. Temporal logic provides a means of representing this problem for the

human factors and systems engineering of interactive control systems. The tractability of the following bi-condition can be increased if **same_object_state**$(\mathbf{o}, \mathbf{o}', \mathbf{os})$, **same_object_methods**$(\mathbf{o}, \mathbf{o}', \mathbf{om})$ and **same_object_view**$(\mathbf{o}, \mathbf{o}', \mathbf{od})$ are replaced by the predicate **inconsistent** (5.6). This is not done in order to represent the relationship between image inconsistency and unpredictability:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{unpredictable\_image}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{od}) \Leftrightarrow \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{os}', \mathbf{os}'' \in \mathbf{Os} \\ &\quad \Diamond(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}', \mathbf{os}'') \wedge (\mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \wedge \\ &\quad \mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \Rightarrow \neg\, \mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}))) \end{aligned} \quad (5.19)$$

Section 5.2.1 described a chemical plant in which pumps, $\mathbf{o}$ and $\mathbf{o}'$, were not all displayed using the same J serial numbers, **od**. Operators assumed that the J and JA serial numbers denoted differences in the methods, **om**, and state, **os**, of $\mathbf{o}$ and $\mathbf{o}'$. Considerable training was required in order to overcome these initial assumptions. Differences in the presentation of these control system objects provided sufficient reason for users to predict different effects, $\mathbf{os}'$ and $\mathbf{os}''$, for input, **m**, issued to either $\mathbf{o}$ or $\mathbf{o}'$. In fact, messages had identical effects on $\mathbf{o}$ and $\mathbf{o}'$. Their different images did not indicate any differences in state or behaviour. Operators might have made more accurate predictions about the effects of their input, **m**, upon objects, $\mathbf{o}$ and $\mathbf{o}'$, if they had been displayed in the same way:

$$\begin{aligned}&\forall\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{predictable\_image}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{od}) \Leftrightarrow \exists\, \mathbf{os}, \mathbf{os}' \in \mathbf{Os} \\ &\quad \Box(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \mathbf{image\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{od})) \end{aligned} \quad (5.20)$$

This predicate illustrates the divide between abstract requirements and specifications that provide a basis for implementation. Abstractions, such as **od**, do not provide detailed information about the graphical and textual images that might be presented to system operators. Section 7.4 will describe techniques that designers can use to introduce gradually the graphical details necessary to clarify the semantics of relations such as **same_object_view**$(\mathbf{o}, \mathbf{o}', \mathbf{od})$.

5.3.2 State Inconsistency And State Unpredictability

State unpredictability occurs if eventually a message can be bound to objects in different states. For instance, Section 5.2.4 described how the Bhopal accident was caused by an operator issuing commands to methyl-isocyanate tanks, $\mathbf{o}$ and $\mathbf{o}'$, that were not in the same state, **os**. The effect of input to repressurise these tanks was safeq, $\mathbf{os}'$, for those which were intact but was unsafe, $\mathbf{os}''$, for those that were leaking:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{os} \in \mathbf{Os} : \\ &\quad \mathbf{unpredictable\_state}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}) \Leftrightarrow \forall\, \mathbf{od} \in \mathbf{Od}, \forall\, \mathbf{om} \in \mathbf{Om}, \exists\, \mathbf{os}', \mathbf{os}'' \in \mathbf{Os} \\ &\quad \Diamond(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}', \mathbf{os}'') \wedge (\mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om}) \wedge \\ &\quad \mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}) \Rightarrow \neg\, \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}))) \end{aligned} \quad (5.21)$$

Designers could reduce the consequences of this problem by ensuring that objects enter an error state if a message is bound to a method which jeopardises safety. For

instance, a command to repressurise a punctured tank might have raised an error. Alternatively, designers might specify that messages are always bound to objects in the same state:

$$\begin{aligned}&\forall\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{os} \in \mathbf{Os} : \\ &\quad \mathbf{predictable\_state}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}) \Leftrightarrow \exists\, \mathbf{os}', \mathbf{os}'' \in \mathbf{Os} \\ &\quad \Box(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}', \mathbf{os}'') \wedge \mathbf{state\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{os}))\end{aligned} \quad (5.22)$$

The previous bi-condition again illustrates the gulf of detail between predictability requirements expressed in terms of high-level abstractions, such as **o** and **o′**, and particular objects, such as methyl-isocyanate tanks. It is important to emphasise that principles are not intended to replace the domain knowledge required when developing a detailed design for a particular interface.

5.3.3 Method Inconsistency And Method Unpredictability

The effects of a message, **m**, are **unpredictable** (5.18) if it can eventually be bound to different objects, **o** and **o′**. Inconsistency exacerbates this problem because **o** and **o′** need not possess the same methods:

$$\begin{aligned}&\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{om} \in \mathbf{Om} : \\ &\quad \mathbf{unpredictable\_method}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{om}) \Leftrightarrow \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od}, \exists\, \mathbf{os}', \mathbf{os}'' \in \mathbf{Os} \\ &\quad \Diamond(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}', \mathbf{os}'') \wedge (\mathbf{same\_object\_view}(\mathbf{o}, \mathbf{o}', \mathbf{od}) \wedge \\ &\quad \mathbf{same\_object\_state}(\mathbf{o}, \mathbf{o}', \mathbf{os}) \Rightarrow \neg\, \mathbf{same\_object\_methods}(\mathbf{o}, \mathbf{o}', \mathbf{om})))\end{aligned} \quad (5.23)$$

Section 5.2.5 described non-return valves that have been developed by systems engineering to prevent pump blow-back from a delivery source. Input, **m**, to a protected pump, **o**, could be bound to a method that activates this safety feature, **os′**. This message might also be bound to a method that generates an error condition, **os″**, for an unprotected pump. Operators could more accurately predict the effects of their intervention if and only if it is always the case that input, **m**, which can be issued to objects, **o** and **o′**, invokes one of a similar set of methods, **om**:

$$\begin{aligned}&\forall\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{om} \in \mathbf{Om} : \\ &\quad \mathbf{predictable\_method}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{om}) \Leftrightarrow \exists\, \mathbf{os}, \mathbf{os}' \in \mathbf{Os} \\ &\quad \Box(\mathbf{unpredictable}(\mathbf{o}, \mathbf{o}', \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \mathbf{method\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}))\end{aligned} \quad (5.24)$$

A number of different techniques might be exploited in order to satisfy the requirements that are imposed by this predicate. For instance, systems engineering might provide blow-back protection for all the pumping equipment in a plant, $\mathbf{method\_consistent}(\mathbf{o}, \mathbf{o}', \mathbf{om})$. Operators could predict that all pumps provided this safety feature. A human factors alternative might be to draft regulations which ensure that all pumps are operated as if they were unprotected.

5.4 Consistency Through Type Instantiation

A number of authors have advocated object orientation as a means of developing consistent control systems. Lazarev's design for a steel furnace control system exploits this architecture because its chief power is "uniformity" [190]. Similar reasons

inspired the use of the Smalltalk object oriented programming language for the implementation of operator assistants in the United States' National Aeronautics and Space Administration's satellite ground-control system [262]. Chin and Chanson describe a range of applications that exploit this architecture to support "uniform access" to the objects which compose distributed control systems [65]. Finkelstein has argued that consistency or "likeness in objects" is the result of family resemblances which can be derived from class hierarchies [99]. Designers might exploit interval temporal logic to clarify the support that the object oriented architecture provides for consistency. Section 5.3 established the relationship between this principle and predictability. Human factors and systems engineers could, therefore, exploit this analysis in order to justify object orientation as a means of supporting operator predictions about the effects of their commands. In order to demonstrate the validity of these assertions, a set, **Cl**, is introduced; its elements are the classes of control objects. Classes provide templates for similar objects. For instance, elements of the class of pumps might offer methods to start and stop operation. The following predicate is true for an object state, **os**, which is legal for an instance of a class, **cl**, in the present interval:

$$\forall \mathbf{cl} \in \mathbf{Cl}, \exists \mathbf{os} \in \mathbf{Os} : \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}) \tag{5.25}$$

The relations **legal_view** (5.26) and **legal_methods** (5.27) are introduced in a similar fashion between elements of **Cl** and elements of **Od** and **Om**. A class can be described in terms of the states, displays and methods that its instances possess:

$$\begin{aligned}&\forall \mathbf{cl} \in \mathbf{Cl}, \exists \mathbf{om} \in \mathbf{Om}, \exists \mathbf{os} \in \mathbf{Os}, \exists \mathbf{od} \in \mathbf{Od} : \mathbf{class\_defined}(\mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})\end{aligned} \tag{5.28}$$

Objects, **o**, possess the type of class **cl** if and only if they have the state, methods and image of that class:

$$\begin{aligned}&\forall \mathbf{o} \in \mathbf{O}, \exists \mathbf{cl} \in \mathbf{Cl}, \exists \mathbf{om} \in \mathbf{Om}, \exists \mathbf{os} \in \mathbf{Os}, \exists \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{class\_type}(\mathbf{o}, \mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{class\_defined}(\mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od})\end{aligned} \tag{5.29}$$

A relation is also introduced between objects and their class:

$$\forall \mathbf{o} \in \mathbf{O}, \exists \mathbf{cl} \in \mathbf{Cl} : \mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \tag{5.30}$$

Reisner argues that consistency depends upon family resemblances between sets of similar objects [256]. Interval temporal logic provides a notation in which to explain why object oriented design supports this form of consistency. Designers could exploit type instantiation to ensure that objects always share the same attributes as other instances of their class:

$$\begin{aligned}&\forall \mathbf{o} \in \mathbf{O}, \exists \mathbf{cl} \in \mathbf{Cl}, \exists \mathbf{om} \in \mathbf{Om}, \exists \mathbf{os} \in \mathbf{Os}, \exists \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{consistent}(\mathbf{o}, \mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{class\_type}(\mathbf{o}, \mathbf{cl}, \mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}))\end{aligned} \tag{5.31}$$

The formal analysis provides a means of representing the constraints that designers must satisfy if they are to achieve consistency through object orientation. For instance, the design of object oriented control systems frequently involves the specialisation and extension of classes. These derived classes inherit some of the attributes

of their parent but need not implement them all [209]. Instances of a derived class are not **consistent** (5.31) with instances of their parent. This is captured by the previous bi-condition because **is_a** (5.30) is not transitive. The consistency provided by type instantiation is also jeopardised by re-classification [231]. Designers can use this technique to ensure that eventually an object is not an instance of its original class. For instance, a tank might be re-classified as a pump. It is hypothesised that such techniques are likely to have profound consequences for the predictability of an interactive control system. This re-classification would violate the $\Box$ quantification of **consistent** (5.31). Some programming languages provide constructs, such as `renames` in Ada, which enable designers to override instantiated methods [26]. Again this violates **consistent** (5.31); objects would not offer the same methods, **om**, as other instances of their class, **cl**. Designers cannot avoid using such mechanisms when developing non-trivial control systems. The use of interval temporal logic helps to identify the trade-offs that exist between dynamic classification, method overriding and consistency.

5.4.1 Image Consistency And Instantiation

Type instantiation provides designers with a means of developing consistent interfaces. An object **o** is image consistent with a class **cl** if it is always the case that **o** is an instance of **cl** and has a legal image for an instance of that class. For example, Hess describes a system which required that operators press a number of buttons in order to confirm their observation of process information [146]. This is an example of the handshaking protocol, described in Section 4.3.2. All of the buttons included the word 'clear', except for one that was labelled 'message acknowledged'. Human factors research found that, under the pressures of flight, pilots failed to recognise that this button belonged to the same class as all the others that were labelled 'clear'. Operators failed to predict common functionality because of the lack of visual similarity. The image, **od**, of this object, **o**, would have been consistent if it had been labelled 'message clear'. Designers might apply the axioms and theorems of interval temporal logic to **consistent** (5.31), see Appendix E.4, in order to represent the way in which type instantiation supports image consistency:

$$\begin{aligned}&\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{instantiate\_image\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{od}) \Leftrightarrow \\ &\quad\quad \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}))\end{aligned} \tag{5.32}$$

The formal analysis encourages human factors and systems engineers to ensure that operators can recognise the images of similar objects as representing instances of the same class. This observation is far from novel; graphical consistency lies at the heart of equipment labelling specifications produced by the International Standards Organisation [327]. At first sight this analysis might appear to do nothing more than restate the argument for image consistency. This is not the case. In particular, it is important to note that image consistency is directly supported by the object oriented architecture. All instances of a class offer the same legal images. The designer, therefore, achieves consistency at a minimal cost, providing they can construct appropriate class structures. The selection of the objects that are to be grouped within a class, therefore, determines the consistency exhibited by an interface.

Figure 5.1: A diagram of a flow valve

5.4.2 State Consistency And Instantiation

Designers might exploit the theorems of interval temporal logic, see Appendix E.4, in order to represent the way in which type instantiation supports state consistency. An object, **o**, is state consistent with a class, **cl**, if and only if it is always the case that **o** is an instance of **cl** and its state, **os**, is valid for that class:

$$\begin{aligned}&\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{os} \in \mathbf{Os} : \\ &\quad \mathbf{instantiate\_state\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{os}) \Leftrightarrow \\ &\quad \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}))\end{aligned} \tag{5.33}$$

For instance, Kletz describes how an ethylene plant control system used readings from solenoids to detect oxygen passing through a flow valve [179]. Figure 5.1 illustrates the composition of these flow valve objects. Section 7.4.3 will describe techniques that designers can employ to represent the structure of these composite objects. For now it is sufficient to observe that consistent states for an instance of the class of flow valves were: solenoid de-energised and trip valve closed, or solenoid energised and trip valve open. The flow of oxygen through one of the solenoids was halted by a wasps' nest in a vent. As a result the state, **os**, of a flow valve, **o**, eventually became inconsistent with those of its class, **cl**. The solenoid was de-energised and its trip valve was open. Oxygen continued to flow into the process even though the solenoid readings indicated that the trip valve was closed. Operators failed to discover that a potentially disastrous mixture of gasses was being supplied to the process. Human factors and systems engineers might guard against such

errors by enumerating potential states, **os**, for instances of the class of flow valves, **cl**. Systems engineers must ensure that the readings supplied by application sensors provide an accurate reflection of the state of the physical objects which operators must control, we shall return to this theme in Section 6.1.3. Human factors engineers must ensure that sufficient information is presented about the state of an object for users to detect error states, such as that caused by the wasps' nest.

5.4.3 Method Consistency And Instantiation

Simpson describes how an engineering company fitted blow-back valves and pressure sensors to upgrade the safety equipment around a number of propylene pumps [286]. Shortly afterwards an operator responded to a major leak by ordering two foremen to manually halt the pumps. Fortunately, the leak did not fire. Later investigations revealed that staff had forgotten about the automated support. In other words the provision of safety equipment had eventually made the operating methods of control objects inconsistent with the class of pumps in the plant. It is possible to apply the axioms and theorems of interval temporal logic, see Appendix E.4, to represent the support that type instantiation provides for method consistency. Operator error might be avoided by ensuring that instances, **o**, of the class of pumps, **cl**, are operated by the same methods, **om**. Safety equipment could be fitted to all pumps so that operators can assume that there is an alternative to manual intervention:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{om} \in \mathbf{Om} : \mathbf{instantiate\_method\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{om}) \Leftrightarrow \\ \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \qquad (5.34)$$

Human factors and systems engineers might exploit type instantiation to ensure that objects share the attributes of their class. It is important to note, however, that this does not guarantee that all operators will view the resulting interface as consistent. For instance, users cannot exploit class consistency unless they know which class an object belongs to and what methods are defined for instances of that class. Consistency and predictability are "user-referenced" principles [304]. What one operator (or designer) views as consistent can appear to be inconsistent to another operator (or designer). Chapter 7 will argue that human factors and systems engineers can use prototypes to glean the empirical evidence necessary to support assertions about the consistency and predictability of an interactive control system.

5.5 Conclusions

Control objects are inconsistent if they do not always share the same methods, states and displays. Image inconsistency exacerbates unpredictability if input can eventually be issued to different objects that are not presented in the same way. State inconsistency exacerbates unpredictability if input can eventually be bound to different objects in different states. Method inconsistency exacerbates unpredictability if input can eventually be bound to different methods of different objects.

It has been argued that both human factors and systems engineering must be recruited in order to develop consistent and predictable control systems. Object orientation supports the integration of both engineering disciplines because type

instantiation provides a means of achieving these common objectives. Rather than specifying common consistency criteria, designers might advocate object orientation as a common structure for the detailed design of consistent control systems. It has not, however, been argued that type instantiation guarantees consistency. Trade-offs have been identified been object oriented techniques, such as dynamic classification and method overriding, and principles, such as consistency and predictability. Human factors and systems engineers must have a clear understanding of these trade-offs if they are to exploit the benefits and avoid the costs of object oriented development.

This chapter has argued that designers might use interval temporal logic to assess the strengths of development architectures for principles, such as predictability and consistency. Chapter 6 argues that human factors and systems engineers might also exploit this notation to represent techniques that are intended to avoid the weaknesses of these architectures.

Chapter 6

Break-Down

> "Objects and properties are not inherent in the world, but arise only in an event of breaking down in which they become present-at-hand" (Winograd and Flores, [326]).

6.1 Introduction

This chapter argues that principles provide human factors and systems engineers with criteria against which to assess the weaknesses of development architectures. In particular, it is argued that:

- object oriented systems are susceptible to break-down, this occurs when control system objects do not accurately represent process components;
- break-down leads to unpredictability because operators are unlikely to make correct predictions about the effects of their intervention on unfamiliar physical components;
- human factors and systems engineering must be integrated in order to avoid the problems caused by break-down.
- users must view the information presented to them with a degree of scepticism if they are to accurately predict the consequences of their interaction.

It is concluded that human factors and systems engineering must be integrated in order to avoid the problems caused by break-down.

6.1.1 Consequences: Alienation And Unpredictability

Break-down leads to alienation. In other words, operators will cease to trust displays that provide misleading information about application components [22]. For instance, proximity warning systems have been developed to alert pilots if they are about to enter another aircraft's air-space. These systems suffer from the problems of predictive presentation techniques, described in Section 2.1.3. They frequently fail to detect approaching aircraft. In such circumstances, the objects presented on the display do not correspond to the planes that are visible from the cockpit. Pilots do not trust the information presented by many of these systems and, in consequence, most have been abandoned [295].

Break-down prevents operators from successfully predicting the effects of their commands. For instance, in 1989 an explosion in the tail engine of a McDonnell-Douglas DC-10 robbed the crew of all cockpit instrumentation [84]. The objects presented by the displays ceased to provide accurate information about the aircraft. The pilot retained limited throttle control but had great difficulty in predicting the consequences of using it. Fortunately, he did manage to land the aircraft.

6.1.2 Causes: Different States; Images And Methods

Break-down can occur if the objects presented by a control system do not accurately represent the state of process components. This is illustrated by an incident in which the systems engineering of an automated avionics application failed to preserve the safety of an aircraft. Norman describes how the automatic pilot of a Boeing 747 compensated for a slow loss of power from its outer-right engine [228]. Avionics displays indicated that the engines were in a normal state. They were, in fact, suffering a serious malfunction. Eventually, the automatic pilot reached a point where it could not correct the yaw and the plane rolled into a thirty-one thousand feet dive. Break-down occurred when the crew discovered that their avionics display did not accurately represent the state of the engine.

Break-down can also occur if the objects presented by a control system do not look like application components. This is illustrated by human factors research conducted by the United States' National Aeronautics and Space Administration's Faultfinder project [8]. Tests were conducted to evaluate the cognitive and perceptual demands imposed upon pilots by a number of different avionics display formats. It was found that pilots make less accurate observations and take more time to respond to error messages if the objects presented by avionics displays do not resemble the physical components of their aircraft. These results can be explained in terms of break-down; each time an error occurred pilots were forced to consider the relationship between the information presented by their displays and the underlying components of their aircraft.

Break-down occurs if control systems are not operated in the same way as the physical components of an application process. This is illustrated by the United States' Federal Aviation Authority's concern that pilots are being distanced from the underlying mechanics of their aircraft [295]. Fuel management systems have automated many of the tasks that were previously performed by flight engineers. Much of the information used by these systems is hidden from the crew in order to reduce display clutter. Break-down occurs when operators are forced to perform low-level fuel management tasks. They are made aware of the differences that exist between the operating methods of automated applications and of manual avionics systems.

6.1.3 Solution: Object Conformance

A number of techniques could be used to avoid the consequences of break-down. For instance, human factors engineers might attempt to ensure that the objects presented by a control system always look like application components. Systems engineering could be recruited to ensure that the states of control system objects always accurately represent the states of physical components. The following sections

use interval temporal logic to represent the requirements that must be satisfied in order to exploit such conformal presentation techniques. This formalisation provides a basis upon which to assess the support that these techniques offer for principles, such as predictability.

6.2 Break-Down And Image Failure

It is important to have a clear understanding of the causes of break-down before attempting to identify solutions to this problem. These causes can be represented by introducing a set, **Pc**, whose elements are the physical components of an application. This set must not be confused with **O**. The elements of **O** should be thought of as abstract representations of the software components that would be developed during implementation in an object oriented programming language. In order to maintain this distinction the elements of the set **O** will be referred to as control system objects. Elements of the set **Pc** will be referred to as physical components. A predicate is introduced between control system objects and the physical components that they represent in the present interval. For example, an instance, **o**, of a class of pumps could be used by an avionics application to represent a fuel pump, **pc**:

$$\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc} : \mathbf{represents}(\mathbf{o}, \mathbf{pc}) \tag{6.1}$$

A relation is also introduced between the state, **os**, of a control system object and those process components that it conforms to in the present interval. For instance, a pump, **pc**, that is transferring fuel from one tank to another is represented by a control system object which is pumping, **os**:

$$\forall\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{os} \in \mathbf{Os} : \mathbf{state\_conforms}(\mathbf{pc}, \mathbf{os}) \tag{6.2}$$

The relations **method_conforms** (6.3) and **image_conforms** (6.4) are introduced in a similar fashion between physical components and elements of **Om** and **Od** respectively. Break-down occurs if and only if eventually a control system object, **o**, represents a physical component, **pc**, and its attributes, **om**, **os** and **od**, do not conform to the appearance, state and behaviour of that physical component. If the reader compares the following bi-condition with **class_defined** (5.28) in Section 5.4 it will readily be apparent that conformance between the attributes of a control system object and a physical component imposes similar constraints to consistency between the attributes of a control system object and its class:

$$\begin{aligned} &\forall\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{om} \in \mathbf{Om}, \exists\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{conforms}(\mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \mathbf{method\_conforms}(\mathbf{pc}, \mathbf{om}) \wedge \\ &\quad \mathbf{state\_conforms}(\mathbf{pc}, \mathbf{os}) \wedge \mathbf{image\_conforms}(\mathbf{pc}, \mathbf{od}) \end{aligned} \tag{6.5}$$

The Lockheed Corporation have recently developed a range of avionics displays that illustrate the potential for break-down in object oriented control systems [291]. Figure 6.1 sketches one of these interfaces. The measuring scales, **o**, represent an aircraft fuel pump, **pc**. Break-down would occur if a pilot realised that the state, **os**, of the scales did not always accurately represent the state of the fuel pump. Similarly, break-down would occur if eventually a pilot realised that the methods, **om**, used to operate the control system object were not the same as the commands issued to

Figure 6.1: An object display for aircraft fuel distribution

the physical component. Finally, break-down would occur if a pilot recognised that the image, **od**, of the scales did not always look like the process component. For instance, the scales might not show that smoke was pouring from an over-heated pump. The causes of break-down are represented by the following predicate:

$$\begin{aligned}&\exists\,\mathbf{o} \in \mathbf{O}, \exists\,\mathbf{pc} \in \mathbf{Pc}, \forall\,\mathbf{om} \in \mathbf{Om}, \forall\,\mathbf{os} \in \mathbf{Os}, \forall\,\mathbf{od} \in \mathbf{Od} :\\ &\quad \mathbf{break\_down}(\mathbf{o}, \mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow\\ &\quad \neg\,\Box(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}))\end{aligned} \tag{6.6}$$

Human factors and systems engineers could manipulate this predicate in order to represent particular forms of break-down. For instance, break-down occurs if control system objects do not look like the physical components that they are intended to represent. Designers might apply the axioms and theorems of interval temporal logic, see Appendix F.1, to derive a formalisation of this image failure from the predicate **break_down** (6.6):

$$\begin{aligned}&\exists\,\mathbf{o} \in \mathbf{O}, \exists\,\mathbf{pc} \in \mathbf{Pc}, \forall\,\mathbf{om} \in \mathbf{Om}, \forall\,\mathbf{os} \in \mathbf{Os}, \forall\,\mathbf{od} \in \mathbf{Od} :\\ &\quad \mathbf{break\_down}(\mathbf{o}, \mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \Diamond(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge\\ &\quad (\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{method\_conforms}(\mathbf{pc}, \mathbf{om})) \wedge\\ &\quad (\mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{state\_conforms}(\mathbf{pc}, \mathbf{os})) \Rightarrow\\ &\quad \neg\,(\mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{image\_conforms}(\mathbf{pc}, \mathbf{od})))\end{aligned} \tag{6.7}$$

This bi-condition can be made more tractable by introducing a predicate that is true if and only if the methods, **om**, of a control system object, **o**, conform to the operating methods of a physical component, **pc**, in the present interval:

$$\begin{aligned}&\exists\,\mathbf{o} \in \mathbf{O}, \exists\,\mathbf{pc} \in \mathbf{Pc}, \exists\,\mathbf{om} \in \mathbf{Om} : \mathbf{accurate\_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om}) \Leftrightarrow\\ &\quad \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{method\_conforms}(\mathbf{pc}, \mathbf{om})\end{aligned} \tag{6.8}$$

The relations **accurate_state** (6.9) and **accurate_image** (6.10) are also introduced for elements of **Os** and **Od**. These predicates can be incorporated into the previous formalisation of **break_down** (6.7) as follows:

$$\exists\,\mathbf{o} \in \mathbf{O}, \exists\,\mathbf{pc} \in \mathbf{Pc}, \forall\,\mathbf{om} \in \mathbf{Om}, \forall\,\mathbf{os} \in \mathbf{Os}, \forall\,\mathbf{od} \in \mathbf{Od} :$$

Figure 6.2: The horizontal situation indicator for Boeing 757 and 767 aircraft

$$\begin{aligned}&\mathbf{break\_down}(\mathbf{o}, \mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \Diamond(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \\ &\mathbf{accurate\_state}(\mathbf{o}, \mathbf{pc}, \mathbf{os}) \wedge \mathbf{accurate\_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om}) \Rightarrow \\ &\neg\, \mathbf{accurate\_image}(\mathbf{o}, \mathbf{pc}, \mathbf{od}))\end{aligned} \tag{6.11}$$

In case the reader is not convinced that image failure is a problem for commercial systems, Figure 6.2 sketches the horizontal situation indicator currently fitted in Boeing 757s and 767s. The pilot's aircraft is presented using the pointed oblong. The stars represent way-points, or navigational markers. The crew input the co-ordinates of these way-points in order to indicate the intended flight-path. The hexagons and squares, labelled `ELN`, `DLS`, `AST` and `HOM`, represent navigational transmitters. The state and behaviour of all of these control system objects faithfully represents the state and behaviour of physical components; *accurate_state(o, pc, os)* $\wedge$ *accurate_methods(o, pc, om)*. For instance, pilots cannot move navigational transmitters by manipulating the objects represented by their control system. None of these control system objects look like the real-world components that they are intended to represent; $\neg$ *accurate_image(o, pc, od)*. Palmer argues that there are a number of human factors problems with these systems [232]. They encourage substitution errors; pilots believe that they are observing one navigational marker when they are actually monitoring another. For instance, if the crew were tracking their path towards `ANGOO` then they could mistake it for way-points that are presented by similar control system objects, such as `WHYTE` or `SWANY`. Such problems led the crew of a McDonnell-Douglas DC-10 to enter incorrect navigational data prior to take-off. The automated navigation system dutifully flew the programmed course into Mount Erebus in Antarctica [225]. Such errors might be avoided if the images of control system objects provide more accurate views of the physical components that they are intended to represent. This would reduce substitution errors by helping operators to differentiate between application objects. For example, designers could have displayed additional information about the terrain surrounding navigational beacons. This approach is already exploited by the Jeppensen approach charts that

Figure 6.3: A plan-view display for aircraft collision detection

pilots consult in order to familiarise themselves with airport layouts [295].

6.2.1 Image Failure And Predictability

The conditions for image failure were derived from a formalisation of **break_down** (6.6). This suggests that an implementation which presents control system objects that do not resemble physical components would suffer from the problems of break-down: unpredictability and alienation. Interval temporal logic provides a means of representing the causes of this form of unpredictability. It should be noted that designers could introduce **break_down** (6.6) into the following predicate in order to explicitly represent the way in which this problem jeopardises predictability. The decision not to do this is justified by the observation that image failure can cause unpredictability in systems for which **accurate_state**$(\mathbf{o}, \mathbf{pc}, \mathbf{os})$ and **accurate_methods**$(\mathbf{o}, \mathbf{pc}, \mathbf{om})$ are not true:

$$\begin{aligned}
&\exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \\
&\quad \mathbf{unpredictable_object_image}(\mathbf{pc}, \mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \exists\, \mathbf{os}' \in \mathbf{Os} \\
&\quad \Diamond(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{message_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \\
&\quad \neg\, \mathbf{accurate_image}(\mathbf{o}, \mathbf{pc}, \mathbf{od}))
\end{aligned} \tag{6.12}$$

Figure 6.3 provides an example of an interface that is prone to this form of unpredictability. It illustrates a plan-view display for a collision avoidance system, developed by Hart and Wempe [135]. The arrow-head and three parallel dotted lines represent the position and flight-path of the operator's plane. The previous flight-paths of aircraft `BA 453`, `GH 765` and `AL 345` are represented by the dotted lines leading to the squares. The predicted flight-paths of these aircraft are represented by the solid lines leading from the squares. These control system objects, **o**, did look like the physical components because they did not represent aircraft altitude, $\neg$ *accurate_image(o, pc, od).* In consequence, it was difficult for pilots to

Figure 6.4: A perspective display for aircraft collision detection

determine the effects, $\mathbf{os}'$, of input, $\mathbf{m}$, to avoid approaching aircraft, $\mathbf{pc}$, from the data displayed by such objects, $\mathbf{o}$.

6.2.2 Image Conformance

During the 1960s pictorial realism emerged as a major design aim for the presentation of complex control systems [259]. This approach attempted to make the objects that were presented to operators look as similar as possible to process components. Pictorial realism provides a potential solution to the unpredictability caused by image failure. For instance, Ellis and McGreevy have developed displays that present information about the direction and altitude of aircraft [96]. Instead of simple squares, planes are represented by icons that look like individual aircraft. Their perspective presentation format is illustrated by Figure 6.4. It is hypothesised that the additional information provided by such displays could reduce the frequency of substitution errors. This would, in turn, decrease the incidence of break-down. Users are less likely to be surprised by aircraft outside their cockpit if they avoid errors when monitoring the control system objects presented by collision avoidance systems. The requirements for image conformance can be represented by the following predicate. It is always the case that the image, $\mathbf{od}$, used to present a control system object, $\mathbf{o}$, provides an accurate picture of a physical component, $\mathbf{pc}$:

$$\begin{aligned} &\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{od} \in \mathbf{Od} : \mathbf{image\_conformal}(\mathbf{o}, \mathbf{pc}, \mathbf{od}) \Leftrightarrow \\ &\quad \Box\ \mathbf{accurate\_image}(\mathbf{o}, \mathbf{pc}, \mathbf{od}) \end{aligned} \tag{6.13}$$

It seems obvious that collision avoidance systems should present information about aircraft altitudes. It is extremely important that such requirements should emerge from an analysis of predictability. Section 1.5 argued that principles can be used to capture fundamental truths, such as the utility of pictorial realism. As Hart and Wempe's plan-view displays illustrate, these truths have been ignored in the past. It is also important that human factors and systems engineers should question the applicability of such fundamental truths. For instance, pictorial realism is unnecessary for many control tasks. Smith and Ellis demonstrate that pilots routinely implement collision avoidance manoeuvres in two dimensions [**?**]. They prefer to make turns rather than ascents or descents. In such circumstances, plan-view displays support pilot predictions about the effects of their interaction even though control system objects do not look like physical components. Further limitations restrict the utility of image conformance. Many control tasks require the presentation of information that cannot easily be represented by control system objects which look like physical components. It is for this reason that head-up displays project alphanumeric data about the velocity and distance of approaching aircraft onto cockpit canopies [293]. A further factor which limits the application of **image_conformal** (6.13) is that it can be extremely expensive for systems engineers to develop the additional sensing devices that are required in order to provide accurate representations of physical components. This discussion re-emphasises the point, made in Section 5.3.2, that principles are not intended to replace the domain knowledge required when developing a detailed design for a particular interface.

6.3 Break-Down And Method Failure

Carroll and Thomas argue that structuring a system around objects aids both the development and the operation of interactive systems [58]. The real-world counterparts of control system objects are part of the everyday experience of users and designers. The utility of this framework is reduced if eventually control system objects are not operated in the same way as the physical components that they represent. Human factors and systems engineers might formalise the causes of this method failure, see Appendix F.2, by re-writing **break_down** (6.6):

$$\begin{aligned}&\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \\ &\quad \mathbf{break\_down}(\mathbf{o}, \mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow \\ &\quad \Diamond(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{accurate\_state}(\mathbf{o}, \mathbf{pc}, \mathbf{os}) \wedge \\ &\quad \mathbf{accurate\_image}(\mathbf{o}, \mathbf{pc}, \mathbf{od}) \Rightarrow \neg\, \mathbf{accurate\_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om}))\end{aligned} \quad (6.14)$$

Stollings proposes an avionics display, illustrated in Figure 6.5, that provides an example of method failure [294]. The control system objects presented to operators resemble application components, $\mathbf{accurate\_image}(\mathbf{o}, \mathbf{pc}, \mathbf{od})$. Systems engineering was assumed to guarantee that the states of these control system objects corresponded to those of the physical components within the aircraft, $\mathbf{accurate\_state}(\mathbf{o}, \mathbf{pc}, \mathbf{os})$. It was not intended that the pilot should operate the control system objects in exactly the same manner as their physical counterparts, $\neg\, \mathbf{accurate\_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om})$. For instance, the crew could select a number of buttons in order to indicate the source and destination of fuel transfers, a numeric keypad could be used to specify

Figure 6.5: A pictorial status display for avionics

the amount of fuel to be transferred. The crew need not issue the many detailed instructions that avionics used to control the pumps and valves during fuel transfers.

6.3.1 Method Failure And Predictability

Stollings' interface, described in the previous paragraph, illustrates how method failure can be used to simplify the task of operating process components. Pilots need not concern themselves with the detailed commands that are issued by on-board avionics. Human factors and systems engineers must, however, be aware of the consequences of such techniques. Method failure can jeopardise predictability. Eventually, an input message, **m**, issued by an operator to a control system object, **o**, is not bound to a method that is available for an application component, **pc**. Interval temporal logic provides a means of representing the causes of this form of unpredictability:

$$\begin{aligned}
&\exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \exists\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{om} \in \mathbf{Om} : \\
&\quad \mathbf{unpredictable_object_methods}(\mathbf{pc}, \mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{om}) \Leftrightarrow \exists\, \mathbf{os}' \in \mathbf{Os} \\
&\quad \Diamond(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{message_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \\
&\quad \neg\, \mathbf{accurate_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om}))
\end{aligned} \tag{6.15}$$

For instance, the United States' Air Force is currently testing a number of aircraft whose ailerons are operated by computer controlled actuators [305]. These fly-by-wire systems intervene to optimise pilot settings. The methods that are invoked upon control system objects by operator commands are not the same as those which

are available for physical components, $\neg$ **accurate_methods**(**o**, **pc**, **om**). Break-down occurs when pilots realise that their commands are mediated and that they are not directly interacting with the physical components of their aircraft. The consequences of this failure for both the design and operation of control systems must not be under-estimated. Section 1.2.1 argued that differences between the commands issued to avionics and the instructions received by aircraft components contributed to both the Habsheim and Air India crashes [244].

6.3.2 Method Conformance

A number of techniques could be used to avoid the unpredictability caused by method failure. For instance, operators might retain direct control over process components. This approach has been exploited in a number of commercial systems. Airbus Industries' A320 has been heralded as the first civilian aircraft to be entirely controlled through fly-by-wire technology. The crew can, however, exclude avionics intervention in their commands to horizontal stabilisers. The methods available through the control system are the same as those which are available for the physical components, **accurate_methods**(**o**, **pc**, **om**). The mediation of on-board computers is reduced, operator input is not optimised. Interval temporal logic provides a notation in which to represent the requirements that must be satisfied in order to exploit such techniques. It is always the case that the methods, **om**, which operators invoke on a control object, **o**, are those that are available for a process component, **pc**:

$$\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{om} \in \mathbf{Om} : \mathbf{method\_conformal}(\mathbf{o}, \mathbf{pc}, \mathbf{om}) \Leftrightarrow \\ \Box\, \mathbf{accurate\_methods}(\mathbf{o}, \mathbf{pc}, \mathbf{om}) \tag{6.16}$$

It is important to note that task and application details determine the extent to which formal requirements, such as those specified by **method_conformal** (6.16), can be achieved within a particular implementation. There exist control system objects that are not always operated by methods which conform to those of physical components. It is a non-trivial task to provide operators with control system objects that accurately represent the many different operating methods provided by increasingly sophisticated avionics. Section 7.2.2 will argue that predicates, such as **method_conformal** (6.16), provide human factors and systems engineers with a framework for more detailed analyses of these problems.

6.4 Break-Down And State Failure

Madsen and Møller-Pedersen argue that domain modelling is a crucial stage in object oriented design [199]. The benefits of this development architecture are jeopardised if designers fail to accurately model the physical components of the application domain. This could occur if the state of a control system object does not resemble the state of the real-world component that it is intended to represent. Human factors and systems engineers might apply the axioms and theorems of interval temporal logic to *break_down* (6.6), see Appendix F.3, in order to represent this state failure:

$$\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} :$$

$$\begin{aligned}&\mathbf{break\_down}(\mathbf{o},\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od}) \Leftrightarrow \\ &\Diamond(\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{accurate\_image}(\mathbf{o},\mathbf{pc},\mathbf{od}) \wedge \\ &\mathbf{accurate\_methods}(\mathbf{o},\mathbf{pc},\mathbf{om}) \Rightarrow \neg\, \mathbf{accurate\_state}(\mathbf{o},\mathbf{pc},\mathbf{os}))\end{aligned} \quad (6.17)$$

Seamster, Baker and Andrews have developed a naval deployment system that is intended to provide its operators with information about the location of ships and aircraft, **pc** [278]. This system presents icons that look like these vehicles; **accurate_image**(**o**, **pc**, **od**). The methods of control system objects correspond to the commands that can be sent to ships and aircraft, **accurate_methods**(**o**, **pc**, **om**). The position and bearing of control system objects must only be changed within the constraints imposed by the speed and location of the physical components. The states of the icons do not, however, always conform to the states of ships and aircraft, ¬ **accurate_state**(**o**, **pc**, **os**). In particular, it is not always possible to determine whether a target is friendly or hostile. During the Gulf War a total of twenty-seven French and nine British soldiers were killed as a result of allied fire that was directed by similar deployment systems [63]. Many of these casualties were sustained because control systems represented allied troops as part of Iraqi units; their state was assumed to be hostile rather than friendly. In consequence, operators failed to predict the effect of their commands to engage those physical objects.

6.4.1 State Failure And Predictability

The operators of deployment systems use the state, **os**, of control system objects, **o**, to make assumptions about the effects of their input messages, **m**, upon physical components, **pc**. If the state of a control system object eventually does not accurately represent the state of a physical component then it will be difficult for users to predict the consequences of their interaction. Directing fire towards a friendly target was clearly not the intended effect in the examples cited in the previous section. Human factors and systems engineers might exploit interval temporal logic to represent the relationship between state failure and unpredictability:

$$\begin{aligned}&\exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{os} \in \mathbf{Os} : \\ &\quad \mathbf{unpredictable\_object\_state}(\mathbf{pc},\mathbf{o},\mathbf{m},\mathbf{os}) \Leftrightarrow \exists\, \mathbf{os}' \in \mathbf{Os} \\ &\quad \Diamond(\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{message\_effect}(\mathbf{o},\mathbf{m},\mathbf{os},\mathbf{os}') \wedge \\ &\quad \neg\, \mathbf{accurate\_state}(\mathbf{o},\mathbf{pc},\mathbf{os}))\end{aligned} \quad (6.18)$$

On the 3rd July 1988, the United States' Ship Vincennes used its Aegis missile system to shoot down Iranian Airline Flight 655. This civil aircraft was mistaken for an attacking fighter. In the aftermath of the accident it was extremely difficult to identify the causes of this control failure [230]. A United States' Navy spokesmen claimed that the "the (entire) system performed flawlessly" [201]. Malone remarks that "the crew had come to believe that the (Vincennes') Aegis combat system was infallible, if it indicated that they were under attack, that had to be the case". Faced with such a response, interface designers could use formal representations of design problems to focus their analysis of the failures that could have led to this type of accident. For example, the Vincennes incident can be interpreted as a consequence of an **unpredictable_object_state** (6.18). Operators were too ready to believe that control system objects, **o**, accurately represented the hostile state, **os**, of the

aircraft, **pc**. In consequence, they made inaccurate predictions about the effects of their intervention.

6.4.2 State Conformance

The Vincennes' Aegis operators might have been better able to predict the consequences of their actions if the system had always been infallible. Designers could exploit systems engineering, if not to guarantee infallibility then, to improve the reliability of the information presented by their system. For instance, the Vincennes did not receive any radio messages from the crew of the aircraft. It did, however, receive an identification message from an on-board radio system. This was incorrectly interpreted as a 'Mode II' alert from an Iranian fighter, rather than a 'Mode III' alert from a civilian aircraft. The Board of Enquiry into the Vincennes' incident recommended that the United States' Navy should pay urgent attention to the improvement of on-board identification systems [201]. Tactical control systems had to be re-designed for a range of naval vessels. Work-station upgrades are underway for all of the United States' Arleigh Burke Destroyers and Aegis Cruisers as well as North Atlantic Treaty Organisation Frigates [229]. These enhancements are intended to guarantee that the state, **os**, of a control system object, **o**, always provides an accurate representation of the state of the physical component, **pc**, which it is intended to represent:

$$\begin{aligned}\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \exists\, \mathbf{os} \in \mathbf{Os} : \mathbf{state\_conformal}(\mathbf{o}, \mathbf{pc}, \mathbf{os}) \Leftrightarrow \\ \Box\ \mathbf{accurate\_state}(\mathbf{o}, \mathbf{pc}, \mathbf{os})\end{aligned} \tag{6.19}$$

Techniques, such as the state conformance represented by the previous bi-condition, require the integration of human factors and systems engineering. Systems engineers must ensure that sensing devices are available in order to detect the states of physical components. Human factors engineering must determine the best means of presenting this information so that system operators can recognise these states. Abstractions, such as **os**, **o** and **pc**, provide a means of representing techniques for avoiding problems, such as break-down, without specifying the details that must be considered in order to implement a potential control system. This level of abstraction is not appropriate for all stages of design. At some stage, human factors engineers must select the images that are to represent process components. At some stage, systems engineers must determine the states of control system objects that are to represent physical components. Section 7.1.3 will argue that prototypes can be used to inform such design decisions. In contrast, the remainder of this chapter continues the formal analysis of break-down in order to explain recent attempts to abandon the pictorial realism of conformal control systems.

6.5 Break-Down And Direct Perception

A puzzling phenomenon in human factors research has been the decline of pictorial realism. A spate of recent articles have abandoned the conformal representation of process components [59, 103, 120]. Instead, they advocate what have become known as direct perception presentation techniques. Much of this work builds upon Gibson's claim that operators can directly derive information from the structure

Figure 6.6: A time-tunnel display for aircraft engine status

of light as it arrives at the eye [118]. He asserted that users can exploit this information without the need for any indirect cognitive processing of visual images. Many attempts to derive pragmatic applications of Gibson's observations exploit his notion of transformational invariants. People use these invariants when attempting to identify the behaviour of a physical entity from its image. For instance, there is a transformational invariant which states that if an object moves away from a viewer at a constant speed then the decrease in its visual area is proportional to the square of its distance from the viewer. This invariant supports the direct perception of changes in speed through changes in the rate at which the visual area increases or decreases. Transformational invariants have been used to support the presentation of production processes. For example, the American Electrical Power Research Institute has developed a display that uses a circle to present information about electricity generation [330]. This exploits a transformational invariant which states that the area of the circle always changes in proportion to the rate of production. Hansen and Skou extend this approach to present information about previous states of process components [131]. Figure 6.6 illustrates the use of perspective information as the transformational invariant in a tunnel of quadrilaterals. Operators are intended to directly perceive the age of the data set from the area of the quadrilaterals. The tunnel on the right represents an engine operating normally. The tunnel on the left represents an engine error as distortions in two of the quadrilaterals. These appear in the second and sixth most recent states, assuming that the oldest is presented as the smallest quadrilateral.

Previous sections have argued that break-down is likely to occur if the states, methods and displays of control system objects do not conform to those of process components. It can be argued that the circles and quadrilaterals of direct perception displays would suffer from this problem because they do not resemble physical parts of an application. The link between break-down and direct perception can be demonstrated by re-writing *break_down* (6.6), see Appendix F.4, in order to repre-

sent properties that are common to the displays described in the previous paragraph:

$$\begin{aligned}&\exists\,\mathbf{o}\in\mathbf{O},\exists\,\mathbf{pc}\in\mathbf{Pc},\forall\,\mathbf{om}\in\mathbf{Om},\forall\,\mathbf{os}\in\mathbf{Os},\forall\,\mathbf{od}\in\mathbf{Od}:\\&\quad\mathbf{break\_down}(\mathbf{o},\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})\Leftrightarrow\\&\quad\Diamond(\mathbf{represents}(\mathbf{o},\mathbf{pc})\wedge\mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od})\Rightarrow\\&\quad\neg\,\mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od}))\end{aligned}\tag{6.20}$$

For example, a quadrilateral, **o**, in one of Hansen and Skou's time-tunnels represents an application component, **pc**, at a particular instant in time. This object does not conform to the attributes of that process component, $\neg$ **conforms**(**pc**, **om**, **os**, **od**). Direct perception displays encourage image failure; the image, **od**, of a control system object, **o**, need not always resemble that of a physical component. State failure could occur if eventually a direct perception object, **o**, gives misleading information about the state of an element of **Pc**. Method failure is possible because the methods, **om**, that are available for circles and quadrilaterals are unlikely to be the same as those available for physical components.

6.5.1 Direct Perception And Predictability

The Vincennes' Aegis operators assumed that if control system objects were represented as hostile then their physical counterparts must also be hostile. Users might have made more accurate predictions about the consequences of their actions if their systems had presented direct perception displays. The impact of break-down is reduced and predictability is supported because operators are not encouraged to believe that their commands directly affect application components. Users could expect control system intervention because the commands issued to quadrilaterals and circles must be translated into the detailed instructions required by application components. Designers might exploit interval temporal logic to represent the requirements that must be satisfied in order to exploit direct perception as a means of supporting principles, such as predictability. If users can issue input messages, **m**, to a control system object, **o**, that represents a physical component, **pc**, then the attributes of the control system object do not correspond to those of the physical component:

$$\begin{aligned}&\exists\,\mathbf{o}\in\mathbf{O},\forall\,\mathbf{pc}\in\mathbf{Pc},\forall\,\mathbf{m}\in\mathbf{M},\forall\,\mathbf{om}\in\mathbf{Om},\forall\,\mathbf{os}\in\mathbf{Os},\forall\,\mathbf{od}\in\mathbf{Od}:\\&\quad\mathbf{predictable\_direct\_perception\_object}(\mathbf{o},\mathbf{pc},\mathbf{m},\mathbf{om},\mathbf{os},\mathbf{od})\Leftrightarrow\exists\,\mathbf{os}'\in\mathbf{Os}\\&\quad\Box(\mathbf{message\_effect}(\mathbf{o},\mathbf{m},\mathbf{os},\mathbf{os}')\wedge(\mathbf{represents}(\mathbf{o},\mathbf{pc})\wedge\mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od})\Rightarrow\\&\quad\neg\,\mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})))\end{aligned}\tag{6.21}$$

A number of limitations restrict the utility of direct perception presentation techniques. Operators must probe beyond circles and quadrilaterals in order to explain any unpredicted behaviour by process components. Aircrew are required to explain unexpected fuel readings in terms of low-level components even if initial symptoms are presented using direct perception displays. A further limitation is that direct perception displays are unlikely to provide adequate support for novice operators. Gillie and Berry argue that direct perception displays deprive users of the structural information that is necessary when learning to control application processes [120]. Direct perception displays have, typically, been developed to support monitoring

activities. The cognitive and perceptual advantages of this technique have yet to be demonstrated for tasks that require the active intervention of system operators. Human factors research must provide additional evidence to support the cognitive and perceptual benefits that are claimed for direct perception if systems engineering is to exploit it as a means of avoiding alienation and unpredictability.

6.5.2 Indirect Presentation Techniques

Previous attempts to exploit Gibson's work on perception have concentrated upon the notion of transformational invariance. His work on the perception of pictures has been neglected. Although Gibson argued that we perceive the world directly through information in the structure of light, he accepted that indirect awareness was possible when looking at a picture [117]. In other words, cognitive processing intervenes in the perception of images that represent physical objects. Reed built upon Gibson's analysis [253]. He argued that the artist chooses the stimulus sources which are represented in a painting. This selection alters the original image. In consequence, viewers are usually aware that they are not looking at the physical objects which are depicted on the canvas. Human factors and systems engineers might exploit this analysis when designing an interactive control system. Operators must be made aware that the stimulus sources provided by control system objects are not directly produced by physical components. This leads to a surprising conclusion; it is not only essential that users trust the information presented by their system, they must also view it with a degree of scepticism. The Vincennes incident shows what can happen when users assume that control system objects provide accurate views of real-world objects. Recall the remarks of the Navy spokesman; "the (entire) system performed flawlessly" [201]. Recall also Malone's comment; "the crew had come to believe that the (Vincennes') Aegis combat system was infallible, if it indicated that they were under attack, that had to be the case" [201]. Operators seemed unaware that the stimulus sources provided by control system objects were not produced by physical components.

Physical components could be represented by a number of different control system objects in order to encourage the critical assessment of process information. In contrast to direct perception, this indirect presentation technique is intended to encourage the cognitive processing of display information. Users must analyse and collate the information presented by different control system objects in order to monitor all the data presented about application components. This technique is illustrated in Figure 6.7. The left engine of the aircraft, **pc**, is represented by a direct perception time-tunnel, **o**, and by one of Stollings' icons, $\mathbf{o}'$, described in Section 6.3. In order to exploit this technique, designers must integrate human factors and system engineering. Indirect presentation techniques draw upon Gibson's human factors research but can only be exploited if systems engineers provide sufficient sensing devices to support alternative representations, **o** and $\mathbf{o}'$, of physical components, **pc**. Temporal logic provides a means of representing the requirements that must be satisfied in order to exploit this technique:

$$\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc} : \mathbf{indirect\_presentation}(\mathbf{o}, \mathbf{o}', \mathbf{pc}) \Leftrightarrow \\ \Box(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{represents}(\mathbf{o}', \mathbf{pc}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad (6.22)$$

Indirect presentation techniques could support operators' predictions about the ef-

Figure 6.7: An indirect display

fects of their commands. For instance, the Aegis system might have used different objects to present radar data, **o**, and scheduled flight information, $\mathbf{o}'$. If these schedules had been presented then the crew might have been more sceptical about the threatening state of the control system object presented by their radar systems. This could have encouraged more accurate predictions about the effects of their input, **m**, to destroy the physical object, **pc**:

$$\begin{aligned}&\exists\,\mathbf{o}\in\mathbf{O},\exists\,\mathbf{pc}\in\mathbf{Pc},\forall\,\mathbf{m}\in\mathbf{M},\forall\,\mathbf{os}\in\mathbf{Os}:\\&\quad\mathbf{predictable\_indirect\_presentation}(\mathbf{o},\mathbf{pc},\mathbf{m},\mathbf{os})\Leftrightarrow\exists\,\mathbf{o}'\in\mathbf{O},\exists\,\mathbf{os}'\in\mathbf{Os}\\&\quad\Box(\mathbf{message\_effect}(\mathbf{o},\mathbf{m},\mathbf{os},\mathbf{os}')\Rightarrow\mathbf{indirect\_presentation}(\mathbf{o},\mathbf{o}',\mathbf{pc}))\end{aligned}\tag{6.23}$$

Principles provide a focus for an analysis of the human factors and systems engineers benefits of indirect presentation techniques. Predictability is supported because these techniques reduce the likelihood of break-down. Principles also provide a focus for an analysis of the costs of indirect presentation techniques. Complexity can be increased by the presentation of multiple sources of control information. A number of different sources of information, such as cathode ray-tube monitors and mimic boards, must be used if designers are to provide alternative representations of process components. Section 3.4 argued that this threatens predictability. Operators frequently have neither the time nor the opportunity to monitor alternative sources of control information. The crew of the Vincennes had less than five minutes in which to monitor all necessary information, make their decision and receive permission to engage. Malone argues that there "were serious problems with the Captain's large screen display, with vital information not presented and too much extraneous information presented in a confusing manner" [201]. Chapter 7 will argue that prototyping can be used to determine whether such problems threaten the benefits of techniques, such as indirect presentation, which are intended to support principles, such as predictability.

6.6 Conclusions

Break-down occurs if the methods, states and displays of control system objects do not resemble those of the process components which they represent. Break-down jeopardises predictability. Operators are unlikely to make accurate predictions about the effects of their commands upon unfamiliar physical components.

Interval temporal logic provides a medium in which to represent techniques that are intended to avoid the unpredictability caused by break-down. Object conformance requires that the states, image and methods of control system objects always resemble those of process components. Unfortunately, it is not always technically or financially possible to achieve object conformance. Direct perception techniques ensure that the state, image and methods of control system objects never resemble those of process components. The utility of this approach is limited by the human factors problems of providing novice operators with sufficient information about the structure of process components.

Indirect presentation techniques provide alternatives to conformance and direct perception. This approach relies upon the integration of human factors and systems engineering. Human factors engineers must ensure that operators can monitor and assimilate data from a number of different control system objects. Systems engineers must provide adequate sources of information in order to support these multiple representations of process components. It is argued that principles provide criteria against which to assess the benefits of techniques for avoiding break-down in object oriented control systems. Indirect presentation supports predictability because users are not encouraged to believe that control system objects provide complete and infallible representations of process components. It is also argued that principles provide a focus for an assessment of the costs of such techniques. Complexity can be increased by indirect presentation if designers display multiple sources of control information.

Further evidence is required in order to substantiate the claimed costs and benefits of the techniques described in this chapter. Chapter 7 argues that human factors and systems engineers might use executable temporal logics to build prototype implementations. These can be shown to operators in order to determine whether design techniques, such as **indirect_presentation** (6.22), support principles, such as predictability.

Chapter 7

Design Bias

> "There is a practical problem with many of the representations used by human factors engineers... there are not necessarily any simple mappings between them. This raises the question about what we expect from a model and about communication between disciplines which use different techniques." (Bainbridge, [23]).

7.1 Introduction

Design bias occurs if formal or experimental design techniques dominate the development of an interface. Experimental design exploits the analytical tools of physiological and social ergonomics, perceptual and cognitive psychology, introduced in Section 1.3. Formal design exploits mathematical notations, such as interval temporal logic, to support systems development. Design bias hinders the integration that is advocated in this thesis. Formal analyses are typically conducted by systems engineers whilst experimental analyses are usually conducted by human factors engineers. This chapter presents techniques that designers might exploit in order to avoid design bias. In particular, it is argued that:

- abstract requirements, derived from a formal analysis of principles, provide a framework for the specification of detailed designs;
- detailed designs provide blue-prints for the development of prototype implementations;
- prototype implementations provide a means of assessing the utility of principles for particular interfaces to particular control systems.

PRELOG, a system for Presenting and REndering LOGic specifications of interactive systems, has been developed to support these arguments. It is concluded that this tool could be used to avoid the design bias which threatens the integration of human factors and systems engineering.

7.1.1 Consequences: Distrust And Unpredictability

Design bias focuses the application of finite development resources upon a dominant analytical technique. Principles can be violated if this focus changes. For instance,

designers might adopt predictability as a goal for an initial formal analysis. Subsequent investigations could reject this principle in favour of properties that are more easily assessed using experimental techniques.

The lack of integration between formal and experimental analyses can exacerbate the distrust that has been observed between human factors and systems engineers [32]. Abstract analyses, conducted in mathematical formalisms, can ignore the cognitive and perceptual demands placed upon system operators. Human factors engineering distrusts formal modelling which lacks the under-pinning provided by experimental results. Systems engineering distrusts the findings of laboratory investigations which fail to consider functional or safety requirements [113]. Unfortunately, the results of formal and experimental analyses are often pitched at different levels and are rarely used to inform each other.

7.1.2 Causes: Different Disciplines And Design Techniques

The quotation that opens this chapter identifies the causes of design bias; different disciplines use different design techniques. The approaches employed by systems engineers, typically, abstract away from device and presentation details that have a profound impact upon operator performance [175]. In contrast, human factors engineering has exploited experimental techniques to assess the influence of these details upon usability.

Winner, Pennel, Bertrand and Slusarczuk argue that common goals must be established if different analytical techniques are to be applied during the development of an interactive system [325]. Hofer and Ruggiero argue that common goals can be established by incorporating the products of human factors and systems engineering into prototype implementations [150].

7.1.3 Solution: Prototyping

Design requirements, expressed in interval temporal logic, provide the non-formalist with little idea of what it would be like to interact with a control system. Prototypes provide a far better impression of the look and feel of a final implementation [133]. They can be shown to system operators. They are amenable to experimental analysis [106]. They can be used to determine the cognitive, perceptual, physiological and sociological demands that a control system might place upon its users. They could, therefore, be used to assess the relevance of design principles for system operators.

7.2 From Abstract Requirements To Specifications

Continuous steel casting provides an example for the remainder of this chapter. The casting process exhibits many of the problems that frustrate the design of interactive control systems. It is complex, dynamic and open. Systems engineering must monitor and respond to hundreds of potential input values from this application. The presentation of casting control systems also poses significant challenges for human factors engineers [149]. Operators must detect and respond to a range of potential errors. The casting process is illustrated by Figure 7.1. Steel from a melting shop is passed through a water cooling mechanism and, in so doing, solidifies into a billet. This is cut to length at the end of the run. The most feared accident in this process

Figure 7.1: The continuous casting process

is a break-out which occurs when the billet tears and steel flows inside the casting plant. There is also a danger that the flow will clog leading to a splash-off either at the ladle or onto the floor. Casting must halt if either a break-out or a splash-off is detected.

7.2.1 Instantiating Architectural Models

Formal models of development architectures provide a framework for the detailed design of interactive control systems. For instance, the model of object orientation introduced in Section 5.2 described objects as having a state, **os**, a graphical image, **od** and some operating methods, **om**:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{om} \in \mathbf{Om}, \exists\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{od} \in \mathbf{Od} : \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \Leftrightarrow$$
$$\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \tag{5.5}$$

Systems engineers might instantiate these abstract predicates in order to represent the operating methods that must be provided by application functionality. Human factors engineers could use them to represent the displays that must be provided for the operators of a particular control system. For example, a **coolant_system** is presented by the **coolant_error_display** in the **error** state and is operated by methods which **halt** it:

$$\mathbf{object}(\mathbf{coolant\_system}, \mathbf{halt}, \mathbf{error}, \mathbf{coolant\_error\_display}) \tag{7.1}$$

Other components of the model of object orientation can be used in a similar fashion. For instance, a predicate was introduced to describe the effect, $\mathbf{os}'$, of messages, **m**,

upon the states, **os**, of objects, **o**:

$$\forall \mathbf{o} \in \mathbf{O}, \forall \mathbf{m} \in \mathbf{M}, \forall \mathbf{os} \in \mathbf{Os}, \exists \mathbf{os}' \in \mathbf{Os} : \mathbf{message\_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \quad (5.17)$$

Designers might instantiate this abstract predicate in order to specify that an '**error**' message transforms a **coolant_system** from the **on** state into the **error** state:

$$\mathbf{message\_effect}(\mathbf{coolant\_system}, \text{`}\mathbf{error}\text{'}, \mathbf{on}, \mathbf{error}) \quad (7.2)$$

Ineffective input, in other words input that does not alter the state of an object, can be described in similar fashion. An '**error**' message does not affect a **coolant_system** that is already in the **error** state:

$$\mathbf{message\_effect}(\mathbf{coolant\_system}, \text{`}\mathbf{error}\text{'}, \mathbf{error}, \mathbf{error}) \quad (7.3)$$

This level of analysis offers a number of advantages for the development of interactive control systems. Human factors and systems engineers might exploit these bare-bones requirements in order to construct a design without necessarily specifying the elements of particular states and displays. For instance, designers need not represent the exact wording of the error messages that a system would use in order to warn its users that their input had been ineffective. Such details can be gradually introduced as development progresses, techniques for achieving this will be discussed in Section 7.4. A further advantage of this level of abstraction is that designers are not immediately forced to specify which input devices will be incorporated into a final implementation. Such decisions can be postponed until procurements contracts are drafted for hardware suppliers [48]. In order to represent the trade-offs that exist between devices, such as mice and joysticks, it is vital that there be some means of introducing interaction details into the products of a formal analysis. Sections 7.5 will demonstrate how this can be achieved. In contrast, the next section demonstrates that techniques which are intended to support principles can be incorporated into the detailed design of particular interfaces to particular control systems.

7.2.2 Instantiating Generic Principles

Section 2.4.4 argued that it is difficult for operators to predict the effects of their commands if they cannot view the consequences of their intervention. Designers could, therefore, support predictability by ensuring that the pre-condition, **s**, and the post-condition, **s′**, of input, **p**, are viewed through different displays, **d** and **d′**:

$$\begin{aligned}\forall \mathbf{p} \in \mathbf{P}, \forall \mathbf{s} \in \mathbf{S}, \exists \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{temporal\_visible\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{d}, \mathbf{d}') \Leftarrow \exists \mathbf{s}' \in \mathbf{S} \\ (\mathbf{input}(\mathbf{p}) \wedge \mathbf{interpret\_effect}(\mathbf{p}, \mathbf{s}, \mathbf{s}') \wedge \mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \\ \bigcirc (\mathbf{view}(\mathbf{s}', \mathbf{d}') \wedge \neg\, \mathbf{same\_display}(\mathbf{d}, \mathbf{d}')))\end{aligned} \quad (2.28)$$

Such predicates can be brought closer to the level of detail that is required in order to support implementation by translating them into the elements of particular development architectures. For instance, the previous bi-condition could be expressed in terms of the object oriented model, introduced in Section 5.2. States, **s** and **s′**, might be represented by the states, **os** and **os′**, of a control system object, **o**. The

displays, **d** and **d′**, could be replaced by the image, **od** and **od′**, of that object:

$$\begin{aligned}\forall\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{o} \in \mathbf{O}, \forall\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{od}, \mathbf{od}' \in \mathbf{Od} :\\ \mathbf{object\_visible\_effect}(\mathbf{m}, \mathbf{o}, \mathbf{os}, \mathbf{od}, \mathbf{od}') \Leftarrow \exists\, \mathbf{os}' \in \mathbf{Os}\\ (\mathbf{message\_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge\\ \bigcirc(\mathbf{object\_view}(\mathbf{o}, \mathbf{od}') \wedge \neg\, \mathbf{same\_display}(\mathbf{od}, \mathbf{od}')))\end{aligned} \tag{7.4}$$

Section 1.5 argued that principles provide a basis for action. In the context of this thesis, they provide a basis for detailed design. Human factors and systems engineers might instantiate the products of an abstract analysis of predictability in order to construct specifications. For instance, **object_visible_effect** (7.4) can be instantiated to represent specific requirements that must be satisfied if operator predictions are to be supported by a particular interface. The effect of a '**coolant ok**' message is presented by a change in the image of a **coolant_system** in the interval after that input is issued. The state, **os**, and display, **od**, parameters are dropped from **effect_visible** (7.5) for the sake of brevity:

$$\begin{aligned}&\mathbf{effect\_visible}(\text{`}\mathbf{coolant\ ok}\text{'}, \mathbf{coolant\_system}) \Leftarrow\\ &\quad \mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display}) \wedge\\ &\quad \mathbf{message\_effect}(\mathbf{coolant\_system}, \text{`}\mathbf{coolant\ ok}\text{'}, \mathbf{error}, \mathbf{on}) \wedge\\ &\quad \bigcirc(\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_on\_display}) \wedge\\ &\quad \neg\, \mathbf{same\_display}(\mathbf{coolant\_error\_display}, \mathbf{coolant\_on\_display}))\end{aligned} \tag{7.5}$$

The transition between generic principles and detailed specifications relies upon the skill and judgement of the designer. The **effect_visible** (7.5), **object_visible_effect** (7.4) and **temporal_visible_effect** (2.28) predicates are not equivalent. The previous predicate abandons the $\forall$ and $\exists$ quantifiers. There could, therefore, exist input messages with 'invisible' effects. Designers might introduce additional constraints, such as $\bigcirc\neg$ **object_view**(**coolant_system**, **coolant_error_display**), into instantiated predicates as development progresses. It is important that such changes should not unwittingly sacrifice the design objectives represented in abstract requirements. Lin and Hunt present a number of techniques that are intended to support this task [196]. Section 8.2.2 will describe future work that might build upon such research. In contrast, the following sections demonstrate that detailed specifications, such as **effect_visible** (7.5), provide a framework for the prototyping of interactive dialogues.

7.3 From Specifications To Executable Systems

The process of instantiation necessary to derive detailed specifications from abstract predicates does not bridge the divide between principles and prototypes. In order for designers to close this gap they must be provided with some means of developing executable systems that implement these specifications.

7.3.1 Object Oriented Programming Languages

Object oriented programming languages could be used to develop systems that implement specifications such as **effect_visible** (7.5). For instance, the C++ programming language might be used to implement a class of `coolant_systems`:

```
class COOLANT_SYSTEMS: public plant_components
{
// Class definition for coolant_systems in casting control,
// type definitions omitted to simplify the exposition.

  STATE_ID state;
  DISPLAY_ID display;

  public:
  coolant_stms(STATE_ID, DISPLAY_ID); // create instance
  ~coolant_stms(); // destroy instance
  void start();// start system
  void stop();// stop system
  void error();// error alert
  void coolant_ok();// resolve error
  mbool state(STATE_ID); // TRUE if state is STATE_ID
  mbool display(DISPLAY_ID); // TRUE if display is DISPLAY_ID
}
```

The implementation of interactive dialogues is less straightforward. For example, the following code implements **effect_visible** (7.5):

```
main()
//Example of C++ implementation of predictability specification,
// type definitions omitted to simplify the exposition.
{
  string USER_INPUT;
  COOLANT_SYSTEMS COOLANT_SYSTEM = coolant_stms(ASTAT, ADISP);
  //create instance, ASTAT and ADISP defined elsewhere

  cin>>USER_INPUT; //read operator input

  if(USER_INPUT=="coolant-ok" &&
  COOLANT_SYSTEM.state(ERROR)&&
  COOLANT_DISPLAY.display(COOLANT_ERROR_DISPLAY))
  //operator input to resolve coolant error.
  {
    COOLANT_SYSTEM.error_ok();
    if(COOLANT_SYSTEM.display(COOLANT_ERROR_DISPLAY))
      {cout << "UNPREDICTABLE\n";}
      //warn of potential unpredictability
  }
}
```

It is difficult for human factors and systems engineers to implement properties that are represented using the $\bigcirc$, $\Diamond$, $\Box$, $\mathcal{U}$ operators in languages such as Smalltalk or C++. This limitation can be resolved. For instance, some of the National Aeronautics and Space Administration's ground-control systems include a scheduler, or

clock, implemented in Smalltalk-80 [262]. Designers could translate predicates, such as **effect_visible** (7.5), into a real-time notation. A scheduler could then be used to ensure that the resulting specification was satisfied by a particular implementation. In order to exploit this approach, designers must be able to translate interval temporal logic predicates into a real-time notation [89]. Section 8.3.1 will argue that further work is needed in order to develop techniques that might support this task. For now it is sufficient to realise that there is no straightforward translation between interval operators, such as $\bigcirc$ and $\Diamond$, and hours, minutes and seconds.

7.3.2 PROLOG

Detailed specifications, derived from formal analyses, might be implemented using executable subsets of first order logic. For instance, designers could exploit one of the environments that have been derived from Colmerauer and others' PROgramming in LOGic system (PROLOG) [75]. There is a close correspondence between specifications written in first order predicate logic and programs that satisfy those specifications implemented in PROLOG [70]. There is a well understood translation between first order logic predicates and the Horn clauses that can be executed by this environment. PROLOG can be used to prototype detailed designs that are structured using any one of a number of development architectures. Human factors and systems engineers might instantiate and execute **temporal_visible_effect** (2.28) without exploiting the object oriented architecture. Alternatively, designers could implement **effect_visible** (7.5) using any one of a number of PROLOG systems that provide access to object oriented facilities [110, 126]. These extensions provide a means of executing the following PROLOG clause. The '$:-$' symbol represents implication ($\Leftarrow$), ',' represents conjunction ($\wedge$), pairs of single right-hand 'quotes' surrounding text represents a string literal, '.' represents the syntactic termination of a clause:

effect_visible('**coolant ok**', **coolant_system**) :-
    object_view(**coolant_system**, **coolant_error_display**),
    message_effect(**coolant_system**, '**coolant ok**', **error**, **on**),
    object_view(**coolant_system**, **coolant_on_display**),
    not(**same_display**(**coolant_error_display**, **coolant_on_display**)). (7.6)

PROLOG suffers from a number of disadvantages that restrict its utility as a means of avoiding design bias. Changing the order of the predicates in **effect_visible** (7.6) would not affect a detailed specification. It would, however, radically affect the behaviour of a prototype. For instance, the **coolant_on_display** might be presented before an operator had provided input to resolve the **error**. This distinction between the specification and the behaviour of a prototype arises because programming languages must enforce an order of evaluation that is not explicitly part of first order predicate logic. Section 2.3.4 introduced the differences between the declarative and procedural interpretations of first order logic. The distinction between the declarative reading of first order logic and the procedural order of evaluation enforced by PROLOG can frustrate prototype development [181]. The ordering of clauses within a detailed design might have to be substantially revised in order to support the control strategy necessary for implementation.

7.3.3 Tempura

Several research groups have provided executable semantics for interval temporal logics [28, 112, 136, 224]. Moszkowski [218] and Hale [128] have developed interpreters for the Tempura programming language using C and Lisp. Tempura implements the `next` operator which is equivalent to ◯. The `sometimes` operator is equivalent to ◇, `always` is equivalent to □. Tempura avoids the problems that limited the utility of PROLOG. Procedural requirements can be made explicit in predicates, using the ◯ operator, and can be satisfied by Tempura prototypes, using the `next` operator:

```
define predict(O, M, S1, D1) =
/* Implements predictable dialogue */

{
               object_view(O, D1) and
               message_effect(O, M, S1, S2) and
               next(object_view(O, D2)) and
               next(not(same_display(D1, D2))) and
               next(next(empty)).  /* end of interval */
run(predict(coolant_system, ‘‘ok’’, error, coolant_error_display)).
/* Implement dialogue cancelling coolant error */

}.
```

Unfortunately, previous Tempura interpreters have only provided access to a small range of input-output primitives. It has not been possible to develop prototypes that exploit graphical presentation techniques. Hale has recently overcome this problem by providing access to external system calls from within Tempura. There are, however, further limitations. Tempura interpreters require left-right determinism before they can evaluate any clause. As a consequence, unification is not well supported and valid interval temporal logic clauses cannot be implemented. For instance, a designer might specify that `D` is a view of the `coolant_system` in the present interval:

```
object_view(coolant_system, D) and D is coolant_error_display.
```

Tempura scans from the left to the right. The evaluation of this statement would fail because the value of `D` is not known during the interpretation of `object_view`. Until unification facilities are improved, human factors and systems engineers must transform abstract requirements into detailed specifications. They must then guarantee that these designs support the left-right determinism required by Tempura.

7.3.4 Tokio

A number of temporal logic programming languages avoid the limitations of Tempura because they have been implemented using PROLOG [269]. For instance, the Tokio interpreter exploits PROLOG's unification facilities in order to avoid the requirement for left-right determinism [16]. Clauses that do not contain any temporal operators are passed directly to PROLOG for evaluation. Clauses that do contain

temporal operators are re-written in first order logic and are asserted over an appropriate interval. In other words, the Tokio interpreter maintains time-variables that are similar to those introduced in Section 2.4.2. Execution progresses when all the clauses for a particular interval are satisfied [109]. This approach has much in common with Abadi's temporal logic proof system, described in Appendix B [1]. Designers might use these facilities to develop prototypes that implement specifications such as **effect_visible** (7.5):

$$\begin{aligned}&\textbf{effect\_visible}(\text{'}\textbf{coolant ok}\text{'}, \textbf{coolant\_system}) :- \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}), \\ &\quad \textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{coolant ok}\text{'}, \textbf{error}, \textbf{on}), \\ &\quad \bigcirc(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_on\_display}), \\ &\quad \textbf{not}(\textbf{same\_display}(\textbf{coolant\_error\_display}, \textbf{coolant\_on\_display}))) \end{aligned} \quad (7.7)$$

Tokio first attempts to satisfy clauses that are specified for the present interval. These are passed to PROLOG. PROLOG evaluates them in left-right order; the error is displayed before the input is evaluated. Tokio then attempts to satisfy clauses that are specified for the next interval; the **coolant_on_display** is a view of the **coolant_system** and this is not the same as the **coolant_error_display**. Designers could use the $\bigcirc$ operator to avoid any reliance upon the evaluation strategy of PROLOG:

$$\begin{aligned}&\textbf{effect\_visible}(\text{'}\textbf{coolant ok}\text{'}, \textbf{coolant\_system}) :- \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}), \\ &\quad \bigcirc(\textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{coolant ok}\text{'}, \textbf{error}, \textbf{on}), \\ &\quad \bigcirc(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_on\_display}), \\ &\quad \textbf{not}(\textbf{same\_display}(\textbf{coolant\_error\_display}, \textbf{coolant\_on\_display})))) \end{aligned} \quad (7.8)$$

Altering the order of clauses in this implication would not affect its declarative interpretation nor would it affect the behaviour of a prototype. This closes the divide between the products of formal analysis and prototypes that are amenable to experimental investigation.

7.3.5 PRELOG And Tokio

In order to demonstrate that executable interval temporal logics can support prototyping we have incorporated Tokio into PRELOG. It is important to note that PRELOG does not force human factors and systems engineers to exploit the object oriented architecture. The intention is to provide a tool that leaves designers free to choose whether or not this framework is suitable for the development of their control system. If object orientation is appropriate then Bowen and Kowalski's Meta-Theory language [39] or Page's Object Oriented PROLOG [231] could be used in conjunction with PRELOG. If object orientation is inappropriate then designers might confine themselves to the facilities implemented by Tokio. Executable interval temporal logics only provide part of the support required if human factors and systems engineers are to avoid the problems of design bias. The Tokio interpreter has previously been applied to the problems of hardware verification [108]. It was not intended to support interface design and does not provide access to graphical presentation facilities.

7.4 From Executable Systems To Graphical Interfaces

Prototypes provide a means of avoiding design bias; the findings of a formal analysis can be embodied in partial implementations that are amenable to experimental analysis. Prototypes must resemble final implementations if users are to gain an accurate impression of what it would be like to operate a potential control system. Designers must instantiate graphical abstractions, such as **coolant_on_display** and **coolant_error_display**, into the instructions that are required in order to generate images on particular presentation devices.

7.4.1 Unstructured Graphics

Unstructured graphical representations do not distinguish between the images of display components, such as menus and icons. For instance, bitmaps represent the image of pixels as bits in a data structure. Designers might use the following bitmap to describe the **coolant_error_display**:

```
DeclareBitmap(coolant_error_display.bit, 42, 49, e_disp.bits);
short error_display.bits[] =
/* Some information omitted for the sake of brevity */
{
 0x0000, 0x0000, 0x0000, 0x000f, 0xff00, 0x0000, 0x007f, 0xffc0,
 0x0000, 0x00ff, 0xfff0, 0x0000, 0x00ff, 0xfff0, 0x0000, 0x00ff,
 0xfff8, 0x0000, 0x01ff, 0xfff8, 0x0000, 0x01ff, 0x8ffc, 0x0000,
 0x01ea, 0x50be, 0x0000, 0x03d0, 0x02be, 0x0000, 0x03a2, 0x003e,
 0x0000, 0x0340, 0x24af, 0x0000, 0x03a8, 0x412f, 0x0000, 0x0350,
 0x001f, 0x0000, 0x0340, 0x006f, 0x0000, 0x03b0, 0x0a97, 0x0000,
 0x037d, 0x3fef, 0x0000, 0x03ee, 0x0a1b, 0x0000, 0x03d7, 0x3ff7,
 0x0000, 0x03fd, 0x87ca, 0x0000, 0x03f7, 0x5616, 0x0000, 0x014b,
 0x0000, 0x0000, 0x0000,
 };
```

Unstructured graphical representations have a limited utility for the development of prototype control systems. In particular, designers must make a one step refinement between graphical abstractions, such as **coolant_error_display**, and data structures, such as the previous bitmap. The **coolant_system** display cannot be decomposed into a number of simpler images. This hinders the gradual introduction of graphical details as development progresses.

7.4.2 Procedural Graphics

Procedural graphics systems construct pictures from sequences of instructions [2]. Human factors and systems engineers might use these systems to generate the images of interface components without describing the entire appearance of a display. Abstractions, such as **coolant_error_display**, can gradually be replaced by the instructions necessary to produce the images that are presented to system operators. For instance, the inlet icon illustrated in Figure 7.2 could form part of the **coolant_error_display**. The dotted lines are intended to indicate the part of the image that is described by the following clause. Figure 7.2 is not drawn to scale.

The **rotate**(90) predicate specifies a ninety degree clockwise rotation in the direction of the **pen** around its current location, **forward**(10) specifies a ten Centimeter forward movement of the **pen**:

$$\begin{aligned}\textbf{draw\_inlet} : -\textbf{pen}(\textbf{down}), \textbf{forward}(10), \textbf{rotate}(90), \textbf{forward}(10),\\ \textbf{rotate}(90), \textbf{forward}(10), \textbf{rotate}(90), \textbf{forward}(10), \textbf{pen}(\textbf{up}).\end{aligned} \tag{7.9}$$

Procedural approaches offer only limited support for the prototyping of interactive

Figure 7.2: An **inlet** image

control systems. Designers would be forced to write many thousands of instructions in order to create complex images. If one instruction were omitted or placed out of sequence then the final image might be corrupted. For instance, Figure 7.3 illustrates the way in which an image can be changed by altering the position of a single instruction. Swapping the first **rotate** and **forward** instructions in **draw_inlet** (7.9) would produce a very different image to that of Figure 7.2. As before, Figure 7.3 is not drawn to scale. The dotted lines indicate the part of the image which is described by the following clause:

$$\begin{aligned}\textbf{draw\_incorrect\_inlet} : -\textbf{pen}(\textbf{down}), \textbf{rotate}(90), \textbf{forward}(10), \textbf{forward}(10),\\ \textbf{rotate}(90), \textbf{forward}(10), \textbf{rotate}(90), \textbf{forward}(10), \textbf{pen}(\textbf{up}).\end{aligned} \tag{7.10}$$

Designers could avoid this order dependence by recruiting interval temporal logic to explicitly represent the instruction sequences of procedural graphics systems. This requires a cumbersome iteration of the $\bigcirc$ operator for any but the most trivial of images:

$$\begin{aligned}\textbf{temporal\_draw\_inlet} : -\textbf{pen}(\textbf{down}), \bigcirc(\textbf{forward}(10), \bigcirc(\textbf{rotate}(90),\\ \bigcirc(\textbf{forward}(10), \bigcirc(\textbf{rotate}(90), \bigcirc(\textbf{forward}(10), \bigcirc(\textbf{rotate}(90),\\ \bigcirc(\textbf{forward}(10), \bigcirc(\textbf{pen}(\textbf{up}))))))))).\end{aligned} \tag{7.11}$$

Szekely and Myers identify a further limitation of procedural graphics systems [298]. If an operator selects part of a display, using a mouse or some cursor keys, then there

Figure 7.3: An incorrect **inlet** image

is no means of identifying the target of their selection using the instructions that generated the image. Designers must, therefore, maintain additional data structures in order to determine which objects are selected by operator input.

7.4.3 Structured Graphics

Structured graphics systems represent complex images in terms of their component parts. For instance, the image of a **coolant_system** might include water jets, coolant tanks and an inlet. Human factors research has identified the benefits that can be gained if users are aware of the display structures exploited by designers. For instance, Rasmussen and Lind argue that structured presentation formats help operators to learn the layout of their plant [249]. Bainbridge suggests that they help users to remember "chunks" of information about process components [23]. Structured graphics systems have also been exploited by software engineering. Blake and Cook [35] and Took [308] introduce graphical hierarchies into object oriented presentation systems. Pereira uses structural decomposition to represent and generate graphical images in a PROLOG based system [235]. Human factors and system engineers might, therefore, exploit this approach to support the design and implementation of prototype displays. For instance, the following clauses describe the image of the **coolant_on_display** for the **coolant_system** illustrated in Figure 7.4:

$$\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_on\_display}). \tag{7.12}$$

$$\mathbf{part}(\mathbf{coolant\_on\_display}, \mathbf{water\_jet\_A}). \tag{7.13}$$

$$\mathbf{part}(\mathbf{coolant\_on\_display}, \mathbf{tank\_A}). \tag{7.14}$$

Figure 7.4: The graphical decomposition of the *coolant_on_display*

$$\textbf{part}(\textbf{coolant\_on\_display}, \textbf{water\_jet\_B}). \tag{7.15}$$

$$\textbf{part}(\textbf{coolant\_on\_display}, \textbf{tank\_B}). \tag{7.16}$$

$$\textbf{part}(\textbf{coolant\_on\_display}, \textbf{inlet}). \tag{7.17}$$

These clauses can be incorporated into specifications that have been derived from formal analyses of principles, such as predictability. For example, **effect_visible** (7.5) required that an operator's view of a system changes in response to the effects of their intervention. The effect of a command to resolve an error can be presented by a change in the image of a coolant system from that shown in Figure 7.5 to that shown in Figure 7.4. The **coolant_error_display** might differ from the **coolant_on_display** because **tank_B** is not part of it. The explicit introduction of graphical details helps designers to clarify the semantics of abstract presentation requirements. For instance, designers could use **part** clauses to replace

Figure 7.5: The graphical decomposition of the *coolant_error_display*

$$\begin{aligned}
&\mathbf{not}(\mathbf{same_display}(\mathbf{coolant_error_display}, \mathbf{coolant_on_display})):\\
&\quad \mathbf{effect_visible}(\text{'}\mathbf{coolant\ ok}\text{'}, \mathbf{coolant_system}) :-\\
&\qquad \mathbf{object_view}(\mathbf{coolant_system}, \mathbf{coolant_error_display}),\\
&\qquad \mathbf{not}(\mathbf{part}(\mathbf{coolant_error_display}, \mathbf{tank_B})),\\
&\qquad \mathbf{message_effect}(\mathbf{coolant_system}, \text{'}\mathbf{coolant\ ok}\text{'}, \mathbf{error}, \mathbf{on}),\\
&\qquad \bigcirc(\mathbf{object_view}(\mathbf{coolant_system}, \mathbf{coolant_on_display}),\\
&\qquad \mathbf{part}(\mathbf{coolant_on_display}, \mathbf{tank_B})). \qquad (7.18)
\end{aligned}$$

Structured graphics systems must be capable of representing a range of different presentation techniques if they are to support the prototyping of interactive control systems. For instance, the circles and squares of direct perception displays, introduced in the Section 6.5, might be presented instead of the tanks and inlets of

Figures 7.4 and 7.5. The conformal representation of components, such as **tank_B**, could be replaced by an **irregular_quadrilateral**:

$$\begin{aligned}&\textbf{effect\_visible}(\text{'}\textbf{coolant ok}\text{'}, \textbf{coolant\_system}) :- \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}), \\ &\quad \textbf{part}(\textbf{coolant\_error\_display}, \textbf{irregular\_quadrilateral}), \\ &\quad \textbf{message\_effect}(\textbf{irregular\_quadrilateral}, \text{'}\textbf{coolant ok}\text{'}, \textbf{error}, \textbf{on}), \\ &\quad \bigcirc(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_on\_display}), \\ &\quad \textbf{part}(\textbf{coolant\_on\_display}, \textbf{regular\_quadrilateral}), \\ &\quad \textbf{not}(\textbf{part}(\textbf{coolant\_on\_display}, \textbf{irregular\_quadrilateral}))). \end{aligned} \tag{7.19}$$

This implication illustrates how designers might easily incorporate changes into the component parts of a graphical structure. It would not have been possible to specify changes to an unstructured bitmap in this manner. It would have been difficult to make such changes using procedural graphics systems because designers must ensure that the new instruction sequences represent the desired image.

7.4.4 Regions

Graphical structures are composed from primitive images. For instance, an **inlet** could be implemented as a primitive image if it had no parts. The graphical appearance of this object might include lines from Cartesian coordinates (0.1,0.2) to (0.6,0.2) and from (0.6,0.2) to (0.6,0.8):

$$\textbf{primitive}(\textbf{inlet}). \tag{7.20}$$
$$\textbf{line}(\textbf{inlet}, 0.1, 0.2, 0.6, 0.2). \tag{7.21}$$
$$\textbf{line}(\textbf{inlet}, 0.6, 0.2, 0.6, 0.8). \tag{7.22}$$

It is a non-trivial task for designers to represent control system displays using individual lines. For example, it would not be easy to define new fonts. A further limitation is that operator input is not usually directed towards lines but towards areas of the screen. A user selecting an inlet icon does not, necessarily, expect to select a particular line of its image. In order to support such interaction, designers must exploit more sophisticated graphical 'building-blocks'. Figure 7.6 illustrates how the image of the *coolant_on_display* for the *coolant_system* can be described in terms of a number of regions: a background region; a text region and an inlet region. The inlet region might be further decomposed into sub-regions containing lines, squares or circles. Each region has properties, such as size and position, attributes, such as font and pattern, and a behaviour, such as whether or not it is selectable. For instance, an **inlet** could be represented as a region with a blank background and dimensions that occupy one twentieth of the screen, positioned at coordinates (0.3, 0.3):

$$\textbf{primitive\_region}(\textbf{inlet}). \tag{7.23}$$
$$\textbf{dimension}(\textbf{inlet}, 0.05, 0.05). \tag{7.24}$$
$$\textbf{position}(\textbf{inlet}, 0.3, 0.3). \tag{7.25}$$
$$\textbf{pattern}(\textbf{inlet}, \textbf{blank}). \tag{7.26}$$

Figure 7.6: The region decomposition for part of the *coolant_on_display*

Designers might describe changes to the attributes of a region using interval temporal logic:

$$\begin{aligned}&\textbf{select\_image}(\textbf{inlet}) : -\textbf{pattern}(\textbf{inlet}, \textbf{blank}),\\ &\quad \bigcirc(\textbf{pattern}(\textbf{inlet}, \textbf{dark}), \textbf{not}(\textbf{pattern}(\textbf{inlet}, \textbf{blank}))).\end{aligned} \tag{7.27}$$

The **inlet** changes its appearance under selection if in the present interval its background is blank and in the next interval its background is not blank but dark (i.e., shaded).

7.4.5 PRELOG And Presenter

In order to demonstrate that structured graphics systems can be incorporated into prototyping tools, we have enhanced PRELOG by linking Tokio with Presenter [307]. The Presenter screen presentation system is an appropriate choice for this task because it constructs images in terms of graphical regions. PRELOG uses Presenter to provide facilities for manipulating region structures and for setting, clearing and interrogating properties of regions. Implementation details, such as raster graphics operations, are isolated within the presentation system. This approach also offers

a high degree of device independence; Presenter has been designed so that it can be ported to any machine offering bitmapped display facilities [307]. PRELOG has been implemented on a network of Sun 3/50s and 3/60s using a client-server model. Shared resources are accessed by a number of clients through a central server. Clients are assumed to be Presenter applications and the resource is the interval temporal logic specification running under Tokio. Communication is via a UNIX socket [193]. Figure 7.7 illustrates the resulting architecture of the PRELOG prototyping system.

Figure 7.7: The PRELOG architecture

Appendix G describes the design and implementation of PRELOG in greater detail.

Figure 7.8 shows a display that was generated using PRELOG. In order to produce such an image, PRELOG constructs a **part** hierarchy using clauses such as (7.16). The region properties, attributes and behaviours of each part, represented by clauses such as (7.25), are then recorded in a tree. This data structure is traversed. Information about each region is passed, via the UNIX socket, to Presenter which runs as an independent process. This is achieved through an interface between PROLOG and the C programming language. A **send**(**String**) predicate is evaluated

Figure 7.8: A PRELOG prototype of the casting control system

as true if the **String** is successfully written by PRELOG to the Presenter socket. For a designer, the net effect of linking PRELOG and Presenter is to provide the impression of a graphical output channel. Despite the provision of these presentation facilities, PRELOG provides inadequate support for the prototyping of interactive control systems. In particular, designers must be able to handle device input if they are to assess the utility of alternative implementations.

7.5 From Graphical Interfaces To Working Prototypes

The choice of input media profoundly affects the usability of many interactive control systems. For instance, Galer and Yap have used prototypes to investigate the costs and benefits of different input devices for intensive care systems [113]. They found that thumb wheels suffer from high error rates, mice were difficult to use in a cluttered clinical environment. In order to avoid design bias it is important that designers can exploit experimental techniques to assess the impact of such devices upon the products of a formal analysis. In order to evaluate the trade-offs that exist between tracker-balls, mice, joysticks and keyboards, human factors and systems engineers must be able to handle a variety of input devices from within the prototypes that embody design principles.

7.5.1 Device Handlers

In safety-critical applications, systems engineers must frequently analyse device behaviour at a low level of detail [49]. It is appropriate, therefore, to consider calling device drivers directly from within a prototype implementation. Device drivers are programs that provide an operating system with a low-level interface to a peripheral unit, such as a mouse or a keyboard. Figure 7.9 represents a data structure that is used by device drivers in Apple Macintosh operating systems [260]. This structure

Figure 7.9: A data structure for a device driver

stores pointers to the `driver routines` that are called when input is received from a particular device. For instance, the following assembler code is part of a routine that 'blinks' the caret when a mouse is moved over a text region:

```
CLR.L           -SP         ;event code for null event is 0
PEA             2(SP)       ;pass null event
CLR.L           -SP         ;pass NIL dialogue pointer
CLR.L           -SP         ;pass NIL pointer
DialogueSelect              ;invoke DialogueSelect
ADDQ.L          #4,SP       ;pop off result and null event
```

Burton, Cook, Gikas, Rowson and Sommerville show how designers might formalise such structures in order to specify human-computer interfaces built using the Apple Macintosh Toolbox [51]. Such descriptions provide an appropriate level of detail for many stages in the development of safety-critical control systems. They are, however, extremely device dependent. The data structure illustrated in Figure 7.9 differs from device to device. The complexity of accessing input at this level of detail might dissuade designers from assessing the costs and benefits of a range of different devices.

7.5.2 Device Specific Models

Julien uses first order logic to model the input mechanisms of the Tektronix 4014 display [168]. He has extended MicroPROLOG to support the implementation of prototypes using input from this device. Operators can select graphical objects by using thumb wheels to move two cross-hairs across the screen. Designers can determine the Cartesian coordinates of the point where these lines cross using a PROLOG query. These can be thought of as questions about the validity of a relation. The following can be read as 'does there exist an **X** and a **Y** such that **X**

is the x-coordinate and **Y** is the y-coordinate of the cross-hairs?':

$$: -\mathbf{cross\_hair}(\mathbf{X}, \mathbf{Y}). \tag{7.28}$$

Input requirements can be represented as constraints upon the position of a graphical object and the cross-hair device. In other words, in order to select an **inlet** users must move the **cross_hair** to the position of that graphical object:

$$\begin{aligned}&\mathbf{effect\_visible}(\mathbf{cross\_hair}(\mathbf{X}, \mathbf{Y}), \mathbf{coolant\_system}) : - \\ &\quad \mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display}), \\ &\quad \mathbf{part}(\mathbf{coolant\_error\_display}, \mathbf{inlet}), \\ &\quad \mathbf{position}(\mathbf{inlet}, \mathbf{X}, \mathbf{Y}), \mathbf{not}(\mathbf{part}(\mathbf{coolant\_error\_display}, \mathbf{tank\_B})), \\ &\quad \mathbf{message\_effect}(\mathbf{coolant\_system}, \mathbf{cross\_hair}(\mathbf{X}, \mathbf{Y}), \mathbf{error}, \mathbf{on}), \\ &\quad \bigcirc(\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_on\_display}), \\ &\quad \mathbf{part}(\mathbf{coolant\_on\_display}, \mathbf{tank\_B})).\end{aligned} \tag{7.29}$$

The problems that limit the utility of device drivers for the prototyping of interactive control systems also limit the utility of device specific models. Relations, such as $\mathbf{cross\_hair}(\mathbf{X}, \mathbf{Y})$, cannot easily be used to represent input from cursor keys. Designers would be forced to change their device model each time they investigated a different input medium. This might dissuade them from assessing what Card, Mackinlay and Robertson call "the design space of input devices" [52].

7.5.3 Input Events

Devices can be represented by the events that they generate. For instance, a keyboard could be represented by a **text_enter** event. This would be generated every time an operator pressed a key. A mouse might be represented by a **select** event. This would be generated every time an operator pressed one of its buttons. Events can be introduced into prototypes by associating them with regions. For example, an operator could be required to respond to an error by selecting the **inlet** using a mouse:

$$\begin{aligned}&\mathbf{effect\_visible}(\mathbf{select}(\mathbf{inlet}), \mathbf{coolant\_system}) : - \\ &\quad \mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display}), \\ &\quad \mathbf{part}(\mathbf{coolant\_error\_display}, \mathbf{inlet}), \\ &\quad \mathbf{not}(\mathbf{part}(\mathbf{coolant\_error\_display}, \mathbf{tank\_B})), \\ &\quad \mathbf{message\_effect}(\mathbf{coolant\_system}, \mathbf{select}(\mathbf{inlet}), \mathbf{error}, \mathbf{on}), \\ &\quad \bigcirc(\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_on\_display}), \\ &\quad \mathbf{part}(\mathbf{coolant\_on\_display}, \mathbf{tank\_B})).\end{aligned} \tag{7.30}$$

An implementation of this clause would not have to be changed if a prototype exploited a tracker-ball or cursor keys instead of a mouse. All of these devices could generate the same **select** event. Such device independence helps to avoid premature commitment to particular hardware platforms [303]. Implementation decisions might be postponed until late in the development cycle when the costs and benefits of a range of different input media have been assessed by the empirical analysis of prototypes.

7.5.4 PRELOG And Input Events

Prototypes provide systems engineers with a tangible representation of the products of formal analyses. They support the experimental analysis which human factors engineers could use to assess the physiological, cognitive, perceptual and sociological demands that are imposed upon operators by potential implementations [293]. In order to reap all of these rewards, PRELOG must translate device primitives into input events. This is done by isolating device details within Presenter. Linking Presenter and PROLOG is complicated because each is intended to run as an independent process. Presenter must notify PRELOG of device input. It is not clear if Presenter should interrupt PROLOG when an input event has been received and, if PROLOG is interrupted, how PRELOG should accommodate this new information within an ongoing proof. For example, if PRELOG was forced to suspend a proof to handle a move event for an icon, it would have to ensure that prior proof steps did not depend on positional information about that graphical object. This would radically affect the nature of the programming environment provided by PROLOG and Tokio. A large number of input events might stretch the resources of any implementation to an unacceptable level. An obvious alternative is to make PRELOG responsible for sampling input from Presenter. In the current implementation, input is accepted by a **receive**(**Event**) predicate which is evaluated as true if **Event** unifies with an input event sent from Presenter via the socket and the C programming language interface to PROLOG. The designer is free to specify when PRELOG should take input events. One drawback to this approach is that it allows important input to be delayed while less important input is processed. Another drawback is that graphical clauses may be inconsistent with respect to the screen image if an operator alters the position of a graphical object during a proof step. This could be partially resolved by 'locking' the screen until PRELOG was free to receive more input but this might prove an unacceptable constraint on any prototype.

Linking Presenter and PROLOG does not resolve all of the problems that might frustrate the prototyping of interactive control systems. In particular, previous prototyping tools have not enabled designers to evaluate the utility of multi-user interfaces. Cleary's graphics system for PROLOG exploits distributed processing to provide co-processor support for geometric transformations [72]. The image of the system is not distributed and interaction is focussed through a single monitor. In contrast, PRELOG is capable of synchronising the presentation of process information over a network of workstations. Input events can be handled from a number of concurrent operators. Figure 7.10 illustrates the revised architecture. This architecture has advantages for user testing; the designer can monitor the execution of a prototype on a remote machine. Appendix H describes the application of PRELOG to support the design of concurrent multi-user systems in greater detail.

7.6 Conclusions

Abstract requirements, derived from formal analyses of principles, provide a framework for the specification of detailed designs. Detailed designs provide blue-prints for the development of prototypes.

Prototypes might be implemented using object oriented programming languages or executable subsets of first order logic. This would impose heavy burdens upon de-

Figure 7.10: The distributed PRELOG architecture

signers who must translate interval temporal logic specifications into the statements of languages, such as C++ or PROLOG. Executable interval temporal logics minimise these burdens. There is a relatively straightforward translation from detailed designs into clauses that can be executed by interpreters such as Tokio.

Bitmaps and procedural graphics notations can represent the images that might be presented by prototypes. It is not easy to alter the displays described by such representations, they cannot be used to identify the target of operator input. Structured graphics systems avoid the limitations of procedural notations and bitmaps. Complex images can be described in terms of their component parts, these might be further decomposed into sub-parts.

Device drivers and device specific models can be used to handle operator input within interactive prototypes. These representations are implementation specific and cannot easily be altered if experimental evidence recommends a different input medium. Device abstractions avoid the limitations of device drivers and device specific models. Devices can be represent by the events that they generate.

In order to demonstrate that designers might exploit interval temporal logics, structured graphics and input events to prototype an interactive control system we have incorporated all of these representations into PRELOG. An important advantage of this tool is that it can run over a local area network. Designers might, therefore, exploit it to assess interaction between the multiple users that are involved in the operation of many interactive control systems.

It has yet to be demonstrated that principles, such as predictability, support the integration of human factors and systems engineering during all stages of the development cycle. Chapter 8 argues that much remains to be done if formal notations, such as interval temporal logic, and prototyping tools, such as PRELOG, are to provide adequate support for the principled design of interactive control systems.

Part IV

Conclusions And Further Work

Introduction To Part IV

This part of the thesis argues that the principled approach, advocated in previous chapters, provides inadequate support for the integration of human factors and systems engineering during some stages in the design of interactive control systems.

Chapter 8 argues that better development techniques must be provided to support the elicitation, verification, refinement and validation of principles. Further work is required in order to capture real-time requirements, risk, and operator expertise in interval temporal logic notations. Existing tools must be improved to support: the generation of detailed specifications; the development of prototype displays and the full implementation of interactive control systems.

Chapter 9 summarises the contribution of this thesis.

Chapter 8

Methodological Inadequacy

> "Extrapolating from today's research programmes it would seem that most advances are likely to come from non-standard logics, e.g. those that have a more sophisticated range of basic notions than first-order predicate calculus... It is useful to be able to represent causality at the requirements level. A number of formalisms, e.g., Petri nets are used to represent data flow oriented causal relations in systems. Similarly, interval temporal logics can be used to represent causal models. Development of sophisticated logic frameworks will be of little value unless we have adequate ways of eliciting and validating such specifications. Consequently, it is to be expected that these logics will be related to more traditional techniques such as fault-tree analysis and failure modes and effects analysis as ways of both deriving the specification and evaluating their fault coverage." (McDermid, [207]).

8.1 Introduction

The following pages argue that the approach advocated in previous chapters of this thesis is methodologically inadequate. An approach to design is methodologically inadequate if it provides insufficient support for some aspects of development. Further research is needed in order to provide designers with:

- better development techniques;
- better notations;
- better tools.

If these are provided then human factors considerations might be introduced into the advanced systems engineering techniques which are enumerated in the quotation that opens this chapter.

8.1.1 Consequences: Ad Hoc Design And Unpredictability

Figure 8.1 presents a three stage model of the human factors and systems engineering of an interactive system [323]. This model has much in common with the structure of this thesis, illustrated in Figure 1.2. Initial design corresponds to the identification of

principles, formative evaluation is similar to detailed design, summative evaluation corresponds to the evaluation of implementations. Different approaches to design

Figure 8.1: A three stage model of interface development

provide different levels of support for each of these stages; principles can guide initial development because they provide objectives prior to the detailed design of a particular interface. It has not been demonstrated that principles can be used during acceptance testing. This lack of support can force designers to rely upon ad hoc expedients. For instance, Bainbridge describes how the tools, development techniques and notations exploited during the design of a casting control system did not support acceptance testing [22]. Plant management had to prevent operators from switching off the automated control systems during night-shifts. If designers had been provided with an adequate methodology then they might have avoided such 'solutions', managers might have avoided many sleepless nights.

Methodological inadequacy threatens principles such as predictability. Human factors and systems engineers could reject the integrated approach advocated by this thesis because it does not support tasks that are considered to be essential during the development of a control system. For instance, the United Kingdom's Atomic Energy Authority's Systems Reliability Service encourages risk analysis as a major activity in the benchmarking of safety-critical applications [154]. It has not been demonstrated that principles can be used in conjunction with risk assessment techniques to support the integration of human factors and systems engineering.

8.1.2 Causes: Inadequate Techniques; Notations And Tools

The approach advocated in this thesis is methodologically inadequate because interval temporal logic cannot represent all of the requirements that must be satisfied by particular control systems. For instance, the development of operational interfaces frequently requires systems engineers to consider the real-time properties of a design. Interval operators, such as $\Diamond$ and $\bigcirc$, do not capture this information. Similarly, it has not been shown that human factors engineers might use interval temporal logic to represent the demands which control tasks place upon operators with different levels of expertise.

The approach advocated in this thesis is inadequate because the PRELOG tool, introduced in Chapter 7, does not provide sufficient support for some stages of development. The specification and implementation of complex displays requires thousands of interval temporal logic clauses. This hinders rapid prototyping. Implementations of PRELOG are slow; this hinders experimental analysis. At present, PRELOG cannot be used to assess presentation techniques that exploit colour graphics, audio output or video images.

The approach advocated in this thesis is methodologically inadequate because it does not provide sufficient support for requirements elicitation. Designers cannot produce more detailed designs or prototype implementations until they have selected the principles that are to guide human factors and systems engineering. Previous chapters have not addressed the problems of refinement and verification. How can designers prove that partial implementations actually satisfy the requirements that are intended to support principles, such as predictability?

8.1.3 Solution: Further Research

The pessimistic review of the previous section provides a necessary corrective to the optimism of previous chapters. The following pages argue that the principled integration of human factors and systems engineering can be salvaged through further research. Designers must be provided with better development techniques, alternative notations and improved tools.

8.2 Improved Development Techniques

The development techniques, advocated in the previous chapters of this thesis, must be improved to support: requirements elicitation; verification; refinement and validation.

8.2.1 Requirements Elicitation

The elicitation, or "drawing forth" [12], of initial design requirements is one of the most critical stages in the development of an interactive control system. If operator requirements are not correctly identified then designers are unlikely to select principles that will be relevant for the users of their interface. Detailed designs and prototype implementations cannot easily be developed until human factors and systems engineers have selected the principles to be embodied within a control system. The problem of eliciting these initial requirements has not been addressed by the

previous chapters of this thesis. Bainbridge argues that these constraints can be identified by interviewing operators about the tasks which they have to perform with existing systems [20]. These interviews are intended to elicit the heuristics, or rules of thumb, that users employ during the operation of their control system:

> "In order to repair the boards of a coolant control system we first have to run a program to test the system, this tells us where the fault is, usually. And we fix it and check with the program again..."

This form of analysis can produce many hundreds of pages of transcripts [271]. Designers, might, therefore, translate them into a more concise representation. For instance, the techniques of Task Analysis for Knowledge Description could be applied to the products of these interviews in order to construct task description hierarchies, such as that illustrated in Figure 8.2 [86]. Designers can trace routes through these

Figure 8.2: A part of a task description hierarchy

hierarchies in order to identify the cognitive and physiological demands of particular tasks. For example, in order to test a coolant control system circuit operators must perform the physical actions of issuing keyboard commands to log onto the system. In order to locate faults users must perform the cognitive actions involved in inspecting component wires. An advantage of this approach is that the hierarchy begins to map out the requirements that must be satisfied by human factors and system engineering. The `ELECTRICAL` branch in Figure 8.2 describes the application functionality to be provided by systems engineering. The `COGNITIVE` branch in Figure 8.2 describes operator requirements that must be supported by human factors engineering. A disadvantage of this approach is that task description hierarchies

do not represent temporal information. For instance, it is not apparent from the hierarchy whether operators can run the test program whilst they are repairing a coolant control system.

Interval temporal logic provides an alternative to task description hierarchies as a means of representing the products of operator interviews. This approach has the advantage that task sequencing can be made explicit. Designers might use predicates to represent the observation that operators run test programs while they repair a circuit:

$$\begin{aligned}&\mathbf{existing\_repair\_task}(\mathbf{board\_A}) \Leftarrow \\ &\quad \mathbf{run}(\mathbf{auto\_test\_program}) \wedge \mathbf{repair}(\mathbf{board\_A}) \wedge \\ &\quad \bigcirc\mathbf{run}(\mathbf{auto\_test\_program})\end{aligned} \tag{8.1}$$

Interval temporal logic also provides a means of avoiding further limitations that restrict the use of task description hierarchies during requirements elicitation. Such products of an initial requirements elicitation contain assumptions that are inappropriate for the early stages of development. Designers should not be bound to develop test programs because they were part of a previous implementation. Similarly, the division of `ELECTRICAL` and `COGNITIVE` requirements between human factors and systems engineering might perpetuate an inappropriate task allocation into future implementations. Such problems can be avoided by translating the products of operator interviews into the object orientated model, introduced in Section 5.2. The **run** task could be interpreted as an operator viewing an application object, repairing a board is analogous to sending a message which alters the state of that object:

$$\begin{aligned}&\mathbf{general\_repair\_task}(\mathbf{repair}, \mathbf{board\_A}) \Leftarrow \\ &\quad \mathbf{object\_view}(\mathbf{board\_A}, \mathbf{board\_A\_error\_display}) \wedge \\ &\quad \mathbf{message\_effect}(\mathbf{board\_A}, \mathbf{repair}, \mathbf{error}, \mathbf{on}) \wedge \\ &\quad \bigcirc\mathbf{object\_view}(\mathbf{board\_A}, \mathbf{board\_A\_on\_display})\end{aligned} \tag{8.2}$$

These products of operator interviews might be used to inform the initial selection of design principles. For instance, designers could take this process of abstraction one step further in order to derive the following predictability requirement:

$$\begin{aligned}&\forall\, \mathbf{m} \in \mathbf{M}, \forall\, \mathbf{o} \in \mathbf{O}, \forall\, \mathbf{os} \in \mathbf{Os}, \exists\, \mathbf{od}, \mathbf{od}' \in \mathbf{Od} : \\ &\quad \mathbf{object\_visible\_effect}(\mathbf{m}, \mathbf{o}, \mathbf{os}, \mathbf{od}, \mathbf{od}') \Leftarrow \exists\, \mathbf{os}' \in \mathbf{Os} \\ &\quad (\mathbf{message\_effect}(\mathbf{o}, \mathbf{m}, \mathbf{os}, \mathbf{os}') \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \\ &\quad \bigcirc(\mathbf{object\_view}(\mathbf{o}, \mathbf{od}') \wedge \neg\, \mathbf{same\_display}(\mathbf{od}, \mathbf{od}')))\end{aligned} \tag{7.4}$$

Johnson is currently using temporal logic to support what he calls "task-based design" [166]. A propositional notation is used to represent operator goals and subgoals in a similar fashion to that shown in the **existing_repair_task** (8.1) predicate. Future work might build upon this research in order to further investigate the use of interval temporal logics during requirements elicitation for principled design.

8.2.2 Verification and Refinement

'Verification' is defined to be "the process or an instance of establishing the truth or validity of something" [12]. In terms of this thesis, verification is the process

of ensuring that the properties or principles recommended by formal and informal analyses are accurately represented in a design. Refinement is defined to be the process of moving from abstract requirements to detailed designs and implementations. Verification and refinement are, therefore, closely related. We are concerned to verify that the products of an abstract design are faithfully carried through successive refinements into an implementation. This is a non-trivial task. For instance, human factors and systems engineers could impose **object_visible_effect** (7.4) as a requirement that is intended to encourage predictability. They must ensure that it is satisfied by control systems that present dozens of displays, handle hundreds of input sequences and have thousands of potential states. Section 7.2.2 neglected this problem in order to demonstrate that such predictates provide a framework for the detailed design of interactive dialogues. This omission can be made good by constructing formal proofs to verify that prototypes satisfy these requirements. In other words, the behaviour of implementations might be shown to conform to the requirements of principled design by reasoning in formalisms, such as interval temporal logic. Tools can help in this task. The Gypsy [123] and m-EVES [80] environments were developed in order to support program verification. There are limits to the scalability of this approach. Automated proofs quickly become difficult to understand and comprehensibility seems to decline as analytical power increases [207].

Specification transformation provides an alternative to program verification. Implementations can be derived from abstract requirements by applying a set of rules that are guaranteed to preserve the properties of an initial specification [306]. This supports verification because rules need only be proven once and can be applied without incurring additional proof obligations. This approach also supports refinement; it can be applied to optimise specifications that are intended to be correct rather than to provide fast or memory efficient execution [265]. Runciman describes how a high-level model of interaction, similar to that presented in Section 2.3.1, can be transformed towards implementation [264]. He exploits partial evaluation which involves the refinement of a general function, or abstract requirement, into a more specific form. Such an approach is particularly appropriate because vernacular requirements, generic principles, detailed designs and prototypes all represent partial evaluations or transformations towards a final implementation. Much of the previous research in this area has concentrated upon transformation towards implementation in functional programming languages. Future work might exploit this research by building upon recent attempts to integrate the functional and logic programming paradigms [60].

8.2.3 Validation

The noun 'validation' is derived from the verb to validate: "to make valid, ratify, confirm" [12]. In terms of this thesis, designers must validate, ratify and confirm the utility of design principles. It is vital not to underemphasise the importance of this task. Verifying that a system satisfies a predictability requirement is of little benefit if operators believe that the system is unpredictable. Validation is complicated by the fact that users, typically, cannot be asked direct questions about properties of a system that are expressed in terms of the $\Diamond$, $\bigcirc$ and $\mathcal{U}$ operators of interval temporal logic. The experimental analysis of prototype implementations,

advocated in Chapter 7, provides one solution to this problem. There are limitations to this approach. The results gained from the experimental analysis of a prototype can be misleading. Differences in the performance and presentation of partial, as opposed to full, implementations restrict the application of findings gained from such analyses. Sections 8.4.2 and 8.4.3 will argue that these limitations affect current versions of PRELOG. The impact of performance and presentation differences upon the experimental evaluation of interactive prototypes remains a subject for human factors research [166].

In anticipation of further research into the experimental analysis of prototype implementations, human factors and systems engineers must have some alternative means of ensuring that users can exploit the principles which have guided design. Thimbleby argues that it is possible to bridge the conceptual gap which separates the designer from the user by expressing principles in a colloquial form [301]. These vernacular representations could be introduced into system documentation and training material in order to highlight the critical decisions that guided development. For example, **object_visible_effect** (7.4) might be incorporated into an on-line help system:

```
   --SYSTEM HELP--
The effects of any input will be presented
by a change in the display immediately after
it has been issued.
```

Vernacular expressions of abstract principles also support design. The annotation of formal specifications with informal explanations has been recommended for interface development in Z [296] and Larch [51]. A weakness of this approach is that it is not easy to translate formal requirements into vernacular error messages, training material or specification annotations. The $\bigcirc$ operator has a clear meaning in terms of a designer's formal analysis. Users are unlikely to interpret `immediately after` in the same way. Does this refer to the next second, minute or hour? Future research intends to avoid such problems by exploiting real-time logic notations in order to support the integration of human factors and systems engineering.

8.3 Improved Notations

The interval temporal logic, introduced in Section 2.4.4, fails to provide adequate means of representing real-time properties, risk assessments and operator expertise.

8.3.1 Real-Time

Real-time is "the actual time (minutes, seconds, hours) during which an event or process occurs" [12]. Real-time deadlines are frequently imposed upon operator input. If users do not respond before certain deadlines then systems engineering can intervene to ensure the safe operation of application processes. Such constraints cannot be expressed using interval temporal logic. This limitation can be avoided by translating temporal operators, such as $\Diamond$, into a real-time logic notation. For instance, designers might require that an error is resolved if the operator stops a coolant system within three seconds of an error being displayed. It should be noted that the following implication could have specified **object_state**(**coolant_system**, **off**) as

a consequent that is true at time **T**1. Section 8.4.1 will briefly describe the problems of temporal perspective that this leads to. For now it is sufficient to realise that designers might refine interval temporal logic requirements to specify real-time constraints:

$$\begin{aligned}&\textbf{error\_resolved}(\text{'}\textbf{off}\text{'}, \textbf{coolant\_system}) \Leftarrow \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}, \mathbf{T}) \wedge \\ &\quad \textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off}, \mathbf{T}1) \wedge (\mathbf{T}1 < \mathbf{T} + 3)\end{aligned} \tag{8.3}$$

This notation suffers from many of the problems associated with the time-variables that were discussed in Section 2.4.2. Designers must construct their model of time using relationships such as $(\mathbf{T}1 < \mathbf{T}+3)$. Although this appears to be a trivial issue for simple requirements, the task of maintaining temporal semantics imposes considerable burdens upon the development of complex control systems. The problems which this can create are illustrated by the following implication in which **T** occurs before and after but not during time **T**1. In other words, there is a hole in time **T**1:

$$\begin{aligned}&\textbf{unusual\_temporal\_model}(\text{'}\textbf{off}\text{'}, \textbf{coolant\_system}) \Leftarrow \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}, \mathbf{T}) \wedge \\ &\quad \textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off}, \mathbf{T}1) \wedge \\ &\quad (\mathbf{T}1 < \mathbf{T}) \wedge (\mathbf{T}1 > \mathbf{T})\end{aligned} \tag{8.4}$$

Interval temporal logics avoid this problem because their model of time is hidden within the definition of operators. Future work might, therefore, develop a notation that hides a real-time model in a similar fashion. For example, the $\bigcirc$ operator was introduced as follows:

$$\forall\, \mathbf{st} \in \mathbf{St} \mid \bigcirc(\mathbf{w}) \mid_{\mathbf{st}} \Leftrightarrow \exists! \mathbf{st}' \in \mathbf{St}[\textbf{immediate\_after}(\mathbf{st}, \mathbf{st}') \wedge \mid \mathbf{w} \mid_{\mathbf{st}'}] \tag{2.27}$$

The predicate $\textbf{immediate\_after}(\mathbf{st}, \mathbf{st}')$ is true at the lowest level of temporal granularity. The state of world knowledge $\mathbf{st}'$ holds in the interval immediately after **st**. The duration of this interval could be linked to a real-time clock. This would provide an alternative interpretation of $\bigcirc(\mathbf{w})$ which might now be read as **w** is true of the state of world knowledge in the next second. This real-time interpretation of $\bigcirc$ is denoted by the # symbol. For example, designers could specify that an error is resolved if a **coolant_error_display** is presented and three seconds afterwards the operator closes the system down:

$$\begin{aligned}&\textbf{real\_time\_error\_resolved}(\text{'}\textbf{off}\text{'}, \textbf{coolant\_system}) \Leftarrow \\ &\quad \textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}) \wedge \\ &\quad \#\#\#(\textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off}))\end{aligned} \tag{8.5}$$

The duration of the # operator might be changed to a minute, an hour or a day depending on the granularity specified by $\textbf{immediate\_after}(\mathbf{st}, \mathbf{st}')$. Ladkin has developed similar techniques for grounding Allen's convex interval logic in real-time [188]. Convex intervals specify periods of time without any gaps, this is analogous to an interval constructed from an unbroken series of **immediate_after** states. This work provides a useful starting point for further research.

8.3.2 Risk

There are a number of variations in the definitions of risk proposed by bodies such as the British Standard's Institute [42], the Institute of Chemical Engineers [156] and the United States' Nuclear Regulatory Commission [313]. Brevity forbids a full analysis of the differences which reflect the differing concerns of these agencies. In contrast, this section exploits a more general definition. 'Risk' is defined to be the "chance or possibility of danger, loss, injury or other adverse consequences" [12]. Assessing the possibility of these adverse consequences is an important task in the systems engineering of many interactive control systems [79]. A number of graphical notations have been developed to support risk assessment. For instance, fault-trees, such as that illustrated in Figure 8.3, represent risk in terms of the conjunctions and disjunctions of events which lead to accidents, such as a `Runaway Reaction`. Rasmussen argues that risk assessment notations are seldom used to rep-

Figure 8.3: A fault-tree

resent the role of human error in system failure [248]. This is a significant omission. If it is rectified then human factors considerations could be integrated into techniques that are in widespread use amongst systems engineers. The notation used in Figure 8.3 conforms to guidelines that have been drawn up by the European Federation of Chemical Engineers [79], similar regulations have been issued by the United

States' Nuclear Regulatory Commission [313]. Designers might use the products of fault-tree analyses to identify high-risk errors that could be simulated during the evaluation of prototype implementations. Human factors engineers could use these simulations to determine whether users can accurately predict the consequences of their intervention during equipment failures. Systems engineers could use them to identify situations in which operators need automated support or advice. In order to do this, it must be possible to represent the causes of high-risk events in a form that can be simulated by prototype implementations. The following implication demonstrates that designers might use first order logic to represent fault-trees, such as that illustrated in Figure 8.3. Such predicates might be used in conjunction with the Prolog prototyping tool, described in Chapter 7:

$$\begin{aligned}&\mathbf{probable\_error}(\mathbf{plant}, \mathbf{runaway\_reaction}) \Leftarrow \\ &\quad (\mathbf{object\_state}(\mathbf{pump}, \mathbf{fail}) \vee \mathbf{object\_state}(\mathbf{line}, \mathbf{block}) \vee \\ &\quad \vee \mathbf{object\_state}(\mathbf{tank}, \mathbf{empty})) \wedge ((\mathbf{object\_state}(\mathbf{tmp\_trip}, \mathbf{high}) \wedge \\ &\quad \mathbf{object\_state}(\mathbf{flow\_trip}, \mathbf{low})) \vee \mathbf{object\_state}(\mathbf{valve}, \mathbf{shut}))\end{aligned} \tag{8.6}$$

A number of limitations restrict the utility of fault-trees for the development of interactive control systems. In particular, they provide an inadequate representation of time. Conjunctions and disjunctions are assumed to hold simultaneously or at some time after their branch conjunctions and disjunctions. The leaves of a tree are assumed to hold simultaneously or at some time before their parents. Figure 8.4 provides an example of a cause-consequence diagram, these explicitly capture the way in which faults develop over time. The notation used in this figure also conforms to the guidelines of the European Federation of Chemical Engineers [79]. The boxes labelled with a `D` represent delays in the sequence of events. As with fault-trees, this notation could be used to guide experimental analyses of operator interaction under error conditions. Designers might exploit interval temporal logic to represent the potential failures identified using cause-consequence diagrams. For instance, the $\Diamond$ operator provides a means of capturing the delay before the `High Temperature Signal` in Figure 8.4:

$$\begin{aligned}&\mathbf{probable\_error}(\mathbf{plant}, \mathbf{dump\_error}) \Leftarrow \\ &\quad \mathbf{object\_state}(\mathbf{coolant\_system}, \mathbf{error}) \wedge \\ &\quad \Diamond \mathbf{message\_effect}(\mathbf{tmp\_trip}, \text{'}\mathbf{increase}\text{'}, \mathbf{low}, \mathbf{high})\end{aligned} \tag{8.7}$$

A limitation of both fault-trees and cause-consequence diagrams is that they do not directly provide a quantitative assessment of potential risks. Bayes' theorem provides a means of avoiding such limitations. This relates the probability of a hypothesis being true to the probability of it being true given some observations of the system [258]. Data about the observation of error can be obtained from sources such as the European Economic Community's Event Data Recording System [204], the National Aeronautics and Space Administration's Aviation Safety Reporting System [319] and the United Kingdom's Atomic Energy Authority's Systems Reliability Service [154]. Bacchus exploits logic to represent the probabilistic risk assessments that might be derived from the application of Bayes' theorem to the data held by these agencies [19]. He uses the following predicate to denote the set of moments, $\mathbf{x}$, when the risk of a **runaway_reaction** exceeds twenty percent:

$$[\mathbf{runaway\_reaction}(\mathbf{x})]_{\mathbf{x}} > 0.2 \tag{8.8}$$

Figure 8.4: A cause-consequence diagram

Bacchus' notation could be used to support the development of interactive control systems. For instance, human factors and systems engineers might identify those moments when a **low_flow_trip** leads to a greater than twenty percent risk of a **runaway_reaction**. Additional resources could then be deployed to alert operators of the potential danger during such intervals of interaction:

$$\mathbf{KB}1 = [\mathbf{runaway\_reaction}(\mathbf{x}) \mid \mathbf{low\_flow\_trip}(\mathbf{x})]_{\mathbf{x}} > 0.2 \qquad (8.9)$$

Bacchus extends his notation to represent the facts that are known about a system at particular moments. For instance, **runaway_reaction**(**t**) represents the fact that a **runaway_reaction** occurs at time **t**. Designers might use this notation to represent changes in the assessment of risks as additional evidence is obtained. For instance, at time **t** it could initially be specified that, given a low flow trip, the likelihood of a **runaway_reaction** is greater than twenty percent. At time **t**1 sensors might indicate that there was a high temperature trip. Designers could, therefore, refine their initial assessment to specify that the probability is greater than twenty percent and less than fifty percent:

$$\mathbf{KB}2 = [\mathbf{runaway\_reaction}(\mathbf{x}) \mid \mathbf{low\_flow\_trip}(\mathbf{x})]_{\mathbf{x}} > 0.2 \wedge$$

$$[\mathbf{runaway\_reaction}(\mathbf{x}) \mid \mathbf{high\_tmp\_trip}(\mathbf{x}) \wedge \mathbf{low\_flow\_trip}(\mathbf{x})]_{\mathbf{x}} < 0.5 \wedge$$
$$\mathbf{high\_tmp\_trip}(\mathbf{t}) \wedge \mathbf{high\_tmp\_trip}(\mathbf{t}1) \wedge \mathbf{low\_flow\_trip}(\mathbf{t}1) \quad (8.10)$$

The time-variables $\mathbf{t}$ and $\mathbf{t}1$ are similar to $\mathbf{T}$ and $\mathbf{T}1$ in **error_resolved** (8.3). They can be instantiated at many different points during interaction. Bacchus does not resolve the problem of maintaining temporal semantics within his notation. He does not explicitly specify the relationship between $\mathbf{t}$ and $\mathbf{t}1$. Designers could use interval temporal operators to avoid such problems. The $\bigcirc$ operator can be used to specify the sequence in which evidence is gathered during interaction:

$$\mathbf{KB}3 = \Diamond([\mathbf{runaway\_reaction} \mid \mathbf{low\_flow\_trip}] > 0.2) \wedge$$
$$\Diamond([\mathbf{runaway\_reaction} \mid \mathbf{high\_tmp\_trip} \wedge \mathbf{low\_flow\_trip}] < 0.5) \wedge$$
$$\mathbf{high\_tmp\_trip} \wedge \bigcirc(\mathbf{high\_tmp\_trip} \wedge \mathbf{low\_flow\_trip}) \quad (8.11)$$

The introduction of interval temporal logic into Bacchus' notation offers a number of benefits over previous representations of risk assessments. For instance, Moray uses discrete numerical probabilities to represent the risk of certain classes of human error [213]. The precision of estimates, such as a fourteen and one half percent probability of a keystroke error per key pressed, is questionable. Such probabilities are better represented as a range of values. Interval temporal logic extensions to Bacchus' notation support this. Low-probability high-cost failures, typically, have a greater impact upon the safe operation of a control system than high-probability low-cost errors. Stochastic utility functions and Markov models have been used to represent the potential costs of system failures [284]. Further research might build upon this work so that a hybrid probabilistic, temporal logic notation could also represent the costs of violating principles, such as predictability [50].

8.3.3 Expertise

'Expertise' is defined to be the "skill, knowledge or judgement" of an expert [12]. The previous chapters of this thesis have not attempted to represent the expertise that is necessary in order to operate an interactive control system. In contrast, user requirements have been expressed in terms of the input that is to be provided in response to the presentation of particular displays. Such obligations can be made explicit by the use of deontic logics. These introduce the $\mathcal{P}$ operator which denotes permissible actions and the $\mathcal{O}$ operator which denotes obligatory actions [174]. Designers might use deontic logic to support the experimental analysis of prototype implementations. Potential interfaces could be assessed in terms of the number of obligations which users fail to fulfill. For instance, designers might require that if a coolant error is displayed then the operator is obliged to turn the system **off**:

$$[\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display})]\mathcal{O}$$
$$\mathbf{message\_effect}(\mathbf{coolant\_system}, \text{'}\mathbf{off}\text{'}, \mathbf{error}, \mathbf{off}) \quad (8.12)$$

The $\mathcal{P}$ operator might be used to formalise the scope of user intervention. For instance, the designers of another control system might assume that if a coolant system error is displayed then operators are permitted to turn the system off:

$$[\mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display})]\mathcal{P}$$
$$\mathbf{message\_effect}(\mathbf{coolant\_system}, \text{'}\mathbf{off}\text{'}, \mathbf{error}, \mathbf{off}) \quad (8.13)$$

The Modal Action Logic (MAL) proposed by the Alvey FOrmal REquirements Specification Techniques (FOREST) project extends deontic logics to represent timed obligations [122]. For example, the following relation specifies that an agent, **ag**, is obliged to complete an action, **ac**, before the end of the interval in which the formula, α, is true:

$$\mathbf{obl}(\mathbf{ag}, \mathbf{ac}, \alpha) \tag{8.14}$$

Using this notation it is possible to specify real-time deadlines. For instance, a **user** must respond to a coolant error before midday on the 4th of December:

$$[\mathbf{coolant\_system}, \mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display})]$$
$$\mathbf{obl}(\mathbf{user}, \mathbf{message\_effect}(\mathbf{coolant\_system}, \text{'}\mathbf{off}\text{'}, \mathbf{error}, \mathbf{off}), 1200\_4\_12) \tag{8.15}$$

Goldsack and Finkelstein note that there is no special treatment of the calendar in MAL [122]. Designers must construct the semantics of deadlines, such as 1200_4_12, using Allen's thirteen temporal relations [11]. This task is not eased by the theoretical flaws in these representations; there are alternative definitions of intervals and instants. For a full discussion of these limitations the interested reader is directed to Galton's critical assessment of Allen's work [115]. Alternatively, designers could integrate deontic operators into interval temporal logics. This would not support the specification of deadlines, such as 1200_4_12. Further work into real-time extensions to the interval temporal logic, proposed in Section 8.3.1, could overcome this limitation. Assuming that the base granularity of # is a second then designers might specify that users are obliged to intervene within two seconds of an error being displayed:

$$\begin{aligned}&\mathbf{deontic\_error\_resolved} \Leftarrow \\ &\quad \mathbf{object\_view}(\mathbf{coolant\_system}, \mathbf{coolant\_error\_display})\mathcal{O} \\ &\quad (\#\#\mathbf{message\_effect}(\mathbf{coolant\_system}, \text{'}\mathbf{off}\text{'}, \mathbf{error}, \mathbf{off}))\end{aligned} \tag{8.16}$$

Operators are unlikely to share the same view of obligatory and permissible actions. Such differences are characteristic of the distinction between novice and expert users. Experienced operators, typically, know when intervention is required [245]. Inexperienced users frequently regard the same intervention as a possible course of action which might, or might not, be pursued. These differences can also affect the utility of design principles. Experts are more likely to make accurate predictions about the effect of their intervention than novices. If human factors engineers could represent these differences then they might formalise the requirements that must be satisfied by operator training. Systems engineers might use these representations to support the implementation of advice giving systems that inform operators of their obligations. Fox provides a notation which designers could exploit to represent different levels of operator expertise [105]. For instance, the following relation specifies that an argument, **arg**, supports an option for intervention, **op**, given some evidence, **ev**. The '+' symbol is used to indicate an argument in favour of an option for intervention, '−' indicates an argument against an option, '++' confirms it and '−−' excludes it:

$$\mathbf{confirm\_argument}(\mathbf{ev}, \mathbf{op}, \mathbf{arg}, +) \tag{8.17}$$

Designers might instantiate this generic relation in order to specify the criteria that operators must satisfy in order to pass the competence examinations that are used in many industries [331]. For instance, users might be asked to describe the appropriate response to an error display for the coolant system. Satisfactory answers could be specified using Fox's logic notation. In order to predict the effect of their intervention experienced operators must know that they are obliged to turn the system **off** when the **coolant_error_display** is presented because there is a coolant system **error**:

$$\begin{aligned}&\textbf{confirm\_argument}(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}),\\ &\qquad \mathcal{O}\textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off}),\\ &\qquad \textbf{object\_state}(\textbf{coolant\_system}, \textbf{error}), +)\end{aligned} \tag{8.18}$$

Interval temporal logic operators could be integrated into Fox's notation. Experienced users might know that they are obliged to respond to a **coolant_error_display** as soon as it is presented. An inexperienced operator might believe that they are permitted to eventually respond to a **coolant_error_display** at any future point in time:

$$\begin{aligned}&\textbf{confirm\_argument}(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}),\\ &\qquad \mathcal{P}(\Diamond\textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off})),\\ &\qquad \textbf{object\_state}(\textbf{coolant\_system}, \textbf{error}), +)\end{aligned} \tag{8.19}$$

It remains to be seen whether such notations have any value for user modelling. Human factors research indicates that relations such as **confirm_argument** (8.17) are at too coarse a level to characterise the cognitive differences between experienced and inexperienced users [257]. Section 1.3.4 introduced Rasmussen's argument that expertise is characterised by the ways in which users store and retrieve control information [246]. Experienced operators exploit skills that are automatically triggered by their observation of a process. Inexperienced users lack these skill and must store options for intervention in the form of rules. If an appropriate rule is not available then they must rely upon their general knowledge. Future work might investigate whether human factors and systems engineers could exploit interval temporal logic to support a fine grained analysis of predictability requirements for different operators.

8.4 Improved Tools

PRELOG provides insufficient support for the generation of detailed specifications, the development of prototype displays and the full implementation of interactive control systems.

8.4.1 Specification Generation

The detailed design of a control system requires many hundreds of interval temporal logic predicates. The difficulty of tracing interactive dialogues through such specifications can prevent human factors and systems engineers from determining the likely behaviour of partial or full implementations. It can also prevent them from determining whether a specification embodies the requirements that are identified

during abstract analyses of design principles. This task can be eased by using tools that provide graphical representations of specifications [132]. For instance, Kramer describes systems that enable designers to represent temporal properties in terms of the passage of tokens through Petri nets [185]. These nets consist of a number

Figure 8.5: A Petri net specification of interaction

of places, represented by hollow circles. Places are connected to transitions, represented by bars. A transition can be 'fired' when all of the places leading to it contain at least one token. A token is represented by a filled dot. Biljon exploits this notation to support the specification of interactive systems [30]. Figure 8.5 illustrates this use of a Petri net. An error is resolved if the `message_effect` and `object_view` places both receive the tokens required to fire their transitions. The passage of time is represented by the flow of tokens from the `object_state` place at the top of the diagram to the `error_resolved` place at the bottom. These representations might support the approach advocated in this thesis if designers are provided with tools that enable them to translate the places and transitions of Petri nets into the predicates of interval temporal logic. The $\Diamond$ and $\bigcirc$ operators could be used to specify that the `object_view` and `message_effect` transitions in Figure 8.5 fire after the coolant system is in the error state:

$$\begin{aligned}&\textbf{error\_resolved}(\text{'}\textbf{off}\text{'}, \textbf{coolant\_system}) \Leftarrow \\ &\quad \textbf{place\_fire}(\textbf{object\_state}(\textbf{coolant\_system}, \textbf{error})) \wedge \\ &\quad \bigcirc(\Diamond(\textbf{place\_fire}(\textbf{object\_view}(\textbf{coolant\_system}, \textbf{coolant\_error\_display}))) \wedge \\ &\quad (\Diamond\textbf{place\_fire}(\textbf{message\_effect}(\textbf{coolant\_system}, \text{'}\textbf{off}\text{'}, \textbf{error}, \textbf{off})))) \end{aligned} \tag{8.20}$$

Designers might derive a number of benefits if PRELOG were enhanced to support Petri net specifications of interactive control systems. Real-time properties might be represented graphically using the timed Petri nets proposed by Chretienne [68]. In the early stages of development it might not be possible to identify all the places and transitions that will be necessary in order to describe a final implementation. The abstractions of interval temporal logic provide a means of representing parts of an initial design without specifying the interconnections between the components of a Petri net. Conversely, interval temporal logic requirements can become intractable in the later stages of development. The graphical representation of Petri nets helps to minimise the problems of tracing complex dialogues through large specifications. Petri nets also provide a bridge between interval temporal logics and

Figure 8.6: The relationship between Petri nets and fault-trees

the products of risk assessment. Figure 8.6 illustrates how the gates of fault-trees can be represented by the places and transitions of Petri nets [155]. Further research might implement a tool that supports these different notations for the human factors and systems engineering of interactive control systems. This would avoid the need to manually translate between different representations of a design. The development of such a tool raises a number of research issues. In particular, future work must consider how tools are to translate between the different temporal perspectives provided by these representations. In Figure 8.5 the `error_resolved('off', coolant_system)` transition fires at the end of the dialogue whilst the truth value of **error_resolved** (8.20) is determined in the first interval. This is the problem of temporal perspective that was alluded to in Section 8.3.1.

8.4.2 Display Development

Section 7.4 has argued that PRELOG's structured graphics system provides a powerful means of implementing prototype displays. It remains a non-trivial task for designers to develop complex graphical images using the predicates that are supported by this tool. Visual programming languages can be used to resolve this

problem [38]. For instance, Figure 8.7 shows how a designer might define the mid-point of a line. The X coordinates of the end-points, `P1` and `P2`, are added and

Figure 8.7: The visual specification of graphical images

then the result is divided by two. This procedure is repeated for the Y axis in order to derive the Cartesian coordinates of the midpoint `M`. Although this form of visual specification is cumbersome, it describes a constraint that can be re-used to specify the mid-point of a line in many different displays. Such tools might be extended to support the specification and generation of the graphical images presented by PRELOG. Designers could rapidly 'mock-up' displays by directly manipulating graphical objects [222]. Figure 8.8 illustrates a primitive tool that has been implemented to demonstrate the feasibility of this approach. The buttons labelled `Circle` and `Line` can be selected to generate simple images. The menu at the bottom left of the figure can be used to manipulate components of the **part** structure described in Section 7.4. The button labelled `Parse` can be selected to write the logic representation of an image to a file for later consultation by PRELOG. The advantages of this approach are illustrated by the contents of a file that describes the image of the inlet icon shown in Figure 8.8. The following clauses are presented using the PROLOG notation implemented by PRELOG:

$$\mathbf{part}(\mathbf{coolant\_on\_display}, \mathbf{inlet}). \tag{8.21}$$
$$\mathbf{line}(\mathbf{inlet}, 0.1, 0.3, 0.45, 0.3). \tag{8.22}$$
$$\mathbf{line}(\mathbf{inlet}, 0.1, 0.35, 0.45, 0.35). \tag{8.23}$$
$$\mathbf{line}(\mathbf{inlet}, 0.45, 0.25, 0.45, 0.4). \tag{8.24}$$
$$\mathbf{line}(\mathbf{inlet}, 0.6, 0.25, 0.6, 0.4). \tag{8.25}$$
$$\mathbf{line}(\mathbf{inlet}, 0.45, 0.25, 0.6, 0.25). \tag{8.26}$$
$$\mathbf{line}(\mathbf{inlet}, 0.45, 0.4, 0.6, 0.4). \tag{8.27}$$
$$\mathbf{line}(\mathbf{inlet}, 0.45, 0.25, 0.6, 0.4). \tag{8.28}$$
$$\mathbf{line}(\mathbf{inlet}, 0.6, 0.25, 0.45, 0.4). \tag{8.29}$$

Figure 8.8: A graphical generation tool for PRELOG

$$\textbf{line}(\textbf{inlet}, 0.6, 0.3, 0.8, 0.3). \tag{8.30}$$
$$\textbf{line}(\textbf{inlet}, 0.6, 0.35, 0.75, 0.35). \tag{8.31}$$
$$\textbf{line}(\textbf{inlet}, 0.8, 0.3, 0.8, 0.7). \tag{8.32}$$
$$\textbf{line}(\textbf{inlet}, 0.75, 0.35, 0.75, 0.71). \tag{8.33}$$

Future research might address the problems of image specification and generation by providing a natural language front-end to PRELOG. Two M.Sc. students have recently extended PRELOG to provide a restricted natural language interface for visually handicapped users [254, 324]. This tool derives a logic representation that can be used to generate graphical images from sentences such as:

```
Draw a square of size 4 units.
Move the square to position (2.0, 4.0).
```

Prototypes which do not closely resemble final implementations have been used to support the human factors and systems engineering of many interactive control systems. For instance, the United States' Air Force employs simple computer generated images to evaluate avionics displays [54]. Further work is required in order to determine whether partial implementations can be employed to assess the physiological demands of different control room layouts and workstation designs. The results gained from experimental analyses of low-fidelity prototypes are not necessarily applicable to a final control system [69]. In order to avoid such problems during the later stages of development, designers might develop high-fidelity prototypes that accurately resemble a full implementation. Future research might

build upon McCrobie's work on the integration of prototype control systems into full-scale simulations of working environments [206]. Current versions of PRELOG provide inadequate support for this because they are monochromatic and mute. Future implementations might be enhanced to provide colour attributes for graphical regions:

$$\mathbf{colour}(\mathbf{inlet}, \mathbf{blue}). \tag{8.34}$$

Many control systems use audio output to present information about application processes [116]. Designers might use PRELOG to evaluate such presentation techniques if it supported auditory displays. For instance, **noise** predicates could represent pre-recorded warnings. Interval temporal logic operators might be recruited to specify the duration of these sounds. For example, audio output could be used to inform operators of normal radiation levels. The sound might continue until contamination is detected:

$$\begin{aligned}&\mathbf{audio\_display}(\mathbf{coolant\_system}, \mathbf{error}) : - \\ &\quad (\mathbf{noise}(\mathbf{ok\_bell}) \mathcal{U} \mathbf{object\_state}(\mathbf{coolant\_system}, \mathbf{error})).\end{aligned} \tag{8.35}$$

Such high-level noises can be refined into sounds possessing attributes such as timbre or character, pitch in Hertz and amplitude in decibels:

$$\mathbf{timbre}(\mathbf{ok\_bell}, \mathbf{bell}). \tag{8.36}$$

$$\mathbf{amplitude}(\mathbf{ok\_bell}, 60\mathbf{dBA}). \tag{8.37}$$

$$\mathbf{pitch}(\mathbf{ok\_bell}, 260\mathbf{Hz}). \tag{8.38}$$

Video output is increasingly important in the presentation of many process control systems. Temporal logic has some potential as a means of supporting image recognition. It is hypothesised that the introduction of temporal dependencies will significantly increase the power of current modelling techniques for image analysis [66]. Future work might investigate the utility of interval temporal logic as a means of representing the video images that are presented to system operators.

8.4.3 Implementation

Prototyping tools provide designers with the means of rapidly developing partial implementations of control systems. These tools frequently achieve this by sacrificing the objectives of fast and efficient execution. These objectives must be satisfied by full implementations. Such sacrifices are a potential source of methodological inadequacy. Designers must ensure that the principles which guide the development of a prototype are propagated into the final control system. This is difficult because interval temporal operators, such as $\Diamond$ or $\bigcirc$, are not provided by programming languages such as Ada or C. There are a number of solutions to this problem. For instance, the speed and efficiency of PRELOG could be enhanced so that it might be used to develop final implementations. The developers of PRELOG's temporal logic interpreter anticipate the advent of hardware support for their environment [108]. PRELOG has recently been ported to run on Sun SPARCstations. Alternatively, existing programming languages might be extended to support the implementation of interval temporal logic specifications. For instance, Donner and Jahanian have developed a version of the ORE programming language that executes temporal logic

formulae [89]. The authors do not provide the full form of their acronym. ORE avoids many of the efficiency problems that restrict the utility of PROLOG based programming environments. Designers might embed predictability requirements, such as **object_visible_effect** (7.4), directly within an implementation. For example, an ORE procedure might be implemented to ensure that an error is always raised if user input, `M`, changes the state of a control system, from `Os` at instant `i-1` to `Os1` at instant `i`, and its displays `Od` and `Od1` are equivalent:

```
proc check_display (M: message_type, O: object_type)
-- Warns operator of silent state transition.
-- Type definitions omitted for sake of brevity.
{
     var Os,Os1:  object_state_type;
     var Od,Od1:  display_type;
     var i :  int;

     maintain forall i -- equivalent to always
     if(@(message_effect(O, M, Os, Os1), i-1) &&
       @(object_view(O, Od), i-1) &&
       @(object_view(O, Od1), i) &&
       @val(Os, i-1) != @val(Os1, i) &&
       @val(Od, i-1) == @val(Od1, i))
           then warn(potential_unpredictability);
}
```

Future research might determine whether real-time operators, such as #, can be implemented using ORE time-variables, such as `i` and `i-1`.

8.5 Conclusion

This chapter has argued that the approach to design, advocated in this thesis, is inadequate for the integration of human factors and systems engineering. Further research has been proposed in order to provide designers with:

- improved development techniques that support requirements elicitation, verification, refinement and validation;
- improved notations that can represent real-time properties, risk assessments and operator expertise;
- improved tools that support the generation of specifications, the design of advanced displays and the full implementation of control systems.

It is hypothesised that if this research is successful then human factors considerations might play an important part in the systems engineering of future control systems. Chapter 9 summarises the contribution that this thesis makes towards the integration of human factors and systems engineering.

Chapter 9

Conclusions

> "The view has been expressed to us, and we find it convincing, that it is impossible for an inspecting body adequately to ensure safety by analysing the completed design of a large, complex plant. Safety is built into the design as it progresses and the primary tasks of the inspecting authority should be to ensure that the proper analysis techniques and disciplines are used at every stage of the design process to achieve clearly defined safety objectives" (The Royal Commission On Environmental Pollution, 6th Report, [139]).

9.1 The Contribution Of This Thesis

Chapter 1 argued that design principles provide a means of integrating the human factors and systems engineering of interactive control systems. It was argued that the task-artifact cycle does not support this integration. Interfaces are assessed after they have been implemented. The costs of making changes to a control system are likely to be prohibitive at this stage of development. It was argued that principled design avoids this limitation by providing heuristics that can be used to guide human factors and systems engineering. Predictability was chosen as an exemplar principle. It was argued that vernacular expressions of such requirements can often be vague and imprecise. It was proposed that designers might avoid this limitation by using formal, mathematically based notations to represent the constraints imposed by principles.

Chapter 2 identified a notation that supports the integration of human factors and systems engineering. Production systems were rejected because their facts, rules, meta-rules and meta-meta-rules cannot easily represent requirements that change over time. First order predicate logic was rejected because it cannot easily represent dialogue sequences. Interval temporal logic was advocated as a notation that avoids these limitations. It was argued that human factors and systems engineers could exploit interval temporal logic in order to represent techniques, such as flexible task allocation and display optimisation, that might be used in order to achieve principles, such as predictability.

Chapter 3 argued that complexity hinders the design and operation of interactive control systems. Users must detect deviations in large amounts of interconnected application data and must select appropriate commands from the many options

that are available to them. Principles provide designers with high-level objectives for techniques that might support these control tasks. It was argued that human factors and systems engineers could use interval temporal logic to represent these techniques without representing thousands of sensor readings and command options. It was concluded that the utility of techniques, such as state restriction, depends upon the degree to which designers can achieve the integration of human factors and systems engineering.

Chapter 4 argued that principles provide generic requirements that can be used to guide the development of many different control systems. In particular, predictability was used as a criterion against which to assess the utility of techniques that are intended to support interaction with a range of open control systems. The input and output protocols of systems engineering were shown to threaten predictability by restricting operator access to shared resources. Techniques that rely upon the human factors observations of social ergonomics threaten safety because they cannot guarantee cooperation between system operators. It was argued that designers must integrate human factors and systems engineering if they are to support predictable interaction with open systems. Interval temporal logic provided a means of representing generic requirements for integrated techniques, such as handshaking and independent views.

Chapter 5 argued that the choice of development architecture can affect the human factors and systems engineering of interactive control systems. It was further argued that designers might use interval temporal logic to represent the support which particular architectures provide for principles, such as predictability. This formalism was used to assess claims that object oriented development supports interface consistency. It was shown that type instantiation does support consistency and that this, in turn, supports predictability. It was argued that some object oriented techniques, such as dynamic classification and method overriding, can sacrifice these benefits.

Chapter 6 argued that principles provide a criterion against which to assess techniques that are intended to resolve the weaknesses of development architectures. In particular, it was argued that the objective of predictability encourages designers to seek solutions for the problem of break-down in object oriented control systems. Conformal and direct perception displays provide only limited support for this principle. Indirect presentation techniques were advocated as alternatives that support predictability by displaying different representations of process components. This approach requires the integration of human factors and systems engineering. Human factors engineers must ensure that operators can monitor and assimilate the information presented by different control system objects. Systems engineers must provide adequate sources of information in order to support multiple representations of physical components.

Chapter 7 argued that design bias occurs when either formal or experimental analyses dominate the development of a control system. This bias hinders integration because formal analyses are typically conducted by systems engineers whilst experimental analyses are usually conducted by human factors engineers. It was argued that design bias can be avoided by developing prototypes which satisfy the requirements gleaned from a formal analysis of design principles. These prototypes are amenable to experimental evaluation and provide a means of assessing whether principles, such as predictability, have any relevance for system operators. Proto-

types can be used to assess the physiological ergonomics of potential implementations. They can also be used to determine the cognitive and perceptual demands that a control system might place upon its operators. Unfortunately, the introduction of implementation and presentation details can sacrifice the tractability of a formal specification. It was shown that these problems might be avoided by the use of executable interval temporal logics, structured graphics systems and generic input events. The implementation of PRELOG, a tool for the Presentation and REndering of LOGic specifications of interactive systems, was briefly described.

Chapter 8 argued that the approach advocated in this thesis provides inadequate support for the integration of human factors and systems engineering. Development techniques must be enhanced to support requirements elicitation, verification, refinement and validation. Improved notations must be developed to represent real-time properties, risk assessments and operator expertise. Improved tools must be provided to support the generation of specifications, the design of advanced displays and the full implementation of interactive control systems.

Chapter 9 has summarised the contribution of this thesis.

Despite the caveats of methodological inadequacy that were raised in Chapter 8, the contribution of this thesis can be summed up by re-iterating the words of the Royal Commission On Environmental Pollution which opened this chapter [139]. The focus of this thesis has moved from the abstract to the concrete. The discussion has ranged from the fundamental problems of control, to the problems of a development architecture, to the problems of designing a particular interface. At each level we have endeavoured to maintain a principled approach to design. This has provided a framework which helps to ensure that “safety is built into design as it progresses”. Interval temporal logic has supported “proper analysis techniques” from the earliest stages of design to partial implementations. Principles, such as predictability, provide the discipline necessary to achieve formally defined objectives for the human factors and systems engineering of interactive control systems.

Part V

Appendices

Appendix A

The Interval Temporal Logic Language

This appendix cites research into the wider applications of interval temporal logic. The syntax and semantics of the language used in this thesis are presented. The choice of a predicate temporal logic is justified.

A.1 Background

Interval temporal logic has been widely applied to represent and reason about computer programs. For instance, Clarke and Emerson have developed their Computation Tree Logic to synthesise the synchronisation skeletons of concurrent programs from proposition temporal logic specifications [71]. Lamport has developed his TIMESET notation to reason about liveness and safety properties in an intuitive manner [189]. Saake and Lipeck have developed a many-sorted first-order temporal logic in order to analyse the update behaviour of databases [267]. The interested reader is directed to [18, 37, 184] and the proceedings of the ACM conferences on the Principles of Programming Languages for background material about the development and application of temporal logic formalisms.

A.2 Syntax

It is important to re-iterate that this thesis does not attempt to develop a new form of interval temporal logic, rather it attempts to apply the findings of previous research. The interval temporal logic described in Section 2.4.4 is based upon that proposed by Manna and Pnueli [205]. This section presents the syntax of the language used in this thesis.

A.2.1 Symbols

The interval temporal logic language uses elements of a set of symbols to represent variables, constants, propositions, functions and predicates. This set can be partitioned into global and local symbols. Global symbols have a uniform interpretation; they do not change their value or meaning from state to state. In contrast, local symbols may assume different meanings and values from state to state. The set of

symbols can also be partitioned into different sorts. These correspond to different domains. For instance, the following symbols are associated with the domain of natural numbers:

$$0, 1, 2, 3, 4, 5, ..., +, -, \text{div}, \times, ..., \geq, \leq$$

For each sort there may be constants that are interpreted over the associated domain. There can also be variables which assume values from that domain, function symbols that represent functions over the domain and predicate symbols which represent predicates over that domain.

A.2.2 Operators And Quantifiers

The logic contains the set of boolean connectives:

$$\wedge, \vee, \neg, \Leftrightarrow, \Rightarrow, \Leftarrow$$

The logic also contains the first-order quantifiers $\exists$ and $\forall$ together with the temporal operators:

$$\bigcirc, \Box, \Diamond, \mathcal{U}$$

The first three of these are unary operators, whilst $\mathcal{U}$ is a binary operator.

A.2.3 Terms

Terms are built from the application of functions and predicates to constants and variables. Applications must satisfy the sort and arity restrictions associated with each of the symbols. It should also be noted that if $\mathbf{t}$ is a term then so is $\bigcirc\mathbf{t}$.

A.2.4 Formulas

Formulas are constructed from the application of boolean connectives, temporal operators or quantifiers to terms. Atomic formulas are propositions and predicates applied to terms which are of an appropriate sort.

A.3 Semantics

This section presents the model-based semantics of the interval temporal logic language used in this thesis [205]. A model $(\mathbf{I}, \alpha, \delta)$ for the predicate temporal logic consists of a global interpretation, $\mathbf{I}$, a global assignment, α, and a sequence of states, δ. $\mathbf{I}$ specifies a domain corresponding to each sort and assigns concrete elements of that domain to symbols. α assigns a value over an appropriate domain to global free variables and propositions. $\delta = \mathbf{s}_0, \mathbf{s}_1, ...$ is an infinite sequence of states where each $\mathbf{s_i}$ assigns values to local free variables and propositions. The following provides an inductive definition of the truth value of temporal formula $\mathbf{w}$ in a model $(\mathbf{I}, \alpha, \delta)$. $\mathbf{I}$ is implicitly assumed, the value of a subformula or term, τ, is denoted by $\tau\ |^{\alpha}_{\delta}$:

- For a local variable or proposition, $\mathbf{y}$:

  $$\mathbf{y} \mid^{\alpha}_{\delta} = \mathbf{y}_{\mathbf{s}_0}$$

  The value assigned to $\mathbf{y}$ in state $\mathbf{s}_0$ is that assigned in the first state of δ.

- For a global variable or proposition $\mathbf{u}$:

  $$\mathbf{u} \mid^{\alpha}_{\delta} = \alpha[\mathbf{u}]$$

  The value assigned to $\mathbf{u}$ is that assigned to $\mathbf{u}$ by α.

- For a constant, $\mathbf{c}$, the evaluation is determined by $\mathbf{I}$:

  $$\mathbf{c} \mid^{\alpha}_{\delta} = \mathbf{I}[\mathbf{c}]$$

- For a function, $\mathbf{f}(\mathbf{x}_1, ..., \mathbf{x_k})$:

  $$\mathbf{f}(\mathbf{t}_1, ..., \mathbf{t_k}) \mid^{\alpha}_{\delta} = \mathbf{I}[\mathbf{f}](\mathbf{t}_1 \mid^{\alpha}_{\delta}, ..., \mathbf{t_k} \mid^{\alpha}_{\delta})$$

  The value of the application is that of the interpreted function, I[f], applied to the values of $\mathbf{t}_1, ..., \mathbf{t_k}$ in the model $(\mathbf{I}, \alpha, \delta)$.

- For a term $\mathbf{t}$:

  $$\bigcirc\mathbf{t} \mid^{\alpha}_{\delta} = \mathbf{t} \mid^{\alpha}_{\delta^1}$$

  The value of $\bigcirc\mathbf{t}$ in $\delta = \mathbf{s}_0, \mathbf{s}_1, ...$ is given by the value of $\mathbf{t}$ in the shifted sequence $\delta^1 = \mathbf{s}_1, \mathbf{s}_2, ...$

- For a predicate $\mathbf{p}(\mathbf{x}_1, ..., \mathbf{x_k})$:

  $$\mathbf{p}(\mathbf{t}_1, ..., \mathbf{t_k}) \mid^{\alpha}_{\delta} = \mathbf{I}[\mathbf{p}](\mathbf{t}_1 \mid^{\alpha}_{\delta}, ..., \mathbf{t_k} \mid^{\alpha}_{\delta})$$

- For a disjunction:

  $$(\mathbf{w}_1 \vee \mathbf{w}_2) \mid^{\alpha}_{\delta} = \texttt{ true iff } \mathbf{w}_1 \mid^{\alpha}_{\delta} = \texttt{ true or } \mathbf{w}_2 \mid^{\alpha}_{\delta} = \texttt{ true}$$

- For a negation:

  $$(\neg\, \mathbf{w}) \mid^{\alpha}_{\delta} = \texttt{ true iff } \mathbf{w} \mid^{\alpha}_{\delta} = \texttt{ false}$$

- For a $\bigcirc$ application:

  $$\bigcirc\mathbf{w} \mid^{\alpha}_{\delta} = \mathbf{w} \mid^{\alpha}_{\delta^1}$$

- For a $\Box$ application:

  $$\Box\mathbf{w} \mid^{\alpha}_{\delta} = \texttt{ true iff for every k} \geq 0, \mathbf{w} \mid^{\alpha}_{\delta^{\mathbf{k}}} = \texttt{true}$$

  $\mathbf{w}$ is true for all suffix sequences of δ.

- For a $\Diamond$ application:

$$\Diamond \mathbf{w} \mid^{\alpha}_{\delta} = \texttt{ true iff there exists k} \geq 0, \mathbf{w} \mid^{\alpha}_{\delta^{\mathbf{k}}} = \texttt{true}$$

  $\mathbf{w}$ is true in at least one suffix of δ.

- For an $\mathcal{U}$ application:

$$\begin{aligned} &\mathbf{w}_1 \mathcal{U} \mathbf{w}_2 \mid^{\alpha}_{\delta} = \texttt{ true} \\ &\texttt{ iff for some k} \geq 0, \mathbf{w}_2 \mid^{\alpha}_{\delta^{\mathbf{k}}} = \texttt{true} \\ &\texttt{ and for all i}, 0 \leq \mathbf{i} < \mathbf{k}, \mathbf{w}_1 \mid^{\alpha}_{\delta^{\mathbf{i}}} = \texttt{true} \end{aligned}$$

- For a universal quantification:

$$\begin{aligned} &(\forall \mathbf{u}.\mathbf{w}) \mid^{\alpha}_{\delta} = \texttt{ true} \\ &\texttt{ iff for every } \mathbf{d} \in \mathbf{D}, \mathbf{w} \mid^{\alpha'}_{\delta} = \texttt{true} \end{aligned}$$

  Where $\alpha' = \alpha \circ [\mathbf{u} \leftarrow \mathbf{d}]$ is the assignment obtained from α by assigning $\mathbf{d}$ to $\mathbf{u}$. $\mathbf{D}$ is the domain over which $\mathbf{u}$ ranges. For notational convenience, universal quantification is represented as $\forall \mathbf{u} \in \mathbf{U} : \mathbf{w}$ where the symbol $\mathbf{U}$ is the sort associated with a domain $\mathbf{D}$.

- For an existential quantification:

$$\begin{aligned} &(\exists \mathbf{u}.\mathbf{w}) \mid^{\alpha}_{\delta} = \texttt{ true} \\ &\texttt{ iff for some } \mathbf{d} \in \mathbf{D}, \mathbf{w} \mid^{\alpha'}_{\delta} = \texttt{true} \end{aligned}$$

  Where $\alpha' = \alpha \circ [\mathbf{u} \leftarrow \mathbf{d}]$. For notational convenience, existential quantification is represented as $\exists \mathbf{u} \in \mathbf{U} : \mathbf{w}$ where the symbol $\mathbf{U}$ is the sort associated with a domain, $\mathbf{D}$, as mentioned in the previous item.

A.4 Predicate Or Propositional Logics?

The decision to choose a predicate temporal logic was based upon a literature survey of the applications cited in Section A.1. Lamport argues that predicate formalisms support "informal mathematical reasoning, and can help us to avoid mistakes in proofs involving quantification" [189]. Saake and Lipeck build upon this argument and suggest that many sorted, first order temporal logics are the "obvious choice" for representing and reasoning about complex, dynamic systems [267].

The decision to adopt a predicate temporal logic helps to derive an intuitive model for our language. For instance, it provides a means of differentiating between a weak and strong semantics for necessitation [268]. This can be illustrated by the statement that 'it is always the case that the safety-system is on'. A strong semantics requires that this is not true if there are intervals in which it is impossible to identify the state of the safety system. This interpretation of $\Box$ is captured by the semantics introduced in the previous section:

$$\Box \mathbf{w} \mid^{\alpha}_{\delta} = \texttt{ true iff for every k} \geq 0, \mathbf{w} \mid^{\alpha}_{\delta^{\mathbf{k}}} = \texttt{true}$$

A weak semantics requires that the statement is true if there are intervals in which it is not possible to identify the state of the safety system:

$$\Box(\exists \mathbf{u} \mid_{\delta}^{\alpha} \Rightarrow \mathbf{w}.\mathbf{u} \mid_{\delta}^{\alpha})$$

The extension of a propositional temporal logic to a first order temporal logic also creates certain problems. The intended semantics of a temporal formula with free variables is that the formula is valid for each possible substitution of the variables with objects during interaction. This leads to reduct quantification; quantification ranges over the lifetime of an object rather than ranging over the object sort carrier. Saake shows how this approach can be used to provide semantics for substitution independent validity which requires quantification over all objects throughout interaction [266]. It should be noted, however, that many of the properties described in this thesis do not require such additional expressive power.

Appendix B

Theorems Of Interval Temporal Logic

This thesis does not attempt to develop a new form of interval temporal logic, rather it attempts to apply the findings of previous research. The interval temporal logic described in Section 2.4.4 is based upon that proposed by Manna and Pnueli [205]. They list a number of valid statements which are presented as axioms or are deduced as theorems from other interval temporal logic statements. For the sake of brevity we only present those statements that are used in this thesis. These hold in addition to the theorems and axioms of first order predicate logic. It should be noted that the interval temporal logic is intended to support particular objectives; namely representing and reasoning about human factors and systems engineering. The following theorems and axioms are, therefore, not intended to provide a minimal framework nor are they intended to provide the most general language possible. Manna and Pnueli's reference scheme is used with an 'M' prefix:

$$\models \quad \Box\neg\, \mathbf{w}_1 \equiv \neg\, \Diamond \mathbf{w}_1 \tag{M1}$$
$$\models \quad \Diamond\neg\, \mathbf{w}_1 \equiv \neg\, \Box \mathbf{w}_1 \tag{M2}$$
$$\models \quad \Box \mathbf{w}_1 \Rightarrow \mathbf{w}_1 \tag{M5}$$
$$\models \quad \Box \mathbf{w}_1 \Rightarrow \Diamond \mathbf{w}_1 \tag{M7}$$
$$\models \quad \Box(\mathbf{w}_1 \wedge \mathbf{w}_2) \equiv (\Box \mathbf{w}_1 \wedge \Box \mathbf{w}_2) \tag{M16}$$
$$\models \quad \Diamond(\mathbf{w}_1 \vee \mathbf{w}_2) \equiv (\Diamond \mathbf{w}_1 \vee \Diamond \mathbf{w}_2) \tag{M17}$$
$$\models (\Box \mathbf{w}_1 \vee \Box \mathbf{w}_2) \Rightarrow \Box(\mathbf{w}_1 \vee \mathbf{w}_2) \tag{M23}$$
$$\models \Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2) \Rightarrow (\Diamond \mathbf{w}_1 \wedge \Diamond \mathbf{w}_2) \tag{M24}$$
$$\models \Box(\mathbf{w}_1 \Rightarrow \mathbf{w}_2) \Rightarrow (\Box \mathbf{w}_1 \Rightarrow \Box \mathbf{w}_2) \tag{M27}$$

Informally, statement M1 says that a formula $\mathbf{w}_1$ is false in all intervals if and only if there is no state in which $\mathbf{w}_1$ is true. Statement M2 says that there is an interval in which $\mathbf{w}_1$ is false if and only if it is not the case that $\mathbf{w}_1$ is always true. Statement M5 says that if $\mathbf{w}_1$ is true in all intervals then it is true in the present interval. Statement M7 says that if $\mathbf{w}_1$ is true in all intervals then it is eventually true. Manna and Pnueli's statement M16 says that the $\Box$ operator distributes over conjunction. Statement M17 says that there is an interval in which either $\mathbf{w}_1$ or $\mathbf{w}_2$ hold if there is an interval in which $\mathbf{w}_1$ is true or an interval in which $\mathbf{w}_2$ is true. Statement M23 states that if either $\mathbf{w}_1$ is true for all intervals or $\mathbf{w}_2$ is true for all intervals then in every interval either $\mathbf{w}_1$ or $\mathbf{w}_2$ holds. Statement M24 states

that if there exists an interval in which both $\mathbf{w}_1$ and $\mathbf{w}_2$ are true then there exists an interval in which $\mathbf{w}_1$ is true and these exists an interval in which $\mathbf{w}_2$ is true. Statement M27 states that if it is always true that $\mathbf{w}_1$ implies $\mathbf{w}_2$ then it is true that if $\mathbf{w}_1$ is always true then $\mathbf{w}_2$ is always true.

The following sections use natural deduction to prove the remaining formulae exploited in the re-writing of Appendices C, D, E and F. It is assumed that the reader is familiar with the notions of premise and assumption, as well as the various introduction and elimination rules for first order logics [328]. Manna and Pnueli's notational conventions for temporal operators are retained. In order to enhance readability, the names of first order theorems are also applied to their counterparts in interval temporal logic. References to theorems such as De Morgan's Laws should be regarded as abbreviations, in this instance for De Morgan's Laws of interval temporal logic. The following proofs are confined to propositions; $\mathbf{w}_1$, $\mathbf{w}_2$ and $\mathbf{w}_3$. This is justified by Hughes and Cresswell's observation that the "derived rules for (modal and temporal) propositional systems hold in the corresponding predicate calculi" [153]. This is only true if our predicate interval temporal logic includes Barcan's formula as a theorem, in the following $\mathbf{p}_1$ is a predicate and $\mathbf{x}$ is a term:

$$\models \forall \mathbf{x} \Box \mathbf{p}_1(\mathbf{x}) \Rightarrow \Box \forall \mathbf{x}.\mathbf{p}_1(\mathbf{x}) \tag{B.1}$$

Manna and Pnueli include the following stronger equivalences in their interval temporal logic:

$$\models \quad \Diamond \exists \mathbf{x}.\mathbf{p}_1(\mathbf{x}) \equiv \exists \mathbf{x} \Diamond \mathbf{p}_1(\mathbf{x}) \tag{M46}$$
$$\models \quad \Box \forall \mathbf{x}.\mathbf{p}_1(\mathbf{x}) \equiv \forall \mathbf{x} \Box \mathbf{p}_1(\mathbf{x}) \tag{M47}$$

Axioms and theorems that have been demonstrated to hold by preceding proofs or which are cited in this section are introduced during the following deductions. This is justified by the observation that theorem and axiom introduction does not incur any additional proof obligations. Modus Ponens for Strict Bi-conditions (M.P.S.B.) and Modus Ponens for Strict Implications (M.P.S.I.) [102] are used as transformation rules:

$$\frac{\mathbf{w}_1, \Box(\mathbf{w}_1 \Rightarrow \mathbf{w}_2)}{\mathbf{w}_2} \quad (\mathbf{M.P.S.I.})$$

$$\frac{\mathbf{w}_1, \Box(\mathbf{w}_1 \equiv \mathbf{w}_2)}{\mathbf{w}_2} \quad (\mathbf{M.P.S.B.})$$

It should be noted that the development of proof techniques for interval temporal logic is a research area in its own right. The following deductions should, therefore, be regarded as outlines that require the support of further work. In particular, the reader's attention is drawn to weaknesses in the proof steps taken during the derivation of the Implication Laws (Theorems B.4 and B.4). The word 'Theorems' is used here as a term of reference; these laws have yet to be proven. These limitations should not be surprising; Abadi observes that "... all effective proof systems for temporal logic are incomplete for the standard semantics, in the sense that some formulas hold in every intended model but cannot be proved" [1]. Future research might resolve this limitation by extending modal logic subordinate proof systems, such as those proposed by Fitch [102] and used by Hughes and Cresswell [153], to

support natural deduction for interval temporal logics. In particular, this work might explore ways of reasoning about the $\bigcirc$ and $\mathcal{U}$ operators using this technique. Alternatively, Abadi's approach of relating temporal proofs to classical systems through the introduction of time parameters into predicates might be adopted [1]. He notes that further work is required in order to establish that this approach can, indeed, prove all the formulas which are intended to hold in his interval temporal logic. It is hypothesised that this research might provide proof techniques that could be integrated into the verification tools discussed in Section 8.2.2.

B.1 Complement Law

Theorem B.1 $\Box\neg\neg\mathbf{w}_1 \equiv \Box\mathbf{w}_1$

Proof:

1. $\Box\neg\neg\mathbf{w}_1$	Premise
2. $\Box\mathbf{w}_1$	Eliminate, $\neg$, 1

B.2 Idempotent Law

Theorem B.2 $\Box\mathbf{w}_1 \equiv \Box(\mathbf{w}_1 \wedge \mathbf{w}_1)$

Proof:

1. $\Box\mathbf{w}_1$	Premise
2. $\Box\mathbf{w}_1 \wedge \Box\mathbf{w}_1$	Introduce, $\wedge$, 1
3. $\Box(\mathbf{w}_1 \wedge \mathbf{w}_1)$	M.P.S.B., (M16), 2

B.3 Implication Laws

Theorem B.3 $\Box(\mathbf{w}_1 \Rightarrow \mathbf{w}_2) \equiv \Box(\neg\,\mathbf{w}_1 \vee \mathbf{w}_2)$

Note: more work is required to support steps 2 and 7.
Sketch of proof:

1. $\Box(\mathbf{w}_1 \Rightarrow \mathbf{w}_2)$	Premise
2. $\Box\mathbf{w}_1$	Assumption
3. $\Box\mathbf{w}_1 \Rightarrow \Box\mathbf{w}_2$	M.P.S.I., (M27), 1
4. $\Box\mathbf{w}_2$	Eliminate, $\Rightarrow$, 3, 2
5. $\Box\neg\,\mathbf{w}_1 \vee \Box\mathbf{w}_2$	Introduce, $\vee$, 4
6. $\Box(\neg\,\mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.I., (M23), 5
7. $\Box\neg\,\mathbf{w}_1$	Assumption
8. $\Box\neg\,\mathbf{w}_1 \vee \Box\mathbf{w}_2$	Introduce, $\vee$, 7
9. $\Box(\neg\,\mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.I., (M23), 8

Theorem B.4 $\Diamond(\mathbf{w}_1 \Rightarrow \mathbf{w}_2) \equiv \Diamond(\neg\,\mathbf{w}_1 \vee \mathbf{w}_2)$

Note: more work is required to support steps 2, 3 and 6.
Sketch of proof:

1. $\Diamond(\mathbf{w}_1 \Rightarrow \mathbf{w}_2)$	Premise
2. $\Box\mathbf{w}_1$	Assumption
3. $\Diamond\mathbf{w}_2$	Eliminate, $\Rightarrow$, 1, 2
4. $\Diamond\neg\ \mathbf{w}_1 \vee \Diamond\mathbf{w}_2$	Introduce, $\vee$, 3
5. $\Diamond(\neg\ \mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.B., (M17), 4
6. $\Box\neg\ \mathbf{w}_1$	Assumption
7. $\Diamond\neg\ \mathbf{w}_1$	M.P.S.I., (M7), 6
8. $\Diamond\neg\ \mathbf{w}_1 \vee \Diamond\mathbf{w}_2$	Introduce, $\vee$, 7
9. $\Diamond(\neg\ \mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.B., (M17), 8

B.4 De Morgan's Laws

Theorem B.5 $\Diamond\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2) \equiv \Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$

In order to prove this we first prove the following lemma:

Lemma B.1 $\Box\mathbf{w}_1 \equiv \neg\ \Diamond\neg\ \mathbf{w}_1$

Proof:

1. $\Box\mathbf{w}_1$	Premise
2. $\Box\neg\ \neg\ \mathbf{w}_1$	M.P.S.B., (Theorem B.1), 1
3. $\neg\ \Diamond\neg\ \mathbf{w}_1$	M.P.S.I., (M1), 2

Proof by contradiction:

1. $\Diamond\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2)$	Premise
2. $\neg\ \Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	Assumption
3. $\neg\ (\Diamond\neg\ \mathbf{w}_1 \vee \Diamond\neg\ \mathbf{w}_2)$	M.P.S.B., (M17), 2
4. $\Diamond\neg\ \mathbf{w}_1$	Assumption
5. $\Diamond\neg\ \mathbf{w}_1 \vee \Diamond\neg\ \mathbf{w}_2$	Introduce, $\vee$, 4
6. $\Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	M.P.S.B., (M17), 5
7. $\Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2) \wedge \neg\ \Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	Introduce, $\wedge$, 2, 6
8. $\neg\ \Diamond\neg\ \mathbf{w}_1$	Introduce, $\neg$, 4, 7
9. $\Box\mathbf{w}_1$	M.P.S.B., (Lemma B.1), 8
10. $\Diamond\neg\ \mathbf{w}_2$	Assumption
11. $\Diamond\neg\ \mathbf{w}_1 \vee \Diamond\neg\ \mathbf{w}_2$	Introduce, $\vee$, 10
12. $\Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	M.P.S.B., (M17), 11
13. $\Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2) \wedge \neg\ \Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	Introduce, $\wedge$, 2, 12
14. $\neg\ \Diamond\neg\ \mathbf{w}_2$	Introduce, $\neg$, 10, 13
15. $\Box\mathbf{w}_2$	M.P.S.B., (Lemma B.1), 14
16. $\Box\mathbf{w}_1 \wedge \Box\mathbf{w}_2$	Introduce, $\wedge$, 9, 16
17. $\Box(\mathbf{w}_1 \wedge \mathbf{w}_2)$	M.P.S.B, (M16), 16
18. $\neg\ \Diamond\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2)$	M.P.S.B., (Lemma B.1), 17
19. $\neg\ \Diamond\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \Diamond\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2)$	Introduce, $\wedge$, 1, 18
20. $\Diamond(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$	Eliminate, $\neg$, 2, 19

Theorem B.6 $\Box\neg\ (\mathbf{w}_1 \wedge \mathbf{w}_2) \equiv \Box(\neg\ \mathbf{w}_1 \vee \neg\ \mathbf{w}_2)$

Proof by contradiction:

1. $\Box(\neg\, \mathbf{w}_1 \vee \neg\, \mathbf{w}_2)$	Premise
2. $\neg\, \Box\neg\, (\mathbf{w}_1 \wedge \mathbf{w}_2)$	Assumption
3. $\Diamond\neg\, \neg\, (\mathbf{w}_1 \wedge \mathbf{w}_2)$	M.P.S.B., (M2), 2
4. $\Diamond\neg\, (\neg\, \mathbf{w}_1 \vee \neg\, \mathbf{w}_2)$	M.P.S.B., (Theorem B.5), 3
5. $\neg\, \Box(\neg\, \mathbf{w}_1 \vee \neg\, \mathbf{w}_2)$	M.P.S.B., (M2), 4
6. $\Box(\neg\, \mathbf{w}_1 \vee \neg\, \mathbf{w}_2) \wedge \neg\, \Box(\neg\, \mathbf{w}_1 \vee \neg\, \mathbf{w}_2)$	Introduce, $\wedge$, 1, 5
7. $\Box\neg\, (\mathbf{w}_1 \wedge \mathbf{w}_2)$	Eliminate, $\neg$, 2, 6

B.5 Commutative Laws

Theorem B.7 $\Box(\mathbf{w}_1 \wedge \mathbf{w}_2) \equiv \Box(\mathbf{w}_2 \wedge \mathbf{w}_1)$

Proof:

1. $\Box(\mathbf{w}_1 \wedge \mathbf{w}_2)$	Premise
2. $\Box\mathbf{w}_1 \wedge \Box\mathbf{w}_2$	M.P.S.B., (M16), 1
3. $\Box\mathbf{w}_1$	Eliminate, $\wedge$, 2
4. $\Box\mathbf{w}_2$	Eliminate, $\wedge$, 2
5. $\Box\mathbf{w}_2 \wedge \Box\mathbf{w}_1$	Introduce, $\wedge$, 3, 4
6. $\Box(\mathbf{w}_2 \wedge \mathbf{w}_1)$	M.P.S.B., (M16), 5

Theorem B.8 $\Diamond(\mathbf{w}_1 \vee \mathbf{w}_2) \equiv \Diamond(\mathbf{w}_2 \vee \mathbf{w}_1)$

Proof:

1. $\Diamond(\mathbf{w}_1 \vee \mathbf{w}_2)$	Premise
2. $\Diamond\mathbf{w}_1 \vee \Diamond\mathbf{w}_2$	M.P.S.B., (M17), 1
3. $\Diamond\mathbf{w}_2$	Assumption
4. $\Diamond\mathbf{w}_2 \vee \Diamond\mathbf{w}_1$	Introduce, $\vee$, 3
5. $\Diamond(\mathbf{w}_2 \vee \mathbf{w}_1)$	M.P.S.B., M(17), 4
6. $\Diamond\mathbf{w}_1$	Assumption
7. $\Diamond\mathbf{w}_2 \vee \Diamond\mathbf{w}_1$	Introduce, $\vee$, 4
8. $\Diamond(\mathbf{w}_2 \vee \mathbf{w}_1)$	M.P.S.B., (M17), 7
9. $\Diamond(\mathbf{w}_2 \vee \mathbf{w}_1)$	Eliminate, $\vee$, 1, 3-5,6-8

Theorem B.9 $\Box(\mathbf{w}_1 \vee \mathbf{w}_2) \equiv \Box(\mathbf{w}_2 \vee \mathbf{w}_1)$

Proof by contradiction:

1. $\Box(\mathbf{w}_1 \vee \mathbf{w}_2)$	Premise
2. $\neg\, \Box(\mathbf{w}_2 \vee \mathbf{w}_1)$	Assumption
3. $\Diamond\neg\, (\mathbf{w}_2 \vee \mathbf{w}_1)$	M.P.S.B., (M2), 2
4. $\Diamond\neg\, (\mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.B., (Theorem B.8), 3
5. $\neg\, \Box(\mathbf{w}_1 \vee \mathbf{w}_2)$	M.P.S.B., (M2), 4
6. $\neg\, \Box(\mathbf{w}_1 \vee \mathbf{w}_2) \wedge \Box(\mathbf{w}_1 \vee \mathbf{w}_2)$	Introduce, $\wedge$, 1, 5
7. $\Box(\mathbf{w}_2 \vee \mathbf{w}_1)$	Eliminate, $\neg$, 2, 6

Theorem B.10 $\Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2) \equiv \Diamond(\mathbf{w}_2 \wedge \mathbf{w}_1)$

Proof by contradiction:

1. $\Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2)$	Premise
2. $\neg \Diamond(\mathbf{w}_2 \wedge \mathbf{w}_1)$	Assumption
3. $\Box\neg (\mathbf{w}_2 \wedge \mathbf{w}_1)$	M.P.S.B., (M1), 2
4. $\Box(\neg \mathbf{w}_2 \vee \neg \mathbf{w}_1)$	M.P.S.B., (Theorem B.6), 3
5. $\Box(\neg \mathbf{w}_1 \vee \neg \mathbf{w}_2)$	M.P.S.B., (Theorem B.9), 4
6. $\Box\neg (\mathbf{w}_1 \wedge \mathbf{w}_2)$	M.P.S.B., (Theorem B.6), 5
7. $\neg \Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2)$	M.P.S.B., (M1), 6
8. $\neg \Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \Diamond(\mathbf{w}_1 \wedge \mathbf{w}_2)$	Introduce, $\wedge$, 7, 1
9. $\Diamond(\mathbf{w}_2 \wedge \mathbf{w}_1)$	Eliminate, $\neg$, 2, 8

B.6 Associative Laws

Theorem B.11 $\Box((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3) \equiv \Box(\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$

Proof:

1. $\Box((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3)$	Premise
2. $\Box(\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \Box\mathbf{w}_3$	M.P.S.B., (M16), 1
3. $\Box(\mathbf{w}_1 \wedge \mathbf{w}_2)$	Eliminate, $\wedge$, 2
4. $\Box\mathbf{w}_1 \wedge \Box\mathbf{w}_2$	M.P.S.B., (M16), 3
5. $\Box\mathbf{w}_1$	Eliminate, $\wedge$, 16
6. $\Box\mathbf{w}_2$	Eliminate, $\wedge$, 16
7. $\Box\mathbf{w}_3$	Eliminate, $\wedge$, 2
8. $\Box\mathbf{w}_2 \wedge \Box\mathbf{w}_3$	Introduce, $\wedge$, 6, 7
9. $\Box(\mathbf{w}_2 \wedge \mathbf{w}_3)$	M.P.S.B., (M16), 8
10. $\Box\mathbf{w}_1 \wedge \Box(\mathbf{w}_2 \wedge \mathbf{w}_3)$	Introduce, $\wedge$, 5, 7
11. $\Box(\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$	M.P.S.B., (M16), 10

Theorem B.12 $\Diamond((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3) \equiv \Diamond(\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$

Proof by contradiction:

1. $\Diamond((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3)$	Premise
2. $\neg \Diamond(\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$	Assumption
3. $\Box\neg (\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$	M.P.S.B., (M1), 2
4. $\Box\neg ((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3)$	M.P.S.B., (Theorem B.11), 3
5. $\neg \Diamond((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3)$	M.P.S.B., (M1), 4
6. $\neg \Diamond((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3) \wedge \Diamond((\mathbf{w}_1 \wedge \mathbf{w}_2) \wedge \mathbf{w}_3)$	Introduce, $\wedge$, 1, 5
7. $\Diamond(\mathbf{w}_1 \wedge (\mathbf{w}_2 \wedge \mathbf{w}_3))$	Eliminate, $\neg$, 2, 6

Theorem B.13 $\Diamond((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3) \equiv \Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$

Proof:

1. $\Diamond((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3)$	Premise
2. $\Diamond(\mathbf{w}_1 \vee \mathbf{w}_2) \vee \Diamond\mathbf{w}_3$	M.P.S.B., (M17), 1
3. $\Diamond(\mathbf{w}_1 \vee \mathbf{w}_2)$	Assumption
4. $\Diamond\mathbf{w}_1 \vee \Diamond\mathbf{w}_2$	M.P.S.B., (M17), 3

5. $\Diamond \mathbf{w}_1$	Assumption
6. $\Diamond \mathbf{w}_1 \vee \Diamond(\mathbf{w}_2 \vee \mathbf{w}_3)$	Introduce, $\vee$, 5
7. $\Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	M.P.S.B., (M17), 6
8. $\Diamond \mathbf{w}_2$	Assumption
9. $\Diamond \mathbf{w}_2 \vee \Diamond \mathbf{w}_3$	Introduce, $\vee$, 8
10. $\Diamond(\mathbf{w}_2 \vee \mathbf{w}_3)$	M.P.S.B., (M17), 9
11. $\Diamond \mathbf{w}_1 \vee \Diamond(\mathbf{w}_2 \vee \mathbf{w}_3)$	Introduce, $\vee$, 10
12. $\Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	M.P.S.B., (M17), 11
13. $\Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	Eliminate, $\vee$, 2, 3-7,8-13
14. $\Diamond \mathbf{w}_3$	Assumption
15. $\Diamond \mathbf{w}_2 \vee \Diamond \mathbf{w}_3$	Introduce, $\vee$, 14
16. $\Diamond(\mathbf{w}_2 \vee \mathbf{w}_3)$	M.P.S.B., (M17), 15
17. $\Diamond \mathbf{w}_1 \vee \Diamond(\mathbf{w}_2 \vee \mathbf{w}_3)$	Introduce, $\vee$, 16
18. $\Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	M.P.S.B., (M17), 17
19. $\Diamond(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	Eliminate, $\vee$, 1, 3-13, 14-18

Theorem B.14 $\Box((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3) \equiv \Box(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$

Proof by contradiction:

1. $\Box((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3)$	Premise
2. $\neg\, \Box(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	Assumption
3. $\Diamond\neg\, (\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	M.P.S.B., (M2), 2
4. $\Diamond(\neg\, \mathbf{w}_1 \wedge \neg\, (\mathbf{w}_2 \vee \mathbf{w}_3))$	M.P.S.B., (Theorem B.5), 3
5. $\Diamond(\neg\, \mathbf{w}_1 \wedge (\neg\, \mathbf{w}_2 \wedge \neg\, \mathbf{w}_3))$	M.P.S.B., (Theorem B.5), 4
6. $\Diamond((\neg\, \mathbf{w}_1 \wedge \neg\, \mathbf{w}_2) \wedge \neg\, \mathbf{w}_3)$	M.P.S.B., (Theorem B.12), 5
7. $\Diamond(\neg\, (\mathbf{w}_1 \vee \mathbf{w}_2) \wedge \neg\, \mathbf{w}_3)$	M.P.S.B., (Theorem B.5), 6
8. $\Diamond\neg\, ((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3)$	M.P.S.B., (Theorem B.5), 7
9. $\neg\, \Box((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3)$	M.P.S.B., (M2), 8
10. $\neg\, \Box((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3) \wedge \Box((\mathbf{w}_1 \vee \mathbf{w}_2) \vee \mathbf{w}_3)$	Introduce, $\wedge$, 1, 9
11. $\Box(\mathbf{w}_1 \vee (\mathbf{w}_2 \vee \mathbf{w}_3))$	Eliminate, $\neg$, 2, 10

Appendix C

Appendices To Chapter 3

C.1 Re-writing Of no_state_correspondence

$$\forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{ps}, \mathbf{ps}' \in \mathbf{Ps} : \mathbf{no\_state\_correspondence}(\mathbf{s}, \mathbf{ps}, \mathbf{ps}')$$

$\Leftrightarrow$ (3.5)

$$\Box\neg\, (\mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \wedge \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \wedge \neg\, \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}')) \tag{C.1}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \vee \neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \vee \neg\, \neg\, \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}')) \tag{C.2}$$

$\Leftrightarrow$ (Complement law)

$$\Box(\neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \vee \neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \vee \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}')) \tag{C.3}$$

$\Leftrightarrow$ (Implication laws)

$$\Box(\neg\, (\neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \vee \neg\, \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}')) \Rightarrow \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}')) \tag{C.4}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}) \wedge \mathbf{state\_represents}(\mathbf{s}, \mathbf{ps}') \Rightarrow \mathbf{same\_process\_state}(\mathbf{ps}, \mathbf{ps}')) \tag{C.5}$$

C.2 Re-writing Of no_display_correspondence

$$\forall\, \mathbf{d} \in \mathbf{D}, \forall\, \mathbf{s}, \mathbf{s}' \in \mathbf{S} : \mathbf{no\_display\_correspondence}(\mathbf{d}, \mathbf{s}, \mathbf{s}')$$

$\Leftrightarrow$ (3.12)

$$\Box\neg\, (\mathbf{view}(\mathbf{s}, \mathbf{d}) \wedge \mathbf{view}(\mathbf{s}', \mathbf{d}) \wedge \neg\, \mathbf{same\_state}(\mathbf{s}, \mathbf{s}')) \tag{C.6}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\neg\, \mathbf{view}(\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{view}(\mathbf{s}',\mathbf{d}) \vee \neg\, \neg\, \mathbf{same\_state}(\mathbf{s},\mathbf{s}')) \tag{C.7}$$

$\Leftrightarrow$ (Complement law)

$$\Box(\neg\, \mathbf{view}(\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{view}(\mathbf{s}',\mathbf{d}) \vee \mathbf{same\_state}(\mathbf{s},\mathbf{s}')) \tag{C.8}$$

$\Leftrightarrow$ (Implication laws)

$$\Box(\neg\, (\neg\, \mathbf{view}(\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{view}(\mathbf{s}',\mathbf{d})) \Rightarrow \mathbf{same\_state}(\mathbf{s},\mathbf{s}')) \tag{C.9}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\mathbf{view}(\mathbf{s},\mathbf{d}) \wedge \mathbf{view}(\mathbf{s}',\mathbf{d}) \Rightarrow \mathbf{same\_state}(\mathbf{s},\mathbf{s}')) \tag{C.10}$$

C.3 Re-writing Of no_command_view_correspondence

$\forall\, \mathbf{c} \in \mathbf{C}, \forall\, \mathbf{s} \in \mathbf{S}, \forall\, \mathbf{d}, \mathbf{d}' \in \mathbf{D} : \mathbf{no\_command\_view\_correspondence}(\mathbf{c},\mathbf{s},\mathbf{d},\mathbf{d}')$

$\Leftrightarrow$ (3.20)

$$\Box\neg\, (\mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}) \wedge \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}') \wedge \neg\, \mathbf{same\_source}(\mathbf{d},\mathbf{d}')) \tag{C.11}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}') \vee \neg\, \neg\, \mathbf{same\_source}(\mathbf{d},\mathbf{d}')) \tag{C.12}$$

$\Leftrightarrow$ (Complement law)

$$\Box(\neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}') \vee \mathbf{same\_source}(\mathbf{d},\mathbf{d}')) \tag{C.13}$$

$\Leftrightarrow$ (Implication laws)

$$\Box(\neg\, (\neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}) \vee \neg\, \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}')) \Rightarrow \mathbf{same\_source}(\mathbf{d},\mathbf{d}')) \tag{C.14}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}) \wedge \mathbf{command\_view}(\mathbf{c},\mathbf{s},\mathbf{d}') \Rightarrow \mathbf{same\_source}(\mathbf{d},\mathbf{d}')) \tag{C.15}$$

Appendix D

Appendices To Chapter 4

D.1 Re-writing Of no_input_contention

$$\forall\, \mathbf{u}, \mathbf{u}' \in \mathbf{U}, \forall\, \mathbf{p}, \mathbf{p}' \in \mathbf{P} : \mathbf{no\_input\_contention}(\mathbf{u}, \mathbf{u}', \mathbf{p}, \mathbf{p}')$$

$\Leftrightarrow$ (4.5)

$$\Box\neg\, (\mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \wedge \mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')) \tag{D.1}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \vee \neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \vee \neg\, \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')) \tag{D.2}$$

$\Leftrightarrow$ (Complement law)

$$\Box(\neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \vee \neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \vee \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')) \tag{D.3}$$

$\Leftrightarrow$ (Implication laws)

$$\Box(\neg\, \neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \Rightarrow \neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \vee \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')) \tag{D.4}$$

$\Leftrightarrow$ (Complement law)

$$\Box(\mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \Rightarrow \neg\, \mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \vee \mathbf{same\_user}(\mathbf{u}, \mathbf{u}')) \tag{D.5}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Box(\mathbf{user\_input}(\mathbf{u}, \mathbf{p}) \Rightarrow \neg\, (\mathbf{user\_input}(\mathbf{u}, \mathbf{p}') \wedge \neg\, \mathbf{same\_user}(\mathbf{u}, \mathbf{u}'))) \tag{D.6}$$

Appendix E

Appendices To Chapter 5

E.1 First Re-writing Of inconsistent

$$\exists\, \mathbf{o}, \mathbf{o}' \in \mathbf{O}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \mathbf{inconsistent}(\mathbf{o}, \mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od})$$

$\Leftrightarrow$ (5.6)

$$\neg\, \Box(\mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{object}(\mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.1)}$$

$\Leftrightarrow$ (M2)

$$\Diamond\neg\, (\mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{object}(\mathbf{o}', \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.2)}$$

$\Leftrightarrow$ ($\mathbf{object}$(5.5))

$$\Diamond\neg\, ((\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od})) \wedge (\mathbf{object\_methods}(\mathbf{o}', \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}', \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}', \mathbf{od})) \wedge \neg\, \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.3)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\neg\, (\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od})) \vee \neg\, (\mathbf{object\_methods}(\mathbf{o}', \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}', \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}', \mathbf{od})) \vee \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.4)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond((\neg\, \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}, \mathbf{od})) \vee (\neg\, \mathbf{object\_methods}(\mathbf{o}', \mathbf{om}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}', \mathbf{os}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}', \mathbf{od})) \vee \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.5)}$$

$\Leftrightarrow$ (Associative laws)

$$\Diamond(\neg\, \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \vee \neg\, \mathbf{object\_methods}(\mathbf{o}', \mathbf{om}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}', \mathbf{os}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}', \mathbf{od}) \vee \mathbf{same\_object}(\mathbf{o}, \mathbf{o}')) \quad \text{(E.6)}$$

$\Leftrightarrow$ (Commutative laws)

$$\Diamond(\neg\,\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\,\mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \vee \neg\,\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\,\mathbf{object\_state}(\mathbf{o'},\mathbf{os}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o'}) \vee \neg\,\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\,\mathbf{object\_view}(\mathbf{o'},\mathbf{od})) \quad \text{(E.7)}$$

$\Leftrightarrow$ (Implication laws)

$$\Diamond(\neg\,(\neg\,\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\,\mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \vee \neg\,\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\,\mathbf{object\_state}(\mathbf{o'},\mathbf{os}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o'})) \Rightarrow (\neg\,\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\,\mathbf{object\_view}(\mathbf{o'},\mathbf{od}))) \quad \text{(E.8)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o'},\mathbf{os}) \wedge \neg\,\mathbf{same\_object}(\mathbf{o},\mathbf{o'}) \Rightarrow \neg\,\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\,\mathbf{object\_view}(\mathbf{o'},\mathbf{od})) \quad \text{(E.9)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o'},\mathbf{os}) \wedge \neg\,\mathbf{same\_object}(\mathbf{o},\mathbf{o'}) \Rightarrow \neg\,(\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o'},\mathbf{od}))) \quad \text{(E.10)}$$

E.2 Second Re-writing Of inconsistent

This re-writing proceeds in identical manner to the first re-writing of **inconsistent** until step (E.6).
$\Leftrightarrow$ (Commutative laws)

$$\Diamond(\neg\,\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\,\mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \vee \neg\,\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\,\mathbf{object\_view}(\mathbf{o'},\mathbf{od}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o'}) \vee \neg\,\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\,\mathbf{object\_state}(\mathbf{o'},\mathbf{os})) \quad \text{(E.11)}$$

$\Leftrightarrow$ (Implication laws)

$$\Diamond(\neg\,(\neg\,\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\,\mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \vee \neg\,\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\,\mathbf{object\_view}(\mathbf{o'},\mathbf{od}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o'})) \Rightarrow (\neg\,\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\,\mathbf{object\_state}(\mathbf{o'},\mathbf{os}))) \quad \text{(E.12)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o'},\mathbf{om}) \wedge \mathbf{object\_view}(\mathbf{o},\mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o'},\mathbf{od}) \wedge \neg\,\mathbf{same\_object}(\mathbf{o},\mathbf{o'}) \Rightarrow \neg\,\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\,\mathbf{object\_state}(\mathbf{o'},\mathbf{os})) \quad \text{(E.13)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o}',\mathbf{om}) \wedge \mathbf{object\_view}(\mathbf{o},\mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o}',\mathbf{od}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\, (\mathbf{object\_state}(\mathbf{o},\mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o}',\mathbf{os}))) \quad \text{(E.14)}$$

$\Leftrightarrow$ (**same_object_state**(5.8), **same_object_view**(5.9), **same_object_methods**(5.10))

$$\Diamond(\mathbf{same\_object\_methods}(\mathbf{o},\mathbf{o}',\mathbf{om}) \wedge \mathbf{same\_object\_view}(\mathbf{o},\mathbf{o}',\mathbf{od}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\, \mathbf{same\_object\_state}(\mathbf{o},\mathbf{o}',\mathbf{os})) \quad \text{(E.15)}$$

E.3 Third Re-writing Of inconsistent

This re-writing proceeds in identical manner to the first re-writing of **inconsistent** until step (E.6).
$\Leftrightarrow$ (Commutative laws)

$$\Diamond(\neg\, \mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}',\mathbf{od}) \vee \neg\, \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}',\mathbf{os}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \vee \neg\, \mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\, \mathbf{object\_methods}(\mathbf{o}',\mathbf{om})) \quad \text{(E.16)}$$

$\Leftrightarrow$ (Implication laws)

$$\Diamond(\neg\, (\neg\, \mathbf{object\_view}(\mathbf{o},\mathbf{od}) \vee \neg\, \mathbf{object\_view}(\mathbf{o}',\mathbf{od}) \vee \neg\, \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \vee \neg\, \mathbf{object\_state}(\mathbf{o}',\mathbf{os}) \vee \mathbf{same\_object}(\mathbf{o},\mathbf{o}')) \Rightarrow (\neg\, \mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\, \mathbf{object\_methods}(\mathbf{o}',\mathbf{om}))) \quad \text{(E.17)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o}',\mathbf{od}) \wedge \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o}',\mathbf{os}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\, \mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \vee \neg\, \mathbf{object\_methods}(\mathbf{o}',\mathbf{om})) \quad \text{(E.18)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{object\_view}(\mathbf{o},\mathbf{od}) \wedge \mathbf{object\_view}(\mathbf{o}',\mathbf{od}) \wedge \mathbf{object\_state}(\mathbf{o},\mathbf{os}) \wedge \mathbf{object\_state}(\mathbf{o}',\mathbf{os}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\, (\mathbf{object\_methods}(\mathbf{o},\mathbf{om}) \wedge \mathbf{object\_methods}(\mathbf{o}',\mathbf{om}))) \quad \text{(E.19)}$$

$\Leftrightarrow$ (**same_object_state**(5.8), **same_object_view**(5.9), **same_object_methods**(5.10))

$$\Diamond(\mathbf{same\_object\_view}(\mathbf{o},\mathbf{o}',\mathbf{od}) \wedge \mathbf{same\_object\_state}(\mathbf{o},\mathbf{o}',\mathbf{os}) \wedge \neg\, \mathbf{same\_object}(\mathbf{o},\mathbf{o}') \Rightarrow \neg\, \mathbf{same\_object\_methods}(\mathbf{o},\mathbf{o}',\mathbf{om})) \quad \text{(E.20)}$$

E.4 Re-writing Of consistent

$$\forall \mathbf{o} \in \mathbf{O}, \exists \mathbf{cl} \in \mathbf{Cl}, \exists \mathbf{om} \in \mathbf{Om}, \exists \mathbf{os} \in \mathbf{Os}, \exists \mathbf{od} \in \mathbf{Od} : \mathbf{consistent}(\mathbf{o}, \mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od})$$

$\Leftrightarrow$ (5.31)

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{class\_type}(\mathbf{o}, \mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od})) \tag{E.21}$$

$\Leftrightarrow$ (**class_type** (5.29))

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{class\_defined}(\mathbf{cl}, \mathbf{om}, \mathbf{os}, \mathbf{od})) \tag{E.22}$$

$\Leftrightarrow$ (**class_defined** (5.28))

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \tag{E.23}$$

$\Leftrightarrow$ (**object** (5.5))

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \tag{E.24}$$

$\Leftrightarrow$ (Commutative laws)

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od})) \tag{E.25}$$

$\Leftrightarrow$ (Associative laws)

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge (\mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \wedge (\mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os})) \wedge (\mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}))) \tag{E.26}$$

$\Leftrightarrow$ (Idempotent law)

$$\Box((\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \wedge (\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os})) \wedge (\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od}))) \tag{E.27}$$

$\Leftrightarrow$ (M16)

$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \wedge \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os})) \wedge \Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od})) \tag{E.28}$$

From this we identify three conjunctions:

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{od} \in \mathbf{Od} :$$
$$\mathbf{instantiate\_image\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{od}) \Leftrightarrow$$
$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{legal\_view}(\mathbf{cl}, \mathbf{od})) \quad \text{(E.29)}$$

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{os} \in \mathbf{Os} :$$
$$\mathbf{instantiate\_state\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{os}) \Leftrightarrow$$
$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{legal\_state}(\mathbf{cl}, \mathbf{os})) \quad \text{(E.30)}$$

$$\forall\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{cl} \in \mathbf{Cl}, \exists\, \mathbf{om} \in \mathbf{Om} :$$
$$\mathbf{instantiate\_method\_consistency}(\mathbf{o}, \mathbf{cl}, \mathbf{om}) \Leftrightarrow$$
$$\Box(\mathbf{is\_a}(\mathbf{o}, \mathbf{cl}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{legal\_methods}(\mathbf{cl}, \mathbf{om})) \quad \text{(E.31)}$$

Appendix F

Appendices To Chapter 6

F.1 First Re-writing Of break_down

$$\exists\,\mathbf{o} \in \mathbf{O}, \exists\,\mathbf{pc} \in \mathbf{Pc}, \forall\,\mathbf{om} \in \mathbf{Om}, \forall\,\mathbf{os} \in \mathbf{Os}, \forall\,\mathbf{od} \in \mathbf{Od} : \mathbf{break\_down}(\mathbf{o}, \mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od})$$

$\Leftrightarrow$ (6.6)

$$\neg\,\Box(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od})) \quad \text{(F.1)}$$

$\Leftrightarrow$ (M2)

$$\Diamond\neg\,(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object}(\mathbf{o}, \mathbf{om}, \mathbf{os}, \mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od})) \quad \text{(F.2)}$$

$\Leftrightarrow$ (**object** (5.5))

$$\Diamond\neg\,(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc}, \mathbf{om}, \mathbf{os}, \mathbf{od})) \quad \text{(F.3)}$$

$\Leftrightarrow$ (**conforms** (6.5))

$$\Diamond\neg\,(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{method\_conforms}(\mathbf{pc}, \mathbf{om}) \wedge \mathbf{state\_conforms}(\mathbf{pc}, \mathbf{os}) \wedge \mathbf{view\_conforms}(\mathbf{pc}, \mathbf{od})) \quad \text{(F.4)}$$

$\Leftrightarrow$ (Commutative laws)

$$\Diamond\neg\,(\mathbf{represents}(\mathbf{o}, \mathbf{pc}) \wedge \mathbf{object\_methods}(\mathbf{o}, \mathbf{om}) \wedge \mathbf{method\_conforms}(\mathbf{pc}, \mathbf{om}) \wedge \mathbf{object\_state}(\mathbf{o}, \mathbf{os}) \wedge \mathbf{state\_conforms}(\mathbf{pc}, \mathbf{os}) \wedge \mathbf{object\_view}(\mathbf{o}, \mathbf{od}) \wedge \mathbf{view\_conforms}(\mathbf{pc}, \mathbf{od})) \quad \text{(F.5)}$$

$\Leftrightarrow$ (Associative laws)

$$\begin{aligned}\Diamond\neg\,(&\mathbf{represents(o,pc)}\wedge\\&(\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)})\wedge\\&(\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)})\wedge\\&(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)}))\end{aligned}\tag{F.6}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\begin{aligned}\Diamond(&\neg\,\mathbf{represents(o,pc)}\vee\\&\neg\,(\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)})\vee\\&\neg\,(\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)})\vee\\&\neg\,(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)}))\end{aligned}\tag{F.7}$$

$\Leftrightarrow$ (Implication laws)

$$\begin{aligned}\Diamond(&\neg\,(\neg\,(\mathbf{represents(o,pc)}\vee\\&\neg\,(\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)})\vee\\&\neg\,(\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)}))\Rightarrow\\&\neg\,(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)}))\end{aligned}\tag{F.8}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\begin{aligned}\Diamond(&\mathbf{represents(o,pc)}\wedge\\&\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)}\wedge\\&\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)}\Rightarrow\\&\neg\,(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)}))\end{aligned}\tag{F.9}$$

F.2 Second Re-writing Of break_down

This re-writing is identical to the previous until step (F.7).
$\Leftrightarrow$ (Commutative laws)

$$\begin{aligned}\Diamond(&\neg\,\mathbf{represents(o,pc)}\vee\\&\neg\,(\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)})\vee\\&\neg\,(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)})\vee\\&\neg\,(\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)}))\end{aligned}\tag{F.10}$$

$\Leftrightarrow$ (Implication laws)

$$\begin{aligned}\Diamond(&\neg\,(\neg\,\mathbf{represents(o,pc)}\vee\\&\neg\,(\mathbf{object\_state(o,os)}\wedge\mathbf{state\_conforms(pc,os)})\vee\\&\neg\,(\mathbf{object\_view(o,od)}\wedge\mathbf{view\_conforms(pc,od)}))\Rightarrow\\&\neg\,(\mathbf{object\_methods(o,om)}\wedge\mathbf{method\_conforms(pc,om)}))\end{aligned}\tag{F.11}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{represents(o, pc)} \wedge (\mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)}) \wedge (\mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)}) \Rightarrow \neg\,(\mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)})) \quad \text{(F.12)}$$

$\Leftrightarrow$ (Associative laws)

$$\Diamond(\mathbf{represents(o, pc)} \wedge \mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)} \wedge \mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)} \Rightarrow \neg\,(\mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)})) \quad \text{(F.13)}$$

$\Leftrightarrow$ (**accurate_methods** (6.8), **accurate_state** (6.9) and **accurate_image** (6.10))

$$\Diamond(\mathbf{represents(o, pc)} \wedge \mathbf{accurate\_state(o, pc, os)} \wedge \mathbf{accurate\_image(o, pc, od)} \Rightarrow \neg\,\mathbf{accurate\_methods(o, pc, om)}) \quad \text{(F.14)}$$

F.3 Third Re-writing Of break_down

This re-writing is identical to the first re-writing of **break_down** until step (F.7).
$\Leftrightarrow$ (Commutative laws)

$$\Diamond(\neg\,\mathbf{represents(o, pc)} \vee \neg\,(\mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)}) \vee \neg\,(\mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)}) \vee \neg\,(\mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)})) \quad \text{(F.15)}$$

$\Leftrightarrow$ (Implication laws)

$$\Diamond(\neg\,(\neg\,\mathbf{represents(o, pc)} \vee \neg\,(\mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)}) \vee \neg\,(\mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)})) \Rightarrow \neg\,(\mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)})) \quad \text{(F.16)}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{represents(o, pc)} \wedge (\mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)}) \wedge (\mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)}) \Rightarrow \neg\,(\mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)})) \quad \text{(F.17)}$$

$\Leftrightarrow$ (Associative laws)

$$\Diamond(\mathbf{represents(o, pc)} \wedge \mathbf{object\_view(o, od)} \wedge \mathbf{view\_conforms(pc, od)} \wedge \mathbf{object\_methods(o, om)} \wedge \mathbf{method\_conforms(pc, om)} \Rightarrow \neg\,(\mathbf{object\_state(o, os)} \wedge \mathbf{state\_conforms(pc, os)})) \quad \text{(F.18)}$$

$\Leftrightarrow$ (**accurate_methods** (6.8), **accurate_state** (6.9) and **accurate_image** (6.10))

$$\Diamond(\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{accurate\_image}(\mathbf{o},\mathbf{pc},\mathbf{od}) \wedge \mathbf{accurate\_methods}(\mathbf{o},\mathbf{pc},\mathbf{om}) \Rightarrow \neg\, \mathbf{accurate\_state}(\mathbf{o},\mathbf{pc},\mathbf{os})) \tag{F.19}$$

F.4 Fourth Re-writing Of break_down

$$\exists\, \mathbf{o} \in \mathbf{O}, \exists\, \mathbf{pc} \in \mathbf{Pc}, \forall\, \mathbf{om} \in \mathbf{Om}, \forall\, \mathbf{os} \in \mathbf{Os}, \forall\, \mathbf{od} \in \mathbf{Od} : \mathbf{break\_down}(\mathbf{o},\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})$$

$\Leftrightarrow$ (6.6)

$$\neg\, \Box(\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})) \tag{F.20}$$

$\Leftrightarrow$ (M2)

$$\Diamond\neg\, (\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od}) \wedge \mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})) \tag{F.21}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\neg\, \mathbf{represents}(\mathbf{o},\mathbf{pc}) \vee \neg\, \mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od}) \vee \neg\, \mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})) \tag{F.22}$$

$\Leftrightarrow$ (Implication laws)

$$\Diamond(\neg\, (\neg\, \mathbf{represents}(\mathbf{o},\mathbf{pc}) \vee \neg\, \mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od})) \Rightarrow \neg\, \mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})) \tag{F.23}$$

$\Leftrightarrow$ (De Morgan's laws)

$$\Diamond(\mathbf{represents}(\mathbf{o},\mathbf{pc}) \wedge \mathbf{object}(\mathbf{o},\mathbf{om},\mathbf{os},\mathbf{od}) \Rightarrow \neg\, \mathbf{conforms}(\mathbf{pc},\mathbf{om},\mathbf{os},\mathbf{od})) \tag{F.24}$$

Appendix G

The Design And Implementation Of PRELOG

In the submitted thesis, this appendix contained a re-formated version of C.W. Johnson and M.D. Harrison, *PRELOG - A System For Presenting And Rendering Logic Specifications Of Interactive Systems*. In C.E. Vandoni and D.A. Duce (eds.) *Eurographics'90*, pp. 469-480, Elsevier Science, North Holland, 1990. This provided more detail about the design and implementation of PRELOG than was provided in Chapter 7. The paper is omitted here in order to avoid violating a copyright agreement with the original publishers.

Appendix H

Temporal Logic And Multi-User Systems

In the submitted thesis, this appendix contained a re-formated version of C.W. Johnson, *Applying Temporal Logic To Support The Design And Prototyping Of Concurrent Multi-User Interfaces*. In D. Diaper and N. Hammond (eds.) *People And Computers VI: Usability Now*, pp. 145-156, Cambridge University Press, Cambridge, United Kingdom, 1991. This provided more detail about the application of PRELOG to support the prototyping of open systems than was provided in Chapters 4 and 7. It is omitted here in order to avoid violating a copyright agreement with the original publishers.

Bibliography

[1] M. Abadi. The power of temporal proofs. *Theoretical Computer Science*, 65(1):35–83, 1989.

[2] H. Abelson. A beginner's guide to LOGO. *BYTE*, 7(8):88–112, 1982.

[3] G. Abowd. Agents : Communicating interactive processes. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 142–148. Elsevier Science Publications, North Holland, Netherlands, 1990.

[4] Agiecut 100D, 200D and 300D machine tools sales brochure. Losone, Switzerland, 1991.

[5] L. Ainsworth. Comparison of U.S. and U.K. practices and philosophies for nuclear plant control room design. In R.E. Edwards, editor, *Proceedings Of The 26th Annual Meeting Of The Human Factors Society*, pages 659–663. Human Factors Society, Santa Monica, United States of America, 1982.

[6] H. Alexander. Specifying and prototyping human-computer interaction. In D. Barnes and P. Brown, editors, *Proceedings Of Software Engineering'86*, pages 336–351. Peter Peregrinus, London, United Kingdom, 1986.

[7] H. Alexander. Structuring dialogues using CSP. In M.D. Harrison and H.W. Thimbleby, editors, *Formal Methods In Human-Computer Interaction*, pages 273–295. Cambridge University Press, Cambridge, United Kingdom, 1990.

[8] D.M. Allen. Investigation of display issues relevant to the presentation of aircraft fault information. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 61–65. Human Factors Society, Santa Monica, United States of America, 1989.

[9] F. Allen. Description of the Chernobyl accident. In N. Worley and J. Lewins, editors, *The Chernobyl Accident And Its Implications For The United Kingdom - Report Number 19 Of The Watt Committee On Energy*, pages 19–24. Elsevier Applied Science, London, United Kingdom, 1988.

[10] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications Of The ACM*, 26(11):832–843, 1983.

[11] J.F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–153, 1984.

[12] R.E. Allen. *The Oxford Concise Dictionary.* The Clarendon Press, Oxford, United Kingdom, 1990.

[13] J.L. Alty. Working within limitations: Computer aided instruction and expert systems. Technical Report AMU8606/01S, Scottish HCI Centre, Strathclyde University, Glasgow, United Kingdom, 1986.

[14] A. Amendola, U. Bersini, P.C. Cacciabue, and G. Mancini. Modelling operators in accident conditions: Advances and perspectives on a cognitive model. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Cognitive Engineering In Complex Dynamic Worlds*, pages 145 – 158. Academic Press, London, United Kingdom, 1988.

[15] S. Anderson and H.W. Thimbleby. Outline of a temporal logic framework for computer supported collaborative work. Technical report, Computer Science Departments of Stirling and Edinburgh Universities, Stirling and Edinburgh, United Kingdom, 1990.

[16] T. Aoyagi, M. Fujita, and T. Moto-Oka. Temporal logic programming language -Tokio- programming in Tokio. In E. Wada, editor, *Proceedings of The 4th Annual Conference - Logic Programming '85*, LNCS 221, pages 128–137. Springer-Verlag, Berlin, FDR, 1986.

[17] J. Arlidge. Hospital admits error in treating cancer patients. *The Independent*, page 3, 7 February 1992.

[18] Association For Computing Machinery. *Proceedings Of The Annual Conference Of The Association For Computing Machinery*, New York, United States of America, 1981. ACM Press.

[19] F. Bacchus. *Representing And Reasoning With Probabilistic Knowledge.* The MIT Press, Cambridge, United States of America, 1990.

[20] L. Bainbridge. Analysis of verbal protocols from a process control task. In E. Edwards and F.P. Lees, editors, *The Human Operator In Process Control*, pages 146–159. Taylor And Francis, London, United Kingdom, 1974.

[21] L. Bainbridge. Mathematical equations or processing routines. In J. Rasmussen and W. Rouse, editors, *Human Detection And Diagnosis Of System Failures*, pages 259–286. Plenum Press, New York, United States Of America, 1981.

[22] L. Bainbridge. Ironies of automation. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology And Human Error*, pages 271 – 283. J. Wiley and Sons, New York, United States of America, 1987.

[23] L. Bainbridge. Types of representation. In L.P. Goodstein, H.B. Anderson, and S.E. Olsen, editors, *Task, Errors And Mental Models*, pages 70 – 91. Taylor and Francis, London, United Kingdom, 1988.

[24] L. Bainbridge. Multiplexed VDT display systems. In G.R.S. Weir and J.L. Alty, editors, *Human Computer Interaction And Complex Systems*, pages 189–221. Academic Press, London, United Kingdom, 1991.

[25] P.J. Barnard, N.V. Hammond, J. Morton, and J.B. Long. Consistency and compatibility in human-computer dialogues. *International Journal of Man-Machine Studies*, 15(1):87–134, 1981.

[26] J.G.P. Barnes. *Programming In Ada*. Addison Wesley, Wokingham, United Kingdom, 1989.

[27] A.C. Barrell. Developments in the control of major hazards. In J. Burgoyne, editor, *The Assessment And Control Of Major Hazards*, pages 1–12. Pergamon Press and The Institution Of Chemical Engineers, Oxford, United Kingdom, 1985.

[28] H. Barringer, M. Fisher, D. M. Gabbay, G. Gough, and R. Owens. MetaM : A framework for programming in temporal logic. Technical report, Department of Computer Science, University of Manchester and Imperial College Of Science and Technology, Manchester and London, United Kingdom, 1989.

[29] V. Bignell and J. Fortune. *Understanding System Failure*. Manchester University Press, Manchester, United Kingdom, 1991.

[30] W.R. Van Biljon. Extending Petri nets for specifying man-machine dialogues. *International Journal of Man-Machine Studies*, 28(4):437–455, 1988.

[31] J. Bindon. Reactor operation and operator training in the United Kingdom. In N. Worley and J. Lewins, editors, *The Chernobyl Accident And Its Implications For The United Kingdom - Report Number 19 Of The Watt Committee On Energy*, pages 61–70. Elsevier Applied Science, London, United Kingdom, 1988.

[32] D. Black. The human element at the core of the disaster. *The Independent*, page 15, 20 March 1989.

[33] D. Black. British Rails' safety record compares well with rest of World. *The Independent*, page 3, 9th January 1991.

[34] D. Black. Signaling failure blamed for Severn Tunnel crash. *The Independent on Sunday*, page 2, 8th December 1991.

[35] E.H. Blake and S. Cook. On including part-hierarchies in object-oriented languages. In J. Bezivin, J.-M. Hullot, P. Cointe, and H. Lieberman, editors, *ECOOP '87: European Conference On Object Oriented Programming*, LNCS 276, pages 41–50. Springer-Verlag, Berlin, FDR, 1987.

[36] J. Borer. *Instrumentation And Control For The Process Industries*. Elsevier Applied Science, London, United Kingdom, 1985.

[37] E. Börger and H. Kleine Büning and M.M. Richter, editors. *Workshop On Computer Science Logic - 1988 Proceedings*. LNCS 385. Springer-Verlag, Berlin, FDR, 1988.

[38] A. Borning and R. Duisberg. Constraint-based tools for building user interfaces. *ACM Transactions on Graphics*, 5(4):345–374, 1986.

[39] K.A. Bowen and R.A. Kowalski. Amalgamating language and metalanguage in logic programming. In K.L. Clark and S.A. Tarnlund, editors, *Logic Programming*, pages 153–172. Academic Press, London, United Kingdom, 1982.

[40] G.A. Boy. Operator assistant systems. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Cognitive Engineering In Complex Dynamic Worlds*, pages 85 – 98. Academic Press, London, United Kingdom, 1988.

[41] R. Braune and C.D. Wickens. The functional age profile: An objective decision criterion for the assessment of pilot performance capacities and capabilities. *Human Factors*, 27(6):681–693, 1985.

[42] British Standard's Institute. *Glossary Of Terms Used In Quality Assurance*, BS 4778, London, United Kingdom, 1979.

[43] D.E. Broadbent. *Decision and Stress*. Academic Press, London, United Kingdom, 1971.

[44] F.P. Brooks. *The Mythical Man-Month: Essays On Software Engineering*. Addison Wesley, Reading, United States of America, 1982.

[45] A.A.F. Brouwers and F.D. Pots. Design process and operator tasks during automation of a sugar factory. In H.-J. Bullinger and B. Shackel, editors, *Human-Computer Interaction—INTERACT'87*, pages 443–452. Elsevier Science Publications, North Holland, Netherlands, 1987.

[46] L. Brownston, R. Farrell, E. Kant, and N. Martin. *Programming Expert Systems In OPS5*. Addison Wesley, Reading, United States of America, 1985.

[47] D.A. Buchanan and J. Bessant. Failure, uncertainty and control : The role of the operator in a computer integrated production system. *Journal Of Management Studies*, 22(3):292 – 308, 1985.

[48] J.P. Bulger, S.G. Hill, and R.E. Christ. Operator workload in the army material acquisition process. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1054–1058. Human Factors Society, Santa Monica, United States of America, 1989.

[49] J. Burgoyne, editor. *The Assessment And Control Of Major Hazards*. Pergamon Press, Oxford, United Kingdom, 1985.

[50] A. Burns. The HCI component of dependable real-time systems. *The Software Engineering Journal*, 6(4):168–174, July 1991.

[51] C.T. Burton, S.J. Cook, S. Gikas, J.R. Rowson, and S.T. Sommerville. Specifying the Apple Macintosh Toolbox Event Manager. *Formal Aspects of Computing*, 1:147–171, 1989.

[52] S.K. Card, J.D. Mackinlay, and G.G. Robertson. The design space of input devices. In J.C. Chew and J. Whiteside, editors, *Proceedings Of The CHI'90 Conference On Human Factors In Computing Systems*, pages 117–124. ACM, New York, United States Of America, 1990.

[53] W.L. Carel. Visual factors in the contact analogue. Technical Report R61ELC60, The General Electric Advanced Electronics Center, Ithica, United States of America, 1961.

[54] P.W. Caro. Flight training and simulation. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 229–261. Academic Press, London, United Kingdom, 1988.

[55] J.M. Carroll. Learning, using and designing file-names and command paradigms. *Behaviour And Information Technology*, 1(4):327–346, 1982.

[56] J.M. Carroll and R.L. Campbell. Artifacts as psychological theories: The case of human computer interaction. *Behaviour And Information Technology*, 8(4):247–236, 1989.

[57] J.M. Carroll and W.A. Kellogg. Artifact as theory nexus : Hermeneutics meets theory based design. In K. Bice and C. Lewis, editors, *Proceedings Of The CHI'89 Conference On Human Factors In Computing Systems*, pages 7–14. ACM, New York, United States of America, 1989.

[58] J.M. Carroll and J.C. Thomas. Metaphor and the cognitive representation of computing systems. *IEEE Transactions On Systems, Man And Cybernetics*, SMC-12(2):107–115, 1982.

[59] C.M. Carswell and C.D. Wickens. Information integration and the object display: An interaction of task demands and display superiority. *Ergonomics*, 30(3):511–527, 1987.

[60] D.M. Cattrall. *The Relational Paradigm*. PhD thesis, Functional Programming Group, Department Of Computer Science, University of York, York, United Kingdom, 1992 (forthcoming).

[61] O. Causse. Superviseur de robot mobile pour la planification et l'exécution réactive de mission. Technical report, DEA Informatique, Institut National Polytechnique de Grenoble, Grenoble, France, 1989.

[62] E.H.P. Chan. Using neural networks to interpret multiple alarms. *IEEE Computer Applications In Power*, 3(2):33–37, 1990.

[63] V. Chaudhary. US warplane blunder kills 9 British troops. *The Guardian*, pages 1–3, 28 February 1991.

[64] P. Checkland. *Systems Thinking, Systems Practice*. J. Wiley And Sons, New York, United States Of America, 1981.

[65] R.S. Chin and S.T. Chanson. Distributed object-based programming systems. *ACM Computing Surveys*, 23(1):91–124, 1991.

[66] R.T. Chin and C.R. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys*, 18(1):67–108, 1986.

[67] K.Y. Choi, J.O. Yang, and S.H. Cheung. The manipulation of time-varying dynamic variables using the rule modification method and performance index in Nuclear Power Plant accident diagnostic expert systems. *IEEE Transactions On Nuclear Science*, 35(5):1121–1125, 1988.

[68] P. Chretienne. Timed Petri nets: A solution to the minimum-time-reachability problem between two states of a timed-event graph. *Journal of Systems and Software*, 6(1-2):95–101, 1986.

[69] R.E. Christ and G.M. Corso. The effects of extended practice on the evaluation of visual display codes. *Human Factors*, 25:71–84, 1983.

[70] K.L. Clark. The synthesis and verification of logic programs. Technical Report 81/36, Department of Computing, Imperial College Of Science And Technology, London, United Kingdom, 1981.

[71] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronisation skeletons using branching time temporal logic. In D. Kozen, editor, *Logic of Programs 1981 - Proceedings*, LNCS 131, pages 52–71. Springer-Verlag, Berlin, FDR, 1982.

[72] J.G. Cleary. A distributed graphics system implemented in PROLOG. Technical Report 84 173 31, Department of Computer Science, University Of Calgary, Calgary, Canada, 1984.

[73] C. Clegg and T.D. Wall. Managing factory automation. In F. Blackler and D. Obourne, editors, *Information Technology And People*, pages 45 – 64. British Psychological Society, Leicester, United Kingdom, 1987.

[74] G. Cockton. Human factors and structured software development: The importance of software structure. In D. Diaper and N. Hammond, editors, *People And Computers VI: Proceedings Of HCI'91*, pages 56–72. Cambridge University Press, Cambridge, United Kingdom, 1991.

[75] A. Colmerauer. Etude et Realisation d'un Système PROLOG. Technical report, Groupe Intelligence Artificielle, Université Aix-Marseille II, Marseille, France, 1973.

[76] Commission Of The European Communities. *Safety Principles For Light Water Reactor Nuclear Power Plants*, Brussels, Belgium, 1981.

[77] S.J. Cook, S. Gikas, W. Roberts, J.R. Rowson, and S.T. Sommerville. Formal aspects of interactive dialogues. Technical Report Alvey Project HI059, Final Report, Queen Mary and Westfield College, London, United Kingdom, March 1989.

[78] R.R. Covey, G.J. Mascetti, W.U. Roessler, and R. Bowles. Operational energy conservation strategies. Technical report, The Institute Of Electrical And Electronic Engineers, Fort Lauderdale, United States of America, 1979.

[79] A.P. Cox, editor. *Risk Analysis In The Process Industries: The Report Of The International Study Group On Risk Analysis.* EFCE No. 45. The Institution Of Chemical Engineers, Rugby, United Kingdom, 1985.

[80] D. Craigen. Strengths and weaknesses of program verification systems. In H.K. Nichols and D. Simpson, editors, *Proceedings Of The First European Software Engineering Conference - ESEC'87*, LNCS 289, pages 396 – 404. Springer-Verlag, Berlin, FDR, 1987.

[81] Cullen. *Proceedings Of The Public Enquiry Into The Piper Alpha Disaster*. The Department of Energy, London, United Kingdom, 1990.

[82] A. Van Daele. Dynamic decision making of control room operators in continuous processes: Some field study results. In E.D. Megaw, editor, *Contemporary Ergonomics*. Taylor and Francis, London, United Kingdom, 1989.

[83] R. Davis. Meta-rules: Reasoning about control. *Artificial Intelligence*, 15(3):179–222, 1980.

[84] J. Davison, J. Cassidy, and M. Hosenball. The right stuff. *The Sunday Times*, page A 17, 23 July 1989.

[85] F. Decortis, V. de Keyser, P.C. Cacciabue, and G. Volta. The temporal dimension of man-machine interaction. In G.R.S. Weir and J.L. Alty, editors, *Human Computer Interaction And Complex Systems*, pages 51–72. Academic Press, London, United Kingdom, 1991.

[86] D. Diaper. Analysing focussed interview data with task analysis for knowledge descriptions (TAKD). In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 277–282. Elsevier Science Publications, North Holland, Netherlands, 1990.

[87] A.J. Dix. *Formal Methods For Interactive Systems*. Academic Press, London, United Kingdom, 1991.

[88] A.J. Dix and C. Runciman. Abstract models of interactive systems. In P. Johnson and S. Cook, editors, *People and Computers : Designing The Interface*, pages 13 – 22. Cambridge University Press, Cambridge, United Kingdom, 1985.

[89] M. Donner and F. Jahanian. RTL meets ORE. Technical report, IBM T.J. Watson Research Centre, Yorktown Heights, United States Of America, 1988.

[90] P.A. Doyle, C.D. Gaddy, D.C. Burgy, and D.A. Topmiller. An investigation of communication problems in nuclear power plants. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 18–22. Human Factors Society, Santa Monica, United States of America, 1981.

[91] K.D. Duncan. Fault diagnosis training for advanced continuous process installations. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology And Human Error*, pages 209 – 221. J. Wiley and Sons, New York, United States of America, 1987.

[92] K.D. Duncan and N. Praetorius. Flow displays representing complex plant for diagnosis and process control. In *Proceedings Of the Second European Meeting*

On Cognitive Approaches To Process Control, pages 259–267, Sienna, Italy, 1989. CEC-JRC Ispra and the University of Sienna.

[93] E.A. Edmonds. The man-computer interface: A note on concepts and design. *International Journal of Man-Machine Studies*, 16(3):231–236, 1982.

[94] C.A. Ellis and S.J. Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, 1989.

[95] C.A. Ellis, S.J. Gibbs, and G. Rein. Design and use of a group editor. Technical Report STP-263-88, M.C.C. Software Technology Programme, Austin, United States of America, 1988.

[96] S.H. Ellis and M.W. McGreevy. Influence of a perspective cockpit traffic display format on pilot avoidance maneuvers. In A.T. Pope and L.D. Haugh, editors, *Proceedings Of The 27th Annual Meeting Of The Human Factors Society*, pages 762–766. Human Factors Society, Santa Monica, United States of America, 1983.

[97] J. Fabry, T. Harding, and K. Mallory. Control-display integration on large multi-system control panels. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 159–162. Human Factors Society, Santa Monica, United States of America, 1981.

[98] D. Fennell. *Investigation Into The Kings Cross Underground Fire*. Department of Transport, London, United Kingdom, 1988.

[99] A.C.W. Finkelstein. Re-use of formatted requirements specifications. *Software Engineering Journal*, 3(5):186–197, 1988.

[100] J. Finlay and M.D. Harrison. Pattern recognition and interaction models. In D. Diaper, D. Gilmore, G. Cockton, and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 149–154. Elsevier Science Publications, North Holland, Netherlands, 1990.

[101] E. Fischer, R. Haines, and T. Proce. Cognitive issues in head-up displays. Technical Report 1711, National Aeronautic and Space Administration (NASA), Washington DC, United States Of America, 1980.

[102] F.B. Fitch. *Symbolic Logic*. The Ronald Press, New York, United States of America, 1952.

[103] J.M. Flach and K.J. Vicente. Complexity, difficulty, direct manipulation and direct perception. Technical Report EPRL-89-03, Engineering Research Laboratory, University of Illinois, Urbana-Champaign, United States of America, 1989.

[104] H.C. Foushee and R.L. Heimrich. Group interaction and flight crew performance. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 189–227. Academic Press, London, United Kingdom, 1988.

[105] J. Fox. Automated assistance for safety critical decisions. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 107–120. Clarendon Press, Oxford, United Kingdom, 1990.

[106] K. Freburger. RAPID: Prototyping control panel interfaces. *ACM SIGPLAN Notices*, 22(12):416–422, 1987.

[107] D. M. Frohlich and P. Luff. Conversational resources for situated action. In K. Bice and C. Lewis, editors, *Proceedings Of The CHI'89 Conference On Human Factors In Computing Systems*, pages 253 – 258. ACM, New York, United States of America, 1989.

[108] M. Fujita, M. Ishisone, H. Nakamura, H. Tanaka, and T. Moto-Aka. Using the temporal logic programming language Tokio for algorithm description and automatic CMOS gate array synthesis. In E. Wada, editor, *Proceedings of The 4th Annual Conference - Logic Programming '85*, LNCS 221, pages 246–255. Springer-Verlag, Berlin, FDR, 1986.

[109] M. Fujita, S. Kono, H. Tanaka, and T. Moto-Aka. Tokio: Logic programming based on temporal logic and its compilation to PROLOG. In E. Shapiro, editor, *Third International Conference On Logic Programming*, LNCS 225, pages 695–708. Springer-Verlag, Berlin, FDR, 1986.

[110] K. Fukunaga and S. Hirose. An experience with a PROLOG based object oriented language. In N. Meyrowitz, editor, *Proceedings Of OOPSLA '86*, pages 224–231. ACM, New York, United States of America, 1986.

[111] D.M. Gabbay. Executing temporal logic for interactive systems. Technical report, Department of Computing, Imperial College Of Science And Technology, London, United Kingdom, 1987.

[112] D.M. Gabbay. Modal and temporal logic programming. In A. Galton, editor, *Temporal Logics And Their Applications*, pages 197–223. Academic Press, London, United Kingdom, 1987.

[113] I.A.R. Galer and B.L. Yap. Ergonomics in intensive care: Applying human factors to the design and evaluation of a patient monitoring system. *Ergonomics*, 23(8):763 – 779, 1980.

[114] D. Gallie. *In Search Of The Working Class: Automation And Social Integration Within The Capitalist Enterprise*. Cambridge University Press, Cambridge, United Kingdom, 1978.

[115] A. Galton. A critical examination of J.F. Allen's theory of action and time. *Artificial Intelligence*, 42(2):159–88, 1990.

[116] W.W. Gaver, R.B. Smith, and T. O'Shea. Effective sounds in complex systems: The Arkola simulation. In S.P. Robertson, G.M. Olson, and J.S. Olson, editors, *Proceedings Of The CHI'91 Conference On Human Factors In Computing Systems*, pages 85–90. ACM, New York, United States of America, 1991.

[117] J.J. Gibson. The information available in pictures. *Leonardo*, 4:27–35, 1971.

[118] J.J. Gibson. *The Ecological Approach To Visual Perception*. Houghton-Mifflin, Boston, United States of America, 1979.

[119] K. Gill. Oil company's safety system 'left men helpless'. *The Times*, page 5, 13 November 1990.

[120] T. Gillie and D. Berry. Direct perception and control of dynamic systems. Technical report, Department of Experimental Psychology, University of Oxford, Oxford, United Kingdom, 1991.

[121] P. Gillman. The railway report. *The Sunday Times*, pages S.54–S.60, 8 December 1991.

[122] S.J. Goldsack and A.C.W. Finkelstein. Requirements engineering for real-time systems. *Software Engineering Journal*, 6(3):101–115, 1991.

[123] D. Good. Mechanical proofs about computer programs. Technical Report TR 41, Institute For Computer Science, University of Texas, Austin, United States of America, 1984.

[124] D. Gopher, M. Olin, Y. Donchin, and M. Bieski. The nature and causes of human errors in a medical intensive care unit. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 956–960. Human Factors Society, Santa Monica, United States of America, 1989.

[125] J. Grudin. The case against user interface consistency. *Communications Of The ACM*, 10(32):1164–1173, 1989.

[126] E. Gullischen. Biggertalk: Object oriented PROLOG. Technical Report STP-125-85, M.C.C., Austin, United States of America, 1985.

[127] S.B. Haber, D.S. Metlay, and D.A. Crouch. Influence of organisational factors on safety. In N.C. Goodwin, editor, *Proceedings Of The 34th Annual Meeting Of The Human Factors Society*, pages 871–875. Human Factors Society, Santa Monica, United States of America, 1990.

[128] R. Hale. Temporal logic programming. In A. Galton, editor, *Temporal Logics And Their Applications*, pages 91–119. Academic Press, London, United Kingdom, 1987.

[129] P. Hammond and M. Sergot. Logic for representing data and expertise. Technical report, Department of Computing, Imperial College of Science And Technology, London, United Kingdom, 1984.

[130] J.P. Hansen. The use of eye mark recordings to support verbal retrospection in software testing. Technical report, The Institute of Psychology, University of Aarhus, Aarhus, Denmark, 1990.

[131] J.P. Hansen and C.V. Skou. Time tunnels: An ecological interface for dynamic data. Technical report, Department of Information Technology, The Risø National Laboratory, Roskilde, Denmark, 1989.

[132] D. Harel. On visual formalisms. *Communications Of The ACM*, 31(5):514–530, 1988.

[133] S. Harker. The use of prototyping and simulation in the development of large-scale applications. *Computer Journal*, 31(5), 1988.

[134] M. D. Harrison, C. R. Roast, and P. C. Wright. Complimentary methods for the iterative design of interactive systems. In G. Salvendy and M.J. Smith, editors, *Designing And Using Human-Computer Interfaces And Knowledge Based Systems*, pages 651 – 658. Elsevier Scientific Publications, North Holland, Netherlands, 1989.

[135] S.G. Hart and T.E. Wempe. Cockpit display of traffic information: Airline pilot's opinion about content symbology and format. Technical Report 78601, National Aeronautic and Space Administration (NASA) Ames Research Center, Moffett Field, United States of America, 1979.

[136] P.F.V. Hasle. Building a temporal logic for natural language understanding with the HOL system. Technical Report 137, Department of Communication, University of Ålborg, Ålborg, Denmark, September 1990.

[137] J.K. Hawley, R.J. dePontbriand, and E.W. Frederickson. Making MANPRINT work: The lessons of the FAADS experience. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1049–1053. Human Factors Society, Santa Monica, United States Of America, 1989.

[138] Health And Safety Executive. *Some Aspects Of The Safety Of Nuclear Installations In Great Britain: Replies From The Secretary of State For Energy*, London, United Kingdom, 1976.

[139] Health And Safety Executive. *The Health And Safety Review Of United Kingdom Nuclear Establishments*, London, United Kingdom, 1979.

[140] Health And Safety Executive. *The Notification Of Installations Handling Hazardous Substances Regulations*, IS1982/1357, London, United Kingdom, 1982.

[141] Health And Safety Executive. *The Control Of Major Accident Hazard Regulations*, IS1984/1902, London, United Kingdom, 1984.

[142] C. Heath and P. Luff. Collaborative activity and technological design: Task coordination in London Underground control rooms. Technical report, Rank Xerox EuroParc, Cambridge, United Kingdom, 1991.

[143] P. Henderson. Functional programming, formal specification and rapid prototyping. *IEEE Transactions In Software Engineering*, SE - 12(2):241 – 250, 1986.

[144] R.L. Henneman and W.B. Rouse. Human performance in monitoring and controlling hierarchical large-scale systems. *IEEE Transactions On Systems, Man And Cybernetics*, SMC-14(2):184–191, 1984.

[145] R.L. Henneman and W.B. Rouse. On measuring the complexity of monitoring and controlling large-scale systems. *IEEE Transactions On Systems, Man And Cybernetics*, SMC-16(2):193–207, March/April 1986.

[146] R.A. Hess. A qualitative model of human interaction with complex dynamic systems. *IEEE Transactions On Systems, Man And Cybernetics*, SMC-17(1):33–51, 1987.

[147] P. Hetherington. Occidental denies undue haste as Piper rebuilding goes ahead. *The Guardian*, page 2, 13 November 1990.

[148] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, London, United Kingdom, 1985.

[149] J.-M. Hoc. Analysis of cognitive activities in process control for the design of computer aids - an example the control of a blast furnace. In H.-J. Bullinger and B. Shackel, editors, *Human-Computer Interaction—INTERACT'87*, pages 257 – 262. Elsevier Science Publications, North Holland, Netherlands, 1987.

[150] E. Hofer and F. Ruggiero. Hypermedia as communication and prototyping tools in the concurrent design of commercial aircraft products. In G. Cockton D. Diaper, D. Gilmore and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 303–308. Elsevier Science Publications, North Holland, Netherlands, 1990.

[151] E. Hollnagel. The GRADIENT project: Technical overview. Technical Report ESPRIT P857, Commission Of The European Communities, Information Technology And Telecommunications Task Force, Brussels, Belgium, 1991.

[152] E. Hollnagel and D.D. Woods. Cognitive systems engineering: New wine in new bottles. *International Journal of Man-Machine Studies*, 18(6):583–600, 1983.

[153] G.E. Hughes and M.J. Cresswell. *An Introduction To Modal Logic.* Methuen, London, United Kingdom, 1968.

[154] D.L.M. Hunt and P.K. Ramskill. The behaviour of tanks engulfed in fire - the development of a computer program. In J. Burgoyne, editor, *The Assessment And Control Of Major Hazards*, pages 71–86. Pergamon Press and The Institution Of Chemical Engineers, Oxford, United Kingdom, 1985.

[155] G.S. Hura and J.W. Attwood. The use of Petri nets to analyse coherent fault trees. *IEEE Transactions On Reliability*, 37(5):469–473, 1988.

[156] Institute Of Chemical Engineers. *Nomenclature For Hazard And Risk Assessment In The Process Industries*, Rugby, United Kingdom, 1985.

[157] International Atomic Energy Agency. *Report Of The Safety Advisory Group On Reactor Design, Operation And Safety Training*, number 75 INSAG-1 in Safety Series, Vienna, Austria, 1986.

[158] International Atomic Energy Agency and The Commission of the European Community. *Critical Survey of Research On Human Factors And The Man-Machine Interaction*, IAEA-SM-26B/29, Vienna, Austria, 1984.

[159] Inveco And The Alberta Community And Occupational Health Office. *Feasibility Of Reducing Injuries To Drilling Rig Employees Through The Use Of Mechanical Pump Handling Systems*, Edmonton, Canada, 1983.

[160] J. Jacky. Formal specification for a clinical cyclotron control system. In M. Moriconi, editor, *Proceedings Of The ACM SIGSOFT International Workshop On Formal Methods In Software Development*, pages 45–54. ACM, New York, United States of America, 1990.

[161] D. John. Vandals cause chaos for 150,000 commuters. *The Guardian*, page 4, 10 December 1991.

[162] C.W. Johnson. Applying temporal logic to support the specification and prototyping of concurrent multi-user interfaces. In D. Diaper and N. Hammond, editors, *People And Computers VI: Usability Now*, pages 145–156. Cambridge University Press, Cambridge, United Kingdom, 1991.

[163] C.W. Johnson and M.D. Harrison. PRELOG - a system for presenting and rendering logic specifications of interactive systems. In C.E. Vandoni and D.A. Duce, editors, *EUROGRAPHICS '90*, pages 469–480. Elsevier Science Publications, North Holland, Netherlands, 1990.

[164] C.W. Johnson and M.D. Harrison. Declarative graphics and dynamic interaction. In F.H. Post and W. Barth, editors, *EUROGRAPHICS '91*, pages 195–207. Elsevier Science Publications, North Holland, Netherlands, 1991.

[165] C.W. Johnson and M.D. Harrison. Using temporal logic to support the specification and prototyping of interactive control systems. *International Journal Of Man-Machine Studies*, 36:357–385, 1992.

[166] P. Johnson. *Human Computer Interaction: Psychology, Task Analysis and Software Engineering.* McGraw-Hill, London, United Kingdom, 1992.

[167] E.J. Joyce. Malfunction 54: Unravelling a deadly medical mystery of a computerised accelerator gone awry. *American Medical News*, page 1, 3 October 1986.

[168] S.M.P. Julien. Graphics in MicroPROLOG. Technical Report 82/17, Department of Computing, Imperial College Of Science and Technology, London, United Kingdom, 1982.

[169] M.K. Junge and M.J. Giacomi. Human factors in equipment development for the Space Shuttle: A study of the general purpose work station. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 218–222. Human Factors Society, Santa Monica, United States of America, 1981.

[170] W.A. Kellogg. Conceptual consistency in the user interface: Effects on user performance. In H.-J. Bullinger and B. Shackel, editors, *Human-Computer Interaction—INTERACT'87*, pages 389–394. Elsevier Science Publications, North Holland, Netherlands, 1987.

[171] R.L. Kershner, J.W. Gebhard, and E.B. Silverman. An evaluation of nuclear power plant operator performance using a safety parameter display system. In R.E. Edwards, editor, *Proceedings Of The 26th Annual Meeting Of The Human Factors Society*, pages 789–793. Human Factors Society, Santa Monica, United States of America, 1982.

[172] V. De Keyser. The structuring of knowledge of operators in continuous processes : A case study of a continuous casting-plant start up. In J. Rasmussen, K. Duncan, and J. Leplat, editors, *New Technology And Human Error*, pages 247 – 259. J. Wiley And Sons, New York, United States Of America, 1987.

[173] V. De Keyser. Temporal decision making in complex environments. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 121–128. Clarendon Press, Oxford, United Kingdom, 1990.

[174] S. Khosla. *System Specification: A Deontic Approach*. PhD thesis, Department of Computing, Imperial College Of Science And Technology, London, United Kingdom, 1988.

[175] D.E. Kieras and P.G. Polson. An approach to the formal analysis of user complexity. *International Journal Of Man-Machine Studies*, 22(4):365–394, 1985.

[176] M. Kirkpatrick and K. Mallory. Substitution error potential in nuclear power plant control rooms. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 163–167. Human Factors Society, Santa Monica, United States of America, 1981.

[177] B. Kirwan. A human factors review and human reliability programme for the design of a large U.K. nuclear chemical plant. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1009–1013. Human Factors Society, Santa Monica, United States of America, 1989.

[178] O. Kivinen. Finns live in fear of ill wind from across the border. *The Times*, page 11, 25 March 1992.

[179] T.A. Kletz. *What Went Wrong? Case Histories Of Process Plant Disasters*. Gulf, Houston, United States Of America, 1985.

[180] M. Kooij. Interface specification with temporal logic. In S.J. Greenspan, editor, *The 5th International Workshop On Software Specification And Design*, pages 104–110. IEEE Computer Society Press, Washington, United States of America, 1989.

[181] R. Kowalski. Algorithm = logic + control. *Communications Of The ACM*, 22(7):424 – 436, 1979.

[182] R. Kowalski. *Logic For Problem Solving.* Elsevier Science Publications, North Holland, Netherlands, 1979.

[183] R. Kowlaski. The relation between logic programming and logic specification. *Philosophical Transactions Of The Royal Society of London*, 312(A):345–361, 1984.

[184] D. Kozen, editor. *Logic of Programs 1981 - Proceedings.* LNCS 131. Springer-Verlag, Berlin, FDR, 1982.

[185] B. Kramer. Introducing the GRASPIN specification language SEGRAS. *Journal of Systems and Software*, 15(1):17–31, 1991.

[186] W. Kuhmann. Stress inducing properties of system response times. *Ergonomics*, 32(3):271 – 280, 1989.

[187] W. Kuhmann, W. Boucsein, F. Schaefer, and J. Alexander. Experimental investigation of psychophysiological stress-reactions induced by different system response times in human-computer interactions. *Ergonomics*, 30(6):933 – 943, 1987.

[188] P. Ladkin. Primitives and units for time specifications. In T. Kehler, S.Rosenschein, R. Filman, and P.F. Patel-Schneider, editors, *Proceedings Of The Fifth Annual Conference On Artificial Intelligence*, pages 354–359. Morgan Kaufman, Los Altos, United States of America, 1986.

[189] L. Lamport. TIMESETS - a new method for temporal reasoning about programs. In D. Kozen, editor, *Logic Of Programs 1981 - Proceedings*, LNCS 131, pages 177–196. Springer-Verlag, Berlin, FDR, 1982.

[190] G.L. Lazarev. Reusability in Smalltalk: A case study. *Journal of Object Oriented Programming*, 4(2):11–18, 1991.

[191] D. Learmont. United Kingdom Air Accidents Investigation Branch slams 737-400 displays. *Flight International*, 6 March 1991.

[192] J. Lee and N. Moray. Trust and the allocation of function in the control of automatic systems. Technical report, Department of Mechanical And Industrial Engineering, University of Illinois, Urbana-Champaign, United States of America, 1990.

[193] S.J. Leffler, M.K. McKusick, M.J. Karels, and J.S. Quarterman. *The Design And Implementation Of The 4.3BSD UNIX Operating System.* Addison Wesley, Reading, United States of America, 1990.

[194] H.W. Leibowitz. Human senses in flight. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 83–110. Academic Press, London, United Kingdom, 1988.

[195] C.H. Lewis. A research agenda for the Nineties in human computer interaction. *Human Computer Interaction*, 5(2-3):125–143, 1990.

[196] F. Lin and F.E. Hunt. LCD-Reification: A formal method for developing PROLOG programs. In S.J. Greenspan, editor, *The 5th International Workshop On Software Specification And Design*, pages 249–256. IEEE Computer Society Press, Washington DC, United States of America, 1989.

[197] Y. Liu and C.D. Wickens. Visual scanning with or without spatial uncertainty and time sharing performance. In N.C. Goodwin, editor, *Proceedings Of The 34th Annual Meeting Of The Human Factors Society*, pages 76–81. Human Factors Society, Santa Monica, United States of America, 1990.

[198] A.M. Madni. The role of human factors in system design and acceptance. *Human Factors*, 30(4):395–414, 1988.

[199] O.L. Madsen and B. Møller-Pedersen. What object-oriented programming may be - and what it does not have to be. In S. Gjessing and K. Nygard, editors, *ECOOP '88: European Conference On Object Oriented Programming*, LNCS 322. Springer-Verlag, Berlin, FRG, 1988.

[200] K. Maguire. Signals staff pay deal will raise safety level, says BR. *The Daily Telegraph*, page 2, 6 March 1991.

[201] T.B. Malone. MPTS methodology in the Navy: Enhanced HARDMAN. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1044–1048. Human Factors Society, Santa Monica, United States of America, 1989.

[202] T.B. Malone, C.C. Heasly, and D.R. Eike. The army MANPRINT idea: Integrated decision engineering aid. In N.C. Goodwin, editor, *Proceedings Of The 34th Annual Meeting Of The Human Factors Society*, pages 1113–1116. Human Factors Society, Santa Monica, United States of America, 1990.

[203] T.B. Malone, M. Kirkpatrick, K. Mallory, D. Eike, J.H. Johnson, and R.W. Walker. Human factors evaluation of control room design and operator performance at Three Mile Island-2. Technical Report NUREG/CR-1270, United States' Regulatory Commission, Washington DC, Unites States of America, January 1980.

[204] G. Mancini. Modelling humans and machines. In L.P. Goodstein, H.B. Anderson, and S.E. Olsen, editors, *Task, Errors and Mental Models*, pages 278 – 292. Taylor and Francis, London, United Kingdom, 1988.

[205] Z. Manna and A. Pnueli. Verification of concurrent programs: The temporal framework. In R.S. Boyer and J. Strother Moore, editors, *The Correctness Problem In Computer Science*, pages 215–273. Academic Press, London, United Kingdom, 1981.

[206] D. McCrobie. Human factors design considerations for military trains. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 536–540. Human Factors Society, Santa Monica, United States of America, 1989.

[207] J.A. McDermid. Skills and technologies for the development and evaluation of safety critical systems. Technical Report YCS 138, Department of Computer Science, University of York, York, United Kingdom, 1990.

[208] B. McGibbon. Case study: A flexible energy management system for Europe. *Usability Now*, 1:2–3, 1990.

[209] B. Meyer. *Object-Oriented Software Construction.* Prentice Hall, London, United Kingdom, 1988.

[210] V.C. Miles, C.W. Johnson, J.C. McCarthy, and M.D. Harrison. Supporting prediction in complex dynamic systems. In D. Diaper and N. Hammond, editors, *People And Computers VI: Usability Now*, pages 133–144. Cambridge University Press, Cambridge, United Kingdom, 1991.

[211] M. De Montmollin and V. De Keyser. Expert logic versus operator logic. In G. Mancini, G. Johannsen, and L. Martensson, editors, *Analysis, Design And Evaluation Of Man-Machine Systems*, pages 43–49. Pergamon Press, Oxford, United Kingdom, 1986.

[212] N. Moray. Human factors research and nuclear safety. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 576–578. Human Factors Society, Santa Monica, United States of America, 1989.

[213] N. Moray. Objective and subjective estimates of human error. *IEEE Transactions On Reliability*, 38(3):301–304, 1989.

[214] N. Moray, P. Lootstein, and J. Pajak. The acquisition of process control skills. *IEEE Transactions On Systems, Man and Cybernetics*, SMC-16(4):497–504, 1986.

[215] W. Morehouse and M.A. Subamaniam. The Bhopal tragedy. Technical report, Council For International And Public Affairs, New York, United States of America, 1986.

[216] J.N. Mosier and S.L. Smith. Application of guidelines for designing user interface software. *Behaviour and Information Technology*, 5(1):39–46, 1985.

[217] C. Moss. Cut and paste : Defining the impure primitives of PROLOG. In E. Shapiro, editor, *Third International Conference On Logic Programming*, LNCS 225, pages 686 – 694. Springer-Verlag, Berlin, FDR, 1986.

[218] B. Moszkowski. *Executing Temporal Logic Programs.* Cambridge University Press, Cambridge, United Kingdom, 1986.

[219] J. Moxon. Airbus offers autothrottle option. *Flight International*, 1 May 1991.

[220] B.M. Muir. Trust between humans and machines and the design of decision aids. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Cognitive Engineering In Complex Dynamic Worlds*, pages 71–83. Academic Press, London, United Kingdom, 1988.

[221] J. Mullin. 'Too many aboard' helicopter crash rig. *The Guardian*, page 2, 17 March 1992.

[222] B.A. Myers. Demonstrational interfaces: A step beyond direct manipulation. In D. Diaper and N. Hammond, editors, *People And Computers VI: Proceedings Of HCI'91*, pages 11–30. Cambridge University Press, Cambridge, United Kingdom, 1991.

[223] C. Neesham. Common codes of behaviour for sympathetic interaction. *Computing*, pages 28–30, 21 March 1991.

[224] L. Ness. L.0 - A parallel executable temporal logic. In M. Moriconi, editor, *Proceedings Of The ACM SIGSOFT International Workshop On Formal Methods In Software Development*, pages 80–89. ACM, New York, United States of America, 1990.

[225] New Zealand Royal Commission. *Report Of The Royal Commission To Inquire Into The Crash On Mt. Erebus, Antartica Of A DC-10 Aircraft Operated By Air New Zealand, Limited.*, Wellington, New Zealand, 1981. Government Printer.

[226] J. Nielsen. *Coordinating User Interfaces for Consistency.* Academic Press, London, United Kingdom, 1989.

[227] D.A. Norman. Cognitive engineering. In D.A. Norman and S.W. Draper, editors, *User Centered System Design*, pages 31–62. Lawrence Erlbaum Associates, Hillsdale, United States of America, 1986.

[228] D.A. Norman. The 'problem' with automation : Inappropriate feedback and interaction not 'over-automation'. In D.E. Broadbent, J. Reason, and A. Baddeley, editors, *Human Factors In Hazardous Situations*, pages 137–145. Clarendon Press, Oxford, United Kingdom, 1990.

[229] G.A. Osga. Human factors issues in future Navy workstation development: Symposium abstract. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1077–1078. Human Factors Society, Santa Monica, United States of America, 1989.

[230] G.A. Osga. User interface issues for future ship combat consoles. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1079–1083. Human Factors Society, Santa Monica, United States of America, 1989.

[231] T.W. Page. *An Object-Oriented Logic Programming Environment For Modelling.* PhD thesis, University of California, 1989.

[232] E. Palmer. Conflict resolution maneuvers during near miss encounters with cockpit traffic displays. In A.T. Pope and L.D. Haugh, editors, *Proceedings Of The 27th Annual Meeting Of The Human Factors Society*, pages 757–761. Human Factors Society, Santa Monica, United States of America, 1981.

[233] S.J. Payne and T.R.G. Green. Task-Action Grammars: A model of the mental representation of task languages. *Human Computer Interaction*, 2(2):93–133, 1986.

[234] S.J. Payne and T.R.G. Green. The structure of command languages: An experiment on Task-Action Grammar. *The International Journal of Man-Machine Studies*, 30(2):213–234, 1989.

[235] F. C. N. Pereira. Can drawing be liberated from the Von Neumann style ? In M. Van Caneghan and D. H. D. Warren, editors, *Logic Programming And Its Application*, pages 175 – 187. Ablex Publishing, Norwood, United States of America, 1986.

[236] R.M. Pew, D.C. Miller, and C.E. Feehrer. Evaluating nuclear control room improvements through analysis of critical operator decisions. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 100–104. Human Factors Society, Santa Monica, United States of America, 1981.

[237] D.W. Plath and P.E. Kolesnik. Readability and operability of three types of digital switches. *Journal of Engineering Psychology*, 5:47–53, 1966.

[238] G. Pólya. *Mathematical Discovery*. J. Wiley, New York, United States of America, 1988.

[239] H. Popitz, H.P. Bahrdt, E.A. Jures, and H. Kesting. Technik und industriearbeit. In J.C.B. Mohr, editor, *Soziologische Untersuchungen In Der Hüttenindustrie*. Tubingen, Berlin, D.D.R., 1957.

[240] E.L. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–268, 1943.

[241] D.C. Poteralski and R.C. Vogel. Status of severe accident research. *IEEE Transactions On Nuclear Science*, 35(1):914–918, 1988.

[242] P. Potter. The design of the Chernobyl Unit 4 reactor. In N. Worley and J. Lewins, editors, *The Chernobyl Accident And Its Implications For The United Kingdom - Report Number 19 Of The Watt Committee On Energy*, pages 9–18. Elsevier Applied Science, London, United Kingdom, 1988.

[243] President's Task Force On Aircraft Crew Compliment, United States' Government. *Report On Aircraft Crew Compliment*, Washington DC, United States of America, 1981.

[244] J. Race. Computer-encouraged pilot error. *Computer Bulletin*, 2(6):13–15, 1990.

[245] J. Rasmussen. On the structure of knowledge - a morphology of mental models in a man-machine system context. Technical Report Risø-M-2192, The Risø National Laboratory, Roskilde, Denmark, 1979.

[246] J. Rasmussen. Skills, rules and knowledge; signals, signs and symbols and other distinctions in human performance models. *IEEE Transactions On Systems, Man And Cybernetics*, SMC-13(3):257–266, 1983.

[247] J. Rasmussen. Cognitive engineering, a new profession? In L.P. Goodstein, H.B. Anderson, and S.E. Olsen, editors, *Task, Errors and Mental Models*, pages 325 – 334. Taylor and Francis, London, United Kingdom, 1988.

[248] J. Rasmussen. Coping safely with complex systems. Technical Report Risø-M-2769, The Risø National Laboratory, Roskilde, Denmark, 1989.

[249] J. Rasmussen and M. Lind. Coping with complexity. Technical Report Risø-M-2293, The Risø National Laboratory, Roskilde, Denmark, 1981.

[250] J. Rasmussen and O.M. Pedersen. Human factors in probabilistic risk analysis and in risk management. Technical Report Risø-N-18-83, The Risø National Laboratory, Roskilde, Denmark, 1983.

[251] J. Reason. The psychology of mistakes: a brief review of planning failures. In K. Duncan J. Rasmussen and J. Leplat, editors, *New Technology And Human Error*, pages 45–53. John Wiley And Sons, Chichester, United Kingdom, 1987.

[252] J. Reason. *Human Error*. Cambridge University Press, Cambridge, United Kingdom, 1990.

[253] E.S. Reed. James Gibson's ecological approach to cognition. In A. Costall and A. Still, editors, *Cognitive Psychology In Question*, pages 142–176. Harvester Press, Brighton, United Kingdom, 1987.

[254] J. Reid. A knowledge based approach to aid the construction and interpretation of computer generated graphics for blind users. Technical report, Department of Computer Science, University of York, York, United Kingdom, 1991. M.Sc. Thesis.

[255] P. Reisner. Formal grammar and human factors design of an interactive graphics system. *IEEE Transactions On Software Engineering*, SE - 7(2):229 – 240, 1981.

[256] P. Reisner. What is inconsistency? In G. Cockton D. Diaper, D. Gilmore and B. Shackel, editors, *Human-Computer Interaction—INTERACT'90*, pages 175–181. Elsevier Science Publications, North Holland, Netherlands, 1990.

[257] E. Rich. Users are individuals: Individualising user models. *International Journal Of Man-Machine Studies*, 18(3):199–214, 1983.

[258] E. Rich and K. Knight. *Artificial Intelligence*. Mc Graw-Hill, London, United Kingdom, 1991.

[259] S.N. Roscoe. Airborne displays for navigation and flight. *Human Factors*, 10:617–629, 1968.

[260] C. Rose. *Inside The Apple Macintosh*, volume I. Addison Wesley, Wokingham, United Kingdom, 1986.

[261] W.B. Rouse. Human-computer interaction in the control of dynamic systems. *ACM Computing Surveys*, 13(1), 1981.

[262] K.S. Rubin, P.M. Jones, C.M. Mitchell, and T.C. Goldstein. A Smalltalk implementation of an intelligent operators assistant. In N. Meyrowitz, editor, *Proceedings of OOPSLA '88*, pages 234–247. ACM, New York, United States of America, 1988.

[263] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling And Design*. Prentice Hall, London, United Kingdom, 1991.

[264] C. Runciman. From abstract models to functional prototypes. In M.D. Harrison and H.W. Thimbleby, editors, *Formal Methods In Human Computer Interaction*, pages 201–232. Cambridge University Press, Cambridge, United Kingdom, 1990.

[265] C. Runciman and M.A. Firth. Formalised development of software by machine-assisted transformation. *Software Engineering Notes*, 15(4):115–117, 1990.

[266] G. Saake. On first order temporal logics with changing domains for information systems specification. Technical report, IBM Wissenschaftliches Zentrum, Heidelberg, Germany, 1988.

[267] G. Saake and U.W. Lipeck. Using finite-linear temporal logic for specifying database dynamics. In E. Börger and H. Kleine Büning and M.M. Richter, editors, *Workshop On Computer Science Logic - 1988 Proceedings*, LNCS 385, pages 288–300, Berlin, FDR, 1988. Springer-Verlag.

[268] M. Sainsbury. *Logical Forms: An Introduction To Philosophical Logics*. Blackwell, Oxford, United Kingdom, 1991.

[269] T. Sakuragawa. Temporal PROLOG - a programming language based on temporal logic. *Computer Software (in Japanese)*, 4(3):15–27, 1987.

[270] T. Sakuragawa. RACCO: A modal-logic programming language for writing models of real-time process-control systems. *Computer Software (in Japanese)*, 5(3):22–33, 1988.

[271] P.M. Sanderson, J.M. James, and K.S. Seidler. SHARPA: An interactive software environment for protocol analysis. *Ergonomics*, 32(11):1271 – 1302, 1989.

[272] T.F. Sanquist and Y. Fujita. Protocol analysis and action classification in the evaluation of an advanced annunciator system design. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1064–1067. Human Factors Society, Santa Monica, United States of America, 1989.

[273] A. Schaafstal. Knowledge and skills of operators in paper mills a comparison between experts and novices. In *Proceedings Of the Second European Meeting On Cognitive Approaches To Process Control*, pages 149–166, Sienna, Italy, 1989. CEC-JRC Ispra and the University of Sienna.

[274] L.M. Schleifer and B.C. Amick. System response times and methods of pay: Stress effects in computer-based tasks. *International Journal Of Human - Computer Interaction*, 1:23 – 39, 1989.

[275] L.M. Schleifer and O.G. Okogbaa. System response times and method of pay: Cardiovascular stress effects in computer-based tasks. *Ergonomics*, 33(12):1459 – 1509, 1990.

[276] K. Schmidt. Modelling cooperative work in complex environments. In *Proceedings Of the Second European Meeting On Cognitive Approaches To Process Control*, pages 173–182, Sienna, Italy, 1989. CEC-JRC Ispra and the University of Sienna.

[277] J.M.C. Schraagen. Requirements for a damage control decision-support system: Implications from expert-novice differences. In *Proceedings Of the Second European Meeting On Cognitive Approaches To Process Control*, pages 315–322, Sienna, Italy, October 1989. CEC-JRC Ispra and the University of Sienna.

[278] T. Seamster, C. Baker, and P.J. Andrews. Tactical symbology for visual displays: The standardisation process. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1094–1098. Human Factors Society, Santa Monica, United States of America, 1989.

[279] B.A. Semenov. Nuclear safety in the Soviet Union. *International Atomic Agency Bulletin*, 25(2), 1983.

[280] J.L. Seminara, W.R. Gonzalez, and S.O. Parsons. Human factors review of nuclear power plant control room design. Technical Report RI-NP-1977-309, Electronic Power Research Institute and Lockheed Missiles And Space Company, Palo Alto, United States of America, 1977.

[281] D. Serrig. Human factors and the medical use of nuclear byproduct material. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 1014–1018. Human Factors Society, Santa Monica, United States of America, 1989.

[282] G.A. Sexton. Cockpit-crew systems design and integration. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 495–526. Academic Press, London, United Kingdom, 1988.

[283] Sheen. *MV Herald Of Free Enterprise*. Court no. 8074. Department of Transport, London, United Kingdom, 1987.

[284] T.B. Sheridan. The system perspective. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 27–51. Academic Press, London, United Kingdom, 1988.

[285] B. Shneiderman. Multiparty grammars and related features for defining interactive systems. *IEEE Transactions On Systems, Man and Cybernetics*, SMC-12:148–154, 1982.

[286] H.G. Simpson. Propylene leakage. *Power And Works Engineering*, page 8, May 1974.

[287] W.T. Singleton. Theoretical approaches to human error. *Ergonomics*, 16(6):727–737, 1973.

[288] M. Sloman and J. Kramer. *Distributed Systems And Computer Networks*. Prentice Hall, Englewood Cliffs, United States Of America, 1987.

[289] A. Sorge, G. Hartmann, M. Warner, and I. Nicholas. *Microelectronics And Manpower In Manufacturing*. Gower Press, Alershot, United Kingdom, 1983.

[290] M. Stefik, D.G. Foster, K. Kahn, and D.G. Tatar. WYSIWIS revised: Early experiences with multiuser interfaces. *ACM Transactions on Office Information Systems*, 5:2:147–167, April 1987.

[291] B.A. Steiner and M.J. Camacho. Situation awareness: Icons vs alphanumerics. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 28–32. Human Factors Society, Santa Monica, United States of America, 1989.

[292] M. Stephens. *Three Mile Island*. Junction Books, London, England, 1980.

[293] A.F. Stokes and C.D. Wickens. Aviation displays. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 433–461. Academic Press, London, United Kingdom, 1988.

[294] M.N. Stollings. Information processing load of graphics versus alphanumeric weapon format displays for advanced cockpit displays. Technical Report AFWAL-TR-84-3037, United States' Air Force Flight Dynamics Laboratory, Wright Patterson Air Force Base, United States of America, 1984.

[295] R.B. Stone and G.L. Babcock. Airline pilot's perspective. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 529–560. Academic Press, London, United Kingdom, 1988.

[296] B. Sufrin. Formal methods and the design of effective user interfaces. In M.D. Harrison and A.F. Monk, editors, *People And Computers : Designing For Usability*, pages 24–43. Cambridge University Press, Cambridge, United Kingdom, 1986.

[297] B. Sufrin and J. He. Specification, refinement and analysis of interactive processes. In M. D. Harrison and H. W. Thimbleby, editors, *Formal methods in Human Computer Interaction*, pages 153–200. Cambridge University Press, Cambridge, United Kingdom, 1990.

[298] P. Szekely and B. Myers. A user interface toolkit based on graphical objects and constraints. *ACM SIGPLAN Notices*, 23(11):36–45, 1988.

[299] D.G. Tatar, G. Foster, and D.G. Bobrow. Design for conversation: lessons from Cognoter. *International Journal of Man-Machine Studies*, 34(2):185–209, 1991.

[300] R.M. Taylor and S.J. Selcon. Cognitive quality and situational awareness with advanced aircraft attitude displays. In N.C. Goodwin, editor, *Proceedings Of The 34th Annual Meeting Of The Human Factors Society*, pages 26–30. Human Factors Society, Santa Monica, United States of America, 1990.

[301] H.W. Thimbleby. Generative user-engineering principles for user interface design. In B. Shackel, editor, *Human-Computer Interaction—INTERACT'84*, pages 102 – 107. Elsevier Science Publications, North Holland, Netherlands, 1984.

[302] H.W. Thimbleby. User interface design. In A.F. Monk, editor, *The Fundamentals Of Human-Computer Interaction*, pages 165–180. Academic Press, London, United Kingdom, 1985.

[303] H.W. Thimbleby. Delaying commitment. *IEEE Software*, 5(3):78–86, May 1988.

[304] H.W. Thimbleby. *User Interface Design*. Addison Wesley, Wokingham, United Kingdom, 1990.

[305] K.C. Thompson. A C-141 roll axis electromechanical primary flight control actuation system. In *Conference On Aircraft Design, Systems And Technology Meeting*, AIAA-83-2488, Fort Worth, United States of America, October 1983.

[306] S. Thompson. Functional programming: Executable specifications and program transformation. In S.J. Greenspan, editor, *The 5th International Workshop On Software Specification And Design*, pages 287–290. IEEE Computer Society Press, Washington DC, United States of America, 1989.

[307] R. Took. Surface interaction a paradigm and model for the presentation level of applications and documents. In J.C. Chew and J. Whiteside, editors, *Proceedings Of The CHI'90 Conference On Human Factors In Computing Systems*, pages 35–42. ACM, New York, United States of America, 1990.

[308] R. Took. Integrating inheritance and composition in an objective presentation model for multiple media. In F.H. Post and W. Barth, editors, *EUROGRAPHICS '91*, pages 291–303. Elsevier Science Publications, North Holland, Netherlands, 1991.

[309] I. Traynor and M. Tran. Pressure grows on Boeing over Thai crash. *The Guardian*, page 22, 3 June 1991.

[310] I.G. Umbers. Models of the process operator. *The International Journal of Man-Machine Studies*, 11(2):263–284, 1979.

[311] Union Carbide Incident Investigation Team, Union Carbide Corporation. *Bhopal Methyl Isocyanate Incident Investigation Team Report*, Danbury, United States of America, 1985.

[312] United States' Nuclear Regulatory Commission, National Technical Information Service. *Guidelines For Control Room Design Review*, NUREG-0700, Springfield, United States Of America, 1981.

[313] W.E. Vesely. The fault tree handbook. Technical Report USNRC NUREG 0492, United States' Nuclear Regulatory Commission, Washington DC, United States of America, January 1981.

[314] W.A. Wagenaar and J. Groeneweg. Accidents at sea : Multiple causes and impossible consequences. In E. Hollnagel, G. Mancini, and D.D. Woods, editors, *Cognitive Engineering In Complex Dynamic Worlds*, pages 133 – 144. Academic Press, London, United Kingdom, 1988.

[315] R.W. Wardell. An ergonomics perspective on safety in the oilfield. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 999–1003. Human Factors Society, Santa Monica, United States of America, 1989.

[316] I.A. Watson. Review of human factors in reliability and risk assessment. In J. Burgoyne, editor, *The Assessment And Control Of Major Hazards*, pages 323–337. Pergamon Press, Oxford, United Kingdom, 1985.

[317] D. Whitfield. HCI in nuclear safety. *Usability Now!*, 1(2):7–8, 1990.

[318] C.D. Wickens. *Engineering Psychology And Human Performance*. C.E. Merrill Publishing Company, London, United Kingdom, 1984.

[319] E.L. Wiener. Cockpit automation. In E.L. Wiener and D.C. Nagel, editors, *Human Factors In Aviation*, pages 433–461. Academic Press, London, United Kingdom, 1988.

[320] E.L. Wiener and R.E. Curry. Flight deck automation : Promises and problems. *Ergonomics*, 23(10):995 – 1011, 1980.

[321] E.L. Wiener and D.C. Nagel, editors. *Human Factors In Aviation*. Academic Press, London, United Kingdom, 1988.

[322] T. Wilkie and S. Watts. Sellafield safety computer fails. *The Independent*, page 1, 24th November 1991.

[323] R.C. Williges, B.H. Williges, and J. Elkerton. Software interface design. In G. Salvendy, editor, *The Handbook Of Human Factors*, pages 1416–1449. Wiley, New York, United States of America, 1987.

[324] S.E. Wilson. The design and implementation of an interactive graphics system with natural language interface, for visually handicapped users. Technical report, Department of Computer Science, University of York, York, United Kingdom, 1991. M.Sc. Thesis.

[325] R. Winner, J. Pennel, H. Bertrand, and M. Slusarczuk. The role of concurrent engineering in weapons system acquisition. Technical Report R-338, United States' Institute Of Defense, Washington DC, United States of America, 1990.

[326] T. Winograd and F. Flores. *Understanding Computers And Cognition*. Addison-Wesley, Reading, United States of America, 1987.

[327] M.S. Wolgater, S.S. Godfrey, G.A. Fontenells, D.R. Desaulniers, P.R. Rothstein, and K.R. Laughery. Effectiveness of warnings. *Human Factors*, 29(5):599–612, 1987.

[328] J. Woodcock and M. Loomes. *Software Engineering Mathematics*. Pitman, London, United Kingdom, 1989.

[329] D.D. Woods. Coping with complexity: The psychology of human behaviour in complex systems. In L.P. Goodstein, H.B. Anderson, and S.E. Olsen, editors, *Task, Errors and Mental Models*, pages 128 – 148. Taylor and Francis, London, United Kingdom, 1988.

[330] D.D. Woods, J.A. Wise, and L.F. Hanes. An evaluation of nuclear power plant safety parameter display systems. In R.C. Sugarman, editor, *Proceedings Of The 25th Annual Meeting Of The Human Factors Society*, pages 110–114. Human Factors Society, Santa Monica, United States of America, 1981.

[331] N. Worley and J. Lewins, editors. *The Chernobyl Accident And Its Implications For The United Kingdom - Report Number 19 Of The Watt Committee on Energy*. Elsevier Applied Science, London, United Kingdom, 1988.

[332] Y.Y. Yeh and C.D. Wickens. The dissociation of subjective measures of mental workload and performance. Technical Report EPL-84-2/NASA-84-2, The Engineering-Psychology Laboratory, Department of Psychology, University of Illinois, Urbana-Champaign, United States of America, 1984.

[333] W. Zachary. A context-based model of attention in computer-human interaction domains. In D.L. Pettigrew, editor, *Proceedings Of The 33rd Annual Meeting Of The Human Factors Society*, pages 286–290. Human Factors Society, Santa Monica, United States of America, 1989.