Observations on solving discrete-event control problems: patterns and strategies

Technical Report No. 2009-558

Lenko Grigorov

grigorov@cs.queensu.ca
School of Computing
Queen's University, Kingston, Canada

Abstract

An observational study of solving discrete-event supervisory control problems is described. Five graduate students with knowledge of discrete-event system control were asked to think aloud while solving two supervisory control problems. The problem-solving sessions were recorded using audio and video equipment, and subsequently analysed using protocol analysis. Patterns and strategies in solving discrete-event supervisory control problems are identified and described, such as the role of control specifications and supervisors, the selection of events and the significance of controllability, the representation of discrete-event system elements and relations in the interface of control software, the flexibility necessary for different styles of problem solving, and the techniques for verification of solutions. The implications for teaching discrete-event systems and for designing discrete-event software interfaces are discussed.

April 27, 2009

Contents

1 Introduction 3

2 Literature overview 5

3 Description of study 9

3.1 Thinking aloud and protocol analysis 9
3.2 Theoretical foundation of discrete-event system control 11
3.3 IDES software application 13
3.4 Observational study setup 14
3.5 Data encoding 18
    3.5.1 Typology 18
    3.5.2 Flowcharts 20
3.6 N-gram analysis 23
3.7 Tracking of attention 23
3.8 Reference solutions 24
    3.8.1 Factory problem 24
    3.8.2 Hospital problem 25

4 Individual performances 28

4.1 Subject 1 28
    4.1.1 Factory problem 28
    4.1.2 Hospital problem 31
    4.1.3 Within-subject analysis 34
4.2 Subject 2 38
    4.2.1 Factory problem 39
    4.2.2 Hospital problem 42
    4.2.3 Within-subject analysis 45
4.3 Subject 3 50
    4.3.1 Factory problem 51
    4.3.2 Hospital problem 54
    4.3.3 Within-subject analysis 57
4.4 Subject 4 63
    4.4.1 Factory problem 63

4.4.2 Hospital problem 67
4.4.3 Within-subject analysis 71
4.5 Subject 5 74
4.5.1 Factory problem 74
4.5.2 Hospital problem 79
4.5.3 Within-subject analysis 84

5 Interviews with experts 91
5.1 Expert 1 91
5.2 Expert 2 93

6 Comparative analysis 95
6.1 Measure of progress in DES problem solving 95
6.2 Rate of errors in incomplete DES problems, perceived mistakes 97
6.3 Factory problem 98
6.4 Hospital problem 103
6.5 Overall observations 108

7 Discussion 118
7.1 Global strategies 118
7.2 Local strategies 121
7.3 Human factors 122
7.4 Events 123
7.5 Control specifications 127
7.6 Supervisors 129
7.7 Computations 130
7.8 Verification 132
7.9 Low-level modelling 135
7.10 Software for DES 136

8 Conclusions 138

Appendices 140

A Problem definitions 140
A.1 Problem 1: Factory problem 140
A.2 Problem 2: Hospital problem 141

B Encoding typology 142

C Flowcharts of problem-solving strategies 150
C.1 Factory problem 150
C.2 Hospital problem 154

Chapter 1

Introduction

This work describes a set of observations of the processes in solving problems in the field of control of Discrete-Event Systems (DESs). It is a part of a larger study aimed at the improvement of the methodologies and the software used in the area of DES control. The main goal of this part of the study is to collect the information needed to guide the proposal of modifications to our DES software [51].

To this end, we decided to perform an observational study of subjects solving a set of DES control problems and then apply mixed qualitative and quantitative analysis within a pragmatic framework of investigation [26, 15]. The investigation is focused on the following questions:

- Is there a difference between solving DES control problems using pen and paper and using a computer? What is this difference? What is the preference of subjects?
- What are the strategies applied in solving such problems? Is there a difference between the strategies applied by experts and by novices?
- What information is produced and consumed by subjects throughout the process of problem solving?

Contemporary DES software offers a range of improvements over its first generation. The most notable advantage is the implementation of a graphical interface which allows direct manipulation of DES models using a pointing device. However, the experience of the authors indicates that the user experience with the software may benefit significantly if other areas are improved as well, potentially leading to an improvement to the overall process of problem solving. Thus, the current investigation is further guided by an *a priori* decision to explore if and how DES software can be bettered in the following aspects:

- Conceptual modeling
- Analogical problem-solving
- Information availability
- Appropriate information visualization for DESs

Finally, this study was conducted to also collect some information to be used later, in the evaluation of the new features of our DES software implementation. The evaluation will examine efficiency and correctness among other things. In order to be able to compare the results, we needed to develop measures of progress towards the solution of a DES problem, as well as consider what constitutes an error in the problem solving in order to measure the rate.

The main contribution of this work is in the establishment of a better understanding of how non-naïve people solve problems in control of DES. The results can be used by developers for the improvement of DES software or by instructors for the design of DES control courses which emphasize good problem-solving strategies. More specific contributions include

- Systematic examination of problem-solving data in the field of DES control
- Development of a typology used to encode such data
- Comparative analysis of performance across subjects and across problem types
- Determination of the most common strategies used in solving DES problems
- Proposal of measures of progress and error rate in DES problem solving

In the rest of this work we provide a brief overview of related literature (Chapter 2), describe the methods used in the setup of the observational study and the data encoding (Chapter 3), and the performance of the individual subjects, including a within-subject comparison (Chapter 4). Subsequently, we make a between-subject comparative analysis (Chapter 6) and we conclude with a discussion of our findings (Chapter 7).

Chapter 2

Literature overview

In the field of Discrete-Event System control there has been a long-recognized need for better software tools [6]. Software tools are especially relevant to this field since usually it is computationally infeasible to perform the necessary computations by hand. In most DES software, the central role is assumed by the implementation of different computational algorithms; such is the case with the TCT [9] and UMDES [62] tools, for example. However, the sole implementation of algorithms proves to be insufficient to aid in real-world problem solving. In [16], the authors evaluate the elementary interface of the TCT software and determine that its usability is very low, despite the excellent implementation of the DES algorithms. Similarly, the development of the next generation, graphical DES software has been driven by the need of users to visualize better the models and operations they work with. For example, the Desco software [20] and later the Supremica software [2] were developed trying to address issues surrounding the application of the DES supervision for the control of example systems. Such issues include the implementation of DES supervisors in Programmable Logic Controller (PLC) code for the control of real hardware and the simulation of control in software.

The IDES software developed at Rudie's research laboratory, [51], also offers a graphical user interface. One of the main goals set at its conception was to center the design of the software around usability. Thus, the development of the tool has been more balanced, where functionality has been introduced at a slower rate compared to other tools (e.g., at the time of writing IDES still does not offer a full array of DES operations) but each feature in the interface has been carefully reviewed. However, it seems that merely providing a graphical environment is insufficient in resolving all problems with the application of DES theory. The modelling, even when done graphically, is still much too sensitive to errors. Even a single error in one of the models may render the whole solution of a control problem incorrect. Adding to this complication, in most cases the solutions to problems are too large to be comprehended in their entirety, and thus verification becomes very hard. Lastly, even if a correct (or desired) solution is obtained, due to the specificity of models and events, it is not simple to reuse the solution in another project. This makes the application of DES control very difficult for humans, even if all underlying functionality is implemented. In order to resolve these issues, we felt it is necessary to investigate in more depth the origins of the difficulties humans experience when working on DES control problems.

Human problem-solving has long been of interest to researchers, especially psychologists. One of the first more rigorous investigations of problem solving is described in [13] where the use of analogies in solving problems was examined. The contemporary investigation of problem solving is done within the field of cognitive psychology, starting with the seminal work of Newell and Simon on general problem solving [39]. Much of such research revolves around the attempt to derive a computational model of the human cognitive processes and to investigate the validity of specific hypotheses about the human cognitive faculties. The results are used to incrementally build and refine our understanding of the cognitive phenomena. For example, in [58, 48], the authors propose a computational model for the use of multiple representations in problem solving (with focus on visual reasoning). The performance of the model is described, but this work touches on problem solving only to discuss the use of multiple representations in the process. Besides the investigation of solving strategies for general and visual problems, research has also looked into the human cognitive performance in expert fields. An empirical study of the use of diagrams in the construction/modification of mental models is described in [44]. Subjects were asked to read Einstein's work on relativity and derive some of the equations. All produced records (drawings, gestures and speech) were examined to interpret the subjects' performance. Even though this work focuses on the use of diagrams and mental imagery, it also provides insight into the use of observational studies to study problem solving in general. Problem solving in Economics and Physics is also discussed in [59]. The experiments described investigate the interpretation of visual representations of data. The diagrams come from Economics (supply-and-demand) and Physics (reel with rope, car motion). The conclusions of the study are interesting, however, there seems to be a lack of an overarching hypothesis/theme and the work offers only cursory discussion of the global process of problem solving in the examined fields. The cognitive activities of computer programmers are studied in [24], with the goal of developing a computational model to simulate the memory of programmers. The subjects are asked to memorize blocks of code under different conditions and their recall rates are examined. Unfortunately, this fails to provide any insight into the high-level process of programming.

Investigation of problem solving and human factors in the field of DES control, to our best knowledge, has not been done. The publications closest to this topic pertain to the teaching of DES to students and the design of DES software. In [21], for example, a graduate course in DES control is described. The article describes the topics covered by the course and the software used by the students, however, it does not discuss the relative difficulty of the topics as experienced by the students or the points of the material which the students consistently had problems understanding. In [2], the authors mention on a number of occasions that the design of their software has been informed by their experience with teaching DES theory. Unfortunately, again, they do not elaborate on their observations and instead only list the features of the software. In [68], the author draws attention to some important aspects of designing and applying DES control. For example, how one constructs the event set to be used in modelling has a significant impact on how it is possible to reason about the problem later on. Furthermore, the work demonstrates that solving a DES problem may involve a number of iterations where not only the problem influences the solution (as expected), but the solution influences the problem as well. This effect seems not to be unique to DES control problems. In [55], the example of looking for a house to buy is given, where the final solution (the chosen house) may not at all represent a solution to the initial problem (what house one looks for). However, these insights do not provide enough information on the DES problem-solving strategies either.

It seems that, at this point, it is not possible to study DES problem-solving using an approach similar to the approaches generally used in the study of cognitive problem-solving processes. In particular, one needs to have some preliminary information which will be used for the formation of hypotheses. However, DES problem-solving is not a topic about which much is known. Furthermore, DES problems are ill-structured, [54], where there is only limited understanding of all the possible ways a problem can be solved and it is hard to define an explicit evaluation procedure for the possible solutions. Thus, we decided instead to do an observational study and examine it according to the methodology of qualitative research, [43], where the goal is to uncover the patterns in a phenomenon through observation and reasoning. The work of Rogers *et al.* on the performance of experts in radiological diagnosis, [50, 49, 46, 47], served as an inspiration. The authors describe a study of X-ray cancer diagnosis by experts and novices for the purpose of designing a computer interface to support the diagnosis process. The study employs the protocol analysis proposed in [17]. It starts with an observation of the work of diagnosticians to collect the information necessary to form a more focused observational study. In the second study, in a number of trials, diagnosticians are asked to diagnose different cases and to concurrently comment verbally on their activities. Then, the verbal reports are encoded and analysed to discover the patterns of actions and the information which is used to make a diagnosis. The researchers then proceed to create a model of the cognitive processes involved in the diagnosis. This model is used to inform the design of the computer interface. Finally, the performance of subjects with the computer interface is evaluated. Another inspirational investigation is the work of Guindon on understanding high-level software design [28]. Using protocol analysis, the author investigates different aspects of problem solving—the exploration of the problem domain, the elaboration of the requirements, the use of design strategies and notations, the application of problem solving schemata and heuristics, and the selection of evaluation criteria. The discoveries are then used to formulate a list of recommendations for software tools made to support high-level software design, e.g., such tools should support the exploration of alternative designs. This work shows that it is possible to employ qualitative analysis of ill-structured, or not well understood, problems to a beneficial end. The recommendations are directly linked, and derived, from the observations in the study. Of course, the use of protocol analysis is not limited to qualitative studies. For example, in [3], the authors use a quantitative approach to compare the problem-solving performance of freshman and senior engineering students. The subjects are given the task of designing a playground and asked to think aloud during problem solving. The protocols are then analysed in terms of steps, activities, objects, and information involved by the subjects during problem solving. All of these are quantified and then correlated with the quality of the corresponding solutions. Similarly, Gero and Mc Neill, [23], use verbal protocol analysis to compare the performance of three engineers when working on electronics and mechanical designs. Each section of the protocols is encoded along different dimensions (e.g., the micro strategy used by a subject at a given point). Then, the data are plotted against time to create graphs of the evolution of the problem solving sessions. It is then argued that the graphs can be used to distinguish between different types of problem solvers (e.g., experts vs. beginners). One main drawback of such quantitative studies is that not much insight is given into the actual processes involved in problem solving. The results are used mostly to allow the numerical comparison between different problem-solving sessions. In our study, we used mainly a qualitative approach, however, quantitative analysis of the performance of the subjects was also employed.

Chapter 3

Description of study

In order to collect data relevant in the context of our bigger project, we decided to perform an exploratory observational study of how subjects solve DES control problems. This decision was based largely on two considerations. First, we could not find in the literature a description of a previous investigation of this topic. Without any preliminary information, it is impossible to design good experimental setups. Second, the nature of our investigation requires the building of a holistic understanding of the process of problem solving. Instead of employing quantitative analysis of isolated aspects of subjects' activities, we decided to use qualitative investigation and try to discover the general trends in the problem-solving processes of the subjects. However, we also adopted an opportunistic approach during the analysis, where quantification was used when appropriate. As the intention of the observational study was to gain understanding of the thought processes of individuals, the approach proposed in [17] was considered.

3.1 Thinking aloud and protocol analysis

The scientific study of cognitive activities such as problem-solving is intrinsically difficult to perform as currently a feasible method to observe mental processes does not exist. In the past, psychology researchers have attempted to acquire information about thought through *introspective reporting* (described briefly in Chapter 1 of Gray's book on Psychology [25]). Specially trained subjects were asked to describe their thoughts as objectively as possible. However, gradually it became apparent that there are serious problems with the reliability of such reports and the lack of a base for validation. Thus, introspection as a method of inquiry has already been rejected by the scientific community. In order to study cognitive processes, Ericsson and Simon propose a new method, *protocol analysis* [17]. In this method, subjects are asked to think aloud while performing the tasks in a given experiment and then the collected information is analysed to uncover the patterns of their mental activities. As protocol analysis is grounded in theories of human cognition, it has become widely accepted as a method to study human cognitive processes. A list of studies employing this methodology can be found in [17]; see [22] for applications in medical research and [5] for applications in usability studies.

Indeed, thinking aloud shares a common base with introspective reporting in that the data collected is the subjects' verbal elaborations on their cognitive processes. However, this is perhaps the only similarity; there are significant differences between the two methods. The first significant difference, as explained in [18], is in the type of verbalization expected of the subjects. In thinking-aloud studies, subjects are not asked to elaborate on their thoughts. Instead, they are asked to simply voice their thoughts as they come. This does not require any training by the subjects, nor does it call for the use of specific terminology, or language not intuitive to the subject. Ericsson and Simon distinguish between three levels of elaboration when subjects think aloud [17]. Level 1 (least interference) is the simple vocalization of any verbalizable thoughts which occur naturally in the process of problem solving. Level 2 is when the subjects transform their inner thoughts into more understandable verbalizations, however, without elaborating. Level 3 (most interference) is when the subjects elaborate on and explain their cognitive processes. The authors argue that verbalizations of levels 1 and 2 provide valid insight into one's thinking as they do not cause one to break one's train of thought to attend to additional tasks such as elaborating on their current thoughts. The authors warn against the use of level 3 verbalizations as data since such verbalizations may influence the thought processes of the subjects. In fact, the authors point out in [18] that level 3 verbalizations often lead to an improvement of the problem-solving performance of subjects. The second significant difference between introspective reports and protocol analysis lies in the way the collected data is used. Unlike introspective reports, thinking-aloud protocols are not automatically considered to be indisputable. Instead, they are used solely as indicators, helping in the interpretation of the behavior manifested by the subjects.

Ericsson and Simon provide detailed instructions on how to perform a think aloud study in order to collect valid data [17]. However, their instructions on how to perform the analysis of the resulting protocols are not as specific. Instead of recommending a given method, the authors give examples of different ways to encode and analyse think aloud protocols. In general, the verbal data should be examined and a universal vocabulary of the terms and operations mentioned by the subjects has to be created. Then, the protocol has to be segmented into statements and each statement has to be encoded using the vocabulary. Finally, the encoded data are analysed according to the goals of the study. For example, the analysis may concern the number of occurrences of a certain operation, the sequence of items one attends to, the type of errors committed when reasoning, etc. In [22], the authors are more specific. They propose a three-step procedure for protocol analysis. In the first step, the *referring phrase analysis*, the analyst identifies the noun phrases used by the subject and builds the vocabulary for encoding. In the second step, the *assertion analysis*, the analyst identifies the assertions made by the subject about how concepts identified in the referring phrase analysis relate. In the third step, the *script analysis*, the analyst identifies operators that describe the predominant reasoning processes (such as "study", "choose", etc.) Then, patterns of cognition can be examined in terms of which concepts are referred to at different parts of reasoning and what relations are established between them. In [49], the authors describe an exploratory study of the cognitive activities involved in diagnosing X-ray images. The protocol analysis procedure, summarized below, is similar to the ones already discussed.

- Use prior information to define a taxonomy of the task;

- Refine the taxonomy from the observed data;
- Encode the observations as per the proposed taxonomy;
- Analyse the encoded form to find patterns and the interleaving of cognitive activities.

In our study we used think aloud protocol analysis as it appeared well-suited for our purposes, namely, to gain understanding of the subjects' thinking in problem-solving DES control problems. Our approach was also informed by previous applied research in understanding cognitive processes for the purpose of designing software [47, 28] and the description of the protocol analysis procedure the authors used [50, 49].

3.2 Theoretical foundation of discrete-event system control

The study described in this work investigates problem solving in the context of DES control, within the framework of Ramadge and Wonham [45]. Here, we provide a brief introduction to this framework.

Discrete-event systems (DESs) are mathematical constructs designed to represent the behavior of real systems at an abstract level. Instead of describing the continuous evolution of systems, e.g., the rate of filling a tank with liquid, the evolution is described as the occurrences of discrete events, e.g., the commencement of filling a tank with liquid and the completion of filling the tank. Events are assumed to be instantaneous and spontaneous, i.e., they describe the moment something important occurs and the advancement of time is irrelevant to the system—only the relative order of events, i.e., the sequencing, is important.

Customarily, such systems are modelled as finite-state automata (FSAs), [7], as such models are well-suited to describe sequences of events. Finite-state automata consist of a number of states and transitions between them. Transitions are labelled with events from an event set. An example FSA is shown in Fig. 3.1. One of the states is chosen to be initial (denoted with a small arrow in the figure). If one follows the transitions between states, starting in the initial state, one will end up with a sequence of events from these transitions. As the transitions one could take from each state are limited, only certain sequences can be constructed. For example, in the automaton from the figure, one can construct the sequences "encounter vending machine, insert coin, receive candy" or "encounter vending machine, walk on", however, the sequence "encounter vending machine, walk on, insert coin, receive candy" cannot be constructed. Some states can be further chosen to be "marked" (denoted with double circles in the figure). Event sequences leading to such states can be interpreted as desirable or important. They may signify the completion of some task in the system. For example, in the automaton from the figure, the sequence "encounter vending machine, walk on" leads to a marked state, while the sequence "encounter vending machine, insert coin, receive candy" does not. The first sequence signifies completion of one's interaction with the vending machine, while the second sequence does not—one could continue inserting coins and receiving candy.

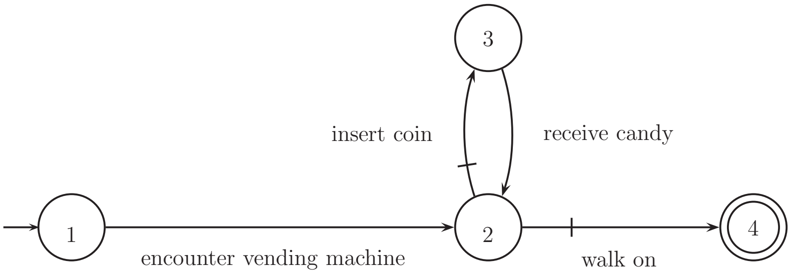


Figure 3.1: Simple example of a finite-state automaton. This model describes the interaction with a vending machine.

The *event set* of an FSA specifies all events which can be used in the model. Control of DES consists of the disablement of certain events at critical states. For example, one might want to prevent their kid from inserting more coins into a vending machine after they have already spent \$10. However, it is not always possible to prevent the occurrence of an event. For example, it is not possible to prevent the delivery of candy after the insertion of coins. Thus, the occurrence of only certain events can be controlled—and these events are called *controllable*. The other events are called *uncontrollable*. If the prevention of an uncontrollable event is desired, one needs to prevent a controllable event preceding the uncontrollable event in the executed event sequence.

The decision of which events to enable and disable to control a system is not very easy, especially when the system is complex. The benefit of the mathematical theory of DES control is that there exists an algorithm which can automatically generate the required control decisions [45]. As input, it takes a model of the system (in the form of an FSA) and a description of the control requirements (also in the form of an FSA, usually a restricted version of the system model). The algorithm outputs what is called a *supervisor*. It is an FSA model which, when used concurrently with the system model (i.e., via the *intersection* operation), contains information about which events should be disabled at which points in an event sequence. In other words, the supervisor defines the control decisions needed to ensure that the control requirements are met. It is important to note that not always all requirements are feasible. Sometimes it may not be possible to prevent the undesired evolution of a system unless large parts of the desired evolution are also prevented. In the worst case, the only way to prevent the execution of an undesirable sequence would be to block the system from operation (e.g., the only way to prevent a robot from breaking down is to disable the robot's operation). The main contribution of the DES algorithm for generation of supervisors is that the output is proved to be not only *correct*, i.e., it does not permit the execution of undesired sequences, but it is also *optimal* (or *most permissive*), i.e., there is no better way to control the system so that more desired sequences are allowed and all undesired sequences are prevented.

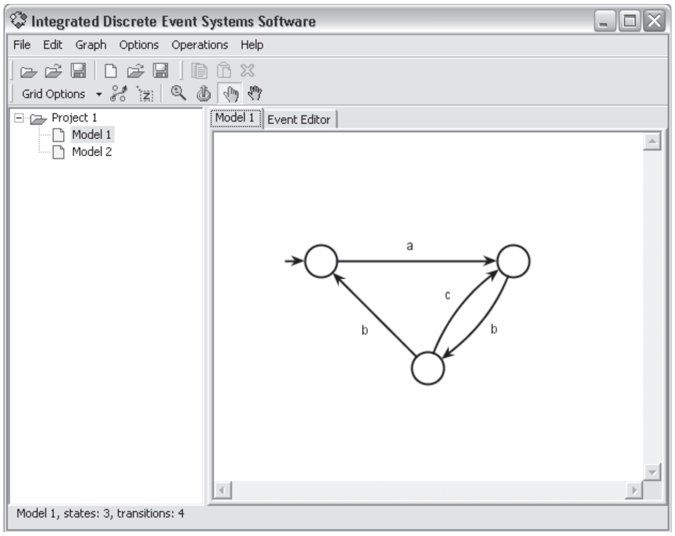
One of the main problems with the application of the DES control algorithm lies in the fact that the size of the model of a system grows exponentially with the number of components of the system. As a result, the standard (or *monolithic*) application of the theory is infeasible when non-trivial systems are considered. In many cases, *modular* supervision [67, 11] can be used to resolve this issue. Instead of creating a single, big model of the system and a single, big model for the control requirements, each system component and each control requirement can be modelled separately, as a module. The interaction between different parts of the system and the specifications is accomplished by using common events in the FSA models. The exponential growth of the model size is avoided when modular design is used. Separate, small supervisors can be generated automatically for each control specification using the original algorithm. The only disadvantage of the modular approach is that the separate supervisors may end up interfering with each other's control decisions, as they are generated independently. Such interference may lead to deadlock or livelock in the controlled system. In order to verify that supervisors do not interfere, one can use different approaches. Two of the most common ones are to compose all supervisors and examine the combined effect, or to use an algorithm to check if the supervisors are *non-conflicting*.

The theory of DES control has received numerous other contributions since its conception. However, we did not expect subjects in this study to use advanced approaches. The only two other approaches, mentioned tangentially by subjects involved *decentralized* control, [52], and *observability*, [35]. Decentralized control refers to cases where each supervisors may be able to control only a small part of the system (i.e., the controllability of events is different for different supervisors). Furthermore, in some cases not all events occurring in a system can be observed by a supervisor, thus, it may be necessary to consider also the observability of events when designing supervisors. Neither decentralized control, nor observability needed to be considered when solving the DES problems in this study.

3.3 IDES software application

During the observational study, subjects had access to DES software to help in their problem solving. The software package made available was IDES version 2. This software was developed at Karen Rudie's research laboratory at the Department of Electrical and Computer Engineering, Queen's University, Canada [31]. There were two reasons for the selection of this package. First, it belongs to the new generation of DES software with a graphical interface. Second, we plan to use IDES as the platform on which to implement the features or improvements which follow from the observations in this study. Thus, direct observations of the use of the software "in the field" were assumed to potentially provide useful feedback.

The IDES software application provides a graphical environment for the modelling of finite-state automata. As well, it includes a number of operations used in DES control, such as *parallel composition*, *intersection*, check for *controllability* or the algorithm for automatic generation of supervisors. For a further explanation of these algorithms, please refer to [7]. The overall interface is shown in Fig. 3.2. It consists of a panel listing all loaded models (see Fig. 3.3). By double-clicking on a model name, the user can activate this model and then edit it inside the graph panel (Fig. 3.4) or the events panel (Fig. 3.5). The graph panel displays finite-state automata graphically and employs a pen-and-paper drawing paradigm [51]. The user need not switch between tools to create states and transitions, as well transitions can be drawn either by clicking or dragging. The event panel lets users specify events to be used in the model. They can click on the "Create event" and "Delete event" buttons to create


Integrated Discrete Event Systems Software
File
Edit
Graph
Options
Operations
Help
Grid Options
Project 1
Model 1
Model 2
Model 1
Event Editor
a
c
b
b

Model 1, states: 3, transitions: 4

Figure 3.2: The user interface of IDES version 2.

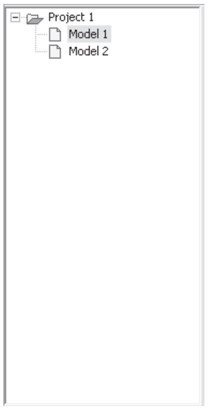

Project 1
Model 1
Model 2

Figure 3.3: The list of loaded models in IDES version 2.

and, respectively, delete events. They can rename the events by typing in the corresponding cells, as well as change the controllability and observability properties of events. Operations can be invoked on the models selected in the model list by selecting from the "Operations" menu (see Fig. 3.6).

The version of IDES used in the study was still under development and on occasions it exhibited some problems when used by the subjects. These occasions are described when the performance of the subjects is discussed.

3.4 Observational study setup

The study was set up so that we could observe and record the performance of subjects during the solution of DES problems.

We recruited five subjects in total. Each subject had at least one university semester of

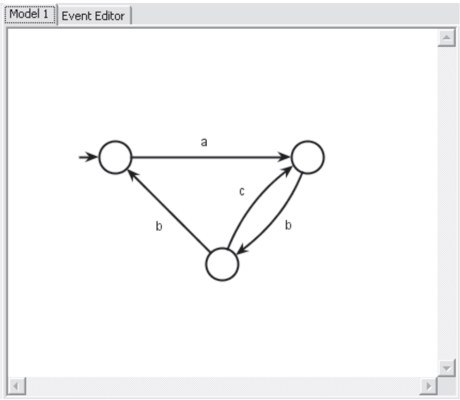
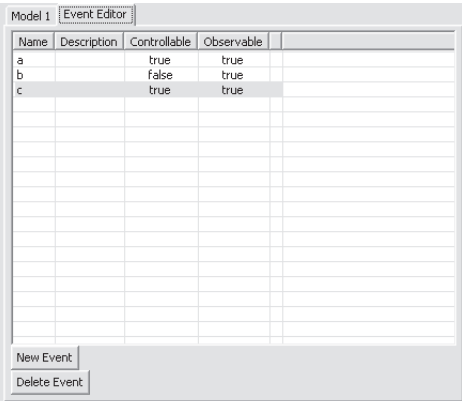

Model 1
Event Editor
a
c
b
b

Figure 3.4: The graph panel in IDES version 2. This is where models can be drawn using the mouse cursor.


Model 1
Event Editor

Name	Description	Controllable	Observable
a		true	true
b		false	true
c		true	true

New Event
Delete Event

Figure 3.5: The events panel in IDES version 2. This is where events can be manipulated.

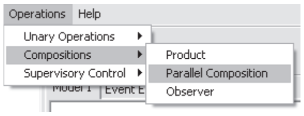

Operations
Help
Unary Operations
Compositions
Supervisory Control
Product
Parallel Composition
Observer
Model 1
Event E

Figure 3.6: The menu providing access to DES operations in IDES version 2.

exposure to supervisory control theory for DESs. All subjects had completed the graduate course, "ELEC843: Control of Discrete-Event Systems", offered by Dr. Rudie at the Department of Electrical and Computer Engineering at Queen's University, Kingston, Ontario, Canada. A more detailed profile of each subject is included before the discussion of their corresponding performances.

Each subject was administered two problems in the field of DES control. One of the problem is an adaptation of a classic problem in the field, the "Transfer Line" [66]. Each subject has seen this problem being solved in the process of instruction—this problem is used as an example in the ELEC843 course. The problem asks for the design of a controller for a system of factory machines and buffers between them. Besides guaranteeing the (rather trivial) lack of underflow and overflow in the buffers, one needs to also consider a more complicated situation where certain conditions in the processing by machines may cause blocking of the system (see Appendix A.1 for the complete statement of the problem). We will refer to this problem as the "Factory problem". The second problem, the "Hospital problem", was modelled after the first one, however, the statement of the problem was modified significantly, so as to hide its similarity. Instead of considering machines and buffers, the problem talks about a patient in a hospital and certain requirements are set on the intake of medication and on the processing of medical reports (see Appendix A.2 for the complete statement of the problem). The second problem was originally designed for our purposes and none of the subjects had seen it before being introduced to it in the study. The reason to use this particular pair of problems was to amplify the difference between "learned" (or mechanized [36]) problem solving and solving new problems, while keeping the fabric underlying the solutions similar to allow for comparison. While it was attempted to make the second problem as close to the first problem as possible, the two problems must not be viewed as completely equivalent. The informal description offers a number of valid interpretations. Furthermore, due to unfortunate wording, the description of the second problem implies that the reception of candy by the patient and eating the candy are two separate events, which makes the problem incompatible with the first problem. This issue was discovered only during the analysis of the performance of the subjects, when it was too late to make corrections. As a result, unless a subject interprets the description so that the reception of candy results in an increase of sugar in the blood of the patient, the solutions to the two problems become incomparable.

The problems were administered in the following fashion. Each subject participated in two separate sessions—one for each problem. Each session lasted about one hour and the subject had up to 50 minutes to solve the particular problem. If they did not complete the problem in this time, they were interrupted. In order to reduce the effect of the first session on the second, there was at least one week between the two sessions for each subject. The two problems were administered in random order (i.e., for some subjects, the Factory problem was first and for others, the Hospital problem was first).

The problems themselves imposed no specific methodology for problem solving. Subjects were instructed to produce a DES supervisory solution and no particular approach (e.g., monolithic or modular [67]) was recommended. Furthermore, subjects were provided with a pen, sufficient amount of paper, and a computer running the IDES software (see Section 3.3).

The problem description advised the use of pen and paper for the modelling, but mentioned that the use of the software is possible, especially for computationally demanding tasks. In practice, the subjects were free to use any tool in any fashion they desired and to switch between tools (i.e., between pen and paper and the computer) whenever they wished and as many times as desired. No restrictions were imposed during the observational sessions.

The subjects in the study were asked to *think aloud* while solving the problems in order to collect data for a subsequent protocol analysis (see Section 3.1). The recommendations for how to carry out a think aloud study, summarized in [5], were heeded whenever feasible.

We decided to let the subjects choose a comfortable level of elaboration, however, our instructions clearly indicated that the subjects are not required to provide any explanation or justification for any thought that they voice. Furthermore, each subject was introduced to thinking aloud and underwent a brief practice session before performing the actual task. This helps the subjects become more comfortable with the process of thinking-aloud and reduces the incurred interference. In our study we noticed that subjects would vary in their level of comfort with voicing thoughts but generally would "get accustomed" within five to ten minutes after the beginning of the session. The level of verbalization was normally what Ericsson and Simon classify as level 2, i.e., the subjects transformed their inner thoughts into more understandable verbalizations, however, without elaborating.

The experimenter was present throughout each session for three reasons: to remind subjects to voice their thought by saying "Keep talking", to observe their performance and gain subjective experience about it, and to provide administrative support such as directing the camera properly when subjects switch between using pen and paper and using software. In almost all cases the experimenter sat in front of the subjects to facilitate the personal observation of the subjects' performance. In one case, however, this proved to be too distracting to the subject as they repeatedly tried to engage in a conversation with the experimenter. In this case, the experimenter remained in the room but moved out of the subject's field of view. In general, as subjects got accustomed to thinking aloud, it was not necessary to encourage them to speak. One subject, however, was not very talkative and it was necessary to repeatedly remind them to keep talking. In that case, the experimenter attempted to subjectively judge when to encourage the subject so that there is not too much interference but, at the same time, data about their thinking is collected. The experimenter did not otherwise interact with the subjects with the exception of answering administrative questions such as which folder subjects should use to save their files when using IDES, or when trying to resolve a bug in the software which had rendered it unusable. Such interferences were duly noted when the data from the study was encoded.

At the end of each observational session, the subjects were asked to answer a few questions in an informal interview. The questions asked included

- Remember about what you thought and explain your approach?
- How confident are you in your solution so far?
- What would be your future steps, were you to continue solving this problem?
- Any comments?

as well as questions seeking to clarify aspects of the answers of the subjects to the above questions. At the end of the second session, each subject was asked to voice also what they think about the study. Potentially, it could be possible to learn if the subjects had realized that the two problems had comparable solutions.

In order to allow for more complete data analysis from this study, the performance of each subject was, consensually, video-taped. The video record (including the audio track), as well as all paper records and computer files produced by the subjects, were retained for analysis.

3.5 Data encoding

After all sessions of the observation study were completed, we proceeded with the encoding of the data. As we envisioned the occasional use of quantitative analysis to accompany the qualitative analysis of the observations, it was necessary to transform the data into a form suitable for automatic processing. As discussed in literature, human performance is influenced by, or happens along, two complementary paths generally characterized as "bottom-up" and "top-down" [38, 46, 25]. This served as an inspiration to approach the data encoding in a similar fashion. One the one hand, we encoded activities at a very low level, using typology developed specifically for the purpose. On the other hand, we encoded activities at a very high level, as hierarchical flowcharts of procedural steps. At the end, both encoding schemes were reconciled by synchronizing them along the dimension of time. This allows us to go back and forth between the low level and the high level and ask questions centered on either aspect. The encoding schemes are presented in more detail next.

3.5.1 Typology

The type of low level data we decided to focus on was determined by the type of questions we had in mind, as well as best practices, i.e., to have a versatile encoding scheme so that initially unforeseen questions will be answerable as the research advances (proposed in private communication with Kathleen Norman, School of Rehabilitation Therapy, Queen's University, Canada). The questions we were interested in concerned mostly two aspects of the problem solving:

1. The fluctuation of the subject's attention and
2. The cognitive processes of the subject.

To this end, we developed an encoding scheme advised by previous work [46, 42] and Chapter 8 of [43]. The first version of our scheme is briefly described in [27]. This version was not, unfortunately, suitable for the encoding of the full range of information that was needed. However, it was very instructive in the design of encoding schemes and helped us develop a much better scheme which is described next.

The essential view that we took was that the low-level performance of subjects consisted of a series of events. Each event represents a specific activity and it has a time stamp of when exactly it occurred relative to the start of problem solving. Three different "streams" of events were encoded:

Visual attention This stream consists of events which describe the shifts of the visual attention of the subject, such as "subject shifts gaze to the computer display" or "subject shifts gaze to the sheet with the problem description".

Physical activity This stream consists of events which describe the physical activities performed by the subject, such as "create an explanatory note on the sheet with the problem description" or "draw a transition between two states in a model". Actions unrelated to problem solving were not encoded, e.g., subject relocating between desk and computer, issuing commands to save files or create projects in IDES, etc.

Verbalization This stream consists of events which describe what the subject verbalizes, such as "subject talks about their intention to model a specific subsystem" or "subject counts the number of states in a model".

In order to encode these events, typology was designed to assign a specific code to each type of event. Each code is in the form 'HT(P)', where the prefix 'H' is a letter indicating a general type of event, the suffix 'T' is a string of one or more letters specifying the type of event further and 'P' is a comma-separated list of parameters. Parameters are used only when relevant. An example of a code is 'CMN(M1)'. Here 'C' stands for a physical activity performed while using the software (computer), 'MN' stands for 'DES module naming' and the parameter 'M1' encodes the fact that the module being named is the module for 'Machine 1' (in the factory problem). The only exception in the format of the code is when certain events in the "verbalization" stream are encoded. Since human expression is very rich, it was determined that sometimes "extended" codes need to be used to capture what the subject says more fully. Thus, when encoding verbalizations, the code may consist of a number of regular codes concatenated with dashes. For example, 'XGY-D(DR)' encodes that the subject voiced their thought (X) considering (Y) their plan (G) in terms of the dynamics (D) of the DES module for the doctor (in the hospital problem). The full encoding typology is described in Appendix B.

Besides developing typology and applying it to encode single events, it is necessary also to record the time when events occur. For this purpose, time stamps were additionally associated with each event to specify when it occurred relative to the start of the observational session. For example, if the event "the subject looks at the computer display" happens fifteen minutes, twenty one seconds and three hundred milliseconds after the start of the observational session, the time stamp used with the code for this event will be "15:21,300". The software used to collect the time stamps, [56], does not guarantee millisecond precision of positioning in the video stream. Thus, milliseconds were used mostly to compute the relative time between very close events, if necessary. The time stamps were collected for the onsets of events.

The encoding itself was carried out by a single person. The resources available for the study, as well as the priorities we had, did not allow for hiring more encoders and then cross-checking to increase the reliability of the encoding scheme.

3.5.2 Flowcharts

Even though it is very valuable to have the low-level data from the observational study, we discovered that, in by itself, such data is insufficient to guide our inquiry. The most important aspect that is missing from low-level data is the overarching *intention* (or pursuit of a goal) in performing a set of actions. It is not possible to say what purpose the low-level actions serve and where the boundary between different goals lies.

In order to understand the intentionality of the subjects' low-level action, and to learn what strategies of problem-solving they employ, we used qualitative analysis. However, in light of our desire to have additional quantitative support, we decided to further encode the observational data in terms of the inferred high-level activities of the subjects. To this end, we decided to employ process diagrams in the form of hierarchical flowcharts, similar to what is used in business and software development [33, 41].

The syntax of the flowcharts we used were based on the *Activity* diagrams defined in UML [41]. However, we did not purposefully attribute any of the UML semantics to the way we used the notation. In our case, *actions* (or boxes) were used to denote a stage of the problem solving of a subject, e.g., "modelling modules", "inputting module 'TU'" or "verifying model". *Transitions* (or arrows between boxes) were used to signify the problem-solving path which a subject follows. *Decisions* (or diamonds, choice points) were used to denote the places where the subject may choose to proceed along two or more alternative paths. The same graphical notation, following the convention in UML, was used also to denote when two or more alternative paths merge back together (the proper UML term for this element is *merge*). Actions (boxes) were considered hierarchical so as to allow the refinement of high-level activities. For example, the box "model modules" may be refined to a flowchart where the subject follows a path through the boxes "model M1", "model M2" and "model TU". *Initial* and *final states* were used to denote the entry and exit points, respectively, in a lower-level (more detailed) diagram. Final states were not used when the encoded high-level task was incomplete during the study, e.g., was interrupted due to the expiry of a session. Examples of hierarchical flowcharts of the type we used can be found in Appendix C.

There were three main goals we had in mind when we started encoding data with high-level flowcharts.

1. The high-level data should make the subjects' strategies more explicit.

2. The flowcharts should "normalize" the data to some extent to allow direct comparisons between subjects.

3. It should be possible to reconcile the high-level encoding with the low-level encoding.

In order to address the first two goals, we decided to use as a base the preliminary taxonomy of problem solving in DES control as described in [27]. Indeed, this taxonomy is very incomplete and has not been validated. However, it was sufficient for our research since our observations are of exploratory nature. Furthermore, we used only the top-level elements in the taxonomy, i.e., the elements which are least likely to fail validation. For our purpose, we encoded the following problem-solving activities:

Understand the subject gains understanding of the problem and the existing situation,

Model the subject creates a (more or less) formal representation of their understanding of a part of the problem,

Input the subject inputs into the software a model which they have created on paper,

Compute the subject invokes the algorithms in the software to obtain a model of a part of the problem,

Verify the subject uses any of the available means to confirm that a part of their solution is correct.

This is more or less in line with the Execution-Evaluation Cycle proposed by Norman in [40].

The encoding of the data proceeded in three stages: subject-specific encoding, generalization, and normalized encoding. In the first stage, the video record of each problem-solving session was observed, and a flowchart with the performance of each subject was created. The encoding was subject-specific. There were no specific criteria for the encoding; instead, all available information (video, diagrams, computer files...) was used by the encoder to intuitively determine what high-level activity the subject is engaged in at any moment. Thus, the process was comparable to what any (knowledgeable) observer would perceive in an everyday setting. At the end of this stage, we obtained ten, subject-specific flowcharts (five subjects solving two problems). These flowcharts had only marginal commonalities and even comparison of the strategies employed by a single subject for the two problems seemed difficult—let alone any comparisons between subjects. This problem was addressed with the second and third stages of encoding. The second stage, generalization, consisted of building two universal flowcharts—one for each problem—in which all problem-solving paths were mapped. The encoder examined all subject-specific flowcharts for a given problem and attempted to recognize general patterns that emerge—and then encode them in the generalized flowchart. The process involved a mix of two approaches. On the one hand, the encoder examined the subject-specific flowcharts, formed a generalization, and amended the universal flowchart. On the other hand, subject-specific data was elevated to the universal flowchart and, as more subject-specific data was amalgamated into the flowchart, a generalized pattern emerged. At the end, two flowcharts were obtained, in which it was possible to trace, with approximation, the problem-solving paths of all subjects. The generalized flowcharts are shown in Appendix C. Unlike the subject-specific flowcharts, the generalized flowcharts contain branching and merging. Since different subjects had different strategies, different paths have to be taken in the generalized flowcharts. However, with the given merging, the generalized flowcharts indicate that there may be many more potential problem-solving strategies which none of the subjects exhibited. The last stage was the most important to allow between-subject comparisons. Here, the performance of individual subjects was encoded using "normalized" flowcharts. In essence, this stage consisted of selecting the specific problem-solving path that each subject takes within the context of the generalized flowchart. Thus, ten flowcharts were obtained which correspond to the ten subject-specific charts from the first stage. However, the flowchart elements are selected from the generalized flowchart for the corresponding problem. Thus, if two subjects performed the same problem-solving step, they would share the same flowchart element in their normalized flowcharts. As a result, the direct comparison of strategies of different subjects became possible. The normalized flowcharts are shown when the subject-specific performances are discussed (Chapter 4). Similar to the initial flowcharts, there is no branching since each subject took only one specific problem-solving path. One of the main benefits of these normalized flowcharts is that, by reducing and unifying the encoded elements, it becomes possible to perceive more easily what a subject's problem-solving strategy is.

The third goal we had in mind when encoding the high-level data was to allow for some reconciliation with the low-level data. The most obvious way to achieve this was to map low-level events to the high-level "blocks" of activity. For each element of a flowchart, timestamps, relative to the video recording, were used to mark the beginning and end of the element. For example, if the subject was modelling the 'DR' module from minute 15:06 to minute 18:43 of the video record, the block "Model DR" in the corresponding normalized flowchart was marked with the timestamps "15:06,000" as the beginning of activity and "18:43,000" as the end of activity. Then, all low-level events with timestamps falling in this interval were mapped onto the "Model DR" block. In this way, it became possible to answer both the question "What specifically was the subject performing while modelling 'DR'?" and the question "In which part of the subject's problem-solving approach did they say that the modules 'M1' and 'M2' are similar?" As the low-level events are the only relevant aspects of the performance of subjects, it was sufficient to choose only approximate chronological boundaries for the blocks in the flowcharts—enough to include the proper low-level events. In this way, we sidestepped the controversial issue of defining precise points in time when a subject transitions from one activity to another.

After all encoding was completed, we proceeded with the analysis of the problem solving as manifested by the subjects. The data we had consisted of:

- Video footage of the problem-solving sessions
- Sheets of paper which the subjects used
- Computer files with the models the subjects input into or generated with the software
- Short interviews
- Subject profiles
- Lists of encoded low-level events, timestamped
- Generalized flowcharts of potential problem-solving strategies
- Flowcharts for every subject specifying their strategies in a comparable way
- Mapping of the low-level events into the high-level flowcharts

The n-gram analysis used for the low-level data, the methods used to track attention, as well as the reference solutions for the two problems are described next.

3.6 N-gram analysis

An important goal of this study was to obtain information to guide the improvement of the interface of DES software. Usually it is not difficult to design software interfaces suitable for work within a particular context (or mode of operation); it is harder, however, to design interfaces for multiple contexts. Thus, information regarding the workflow of users (how they change of contexts) is very important. For this reason, we decided to analyse the workflow of the subjects in the observational study. We considered the context of the physical actions from the low-level encoding. Five different contexts were recognized: module, state, transition, event and computation. These were denoted M, S, T, E and C, respectively, following the notation in the typology from Appendix B.

The sequence of contexts in the low-level physical activities for each session was analysed using n-grams. N-gram analysis [57] is the determination of the frequency of occurrence of a specific sub-sequence of n items in a larger sequence. For example, in the sequence "MSTESST", the 2-gram (or *bigram*) 'ST' occurs two times, while the 3-gram 'STE' occurs only once. In our study we computed both absolute and relative ratios of n-grams. An absolute ratio is the ratio of the number of occurrences of a given n-gram to the total number of n-grams in the sequence. We use the term "relative ratio" to refer to the ratio of the number of occurrences of an n-gram to the number of occurrences of all n-grams which start with the same $n-1$ items. For example, in the sequence "MSTESST", the relative ratio of the bigram 'ST' is 2/3 since there are two occurrences of 'ST', one occurrence of 'SS' and no other bigrams starting with 'S'. In comparison, the absolute ratio of 'ST' is 2/6 since 'ST' occurs twice and in the sequence there are six bigrams in total.

By using n-gram analysis, it was possible to determine how frequently the subjects worked on different elements of their models and what were the most common changes of contexts. Initially, we considered n-gram analysis for multiple values of 'n'. However, the ratio distributions for n$\geq$ 3 were uniform and did not provide any information. Thus, we only investigated the results of the 2-gram (or bigram) analysis.

3.7 Tracking of attention

The attention of the subjects throughout the problem solving was partially indicated by their verbalizations. As one of our goals is to discover what information is used to solve DES problems, we decided to track two types of events related to attention: perceptually-triggered data discovery and cognitively-driven data discovery. Perceptually-triggered data discovery refers to the event when the subject discovers some information relevant to the problem without actively seeking it, e.g., noticing that a model does not have an initial state. Cognitively-driven data discovery refers to the event when the subject purposefully decides to seek out some information relevant to the problem, e.g., deciding to check all models if they have properly defined initial states. The typology developed for the encoding of the low-level data (see Appendix B) provides convenient codes: 'XP' for perceptually-triggered data discovery and 'XD' and 'XHQ' for cognitively-driven data discovery. Whenever possible, these codes include the type of entity that the subject considers (e.g., states, dynamics, modules, etc.) and the specific DES module being considered.

Since the shift of attention was not always announced verbally by the subject, we also performed an analysis of the visual attention of the subject. More specifically, we took note of what sheets of paper the subjects looked at while working on a given model, or what other models in the software they looked at while using the program to work on a given model. Unfortunately, when analysing the visual attention directed at sheets of paper, it cannot be established with certainty which model the subject paid attention to specifically as there could be more than one model per sheet.

In each section describing the within-subject analysis of performance, the shifts of attention demonstrated by the corresponding subject during problem solving are discussed.

3.8 Reference solutions

The problems which the subjects in this study had to solve do not have a trivial solution, however, they can be categorized as very simple. Even so, the space of possible solutions is very large and only becomes larger when various interpretations of the problem description are considered. Nevertheless, drawing as a source on the solutions of problems in the ELEC843 course, it was possible to come up with reference solutions for the problems in this study. These solutions were used later on as a base of comparison when the solutions of the subjects were considered.

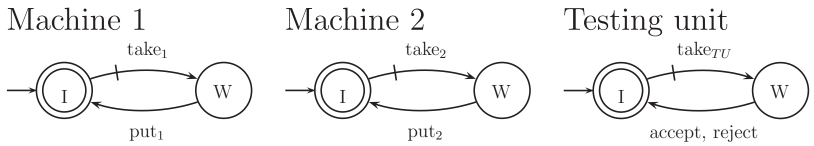
3.8.1 Factory problem

The factory problem is described in Appendix A.1. As already discussed, it is very similar to the "Transfer line" problem from the literature [66]. Thus, the reference solution follows closely the solution to the aforementioned problem.

The modelling of the system is modular, consisting of three modules (see Fig. 3.7). There are two models for the two machines and they are identical, save for the naming of the events. The model of the testing unit is also similar, however, the testing unit makes a choice between two actions after testing a part.

The buffers mentioned in the problem are not components of the system. In fact, they only impose limitations to the functioning of the system. Thus, the buffers comprise the control specifications. The modelling of the control specifications is also modular. There are two models for the two buffers (see Fig. 3.8). The first model counts the number of parts deposited in buffer 1. Whenever machine 1 outputs a part or the testing unit rejects a part, the number of parts increases. Conversely, whenever machine 2 takes a part, the number of parts decreases. In an analogous way, the second model counts the number of parts deposited in buffer 2. As it is necessary to explicitly mention at each state all events which need not be disabled, the specifications contain self-loops listing the events which are irrelevant for the control (e.g., the specification for buffer 2 contains also the events from machine 1 even though they are irrelevant for this buffer).

In the reference solution, the modules of the system are composed into a monolithic system by using the *synchronous product* operation. Similarly, the control specifications are



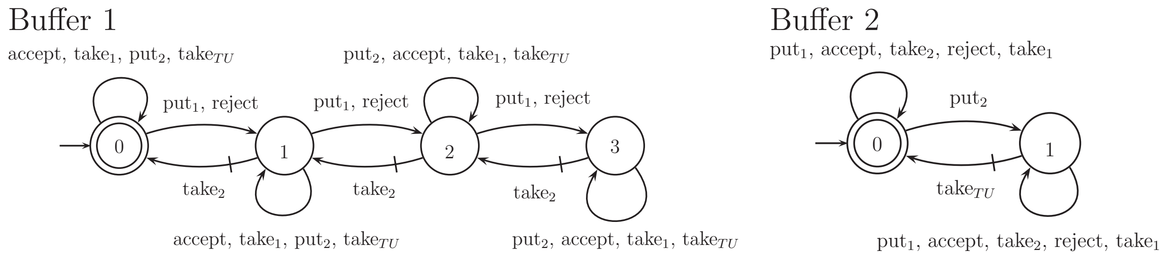
Event	Description
take$_1$	machine 1 takes a part for processing, from the environment
put$_1$	machine 1 outputs a part after processing it, into buffer 1
take$_2$	machine 2 takes a part for processing, from buffer 1
put$_2$	machine 2 outputs a part after processing it, into buffer 2
take$_{TU}$	testing unit takes a part for testing, from buffer 2
accept	testing unit outputs a part after a successful test, into the environment
reject	testing unit outputs a part after an unsuccessful test, into buffer 1

Figure 3.7: The models of the system components for the Factory problem and a description of the events used in the models.

composed into a monolithic model using the *intersection* operation. Finally, the monolithic models are input into the algorithm for automatic generation of supervisors to obtain the appropriate supervisor.

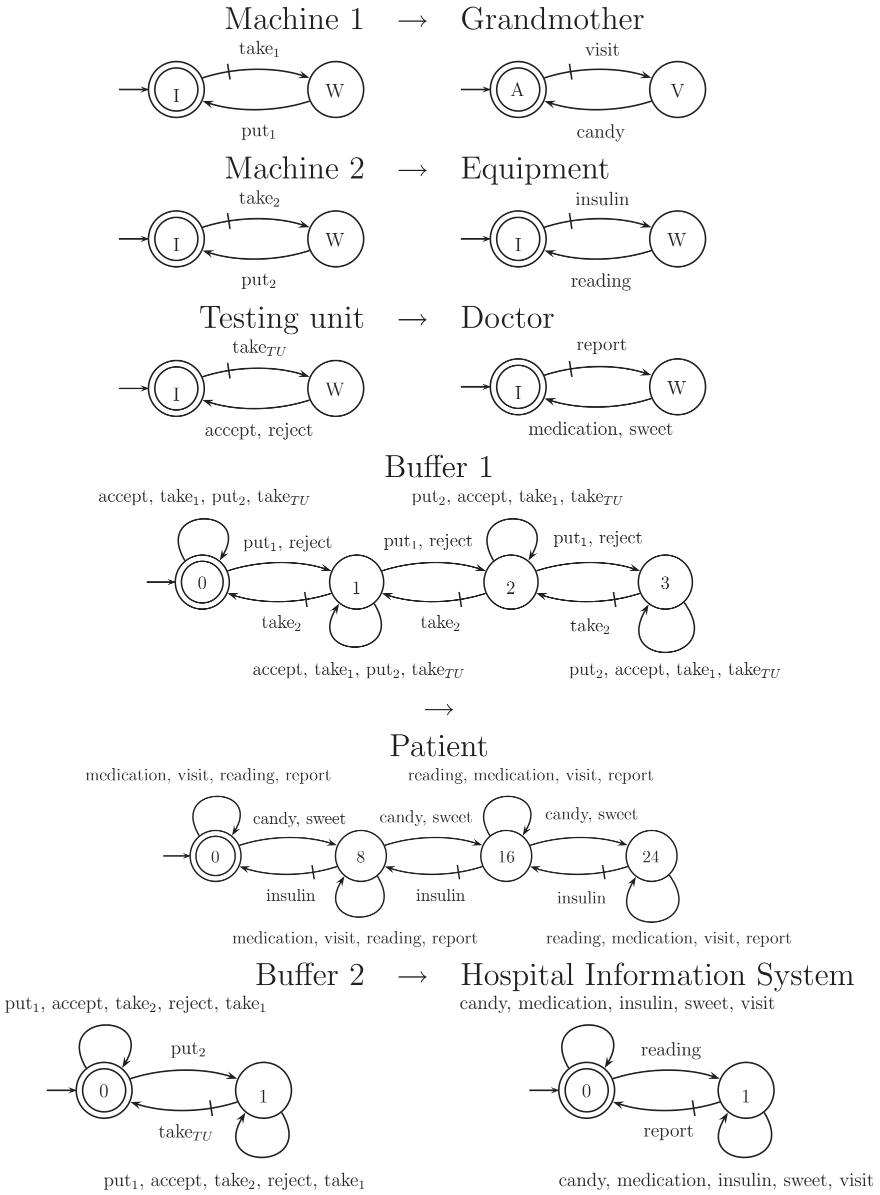
3.8.2 Hospital problem

The hospital problem is analogous to the factory problem. Thus, its reference solution resembles the reference solution of the factory problem. It is sufficient to cast components and events of the factory problem into the new setting. For example, the restrictions on the blood sugar level of the patient in the hospital problem are identical to the restrictions for buffer 1. In Fig. 3.9, the correspondence of the two solutions is demonstrated.


Buffer 1
accept, take$_1$, put$_2$, take$_{TU}$
put$_1$, reject
put$_1$, reject
put$_2$, accept, take$_1$, take$_{TU}$
put$_1$, reject
0
1
2
3
take$_2$
take$_2$
take$_2$
accept, take$_1$, put$_2$, take$_{TU}$
put$_2$, accept, take$_1$, take$_{TU}$
Buffer 2
put$_1$, accept, take$_2$, reject, take$_1$
put$_2$
0
1
take$_{TU}$
put$_1$, accept, take$_2$, reject, take$_1$

Event	Description
take$_1$	machine 1 takes a part for processing, from the environment
put$_1$	machine 1 outputs a part after processing it, into buffer 1
take$_2$	machine 2 takes a part for processing, from buffer 1
put$_2$	machine 2 outputs a part after processing it, into buffer 2
take$_{TU}$	testing unit takes a part for testing, from buffer 2
accept	testing unit outputs a part after a successful test, into the environment
reject	testing unit outputs a part after an unsuccessful test, into buffer 1

Figure 3.8: The models of the control specifications for the factory problem and a description of the events used in the models.


Machine 1 → Grandmother
take$_1$
put$_1$
I
W
visit
candy
A
V
Machine 2 → Equipment
take$_2$
put$_2$
I
W
insulin
reading
I
W
Testing unit → Doctor
take$_{TU}$
accept, reject
I
W
report
medication, sweet
I
W
Buffer 1
accept, take$_1$, put$_2$, take$_{TU}$
put$_2$, accept, take$_1$, take$_{TU}$
put$_1$, reject
put$_1$, reject
put$_1$, reject
0
1
2
3
take$_2$
take$_2$
take$_2$
accept, take$_1$, put$_2$, take$_{TU}$
put$_2$, accept, take$_1$, take$_{TU}$
→
Patient
medication, visit, reading, report
reading, medication, visit, report
candy, sweet
candy, sweet
candy, sweet
0
8
16
24
insulin
insulin
insulin
medication, visit, reading, report
reading, medication, visit, report
Buffer 2 → Hospital Information System
put$_1$, accept, take$_2$, reject, take$_1$
put$_2$
0
1
take$_{TU}$
put$_1$, accept, take$_2$, reject, take$_1$
candy, medication, insulin, sweet, visit
reading
0
1
report
candy, medication, insulin, sweet, visit

take$_1$ → visit
put$_1$ → candy
take$_2$ → insulin
put$_2$ → reading
take$_{TU}$ → report
accept → medication
reject → sweet

Event	Description
visit	grandmother visits the patient
candy	patient gets candy from grandmother
insulin	equipment administers insulin to patient
reading	equipment takes reading of blood sugar level
report	doctor reads report about patient
medication	doctor prescribes medication to patient
sweet	patient gets candy from doctor

Figure 3.9: Demonstration of the recasting of the models from the factory problem in the setting of the hospital problem, and a description of the events used in the new models.

Chapter 4

Individual performances

In this chapter we will introduce the subjects who participated in the observational study and we will describe their individual performances when solving the two problems.

4.1 Subject 1

At the time of their participation, subject 1 was a fifth-year student of Control Engineering. The subject rated their background, on a scale from 1 (very little) to 5 (very much), as follows:

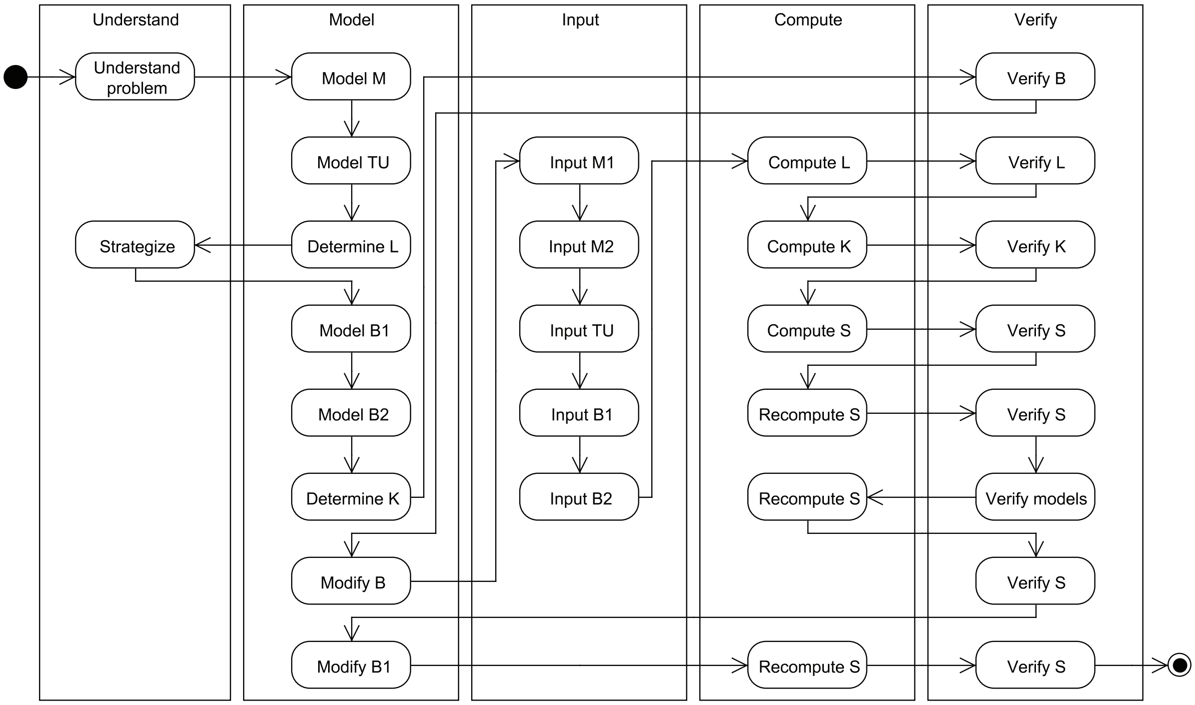
- Knowledge of some natural science: 5,
- Engineering background: 5,
- Knowledge of DES control theory: 3,
- Experience using software for DES: 4,
- Experience using the IDES software: 5.

Using the same scale, they rated their comfort using English as 5. At the time of participation, the subject was enrolled at the ELEC843 course offered by Dr. Rudie at Queen's University.

The observations of the experimenter were that during the study, the subject felt comfortable with thinking aloud. During the factory problem, the experimenter reminded them to keep talking 15 times; during the hospital problem—9 times. The subject occasionally struggled to find the right words to express their thoughts but, overall, it appeared that the the act of speaking did not have a significant impact on their problem solving.

4.1.1 Factory problem

The strategy used to solve the problem can be seen graphically in Fig. 4.1. The subject modelled both the system and the control specifications in a modular way. However, they used the IDES software to combine the modules into monolithic models and to produce a



Understand
Model
Input
Compute
Verify
Understand problem
Model M
Verify B
Model TU
Input M1
Compute L
Verify L
Strategize
Determine L
Input M2
Compute K
Verify K
Model B1
Input TU
Compute S
Verify S
Model B2
Input B1
Recompute S
Verify S
Determine K
Input B2
Recompute S
Verify models
Modify B
Verify S
Modify B1
Recompute S
Verify S

The following abbreviations are used in the chart: M1 – machine 1, M2 – machine 2, M – machines 1 and 2, TU – testing unit, L – monolithic system, B1 – buffer 1, B2 – buffer 2, B – buffers 1 and 2, K – monolithic control specification, S – monolithic supervisor.

Figure 4.1: Flowchart of the strategy used by subject 1 when solving the factory problem.

monolithic supervisor for the system. Apparently, the subject had had this strategy in mind from the beginning, since they formally defined how the monolithic system and specifications can be obtained after they finished modelling, correspondingly, the system modules and the specification modules.

Upon receiving the description of the problem, the subject almost immediately noticed that this problem is similar to, if not the same as, the problem discussed in the ELEC843 course. They read the description of the problem and examined the supplied diagram. They made no additional diagrams and did not write down any notes.

The subject used exclusively pen and paper to model the DES modules (system and control specifications). The modelling proceeded fast and the subject seemed to be very confident in what they did. From the beginning they recognized that the models for the two machines are identical and they created a single, indexed model. On the other hand, they chose to include extra behavior in their models for the machines and the testing unit. In the problem description, there is no mention of machines breaking down or of failing to process a part. However, the models the subject made allow for a part to be “lost” by a machine or by the testing unit, once it is accepted for processing. This appears to be an influence from the problem discussed in the ELEC843 course, where machines can break down and, subsequently, fail to process a part. We feel there is evidence for this influence in both the subject mentioning that they remember the problem from class, and in observing a similar design decision made by another subject. During the modelling of the control specifications, the buffers, the subject experienced uncertainty only once. When they finished modelling the modules of the specifications, they verified them against the requirements in the problem description. The subject became unsure about how exactly to mark the states in the models and what exactly "blocking" means. It seemed that this induced uncertainty in their judgment of whether the system would block or not under different marking schemes. They considered first marking all states of the buffers but finally settled for marking only the states where the buffers are empty.

During the modelling, the subject used the diagram provided with the problem description to write down the events they decided to use. Each event was spatially associated with the modules it belongs to. The event names were based on the verbal descriptions of the events, however, not to an extent where their semantics would be obvious. For example, the event when machine 1 takes a part for processing was named "t_1", the event when machine 2 puts a part in the output buffer was named "p_2", the events when the testing of the quality of a part by the testing unit is positive or negative were named "p" and "n", respectively. The subject did not seem to consider event controllability at the modelling stage.

After the basic modelling, the subject requested the use of the software. They first input all modules and then computed the monolithic system and specification. The entry of events seemed to be the biggest hurdle and a few times the subject needed to switch the view to modules which were already input to examine their events. This was most prominent when inputting the models of the buffers, as the consistent naming of the events is crucial for producing a meaningful solution. After inputting the models, the subject used the available DES operations to obtain the monolithic models. Each monolithic model was visually examined to confirm that it was the desired result. It appeared that the "proper" layout of the models was of very high importance to the subject, who commented that they needed to improve the layout to understand the models more clearly. Correspondingly, they spent a significant amount of time modifying the layout of all models generated by the software. Another common verification procedure was checking out if the number of states in the generated models was consistent with the subject's expectations.

When all models were ready, the subject proceeded with the computation of a supervisor. The first step was to check if the monolithic specification is controllable with respect to the monolithic system, by using the corresponding operation in IDES. Then, the subject used IDES to obtain the supremal controllable sublanguage of the specification. At this point, due to a bug in the way large graphs are visualized in the software, IDES crashed. After IDES was restarted, the subject decided to verify their solution without actually displaying it graphically. They checked the number of transitions and states in the generated supervisor (this information is available without laying out a graph of the model), and then checked to see if the supervisor is controllable with respect to the system—a negative answer would indicate a flaw in the DES algorithms. After the positive answer, the subject concluded that the supervisor should be OK. However, they voiced their strong dislike of the fact that they could not see the graphical structure of the result.

Instead of simply ending the session at this point, the experimenter proposed a way to get around the bug in IDES so that the subject could actually see the graph of the supervisor.

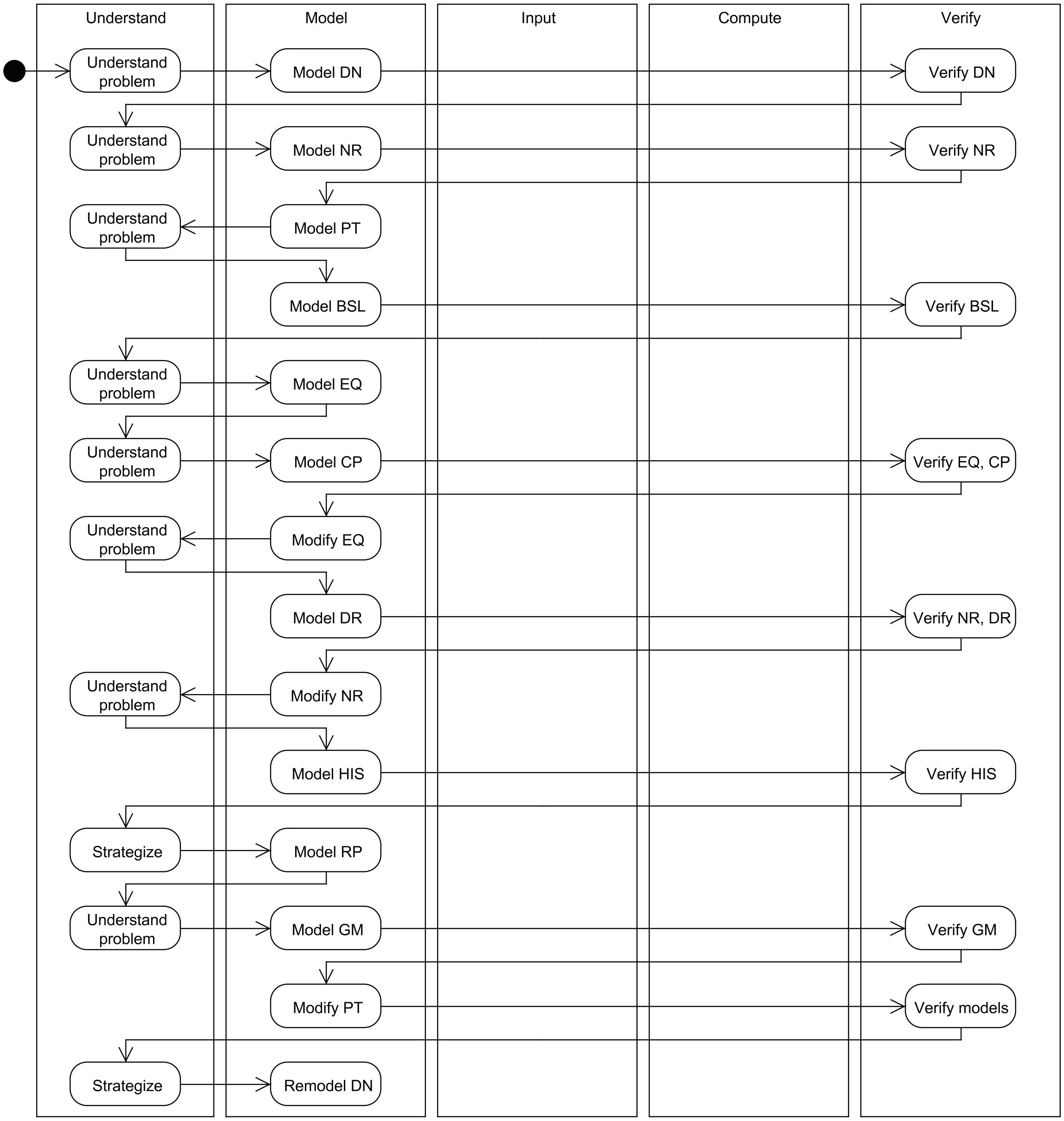
The subject used the advice and obtained the automatically laid-out graph. Then, similar to what they did for the monolithic models, they started reorganizing the layout so that they could acquire a better understanding of the behavior encoded in the structure. After some time of moving states and straightening transitions, the subject became unhappy with the way they had changed the layout and decided to get a new, automatically laid-out copy of the supervisor. Unfortunately, apparently the monolithic system model had not loaded correctly after IDES crashed and the DES algorithm failed to reproduce the correct supervisor. At first, the subject thought that the mistake was in the parameters they had provided to the algorithm, but after repeatedly calling the algorithm and obtaining the same incorrect result, they checked the monolithic models and saw the problem—which they fixed and then obtained the expected supervisor.

The second time the subject verified the supervisor, their actions seemed more targeted. They first looked up the initial state of the supervisor and then started tracing different event strings to see if they lead to the expected states. During one of these verifications, they discovered a string where instead of the testing unit sending a part back to buffer 1 for reprocessing when the part fails the test, the part is removed from the system. The subject diagnosed this as a potential issue with the model for buffer 1. Upon examination, this was confirmed. The subject corrected the problem, regenerated the supervisor and proceeded to verify it once more. They traced the same sequence of events which led to the discovery of the problem in the previous instance. After they confirmed that, this time, it works as expected, the subject seemed unwilling to perform more extensive testing. They announced that, for the purpose of the study, they feel their solution is complete.

During the interview, the subject explained briefly their problem solving. The subject mentioned that they did not make any diagrams describing the situation in the problem since a diagram was already provided. They tried to make models with the smallest number of states possible in order to reduce the complexity of the solution. They pointed out that even with only about thirty states, it was very hard to find the error in the model of the supervisor. Furthermore, the subject tried to model everything as things they already intimately knew, so that they would have a better idea of what to expect from all operations. On the other hand, coming up with a novel solution is much harder when one is already acquainted with a solution, the subject noted. Had they not seen a solution for this problem, they would have had a much harder time deciding what is a part of the system and what is a part of the specifications. The subject theorized that they would have most probably modelled the buffers as modules of the system, similar to the machines. The control specification then would have been the "trim" version of the composed system. Having the current solution, the subject indicated that as a future step in the problem solving they would like to implement their solution and run a black-box test on it.

4.1.2 Hospital problem

The strategy used to solve the problem can be seen graphically in Fig. 4.2. The subject modelled both the system and the control specifications in a modular way. They did not use the IDES software at any point. At the very end of the session they indicated they would like to use it for their next problem-solving steps, however, the session was over.


Understand
Model
Input
Compute
Verify
Understand problem
Model DN
Verify DN
Understand problem
Model NR
Verify NR
Understand problem
Model PT
Model BSL
Verify BSL
Understand problem
Model EQ
Understand problem
Model CP
Verify EQ, CP
Understand problem
Modify EQ
Model DR
Verify NR, DR
Understand problem
Modify NR
Model HIS
Verify HIS
Strategize
Model RP
Understand problem
Model GM
Verify GM
Modify PT
Verify models
Strategize
Remodel DN

The following abbreviations are used in the chart: GM – grandmother, PT – patient, NR – nurse, DR – doctor, DN – personnel (doctor and nurse), EQ – prototype equipment, CP – computer, HIS – Hospital Information System, BSL – blood sugar level, RP – report.

Figure 4.2: Flowchart of the strategy used by subject 1 when solving the hospital problem.

The subject started by reading the description of the problem and then drawing a diagram of the entities participating in the described situation. They focused their attention roughly in the following order: patient, candy, nurse, doctor, equipment, computer, information system, doctor, grandmother, nurse, candy. The nurse and the doctor were amalgamated into a single subsystem called “personnel”.

The subject kept track of the events they used on a separate sheet of paper. They listed each new event as they started using it in their models. The event names were based on the verbal descriptions of the events, however, not to an extent where their semantics would be obvious. For example, the event when the child eats a candy was called “ec”, the event when the computer transmits the information to the hospital information system was called “tr”, the event when the doctor read the report was called “rr”, etc. The controllability of the events was not considered during the problem solving. It seemed that the subject found it particularly difficult to deal with the proper way to model the increase of sugar level of the patient. Apparently, this was a result of the incorrect wording in the problem (described in Section 3.4). The subject chose to use three different events: the grandmother brings candy to the child, the child eats a candy, and the doctor gives a candy to the child, which made the problem much more difficult to solve.

The subject started modelling by drawing the model for the personnel. However, they soon gave up modelling the personnel as a single subsystem, amid uncertainty about how to keep track of when more insulin should be administered to the patient. They proceeded by modelling the nurse and, later, the doctor modules separately. Then they modelled the module for the patient and the specifications for the control of the sugar level in the blood (i.e., to keep the sugar level between 0 and 24 grams). After modelling the modules for the equipment and the computer, the subject realized that they are identical in behavior and one is sufficient.

The subject was most uncertain how to model the transmitting of the blood sugar level detected by the equipment to the doctor. Indeed, they seemed to think that it is insufficient to keep track of how much candy the child consumes, and that the doctor has to base their treatment decision (candy or medication) on the knowledge of the exact blood sugar level. Thus, the subject created a model called “report” where the result of the measurement should be captured. The creation was preceded by a two-minute period of deliberation. The “report” model was least rigorous of all models and necessitated the introduction of many new events: a separate “administer insulin” event for all possible levels of sugar in the blood. The subject was not very certain how to model the behavior of the grandmother either. They proposed a model where the event “bring candy” was repeated in a loop.

Towards the end of the session, the subject considered that most parts of the system were modelled to a degree sufficient to move on with the solution. They started composing the doctor and nurse modules manually, however, soon they requested the use of IDES to complete the task. At that point the time limit for the session was reached and the session was terminated.

During the modelling, the subject decided to correct or remodel some modules. When modelling the computer module, the subject presumably realized that the equipment module which they had modelled before, would need modifications to reconcile with the computer module. Similarly, the modelling of the doctor module resulted in the subject rethinking what the dynamics of the nurse module should be. When the subject modelled the grandmother module, they became uncertain about how the patient module would keep track of candy and they reconsidered the original design for that module. Again, the confusion revolved mostly around which "candy" events should be used and when (i.e., "grandmother brings candy", "child eats candy", and "doctor gives candy to child").

During the interview, the subject explained briefly their problem solving. The subject mentioned that they started by trying to get an overview of the situation described in the problem, without trying to focus on details. The next step was to try to model all subsystems identified during the overview. The subject complained about the way the problem is stated—that much of the information is scattered throughout the description and they had to hunt for it. For example, the subject did not initially realize that the prototype equipment is so dependent on the computer and, consequently, they had to correct the model of the equipment. They would have preferred if the information was organized similar to the way they had grouped it during the overview. A further complaint was that they did not have all the necessary information, e.g., what blood sugar level corresponds to the doctor deciding between giving candy to the patient and prescribing medication. The comments of the subject confirmed the observations that they had most trouble with the model for the report, but also indicated that they were not happy with the models of the patient and the blood sugar level, more specifically, with the fact that the models seem to be unbounded (infinite). The model of the grandmother and its interaction with the model of the patient was not resolved to satisfaction. The subject expressed their low confidence in the correctness of the solution, noting their belief that it was full of errors. As future, hypothetical steps they mentioned that they would like to talk to the doctor and nurse involved and try to discover the relevant information missing from the description. Furthermore, the subject would try to analyse the composed subsystems to see if there were any further errors in the models. More specifically, they would try to follow the behavior of the composed models and see if it makes sense in a real-world setting. As well, before any deployment, extensive tests should be made.

4.1.3 Within-subject analysis

For this subject, the hospital problem was administered before the factory problem. No conscious transfer of knowledge between the two problems was observed and during the interviews the subject did not mention noticing a degree of similarity between the problems. Their interpretation of the study was that it was designed to examine to what kind of problems discrete-event system control theory is applicable. They noted that the hospital problem was in a sense "softer" than the factory problem—and that human interaction seemed to be much harder to model.

This subject did not discuss what their motivation was for using pen and paper versus software to solve the DES problems. Our observation shows that the subject in both cases started problem solving by using pen and paper. They completed all modeling of individual modules on paper and only when they needed to perform a DES operation, such as composition of modules, did they decide to use the computer. In the hospital problem, the subject even tried first to compose two modules manually before requesting the use of the software.

In general, to the extent to which the hospital problem was advanced, the subject used a similar problem-solving strategy. First, they read the description of the problem and tried to get an overview of the situation. To this end, they employed a diagram showing the different participants or entities and how they are linked. In the case of the factory problem, a diagram was already provided and the subject deemed it acceptable for their purposes. In the second step, the subject modelled all subsystems and all control specifications, as identified during the first step. In the factory problem, there was a clear separation between modelling subsystems (machines and testing unit) and modelling control specifications (buffers). In the hospital problem, such a separation was not evident. Rather, it seemed that the subject modelled modules in a sequence according to how they relate to each other. For example, the control specification for the blood sugar level was modelled after the patient module; and the modelling advanced from prototype equipment to computer to information system—in the logical sequence of information transfer.

A significant, and expected difference between the solving of the two problems was in the speed and confidence which the subject manifested. The subject was very fast in solving the factory problem and they did not make any recognizable mistakes. Within the time limit of the observational session, even faced with problems in the implementation of the software, they succeeded in completing a solution. On the other hand, when solving the hospital problem, the subject advanced more slowly and frequently paused in order to consider their next steps. They demonstrated occasional uncertainty about the models they had produced and, on a few occasions, they had to return to previously designed models to correct perceived problems. In the allotted time, they did not succeed in advancing further than modelling the relevant components. An objective evaluation of the produced models also distinguishes the solutions to the two problems. In the factory problem, the models produced by the subject were, within a reasonable level of tolerance, correct. In the hospital problem, the subject did not succeed in modelling all system behavior correctly. They had notable problems with the granularity of the events they considered (they used three separate "candy" events) and they did not realize that the nurse does not have to be modelled to solve this problem. In general, it seems that the subject had a much harder time recognizing the parts of the system which are relevant to the solution. Furthermore, they struggled to model aspects of interaction observed in real-life even when these aspects were not required for solving the problem.

While all the observations discussed in the previous paragraph point to a much higher cognitive load in the hospital problem, it is interesting to note that thinking aloud seems to be more impacted in the factory problem. During the hospital problem (the first problem the subject solved—and thus with less experience thinking aloud), the experimenter reminded the subject to keep talking nine times. During the factory problem, the experimenter reminded the subject to keep talking fifteen times. Two potential explanations are that, during the factory problem, the subject was more focused (as they worked faster), or that there was a discrepancy in the criteria the experimenter used to decide when to remind the subject. Our belief is that, considering also the behavior of other subjects, the subject found it more demanding to voice their thoughts when they had a clear idea of what to do. Most probably such ideas are largely non-verbal. There are two inconveniences to the subject in such a case: first, they have to spend mental efforts translating their thoughts into verbal form and, second, they are slowed down if they have to voice their thoughts. This interpretation is consistent with the hypothesized impact of level 2 verbalization [17].

Lower-level activities of the subject, examined using n-gram analysis (see Section 3.6), also indicate patterns of similarity and difference. For example, the way the subject dealt with events during the two problems is different. In the factory problem, the subject used the diagram provided with the problem description to write down the events, associating them graphically with the relevant components of the system. There were no descriptions for the events. In the hospital problem, the subject kept a separate list of events and there was a description for each event. The n-gram analysis of the actions of the subject (see Fig. 4.3) further shows that during the factory problem, the subject specified events in bulk (high relative ratio of the bigram 'EE'). Conversely, during the hospital problem, events were specified on-the-go (the relative ratios of the bigrams 'ES' and 'ET' are higher than the one of 'EE'). This may indicate that the subject was confident in which events will be used in their factory model, but was less confident about the events for the hospital problem.

The results of the n-gram analysis show that when the subject modelled using pen and paper, they were likely to continue working on the same aspect of the model, e.g., when working on transitions they were likely to continue working on transitions. This is visible in the high absolute and relative ratios of the bigrams 'TT' and 'SS'. This pattern is not as pronounced for the hospital problem, where the relative ratio of the bigram 'ST' is a little bit higher than that of 'SS'. This may be the result of the subject being less certain of which states a model should consist of—and thus considering states incrementally, as opposed to an uninterrupted sequence of state-related actions. An interesting observation is the fact that, in both problems, the bigram 'MS' has a very high relative ratio. This means that, after considering a module, the subject very reliably proceeded working on the states of a model.

Fewer differences can be observed when examining the results of the n-gram analysis for the factory problem, when solving with pen-and-paper or software (see Fig. 4.4). There are only two facts which stand out. First, the relative ratio of the bigram 'MS' when using the software is quite small when compared to that of the bigrams 'MM', 'ME' and 'MC'. It appears that when using the software, the consideration of modules was within a higher-level context (such as events and algorithms)—and not concerned with the actual modeling (i.e., having to deal with states and transitions). The second fact which stands out is that when dealing with DES algorithms (bigrams including the code 'C'), the subject tended to stay in at the higher-level context of modules and algorithms (notice the high relative ratios of the bigrams 'CM', 'MC' and 'CC'). The elements of the low-level models that seem to be most relevant to the DES algorithms are the states (high relative ratio of 'CS' and no 'CT' or 'CE' bigrams). These results are within our expectations. When working with pen and paper, the subject was creating models of the subsystems from the problem, while when using software, they were simply inputting these models and then using the DES operations for higher-level modelling such as composition of subsystems.

The attention of the subject throughout the problem solving was partially indicated by their verbalizations. As one of our goals is to discover what information is used to solve DES problems, we decided to track two types of events related to attention: perceptually-triggered data discovery and cognitively-driven data discovery. The description of how this

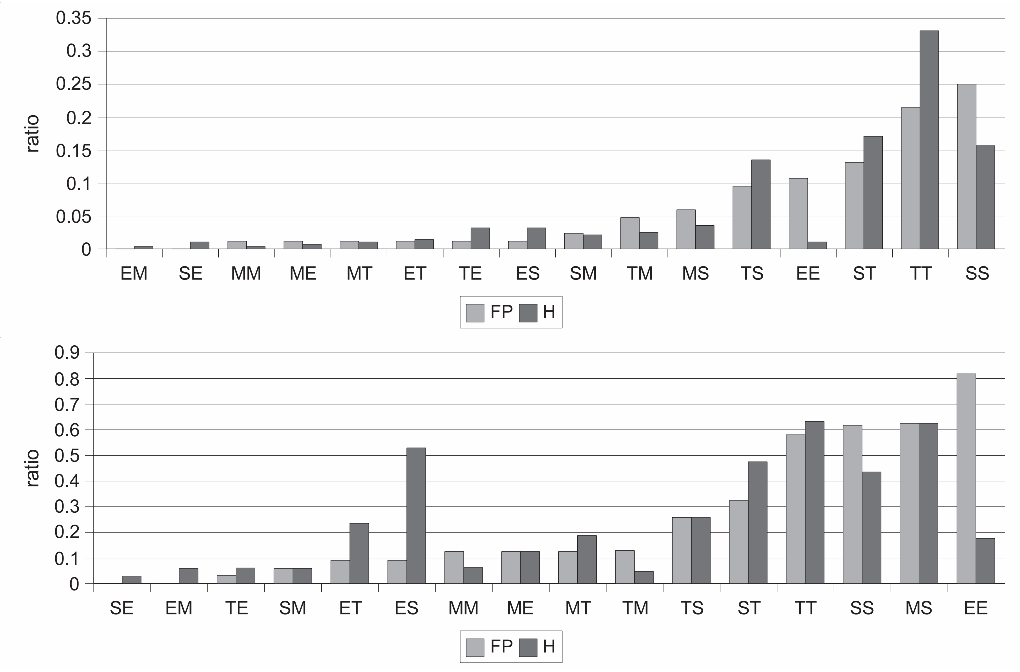


Figure 4.3: The absolute and relative ratios of bigrams for pen-and-paper problem solving during factory problem (FP) and the hospital problem (H). Data are sorted according to the ratios in the factory problem.

analysis is performed can be found in Section 3.7. All events were examined and the following was discovered. During the factory problem, mostly the states of the models contributed to perceptually-triggered data discovery. Such discoveries usually occurred within the current context of the subject (e.g., when working on the testing unit, they discovered something about the same module, the testing unit). Cognitively-driven data discovery concerned predominantly states and events, and the intention was to collect it most frequently by visual inspection and/or by counting the elements. When counting, they counted states and events. During the hospital problem, no low-level elements triggered data discovery. Instead, the subject's attention seemed to be drawn by the general dynamics of subsystems and the figures displaying how the subsystems interact. On occasion, the discovered data would concern parts of the system different from what the subject was working on (e.g., information about the hospital information system while working on the module of the doctor), however, no stable pattern was discovered. The cognitively-driven data discovery concerned mostly the dynamics of modules and the subject most frequently expressed intentions to obtain it through visual inspection or reading (the problem description). These results show that graphical data representation was helpful to the subject.

The analysis of the visual attention of the subject revealed the following facts. During the factory problem, the problem description was the most common target of visual attention, especially during the modelling of the modules. Otherwise, the subject usually did not focus on a specific sheet of paper but rather examined all models. Thus, no specific patterns of attention could be established. During the hospital problem, the subject's attention was

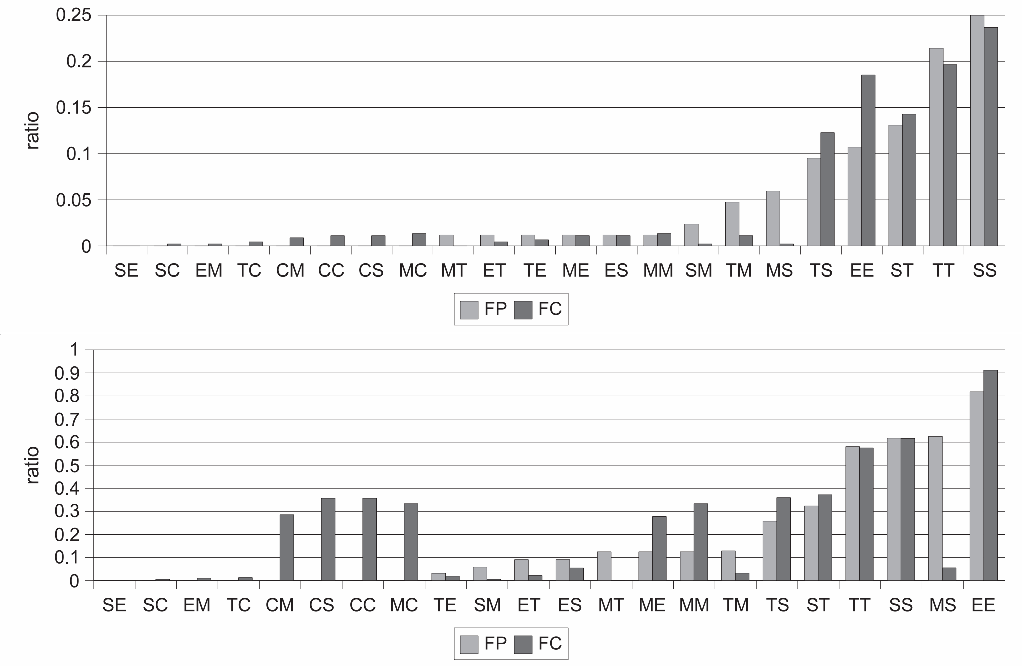


Figure 4.4: The absolute and relative ratios of bigrams for pen-and-paper problem solving (FP) and solving using software (FC) during the factory problem. Data are sorted according to the ratios in pen-and-paper solving.

not, proportionally, attracted as much by the problem description. The following patterns emerged. When modelling the module of the patient, the subject frequently paid attention to the sheets of paper with the models of the equipment, computer, hospital information system, report and the grandmother. When modelling the hospital information system, they frequently paid attention to the sheet with the models of the doctor, nurse, patient and blood sugar level. When modelling the report, they paid attention to the same sheet—with models of the doctor, nurse, patient and blood sugar level. The modelling of the blood sugar level, computer, doctor, grandmother, nurse and prototype equipment was much more self-contained, without switching the visual attention to other sheets of paper.

4.2 Subject 2

At the time of their participation, subject 2 was a fifth-year student of Control Engineering. The subject rated their background, on a scale from 1 (very little) to 5 (very much), as follows:

- Knowledge of some natural science: 4,
- Engineering background: 5,
- Knowledge of DES control theory: 4,

- Experience using software for DES: 5,
- Experience using the IDES software: 5.

Using the same scale, they rated their comfort using English as 4. At the time of participation, the subject was enrolled at the ELEC843 course offered by Dr. Rudie at Queen's University.

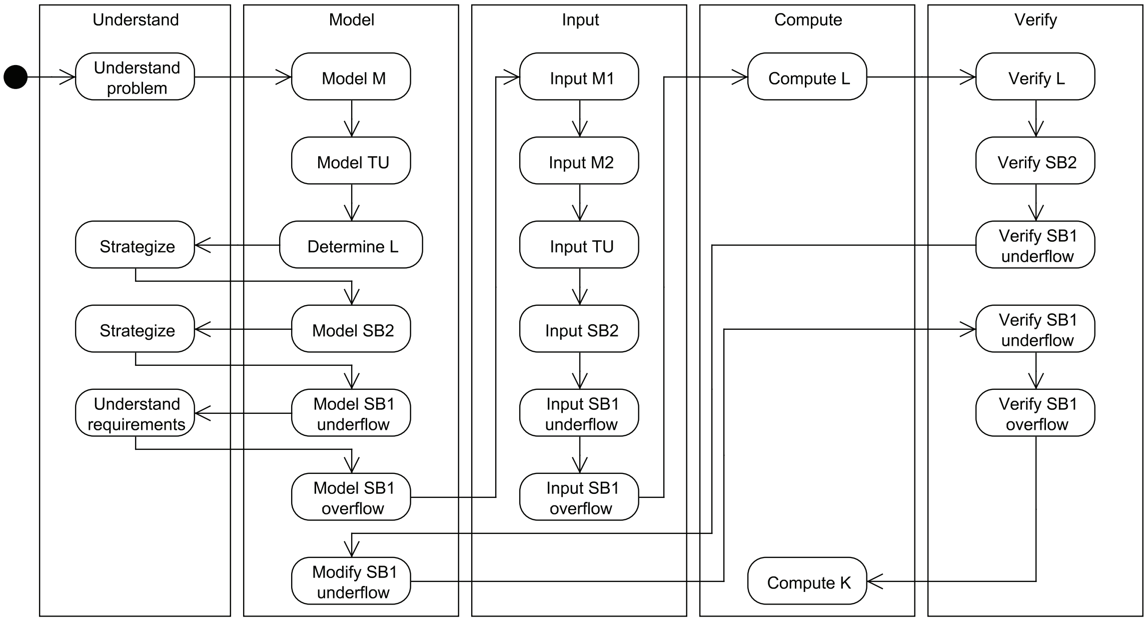
The observations of the experimenter were that, after an initial period of getting accustomed, the subject felt comfortable with thinking aloud and kept a very steady pace. During the factory problem, the experimenter reminded them to keep talking 6 times; during the hospital problem—3 times. The subject rarely used well-formed sentences and did not seem to focus on explaining their thoughts. However, in one of the short interviews, the subject mentioned that they felt uncomfortable working in front of a camera. Additionally, they felt it took more effort to work on the problem while having to voice their thoughts. Nevertheless, they felt thinking aloud did not affect their "train of thought" beyond slowing down the problem solving.

4.2.1 Factory problem

The strategy used to solve the problem can be seen graphically in Fig. 4.5. The subject modelled both the system and the supervisors in a modular way. However, they used the IDES software to combine the system modules into a monolithic model. Apparently, the subject had had this strategy in mind from the beginning, since they formally defined how the monolithic system can be obtained after they finish modelling the system modules. On the other hand, the subject never combined the supervisors into a monolithic model and seemed to act so as to avoid doing this. The subject did not create models of the control specifications during their problem solving.

After receiving the description of the problem and examining it, the subject recognized that the problem is similar to, if not the same as, the problem discussed in the ELEC843 course. They read the description of the problem and examined the supplied diagram. They expressed dislike of the diagram. However, apparently the description and the diagram were enough for them to gain understanding of the problem. They made no additional diagrams and did not write down any notes.

The subject used exclusively pen and paper to model the DES modules (system and supervisors). The modelling proceeded fast and, except for the modelling of the first buffer, the subject seemed to be very confident in what they did. From the beginning they recognized that the models for the two machines are identical and they created a single, indexed model. When modelling the supervisors for the buffers, the subject decided to model the second buffer first. They mention at one point that this was "the simple part". We assume that they wanted to complete modelling the parts they felt confident about before moving to the more difficult parts. When modelling the first buffer, the subject decided to split the model into two: one model preventing the underflow of the buffer and one model preventing the overflow. Again, the subject rated the underflow control as simpler than the overflow control. The production of the specification for the underflow prevention did not appear to cause significant difficulty to the subject, even though it was preceded by a longer period of


Understand
Model
Input
Compute
Verify
Understand problem
Model M
Input M1
Compute L
Verify L
Model TU
Input M2
Verify SB2
Strategize
Determine L
Input TU
Verify SB1 underflow
Strategize
Model SB2
Input SB2
Verify SB1 underflow
Understand requirements
Model SB1 underflow
Input SB1 underflow
Verify SB1 overflow
Model SB1 overflow
Input SB1 overflow
Modify SB1 underflow
Compute K

The following abbreviations are used in the chart: M1 – machine 1, M2 – machine 2, M – machines 1 and 2, TU – testing unit, L – monolithic system, K – monolithic control specification, SB1 – supervisor for buffer 1, SB2 – supervisor for buffer 2.

Figure 4.5: Flowchart of the strategy used by subject 2 when solving the factory problem.

planning. The specification for overflow prevention appeared troublesome to the subject and they had to discard their first attempt and re-model it. They determined their initial model was incorrect by considering the dynamics of the different subsystems and realizing that they had used the wrong events to control the load of the buffer. While the analysis of the subject's work shows that they were modelling supervisors instead of control specifications for the buffers, the subject did not appear to be aware of this explicitly. They were terming the models "specifications" instead of "supervisors". Implicitly, however, the subject was aware that these are supervisors as they used them as such when employing the software later on. None of the models produced on paper contained marked states; the subject appeared not to consider marking at this stage.

The event names used by the subject were based on the verbal descriptions of the events, however, not to an extent where their semantics would be obvious. For example, the event when machine 1 takes a part for processing was named "t_1" and the event when machine 2 puts a part in the output buffer was named "p_2". The events of the testing unit were less clear. While "p_U", the negative outcome of testing part quality, was consistent with the naming scheme (i.e., it stands for "the testing unit puts a part in buffer 1"), the event for a positive outcome of the test was "f_U". The subject did not make any notes about what events were used in the models besides the occurrences of the events as labels of transitions. The controllability of events was considered only verbally and the models did not include any indication of which events were controllable and which were not.

After the basic modelling, the subject requested the use of the software. They first input all modules. This process took longer than what would be expected. The subject wished to use the “copy” function of the software to copy similar models and speed up the process of inputting. For example, they attempted to create the model for machine 2 by copying the model of machine 1, with the intention of modifying only the relevant parts, e.g., changing the event names. Unfortunately, due to a bug in the implementation of IDES, the “copy” function did not always produce the desired result. In order to resolve this issue, the subject was offered to save their current work and then open it within an older version of IDES which did not exhibit this particular bug. Another shortcoming of the software discovered by the subject was that the software does not have the capability to automatically create self-loops of events in a model, as required by the computational algorithms used. During the inputting of the models, the subject decided which states had to be initial states and which states had to be marked.

When all modules were input, the subject proceeded by constructing the monolithic model of the system behavior. Then, they decided to check the controllability of the supervisors. First, they checked the controllability of buffer 2. The older version of IDES which the user started using to resolve the bug with the input of automata, unfortunately did not contain a correct implementation of the algorithm for checking controllability. Thus, it gave an incorrect result saying that the model of the buffer was not controllable. The subject apparently had the opposite expectation and they briefly checked all modules to confirm that their model ought to be controllable. Then, they announced that there was a problem with the software. To resolve this new issue, the subject saved their work and opened it once again in the version of IDES which was originally used in the observational study. Then, they checked the controllability of the supervisor again and they obtained the, correct, result that the model is, indeed, controllable. The controllability check of the underflow prevention for buffer 1 revealed that the model is not controllable. Again, this was contrary to the subject’s expectations. This triggered a cycle of attempts to correct the model and then check its controllability until the model was verified as controllable. The issue lay with the subject forgetting to set the controllability properties of the events of the model. The controllability check for the overflow prevention for buffer one revealed that it is, indeed, controllable. The subject then voiced their desire to check if the specifications were *not conflicting*, a standard way to verify that supervisors will not cause a deadlock or a livelock in a system. The software does not offer this capability, so the subject tried to come up with another way of verifying that the controlled system will be deadlock- and livelock-free. Eventually, they converged to a solution where all supervisors will be composed into a monolithic model and then its action on the system examined. At this point the time for the observational session expired.

During the interview, the subject explained briefly their problem solving. They mentioned that in the beginning they realized that they had seen the solution to this problem before, however, they could not remember the details. Thus, they had to produce the solution anew. They modelled the modules on paper and that served as a way to “offload” them from their mind. They preferred to use pen and paper because they find modeling like that easier than using mouse and software. However, when there were too many states, the subject preferred using the computer. They mentioned that even the combination of of the system components, which would result in eight states, seemed to call for the use of the software. The software itself was evaluated as mostly intuitive, but lacking a number of desired features. For example, modelling a number of modules was tedious since event names cannot be copied from existing models. As well, the subject theorized that modelling a large system would be problematic. When asked what they would do to complete the solution to the factory problem, the subject indicated that they would like to check if the supervisors they created are non-conflicting and that potentially they would like to create a new, monolithic supervisor instead of the modular ones.

4.2.2 Hospital problem

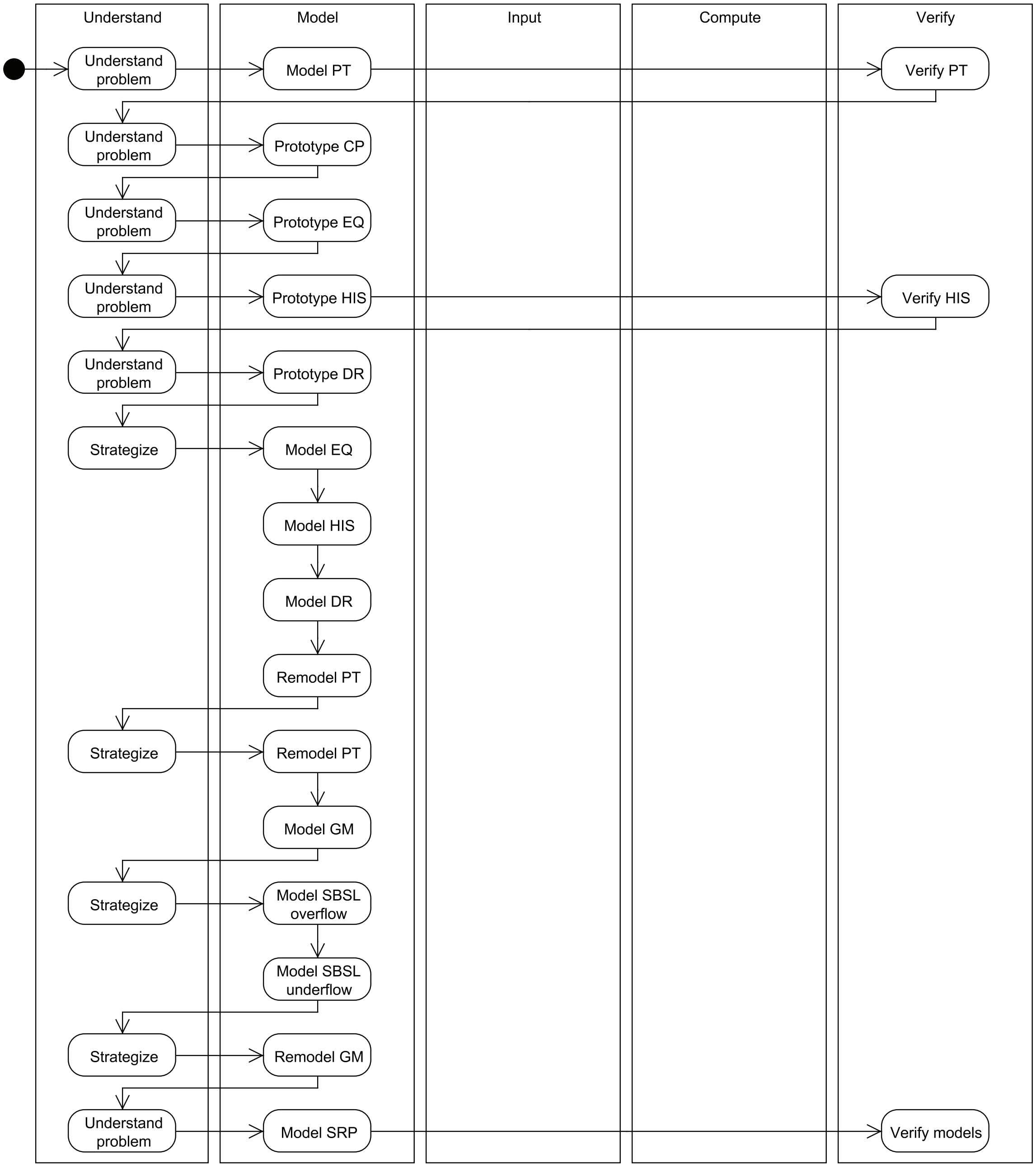
The strategy used to solve the problem can be seen graphically in Fig. 4.6. The subject modelled both the system and the supervisors in a modular way. They did not use the IDES software at any point. At the very end of the session they indicated they would like to use it for their next problem solving steps, however, the session was over. The subject did not create models of the control specifications during their problem solving.

The subject started by reading the description of the problem. From what they said, their attention was focused on the patient and on the interaction between the computer and prototype equipment (which they could not understand entirely).

The subject did not make any notes about what events were used in the models besides the occurrences of the events as labels of transitions. The event names were descriptive enough to make their semantics obvious. For example, the event when the child eats a candy was called “eat candy”, the event when the computer transmits the information to the hospital information system was called “send report”, the event when the doctor read the report was called “read”, etc. The controllability of the events was not considered initially. Only after the subject tried to establish the specifications for the controlled system did they start considering event controllability. The subject correctly identified most of the relevant events in the system, as well as their controllability. Apparently, due to the incorrect wording in the problem (described in Section 3.4), they chose to use two events, “give candy” and “eat candy” to describe the behavior of the patient. This significantly complicated the problem. Furthermore, they identified the outcomes of the doctor’s evaluation (giving candy or prescribing medicine) as controllable events. While, from the doctor’s perspective, they are, indeed, controllable, within the context of the problem, they are not. The doctor’s expert opinion cannot be superseded by that of an automatic controller. Finally, the subject also interpreted the description of the problem such that prescribing medication was in fact prescribing insulin. While the description does not mention what the prescribed medication is, or what effect it has, there is no indication that it is equivalent to administering insulin.

The process of modelling did not advance continuously. Rather, the subject modelled a few modules and then spent time analysing how to proceed. They seemed to consider extensively their options for what to do before advancing. As well, they seemed to prioritize their actions according to their depth of understanding of different aspects of the problem.

The subject started modelling by drawing the model for the patient, describing the complementary function of consuming candy and receiving a shot of insulin. Then, they focused attention on the flow of information from the prototype equipment to the doctor. They drew a small diagram and then sketched out the models for the equipment, the computer, the


Understand
Model
Input
Compute
Verify
Understand problem
Model PT
Verify PT
Understand problem
Prototype CP
Understand problem
Prototype EQ
Understand problem
Prototype HIS
Verify HIS
Understand problem
Prototype DR
Strategize
Model EQ
Model HIS
Model DR
Remodel PT
Strategize
Remodel PT
Model GM
Strategize
Model SBSL overflow
Model SBSL underflow
Strategize
Remodel GM
Understand problem
Model SRP
Verify models

The following abbreviations are used in the chart: GM – grandmother, PT – patient, DR – doctor, EQ – prototype equipment, CP – computer, HIS – Hospital Information System, SBSL – supervisor for the blood sugar level, SRP – supervisor for the report.

Figure 4.6: Flowchart of the strategy used by subject 2 when solving the hospital problem.

hospital information system and for the doctor.

After these models were complete, the subject focused on the actions of the grandmother and what control can be exercised to prevent the child from eating dangerous amounts of candy. At some point, however, it seemed that the subject gave up this train of thought and instead proceeded with remodelling the modules responsible for the processing of the information from the equipment. They correctly noticed that it is redundant to consider the equipment and computer modules separately, thus they modelled them as a single module. After remodelling these modules, they copied the initial model of the patient. Then, the subject again shifted their attention to the actions of the grandmother. They seemed to struggle trying to decide how to model this part of the system. At the end, they announced that they will instead try to deal first with the other aspects which they understand better. In the next stage, they wrote down a more formal version of the specifications for which the problem calls, and they determined the controllability of events. That led to a re-evaluation of their understanding of the model of the patient. They modelled an additional component that described another aspect of the patient's activities. Apparently, the latest considerations also made it more clear for the subject how to model the grandmother—they created a model right away after finishing the additional model for the patient.

When all relevant subsystems were modelled, the subject started considering the control necessary to protect the patient from overdosing on sugar or receiving too much insulin. They seemed to have a hard time deciding how to model the corresponding specifications. The main stumbling block was how to keep track of how much candy the child has been given and how much candy has been consumed. The impasse was resolved when the subject decided to use the same approach as the one they used with the buffer in the factory problem. They modelled two supervisors: one for preventing "overflow", i.e., eating too much candy, and one for preventing "underflow", i.e., administering too much insulin. While it did not appear that the subject realized that the factory problem and the hospital problem are very similar (neither did they mention this in the interview), there was clearly transfer of knowledge from the solution of the first problem. In this case, this transfer was particularly important since this was the knowledge that helped the subject advance.

The next realization by the subject was that in some of the their models, the event "give candy" is controllable while, in others, it is not. They spent a long period trying to reconcile their dualistic approach—considering both what it would mean to make all instances of the event controllable and what it would mean to make all instances uncontrollable. At the end, they resolved the issue by remodelling the model of the grandmother and making all instances of "give candy" controllable. The subject seemed to realize that there is some redundancy in their models but they did not seem to be able to pinpoint it.

In the last problem-solving step before the end of the session, the subject reviewed all modules they had designed and determined that they were missing the supervisor which would prevent the failure of the hospital information system. They created a very rough sketch of the model, labelling states but not using any events. Then, they requested to use the software, however, the time for the session expired.

During the modelling, the subject frequently modified or remodelled their models. However, it seems that there were two main causes for that behavior. First, the subject apparently decided to initially sketch out rough models, and then create the "real", more elaborate versions. Thus, remodelling seemed to be part of their plan. Second, a lot of the modifications to existing models were induced by the changes of the subject's understanding of the nature of the system events. There was a variety of such modifications, from simple changes to the controllability of events, to remodelling a complete module (for example, the module for the grandmother was remodelled to reconcile it with the new understanding of the semantics of the event "give candy").

During the interview, the subject explained briefly their problem solving. They mentioned that first they read the whole problem description, but afterwards they focused on different parts, as the amount of information felt overwhelming. They attempted to model the parts that they understood in order to "free" their mind from having to think about them. Then, they could refer to the models when they needed to recall details (referring to the problem description was not considered necessary since they trusted their models). The subject said that the two most difficult aspects of the problem were determining what is part of the system and what is part of the specifications, and what events should be used and what their controllability should be. Having the current solution, the subject indicated that as future steps in the problem solving they would like to check if the supervisors they designed are controllable and if they are non-conflicting. If both tests are positive, they would announce a successful solution. If not, they would go back to review if the models are correct and to check if their interpretation of the control requirements is correct.

4.2.3 Within-subject analysis

For this subject, the factory problem was administered before the hospital problem. Transfer of knowledge between the two problems was observed. The subject resolved a design issue during the hospital problem by recalling a part of their solution of the factory problem. However, there was no indication that the subject noticed a deeper-running similarity between the two problems. For example, except on this one occasion, they never mentioned the factory problem. Their interpretation of the study was that it was designed to examine to how people interpret DES problems and how they convert informal verbal descriptions into formal DES models.

This subject started using pen and paper for both problems, and used the software only in the factory problem, when the process called for the composition of simple models into more complex entities. The subject explained that they prefer the use of pen and paper over a computer and they were "forced" to use software only due to the complexity of the problem. However, their evaluation of the software was not entirely negative. They found the the main interface intuitive and they liked the fact that they can drag elements of a model to change its layout. The main complaints were the lack of some DES operations, such as *inverse projection* and the check for *conflict*, and the fact that copying of events between models was not supported.

When comparing the problem-solving strategies of the subject for both problems, they were quite similar, however, there were also some differences. Both strategies were based on the idea of first modelling all subsystems, then modelling supervisors and then checking if the supervisors are controllable and non-conflicting. The subsystems were modelled in sequence according to the "flow" of material or data through the system. However, due to the perceived greater difficulty of the hospital problem, the subject employed an additional strategy. When they encountered a subsystem which the subject did not have a good idea how to model, they decided to postpone modelling it and to first complete modelling the subsystems they were more confident about. This shift of priority was demonstrated two times when the subject considered how to model the grandmother and then chose to first model another aspect of the system. However, when modelling the supervisors, the subject experienced difficulty in both problems—and subsequently applied the same strategy of first modelling the simplest parts and last the hardest. A difference in problem solving was observed in the management of information. In the factory problem, the subject read the description of the problem thoroughly once and then all modelling proceeded without a major interruption. During the hospital problem, the subject seemed to "partition" the problem description into sections and then focus on each section separately—first examining it, and then modelling the modules derived from the section. This was later confirmed by the subject in the interview, where they explained that the amount of information was overwhelming for them, so they decided to model the parts they understood so that they would not have to think about them anymore. Another observed difference is the fact that during the hospital problem the subject on a few occasions sketched very rough models which were later refined and remodelled in greater detail. This approach was not used in the factory problem.

In both sessions, the subject never called the supervisor models "supervisors". Instead, they referred to them as "specifications". However, in the last stages of problem solving it became most clear that, in fact, these specifications were, or were being treated as, supervisors. There are three indicators for this:

- The models of the specifications in the factory problem expressed not only the desired behavior of the system, but also gave prescriptions for how to accomplish this behavior. While there is quite a lot of flexibility in the extent to which control prescriptions are included in the specifications, usually it is up to the supervisor construction algorithm (in the software) to determine the best way to satisfy restrictions on system behavior.

- The subject was very concerned with having controllable specifications. A controllable specification implies that the supervisor for this specification will be identical with the specification, i.e., no additional control has to be exercised. Usually, specifications are *not* controllable and the purpose of the supervisor construction algorithm is to find the proper control policy to satisfy the specification. In this sense, the subject was aiming for specifications which can be directly applied as supervisors.

- The check for lack of *conflict* is applied to supervisors. It has no significance when applied to specifications.

Thus, we conclude that this subject did not have distinct concepts for specifications and for supervisors. Designing supervisors is, normally, more demanding that designing control specifications. This may also explain why this subject consistently experienced more difficulty modelling, in their eyes, the "specifications" for the systems, compared to modelling the subsystem modules.

A significant, and expected difference between the solving of the two problems was in the speed and confidence which the subject manifested. The subject was fast in solving the factory problem until they got to the modelling of the supervisors. They did not make any recognizable mistakes with the subsystem modules. They managed to resolve most issues with the specifications/supervisors and almost completed the problem within the time allotment. On the other hand, when solving the hospital problem, the subject advanced more slowly and made longer pauses between solving different parts of the problem. They referred to the problem description to gain better understanding of the specific parts they were working on. In the allotted time, they did not succeed in advancing further than modelling the relevant components, some of them only very roughly. An objective evaluation of the produced models also distinguishes the solutions to the two problems. In the factory problem, the models and supervisors produced by the subject were, within a reasonable level of tolerance, correct. In the hospital problem, the subject did not succeed in modelling all system behavior correctly. They had notable problems with the models for the patient and the grandmother. They considered two separate modules for the patient and for the level of sugar in the blood of the patient, while a single model would have resolved some of the issues the subject experienced. Furthermore, the module for the patient was modelled using two separate, complementary models. This indicates that the complete function of the patient was not apparent to the subject from the beginning. The module for the grandmother was, eventually (and incorrectly), reduced to a single self-looping event. The subject indicated that they had trouble discerning which parts of the problem should be part of the system description and which ones should be part of the control specifications.

Lower-level activities of the subject also indicate patterns of similarity and difference. For example, when comparing the results of the n-gram analysis for the pen-and-paper part of problem solving (see Fig. 4.7), it is clear that the subject followed similar patterns of actions during both problems. The only exception is in the actions related to events. It was only during the hospital problem that the subject considered event controllability—and thus performed event-specific actions. As a result, only during the hospital problem did event-related actions have non-zero ratios (which is clearly visible in the chart with relative ratios).

The results of the n-gram analysis show that when the subject modelled using pen and paper, they were likely to continue working on the same aspect of the model, e.g., when working on transitions they were likely to continue working on transitions. This is visible in the high absolute and relative ratios of the bigrams 'TT', 'SS', 'EE' (for the hospital problem) and 'MM'. The relative ratio of the bigram 'ST' is a little bit higher than that of 'TS'. This indicates that usually the subject would first draw all states and then start drawing the all transitions, i.e., they would normally not return to drawing states after drawing transitions. The bigram 'MS' has a high relative ratio. This means that, after considering a module, the subject very reliably proceeded working on the states of a model.

More differences can be observed when examining the results of the n-gram analysis for the factory problem, when solving with pen-and-paper or software (see Fig. 4.8). The first difference, as expected, is the fact that only when the software was used did the DES-algorithm-related bigrams (including the code 'C') have non-zero ratios. Similarly, as discussed previously, no event-related actions were recorded during solving with pen and paper. For both stages of problem solving, the relative ratios show that continuing working on the same aspect of the problem is very likely (even the occurrence of ‘CC’ is more likely than any other bigram containing ‘C’). However, there is an interesting exception: during the use of the software, both the ‘ST’ and ‘TS’ bigrams occur more frequently than the ‘SS’ bigram (see the absolute ratios), and the relative ratio of ‘ST’ is higher than that of ‘SS’. This means that, when using the software, the subject did not tend to create states in bulk. Rather, they would alternate between creating states and transitions.

When dealing with DES algorithms (bigrams including the code ‘C’), the subject tended to stay in at the higher-level context of modules and algorithms (notice the elevated relative ratios of the bigrams ‘CM’, ’MC’ and ’CC’). The elements of the low-level models that seem to be most relevant to the DES algorithms are the transitions (higher relative ratio of ‘CT’ and lower ‘CS’ or ‘CE’ ratios). However, the number of algorithm-related bigrams is quite small (see low absolute ratios) and thus these conclusions may not be reliable.

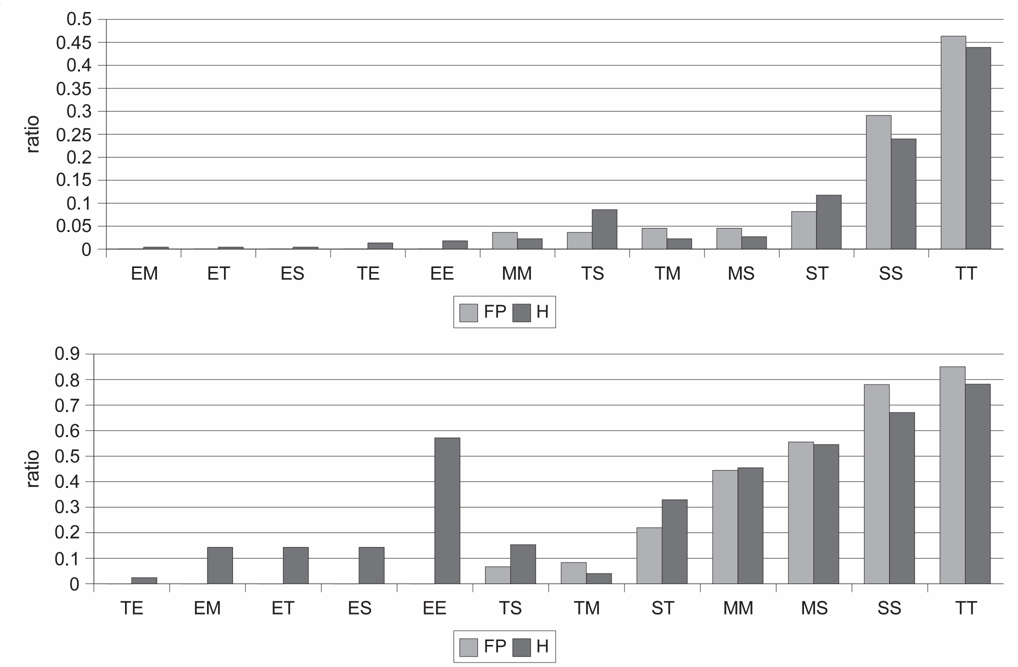


Figure 4.7: The absolute and relative ratios of bigrams for pen-and-paper problem solving during factory problem (FP) and the hospital problem (H). Data are sorted according to the ratios in the factory problem.

The attention of the subject throughout the problem solving was analysed as described in Section 3.7. During the factory problem, the subject was very verbal about their discoveries about the functionality of IDES. Besides that, most of the times, the perceptually-triggered data discovery concerned modules and their control. Such discoveries usually occurred within the current context of the subject (e.g., when working on the supervisor for buffer 1 underflow, they discovered something about the same model, the supervisor for buffer 1 underflow). There was some indication that discoveries about the problem description were also made

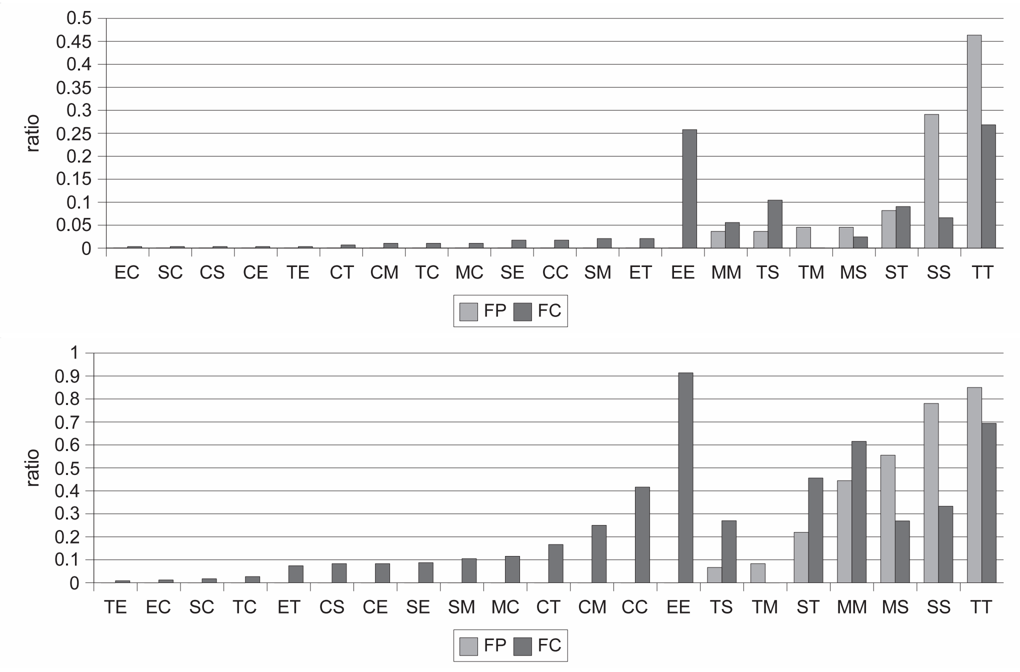


Figure 4.8: The absolute and relative ratios of bigrams for pen-and-paper problem solving (FP) and solving using software (FC) during the factory problem. Data are sorted according to the ratios in pen-and-paper solving.

from within a number of contexts, most prominently when working on the supervisors for the buffers. Cognitively-driven data discovery concerned all possible aspects and entities which were used in the encoding, evenly. The subject voiced intentions to collect the data using all ways which were encoded, evenly. When counting, they counted states, events and models. Again, discovery targeted most frequently aspects of the current context. During the hospital problem, the subject did not report almost any perception-triggered data discovery. The reported discovery concerned dynamics of subsystems and usually referred to the problem description. The cognitively-driven data discovery concerned mostly the problem-solving method, modules and events. When counting, the subject counted states or modules. Examining the contexts of the cognitively-driven data discovery, a clear pattern emerged: the subject seemed to be interested in data about all modules while working on the patient module. The most frequent modules of interest were the module for the grandmother and the module for the doctor. The subject also seemed to be interested in information about other modules while working on the modules for the doctor and for the hospital information system. No other reliable patterns were discovered.

The analysis of the visual attention of the subject revealed the following facts. During the factory problem, the problem description was the most common target of visual attention, especially during the modelling of the supervisors for the buffers. Otherwise, the subject usually did not focus on a specific sheet of paper but rather examined all models. Thus, no specific patterns of attention could be established. During the hospital problem, the subject's attention was, again, attracted most by the problem description. The following patterns emerged. When remodelling the module of the patient, the subject frequently paid attention to the sheets of paper with the initial, prototype models of the patient, equipment, computer and doctor, and the models of the supervisors for the control of the blood sugar level. When modelling the supervisors for the control of the blood sugar level and for the report, they paid attention to the sheet with the proper models of the equipment, hospital information system, doctor, grandmother and patient. When modelling properly the equipment and the hospital information system, they paid attention to the sheet with the initial, prototype models of the patient, equipment, computer and doctor. When remodelling the grandmother, they paid attention to the sheet with the models of the supervisors for the control of the blood sugar level. The modelling of the doctor was much more self-contained, without switching the visual attention to other sheets of paper.

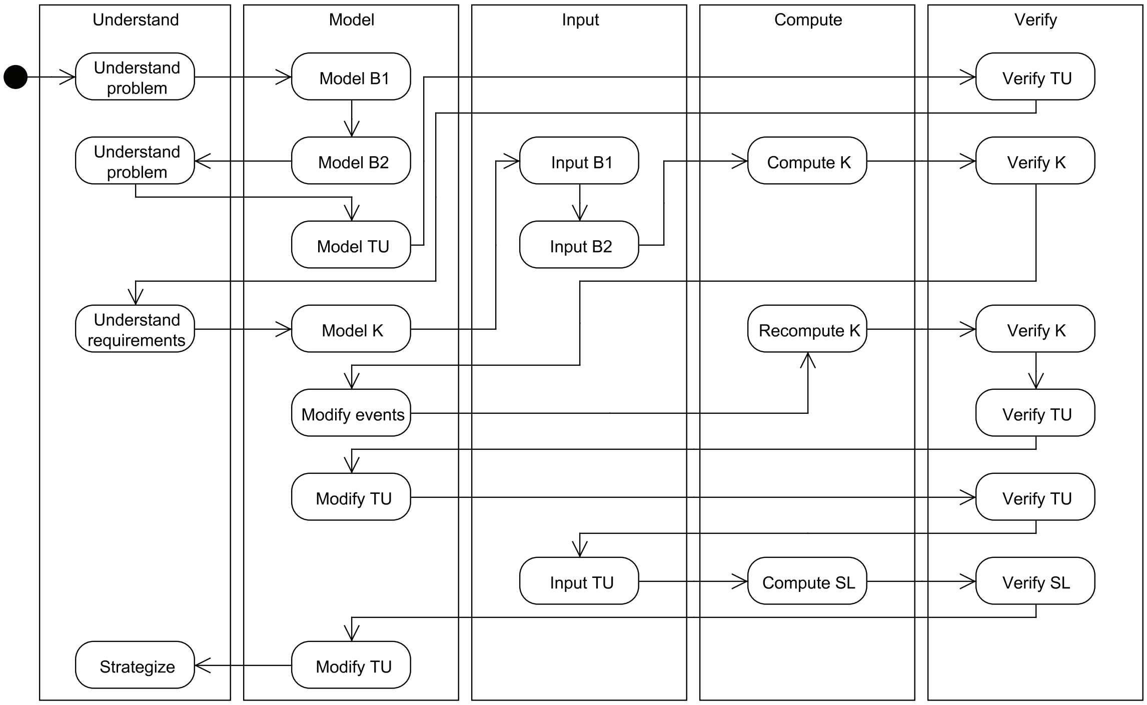
4.3 Subject 3

At the time of their participation, subject 3 was a Master's student in Computer Science. The subject rated their background, on a scale from 1 (very little) to 5 (very much), as follows:

- Knowledge of some natural science: 5,
- Engineering background: 3,
- Knowledge of DES control theory: 4,
- Experience using software for DES: 5,
- Experience using the IDES software: 4.

Using the same scale, they rated their comfort using English as 5. At the time of participation, one semester had passed since the subject was enrolled at the ELEC843 course offered by Dr. Rudie at Queen's University.

The observations of the experimenter were that during the study, the subject felt comfortable with thinking aloud, however, they tended to be very talkative and they often interrupted their problem solving to elaborate on their thoughts. In the beginning, the subject aimed their speech at the experimenter and often attempted to initiate a conversation. For this reason, the experimenter decided to forgo the opportunity to observe the actions of the subject directly and instead moved out of the field of view of the subject. This significantly reduced the interactivity in the speech of the subject. During the factory problem, the experimenter reminded them to keep talking 9 times; during the hospital problem—12 times. Thinking aloud at a low level of elaboration seemed to be difficult for the subject. Thus, it is expected that this subject's problem solving was impacted by the act of thinking aloud—at the very least, by decreasing the speed of the subject's performance. Furthermore, it is possible that the problem-solving strategy of the subject might have also been altered due to verbalizing. However, the results for this subject are also included in the study since they exhibited some unusual points of view and since this seems to be the subject with the least expertise. Thus, their approach is useful in illustrating problems less experienced users of DES tools may have.


Understand
Model
Input
Compute
Verify
Understand problem
Model B1
Verify TU
Understand problem
Model B2
Input B1
Compute K
Verify K
Model TU
Input B2
Understand requirements
Model K
Recompute K
Verify K
Modify events
Verify TU
Modify TU
Verify TU
Input TU
Compute SL
Verify SL
Strategize
Modify TU

The following abbreviations are used in the chart: TU – testing unit, B1 – buffer 1, B2 – buffer 2, K – monolithic control specification, SL – composition of testing unit with monolithic control specification.

Figure 4.9: Flowchart of the strategy used by subject 1 when solving the factory problem.

4.3.1 Factory problem

The strategy used to solve the problem can be seen graphically in Fig. 4.9. While the subject did not make a distinction between modules of the system and modules of the control specifications, they modelled the problem in a modular way. Then, they tried to combine the modules into a monolithic model. Before the time allotment for the session was over, the subject announced that they "give up" solving the problem and the session was terminated.

After receiving the description of the problem and examining it, the subject noticed that this problem is very similar to, if not the same as, the problem discussed in the ELEC843 course. They read the description of the problem and examined the supplied diagram. They made no additional diagrams and did not write down any notes.

The subject started modelling using pen and paper. First, they announced that they would model the machines mentioned in the description, but almost immediately they reinterpreted their actions as modelling the buffers from the problem. First, they modelled buffer 1 and then buffer 2. Then, the subject modelled the testing unit. Unlike modelling the buffers, modelling the testing unit seemed more demanding for the subject. After being done with this model, they made an overview of all the models they had, also checking if the problem mentions machines breaking down. Since this is never mentioned in the problem description, one can assume the subject was recalling the problem solved in the ELEC483 course where machines, indeed, could break down. They also made a guess that the main issue in this problem is the prevention of deadlock.

The next step of the subject was to compose the models of the buffers, announcing that they would use the operation "meet" (or *intersection*). They started composing the modules manually, however, soon they requested the use of the computer. After inputting the two buffers, the subject attempted to compose the models by using the "meet" operation. Repeated attempts did not produce the results the subject expected, e.g., the outcome was an automaton with a single state and no transitions. The subject, for a brief period of time, discontinued using the software and tried instead to "see the solution in their head". They analysed the problem and came up with the idea of keeping an empty slot in the first buffer. Soon afterward, however, they returned to their attempts to compose the two buffers using both pen and paper and software in an iterative process. The first issue they addressed was the question of whether the events have to be shared between the two models or whether the models should use different events. The subject recalled that this is a problem they also had in solving the hospital problem, however, they could not remember what they had decided at that time. Finally, they decided to use shared events. The second issue they addressed was the issue of self-loops in the model of control specifications. They added self-loops to the models of the buffers and, after another iteration, the result of the module composition was acceptable. The subject referred to the outputs of the DES operations as being "interesting" with various degrees. It seemed that being "interesting", or having a rich graphical representation (such as many states and transitions), was the criterion used to determine if a result was acceptable or not. Later, in the interview, the subject explained more specifically what cues they used to judge the validity of the models output by the software. They said that having only one or two states was an obvious indication that a model is incorrect. When there were more states, the subject looked at the transitions, following short traces to see if the traces lead to states in a way that made sense. In the model they accepted as correct, they noticed that all transitions seem to follow a pattern and to generally flow in the same direction.

When the composition of the buffers was obtained, the subject announced that would like to verify it. To that end, they wanted to obtain a paper copy of the computed model. A natural way to do it would be to print it. However, IDES does not provide this functionality and so the subject had to manually copy the model from the screen. During the copying, they seemed to pay extra attention to the structure of the model. Furthermore, they did not perform any verification activities after finishing the copying. Thus, it seemed that they verified the model during the process of copying.

In the next step, the subject wanted to consider the testing unit in their overall solution. They used pen and paper and the model of the buffers copied from the software. Again, the subject focused their attention on the problematic state during the execution of the system, where both buffers are full and a failed test of a part would result in a deadlock, the testing unit not being able to deposit the part in buffer 1. Then, they remodelled the testing unit in terms of the events used in the buffers. Choosing which events to use appeared very problematic for the subject. As well, they indicated that they were not sure how to "combine" the model of the testing unit with the model of the buffers. After finishing the model, they proceeded to inputting it into the software and composing it with the combined model of the buffers using “meet”. Then, they tried to interpret the result and see if it “makes sense”. The subject did not find the automatically generated layout helpful so they spent time moving nodes and transitions to make the layout more clear. They followed a trace of events from the initial state and soon concluded that “something is wrong.” They returned to examining their models designed on paper.

The subject decided to try remodelling the testing unit once again. While doing that, they proceeded to modify the model of the first buffer to include a separate event denoting the depositing of a part from the testing unit. Initially, there was only a single event for part depositing by both machine one and the testing unit. Then, their attention seemed to be exclusively focused on trying to figure out how to implement their understanding of how to solve the problem of deadlock when buffer 1 is full and the testing unit rejects a part. They examined all models, including already rejected models (such as their first model of the testing unit). After some time struggling to come up with a strategy, the subject discontinued solving the problem—well before the expiration of the session.

The event names used by the subject were abstract and were not based on the verbal descriptions of the events. For example, the event when machine 1 puts a part into buffer 1 was named “a” and the event when machine 2 puts a part into buffer 2 was named initially “c” (and later was renamed to “a”). Initially, the subject did not make any notes about what events were used in the models besides the occurrences of the events as labels of transitions. However, when they started modelling the testing unit, they wrote down a legend for the used events (in this case, from “a” to “f”). The use of abstract names did not seem to pose a problem for the subject, in the sense of causing confusion over which event is denoted by a given name. The controllability of events was not considered at all during the session.

During the modelling, the subject decided to correct or remodel some modules. Some of the changes to models were inspired by working on a different module and recognizing an inconsistency. For example, while working at the testing unit, the subject realized that the model for the first buffer did not have an event to denote the depositing of a part which has failed the test. However, most of the other modifications and remodelling occurred only in response to specific issues, such as the unsuccessful composition of two models. The subject did not appear to have much justification for their activities; it seemed that they were trying to recall what had worked in the past and then they applied it to the current situation. They tried using both shared and disparate events, substantiating their choice by “checking to see what the result would be”. Similarly, the addition of self-looped events in the models was considered when the subject remembered that this had been a successful solution in the past. It was not the description of the problem that led them to the conclusion that self-loops are necessary. Most of the difficulties the subject had and most of the modifications they did revolved around the events used in the models. It seemed the subject was not very sure how to model the dynamics of the system using events.

During the interview, the subject explained briefly their problem solving. The subject mentioned that they tried to model all subsystems, namely the buffers and the testing unit separately and then to compose them. They reiterated that the modular approach is very important to deal with difficult-to-model systems. They expressed confidence in their models of the buffers and the composition of the buffers, however, they confirmed that the testing unit was difficult to model. The subject said that they understand the problem that needs to be resolved (i.e., the potential for a deadlock when all buffers are full), however, they were not able to express it using DES theory. They explained that they realized where there could be a problem by considering the different cases of what could happen in the system. Their attention had been attracted, according to them logically, by the state when all buffers are full since this is the only case that could result in a deadlock (i.e., when the testing unit attempts to return a part for re-processing). The machines in the problem do not break down, so the flow of parts through the system would be fluid if there were no feedback loop from the testing unit. Having the current solution, the subject indicated that as a future step in the problem solving they would like to consider the controllability of events. As well, they would like to review the solution for the similar system, given as an example in the ELEC843 course.

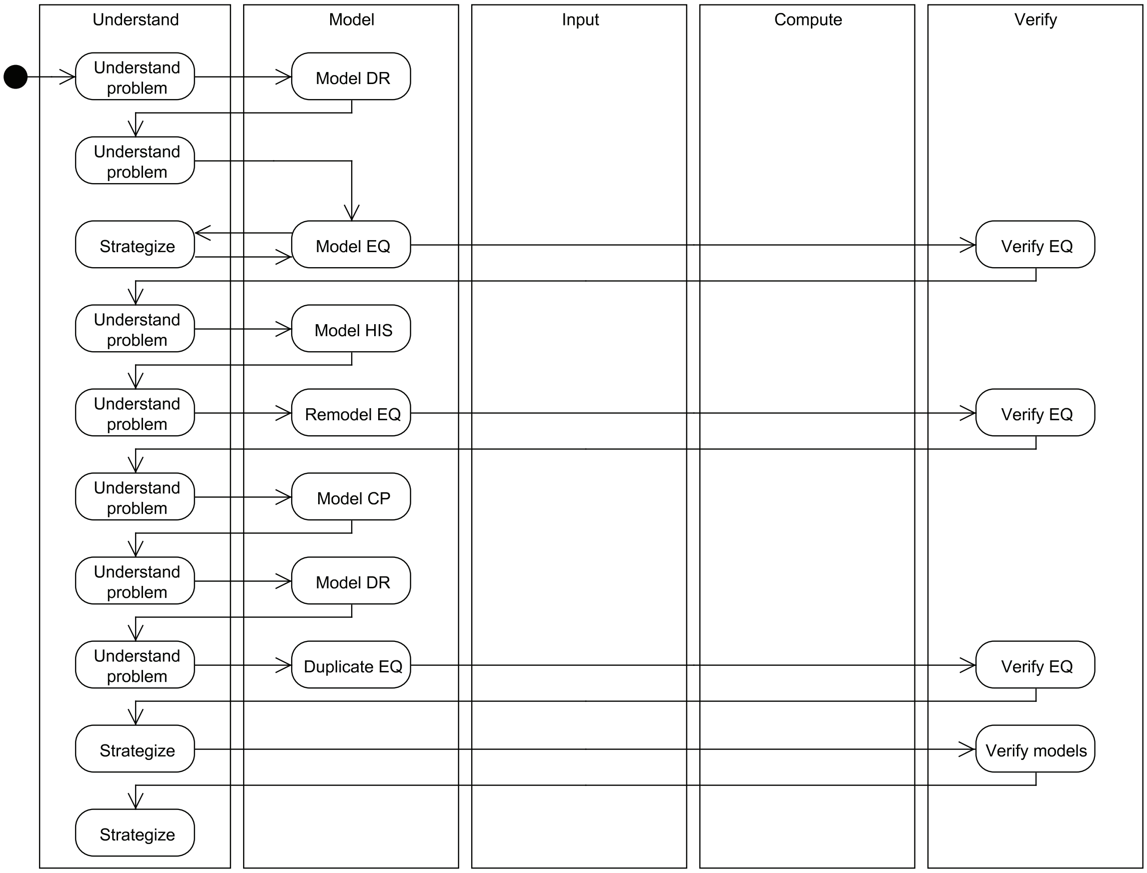
4.3.2 Hospital problem

The strategy used to solve the problem can be seen graphically in Fig. 4.10. The subject modelled only part of the system, in a modular way. They did not use the IDES software at any point. Before the time allotment for the session was over, the subject decided to stop solving the problem.

The subject started solving by reading the description of the problem and then taking notes, summarizing the points they found important. They focused their attention roughly in the following order: candy, computer, equipment, patient, equipment, doctor, grandmother, patient, report, sugar level.

For each model, the subject created a separate legend to keep track of the events that they used. They listed each new event as they started using it in a model. The event names used by the subject were abstract and were not based on the verbal descriptions of the events. For example, the event when the doctor gives a candy to the child was called “f”, the event when the computer transmits the information to the hospital information system was called “b”, etc. In general, the subject named the first event they started using in a model “a”, the second one “b”, and so on using consecutive letters from the alphabet. When a new event was listed in a legend, a short description of the event was included, e.g., “c – inject insulin”. The controllability of the events was not considered during the problem solving. Frequently, events would signify actions, such as “send data” or “order insulin injection”. However, there were also events signifying a state of the system, such as “blood sugar level outside limit” or “good report”; and even events such as “do nothing”.

The subject started modelling by drawing the model for the doctor. They interrupted working on this model to consider the dynamics of, and then model, the prototype equipment. After this, the subject decided to create a more high-level block diagram of the entities in the problem. They considered the prototype equipment, the computer and the hospital information system. Then, the subject modelled the hospital information system. The model seemed to be more of a sketch rather than a complete model, however, the subject moved on to review the role of the nurse in the problem. They did not produce any model for the nurse, but instead they started remodelling the equipment. They remarked that they will not consider marked states since the system is non-terminating. They proceeded by creating a


Understand
Model
Input
Compute
Verify
Understand problem
Model DR
Understand problem
Strategize
Model EQ
Verify EQ
Understand problem
Model HIS
Understand problem
Remodel EQ
Verify EQ
Understand problem
Model CP
Understand problem
Model DR
Understand problem
Duplicate EQ
Verify EQ
Strategize
Verify models
Strategize

The following abbreviations are used in the chart: DR – doctor, EQ – prototype equipment, CP – computer, HIS – Hospital Information System.

Figure 4.10: Flowchart of the strategy used by subject 3 when solving the hospital problem.

model for the computer and then shifted their attention back to the incomplete model for the doctor. The subject did not seem to interpret some of the requirements from the description of the problem correctly. In their model of the doctor, the doctor can access the information system before a report is available and only has to wait until the report appears. In the problem description, it is explicitly stated that the doctor should not access the information system if no report is available.

When the subject finished modelling the doctor, they briefly reviewed all the models and decided to copy the good ones onto a single page. They copied only the equipment model and then they again preoccupied themselves with trying to come understand the problem and what needs to be done to arrive at a solution. They complained about the complexity of the problem or, more specifically, with the amount of information that they need to consider. As an example, they mentioned that they had forgotten about the requirement to keep the level of sugar in the patient's bloodstream limited. They entertained the idea of having a "rollback" system to keep track of sugar and insulin. However, no formalizations were produced. The subject reviewed once again all the models they had and announced that they would try to establish if the models can "work together"—especially if the "meet" operation is applied. While they were examining the models, they came to the realization that they had not considered controllability and observability of events. Soon after, the subject conceded defeat and stopped solving the problem.

During the modelling, the subject decided to remodel only one module—that of the prototype equipment. It is not clear what exactly is the reason for this decision. The subject suddenly ventured into this task after examining the description of the problem. They did not seem to reference their previous model for the equipment at all. Furthermore, the subject initially designed two separate models for the equipment and for the computer. They did not seem to realize that it is sufficient to model the functionality of these two entities using a single, small model. However, there was some indication that, implicitly, the subject was aware that it is not important to consider the computer separately. They never referenced the model of the computer in their discussions and they did not include it in their review of the modules which were "ready". We feel this omission of the computer model cannot be explained simply by the fact that the computer was on a separate sheet of paper since the subject always examined this sheet as well.

During the interview, the subject explained briefly their problem solving. The subject mentioned that they tried to model each entity separately. However, after a while of working, they noticed that their approach was flawed. They should have spent more time trying to abstract away details of the system and they should have considered relationships between entities earlier on. Also, they recognized that they did not consider the grandmother in their solution, which was to them obviously an omission. The subject noted that they did not have problems recalling concepts from DES theory, however, they did not know how to use them in modelling. They recommended that courses teaching DES theory focus more on modelling specific systems. The two ideas put forth for how to achieve this were either to have a two-semester course, or to move the theory into an undergraduate course and then expect graduate students to have this prerequisite. As future, hypothetical steps in problem solving, the subject mentioned that they would like to input the models into software like TCT [9], define the controllability and observability of events, compose the models and examine them to see if all states are reachable and if there is a deadlock. In order to conclude that the problem has been solved, to them it was necessary to make sure that the solution is safe for the patient. However, the subject said that, in order to solve the problem, they would most probably have to review again the material from the ELEC843 course. Specifically, they would have to examine other problems which have been solved, e.g., the problem with machines (here the subject most probably referred to one of the factory problems in [66]) and they would have to study how events interact when models are composed.

4.3.3 Within-subject analysis

For this subject, the hospital problem was administered before the factory problem. Transfer of knowledge between the two problems was observed. They applied knowledge from solving the hospital problem to resolve an issue in the factory problem. More specifically, they recalled that shared events have to be considered when composing modules which work together. However, there was no indication that the subject noticed a deeper-running similarity between the two problems. During the final interview, the subject noted that they felt better about their solution to the factory problem in comparison to the hospital problem. They explained that the hospital problem seemed to have significantly more information to attend to and that it was the largest problem they had had to solve up to that point.

This subject did not discuss what their motivation was to use pen and paper or software to solve the DES problems. Our observation shows that the subject in both cases started problem solving by using pen and paper. Only when they needed to perform a DES operation, such as composition of modules, would they decide to use the computer. In the factory problem, the subject even tried first to compose two modules manually before requesting the use of the software. Furthermore, after obtaining the composition of modules, they preferred to transfer the new model from the software onto paper and to continue working with pen and paper. On the other hand, the application of a DES operation—selecting the relevant modules and then choosing the operation—was simple according to the subject. They appeared to use the software interface with confidence and they spent a portion of the problem-solving session working simultaneously with the software and with pen and paper with an apparently fluid workflow. In the interviews, the subject had comments about the software. They would have liked to have support for diagrams and to have a central repository of events from where events would be selected during modelling. Furthermore, the subject imagined software where hovering over event names, or zooming into small sections of a model, would reveal the descriptions for the events. In this way, short (and non-intuitive) event names would be used in complex models to reduce clutter but descriptions of the events will be available upon request.

The comparison of the problem-solving strategies used by the subject in the two problems indicates similarity only on the surface level. First, in both problems the subject used a modular approach to modelling instead of trying to create a monolithic model. Furthermore, in both problems the subject did not seem to clearly distinguish between subsystems and specifications for the control of the subsystems. The fact that sometimes specifications appear like real units may be contributing to the difficulty of making a distinction. For example, in the factory problem buffers are real entities, parts of the factory floor. However, in a DES system description they should be modelled as parts of the control specifications since they are passive units and only impose limitations on the units interacting with them. Beyond these similarities, the strategies employed by the subject are largely incomparable. The actions of the subject were predominantly opportunistic, i.e., they seemed to be motivated by what was possible to do at a given situation rather than by the procedure of a thought-out plan. It is clear that the subject was uncertain how to approach DES problem solving and they had difficulty expressing the situation described for a problem in terms of DES models. Most of the models they created do not have an interpretation helpful for solving the problem. Similarly, the events used in the models do not reflect good understanding of DES modelling methodologies. As a result, the attempts of the subject to compose models did not work out properly since correctly defined events are crucial for the success of the composition operation.

An examination of the models produced by the subject indicates that they have a different interpretation of finite-state automata than the one employed in DES design. In the DES framework, the states of a model represent specific configurations of the controlled system. The events on the transitions signify actions executed by the system or changes of the system configuration due to events in the environment, e.g., a water tank filling up above a threshold value. The subject, however, had a different interpretation of finite-state automata, much closer to that of process diagrams. In process diagrams, states (or boxes) signify steps or tasks of a process, e.g., "transport a part between two factory robots" or "fill up water tank". In this sense, the system may be executing a number of actions in order to complete the task. The transitions between states signify potential outcomes of a process step and lead to the steps that have to follow each respective outcome. For example, there can be a transition from "fill up water tank" to "water garden", labelled with "tank is full", and a transition from "fill up water tank" back to the same task, labelled with "tank is not full". In this sense, transitions are interpreted as states of the system and lead to the actions which have to be taken under different circumstances. This is different from the DES interpretation, where transitions are the actions which lead to different states. The models of the subject included states labelled, for example, "read report" and "send data" and transitions labelled, for example, "report is not available" and "part is present". This is completely contrary to how the system would be modelled as a DES, where "read report" and "send data" would be the transitions and "report is available" and "part is present" would be states. However, this confusion was not consistent in the models of the subject. In many cases, the models were much more along the lines of the DES approach. For example, in the buffer models, the states indicated how many parts are in the buffer and the transitions stood for the actions of depositing or removing parts from the buffer; or, in the model of the doctor, there would be events for the prescription of insulin or for giving candy to the patient. The subject occasionally employed in their models transitions labelled "do nothing" or "wait for…". These transitions usually led between states with different tasks but where no action has to be taken to move between these states. For example, in the model for the prototype equipment from the hospital problem, the subject drew a "do nothing" transition between the "send data" and "take reading" states (since the equipment can take a new reading after it sends

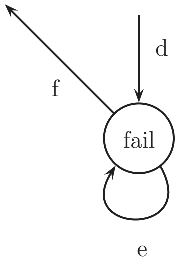


Figure 4.11: A reconstruction of a part of the model of the testing unit made by subject 3 during the factory problem. In the associated legend, the events were listed as follows: "d – part failed", "e – no space in buffer", "f – reject back to B1".

the data from the previous reading). The confusion between states and transitions not only resulted in models which could not be used in the DES framework. Apparently, sometimes it also led to incorrect interpretations of the behavior of the system. For example, during the factory problem, in one of the model of the testing unit, the subject had the structure shown in Fig. 4.11. According to this structure, and according to the verbal interpretation given by the subject, when a part fails the quality test, the testing unit has to wait until there is an empty spot in buffer 1 in order to deposit it there. This, according to the subject, would be the cause of a deadlock in the system. In the problem description, however, it is explained that one can only control when a part is picked up by the testing unit and that the depositing into buffer 1 cannot be prevented in case the part fails the test. Thus, in the system there will be a deadlock if and only if the control strategy, designed by the control engineer (in the case of our study—by the subject), results in preventing both machines and the testing unit from picking a part for processing.

Besides the problematic models, the way the subject dealt with the events in the system also contributed, in our opinion, to the failure of their problem solving. The first observation is that, especially in the hospital problem, the subject modelled each module completely separately. As they indicated in their interview, they failed to consider module interactions early on in the process of problem solving and subsequently they felt they would have to remodel everything—contributing to their decision to give up problem solving. A similar lack of understanding of how to couple interacting modules via shared events was demonstrated in the factory problem where the attempts of the subject to compose modules were consistently unsuccessful (and even in the cases when the subject believed they had the proper composition, it was incorrect). In our opinion, these problems were also, to a certain extent, due to the use of abstract event names by the subject. It can be observed that generic names (such as "a", "b", etc.) generally contribute to a more difficult process of problem solving since the subject has to maintain a mapping between arbitrary symbols and the significance of the symbols. More descriptive event names may serve as retrieval keys and make this task easier. During the factory problem, when the subject (correctly) observed that modules have to share event names, they (incorrectly) decided to make the events of the buffers identical. It seems that this mistake would have been more easily avoided if event names were more specific, e.g., containing information on which machine is executing the event (and thus, distinguishing the "put" event in buffer one from the "put" event in buffer two by virtue of which machine is depositing a part). During the hospital problem, the same mistake was made, inadvertently. Since the events in all modules were labelled with letters from the beginning of the alphabet, all modules shared events that should have been completely separate. Had the subject attempted any composition of modules, they would have ended with a meaningless model.

In terms of the speed and confidence of problem solving, there did not seem to be a marked difference between the two problems. Only the modelling of the buffers in the factory problem seemed to be accomplished with a degree of confidence. Considering that these two models were also closest to the expected, correct versions of the models, we believe that the subject had the best recollection of the buffer models from the problem that they had seen in the ELEC843 course. The quality of the solution deteriorated significantly for the rest of the models, presumably because they could not remember the other parts of the solution from the course. This interpretation is substantiated to some extent from a number of observations. For example, the subject indicated on a few occasions that they were thinking about the previously seen solutions and that they could not recall many details from the course material. Also, during the interview, they expressed interest in reviewing examples from the course. Overall, the models produced of the subject were incorrect and could not be used to correctly solve the problem. The subject did not work fast and frequently struggled with different issues. In both problems, they discontinued their solving before the session expired.

Lower-level activities of the subject also indicate patterns of similarity and difference. For example, when comparing the results of the n-gram analysis for the pen-and-paper part of problem solving (see Fig. 4.12), it is clear that the subject followed similar patterns of actions during both problems. When the subject modelled using pen and paper, they were likely to continue working on the same aspect of the model, e.g., when working on transitions they were likely to continue working on transitions. This is visible in the high absolute and relative ratios of the bigrams ‘TT’, ‘SS’, ‘EE’ and ‘MM’ (for the hospital problem). However, a general trend can be observed in the data from the hospital problem which indicates that the subject was less likely to work with a specific entity in bulk. For example, the absolute ratio of the bigram ‘TT’ is reduced while that of ‘ST’ and ‘TS’ is elevated, in comparison to the factory problem. Similarly, the relative ratios of the bigrams ‘SS’, ‘EE’ and ‘TT’ are lower. The only exception is the relative ratio of the bigram ‘MM’ which is higher for the hospital problem. This indicates longer periods in which the subject considered modules without mentioning building blocks (such as states and transitions). The bigram ‘MS’ has a very high relative ratio for both problems. This means that, after considering a module (or a number of modules in the case of the hospital problem), the subject very reliably proceeded working on the states of a model.

More differences can be observed when examining the results of the n-gram analysis for the factory problem, when solving with pen-and-paper or software (see Fig. 4.13). The first difference, as expected, is the fact that only when the software was used, the DES-algorithm-related bigrams (including the code ‘C’) have non-zero ratios. For both stages of problem solving, the relative ratios show that continuing working on the same aspect of the problem is very likely, with the exception of the use of computational algorithms. The relative ratio of the bigram ‘CM’ is much higher than that of the bigram ‘CC’. Furthermore, when working with the software, the subject was much more likely to continue examining a module once they started looking at it (as shown by the high absolute and relative ratios of the bigram 'MM', and by the large decrease in the relative ratio of the bigram 'MS'). The subject used the software mainly to compose modules and they were struggling to determine how to do it properly. Thus, it is not a surprise that the data indicate a workflow where first a DES operation is called and then the result is extensively examined.

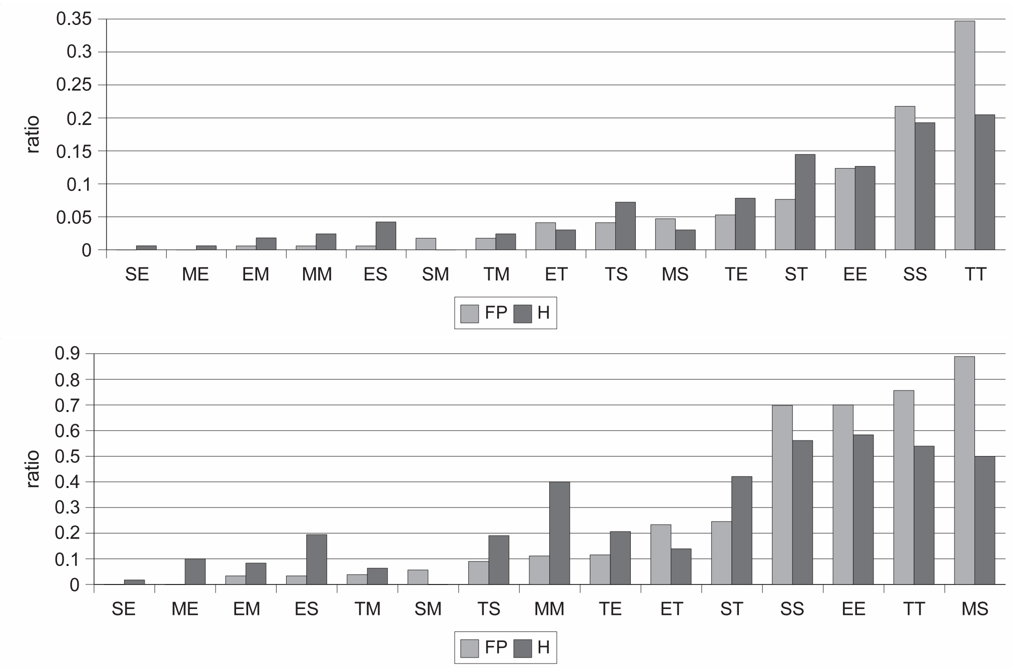


Figure 4.12: The absolute and relative ratios of bigrams for pen-and-paper problem solving during factory problem (FP) and the hospital problem (H). Data are sorted according to the ratios in the factory problem.

The attention of the subject throughout the problem solving was analysed as described in Section 3.7. During the factory problem, the subject did not report almost any perception-triggered data discovery. Cognitively-driven data discovery concerned mostly relationships between modules and the dynamics of modules. The functionality of the software was also frequently of interest. This may be largely due to the fact that the subject was struggling with the composition of modules. The subject voiced intentions to collect the data using mostly visual inspection or counting. When counting, they counted states and transitions. However, the most common method the subject used to discover information was to ask the experimenter. As already mentioned, this heightened level of interaction was also observed by the experimenter. The cognitive-driven data discovery frequently concerned modules other than the one that the subject was working on, e.g., there was cross-context reference between the testing unit and the buffers and between the testing unit and the composition of the buffers. This indicates that the subject was actively searching for disparate pieces of information. This is consistent with the observation that the problem solving approach of the subject was mostly opportunistic. During the hospital problem, the subject did not

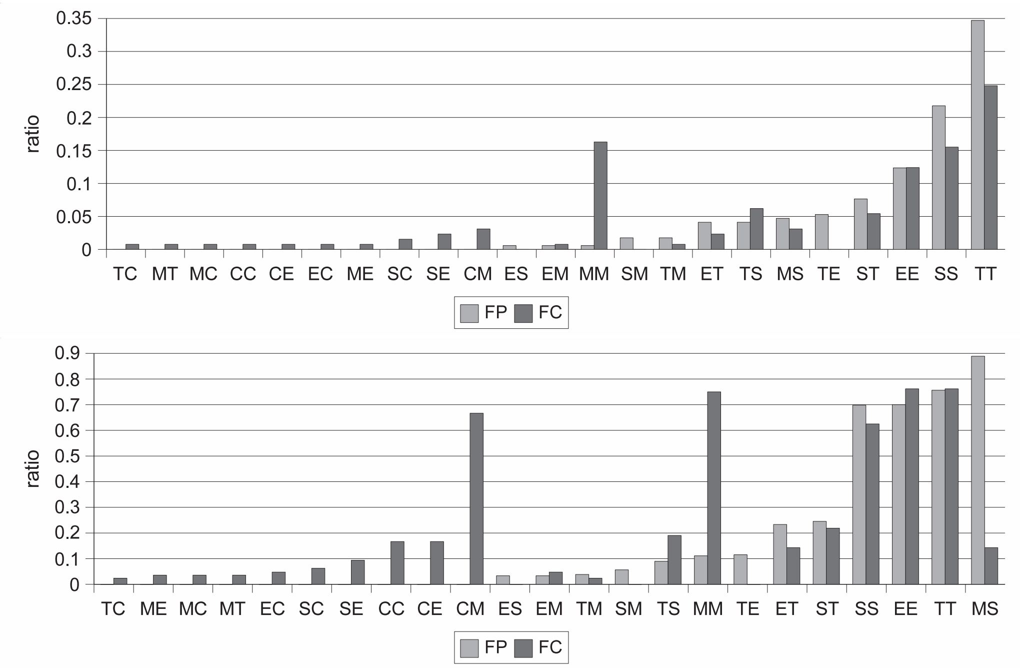


Figure 4.13: The absolute and relative ratios of bigrams for pen-and-paper problem solving (FP) and solving using software (FC) during the factory problem. Data are sorted according to the ratios in pen-and-paper solving.

report almost any perception-triggered data discovery either. The reported discovery usually concerned the problem description and they were made while working on the model of the prototype equipment. There were only a limited number of occasions when the subject voiced their intention of cognitively-driven data discovery. The most frequent item of interest was the relationship between modules. The methods for data collection usually involved reading the description or asking the experimenter. There was no compelling evidence for cross-context data discovery.

The analysis of the visual attention of the subject revealed the following facts. During the factory problem, the subject did not attend to the problem description very frequently, except when modelling the testing unit. The following other patterns emerged. When modifying the specification for buffer 1, the subject paid attention to the sheet of paper with the latest version of the model for the testing unit. When modifying the specification for buffer 2, they paid attention to the composition of the two buffers—both as displayed on the computer screen and as re-drawn on a sheet of paper. When composing the models of the buffers using the software, the subject attended to the sheet with the models of the buffers and the testing unit. When remodelling the testing unit, the they referred to the sheets with the models of the buffers, the composition of the buffers, and the initial model of the testing unit. During the hospital problem, the subject's attention was attracted more frequently by the problem description, however, most of these references occurred during the modelling of the module for the equipment. The following other patterns emerged. When remodelling the module of the computer, the subject paid attention also to the sheets of paper with the models of the doctor, equipment and hospital information system. When modelling the module for the doctor, they paid attention to the sheets with the models of the equipment, the computer, and the summary of the problem description. When remodelling the equipment, they paid attention to the sheets with the previous models of the equipment, the models of the doctor, hospital information system, computer, the summary of the problem description and the diagram the subject had created. They referred to this diagram also during the modelling of the hospital information system and the initial modelling of the equipment.

4.4 Subject 4

At the time of their participation, subject 4 was a Master's student in Computer Science. The subject rated their background, on a scale from 1 (very little) to 5 (very much), as follows:

- Knowledge of some natural science: 4,
- Engineering background: 2,
- Knowledge of DES control theory: 4,
- Experience using software for DES: 4,
- Experience using the IDES software: 4.

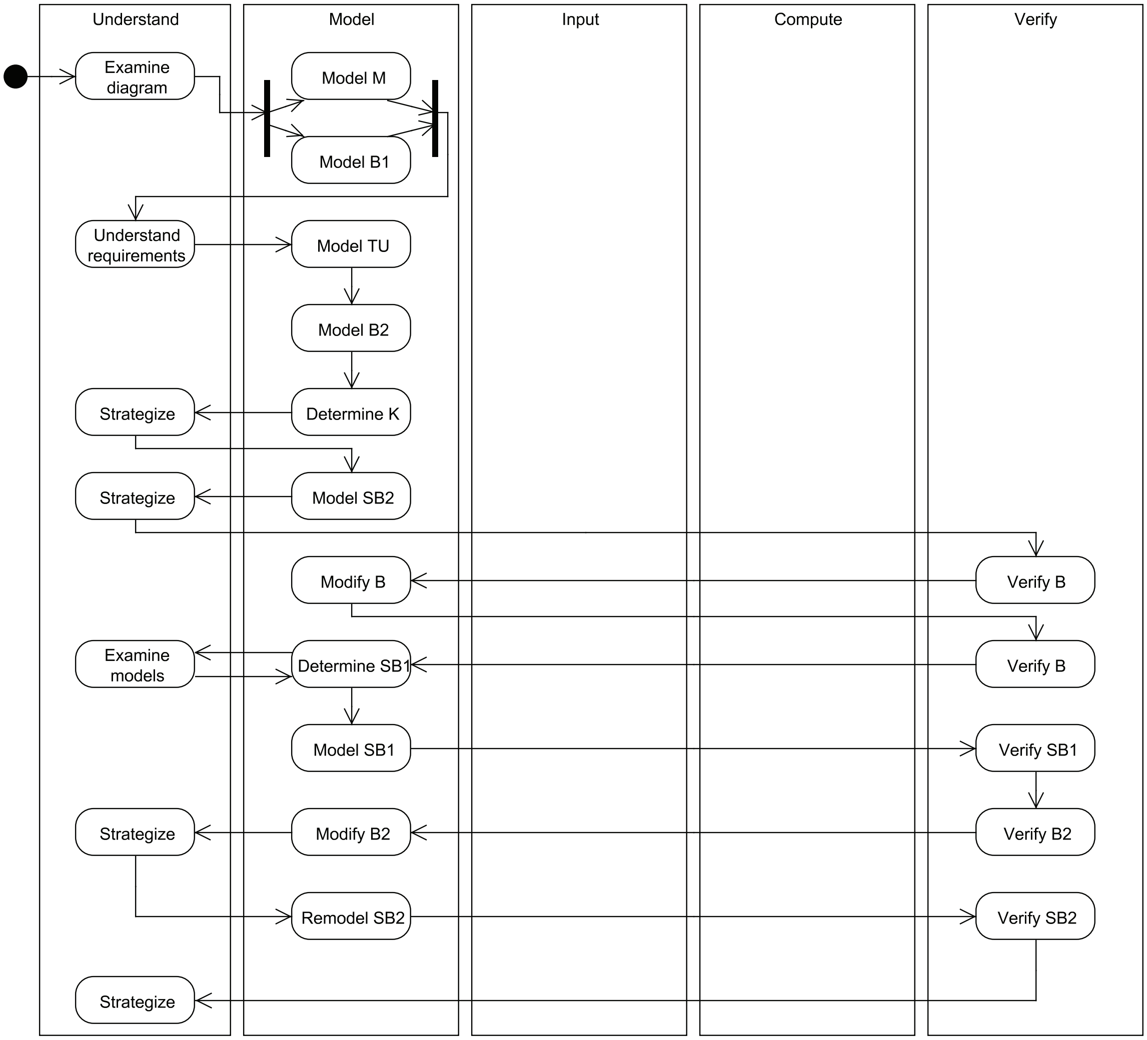
Using the same scale, they rated their comfort using English as 5. At the time of participation, one semester had passed since the subject was enrolled at the ELEC843 course offered by Dr. Rudie at Queen's University.

The observations of the experimenter were that during the study, the subject did not feel entirely confident thinking aloud. It seemed that they talk due to the requirements of the study (i.e., they were asked to think aloud) instead of using natural flow of speech. During the factory problem, the experimenter reminded them to keep talking three times; during the hospital problem no reminders were given. The impact of thinking aloud on the problem solving of the subject was hard to estimate. The impression is that the process of problem solving exhibited by the subject might have been more structured than it would normally be since the subject proceeded slowly and did not make many corrections. The subjective impression of the subject was that thinking aloud was not as difficult as they had anticipated.

4.4.1 Factory problem

The strategy used to solve the problem can be seen graphically in Fig. 4.14. The subject modelled both the system and the control specifications in a modular way. They started designing modular supervisors manually, however, time ran out before they could finish their work.

Upon receiving the description of the problem, the subject read it and immediately ventured into modelling the system. They made no additional diagrams and did not write down


Understand
Model
Input
Compute
Verify
Examine diagram
Model M
Model B1
Understand requirements
Model TU
Model B2
Strategize
Determine K
Strategize
Model SB2
Modify B
Verify B
Examine models
Determine SB1
Verify B
Model SB1
Verify SB1
Strategize
Modify B2
Verify B2
Remodel SB2
Verify SB2
Strategize

The following abbreviations are used in the chart: M – machines 1 and 2, TU – testing unit, B1 – buffer 1, B2 – buffer 2, B – buffers 1 and 2, K – monolithic control specification, SB1 – supervisor for buffer 1, SB2 – supervisor for buffer 2.

Figure 4.14: Flowchart of the strategy used by subject 4 when solving the factory problem.

any notes. Apparently reading was enough for them to gain understanding of the problem. The subject did not mention that they had recognized any similarity between this problem and the problem that had been discussed in the ELEC843 course. However, during the interview at the end, they said that they remembered the problem from the course and they were trying to recall how the solution should proceed.

The subject used exclusively pen and paper to model the DES modules and, later, the supervisors for the system. The modelling proceeded at a slow but steady pace and the subject did not need to correct their work very often. They began by considering how to model the first buffer and machine 1. However, soon their attention was subsumed by the events they should use—what names to use and which ones are controllable. These issues were the most significant uncertainties experienced by the subject during the modelling. After the issues were resolved, the subject modelled machine 1 and then, instead of creating another model for machine 2, they copied the model, generalizing it for both machines using indexed events. The subject considered marking of states in all modules during the modelling. When working on buffer 1, the subject wondered what the requirements are for the marking in the specifications and they re-examined the problem description. They did not seem to find the information they needed so they decided to mark the state where the buffer is empty. When marking states in buffer 2, they referred to their decision with buffer 1 and did the same to "be consistent". A similar justification was given for the general design of buffer 2, which indicates that the subject was reusing their design decisions.

The event names used by the subject were abstract and were not based on the verbal descriptions of the events. For example, the event when machine 1 takes a part for processing was named "α_1", the event when machine 2 puts a part in the output buffer was named "β_2", the events when the testing of the quality of a part by the testing unit is positive or negative were named "β_3" and "ρ", respectively. The only potential argument for some influence of speech on the event names could be the event "ρ" where the Greek letter could stand for the first letter of the word "reject" (as "reject" was frequently used by the subject in their verbal references to "ρ"). Initially, the subject did not make any notes about what events are used in the models besides the occurrences of the events as labels of transitions. However, as they introduced more events in their models, they wrote them down. After finishing all models, the subject amalgamated all events that had been used into a separate list where their controllability was noted as well. The use of abstract names did not seem to pose a problem to the subject, in the sense of being confused about which event is denoted by a given name. In the interview they mentioned that they were coming up with event names as needed. However, they tried to use a scheme which helped them remember the names. They said that they became completely comfortable with the event names half way into the session.

After the basic modelling, the subject proceeded in their problem solving by building supervisors for the system manually. They tried to recall what the solution had been in the problem they had seen in the ELEC843 course. They decided to create modular supervisors (a separate supervisor for each specification). One point of concern they had was whether during the modelling they had managed to separate the control requirements from the system model.

The subject started modelling the supervisor for buffer 2 because, in their words, this supervisor should be easier as the feedback from the testing unit does not have to be considered. This fact had been pointed out in the ELEC843 course and the subject could have recalled it. Before designing the finite-state representation, they wrote down a more formal version of the requirements from the problem description and they drew a small diagram of the relevant modules (basically replicating a subsection of the diagram provided with the problem). While drawing the finite-state model, they remarked a number of time that the task is similar to drawing the intersection of the models for machine 1 and buffer 2. The modelling task made the subject question again if there is enough separation between their models of the specifications and the system. From the verbalizations of the subject and from their subsequent modifications of the models of the buffers it became clear that they were confused about the interpretation of the buffer models. The subject thought that they had embedded the control decisions into the buffer models since these models did not describe what is physically possible in the system. E.g., the model for buffer 2 did not reflect that it is physically possible for machine 2 to deposit a part into it after one part has already been deposited (the buffer has a capacity of one). As a result, the subject modified the buffer modules to indicate potential evolutions which lead to "bad" (or undesirable) states. In doing this, the subject forfeited the purpose of having separate buffer modules since the only purpose for having them is to model the specifications for the system. When all possible system behavior is modelled, implicitly no restrictions are imposed on the system. Apparently, the subject experienced doubts about the role of the buffer modules and gradually converged onto considering them as subsystem modules.

After the subject modified the models of the buffers to include all possible system behavior, they focused their attention on the supervisor for the first buffer. They first labelled with events the connections in the diagram provided with the problem description. This activity did not appear to be essential to the problem solving and seemed to happen only so that the subject could get a "mental break" from the more demanding task of modelling. The subject proceeded by writing down a more formal version of the requirements from the problem description, just as they did before modelling the supervisor for buffer 2. However, this time they were more detailed. They announced they would first look at the task of preventing buffer overflow. They came up with the following equation, similar to what had been discussed in the ELEC843 course:

$$0 \leq \#\beta_1 - \#\alpha_2 + \#\alpha_3 - \#\rho - \#\beta_3 \leq 3,$$

where # stands for the number of occurrences of a given event, β_1 is the output of a part by machine 1, α_2 is the pickup of a part by machine 2, α_3 is the pickup of a part by the testing unit, ρ is the rejection of a part by the testing unit (the part is deposited back into buffer 1), and β_3 is the successful test of a part (the part is delivered to the environment). This equation will prevent the overflow of buffer 1, however, it will not be able to prevent a deadlock in the system if used concurrently with the specification for buffer 2. For example, if both buffers fill up (there are four occurrences of β_1 and one occurrence of α_2), the event α_3 will be disabled, however, so will be α_2 by the other specification. Thus, the processing of parts will halt. The subject experienced difficulty designing the supervisor for this specification; as well, they ended up considering both overflow and underflow in their model. After struggling with the supervisor for some time and not knowing how to complete the model, they again shifted attention to the control for buffer 2. They realized that the supervisors for the two buffers will interact, or "conflict" in the words of the subject, through the event α_3 (the testing unit picks up a part). The subject modelled one more version of the supervisor for buffer 2 but their model was not capable of preventing buffer overflow. It seems that the subject did not consider all events necessary for such control.

The subject apparently timed their progress since they completed a brief overview of all models they had produced as the time for the session expired.

In summary, during the modelling, the subject made only two significant changes to their work. First, when they started modelling the supervisors, they reinterpreted the purpose of the buffer modules. Consequently, they modified to models of the buffers to include all possible system behavior. Second, after attempting to model the supervisor for buffer 1, the subject felt they had not addressed the specifications for buffer 2 properly. Thus, they ventured in remodelling the supervisor for buffer 2.

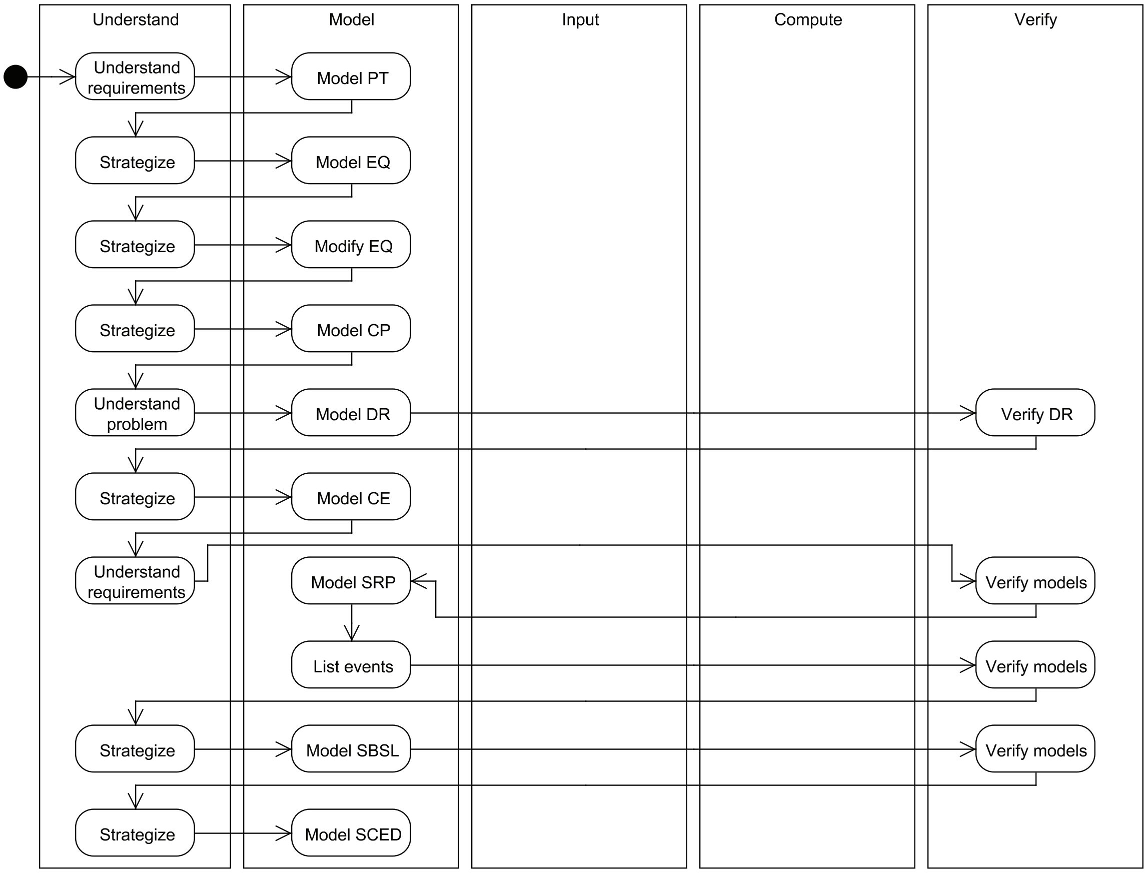
During the interview, the subject explained briefly their problem solving. The subject mentioned that they started by modelling the different components of the system and they tried to model what is physically possible to occur. They explained that this approach had been taught in the ELEC843 course and that the models of what is physically possible may include behavior which needs to be restricted. Similarly, the subject had tried to describe the control requirements also according to what they had seen before—using equations and event counts. They felt their solution is incomplete and they expressed lack of confidence in the way they had modelled what is physically possible in the system. Having the current solution, the subject indicated that as future steps in the problem solving they would like to design modular supervisors to prevent underflow and overflow of the buffers, and then to merge the supervisors—after checking background literature on how to do this step. These intentions show that the subject self-evaluated the models for the supervisors as incomplete and/or incorrect.

4.4.2 Hospital problem

The strategy used to solve the problem can be seen graphically in Fig. 4.15. The subject modelled the system and the supervisors in a modular way. They did not use the IDES software at any point. At the very end of the session they indicated they would like to use it for their next problem solving steps, however, the session was over.

The subject started by reading the description of the problem and then highlighting different parts of the text—mostly relating to the control specifications for the system. After finishing reading, they started modelling.

The subject used exclusively pen and paper to model the DES modules and, later, the supervisors for the system. The modelling proceeded at a slow but steady pace. The subject started by modelling the module describing the intake of glucose and insulin by the patient. They considered what is physically possible in the system and their model included undesirable states where the level of blood sugar is too high or too low. However, at the end, they drew a box around the model which delimited the acceptable states. From the verbalizations


Understand
Model
Input
Compute
Verify
Understand requirements
Model PT
Strategize
Model EQ
Strategize
Modify EQ
Strategize
Model CP
Understand problem
Model DR
Verify DR
Strategize
Model CE
Understand requirements
Model SRP
Verify models
List events
Verify models
Strategize
Model SBSL
Verify models
Strategize
Model SCED

The following abbreviations are used in the chart: PT – patient, DR – doctor, EQ – prototype equipment, CP – computer, CE – composition of computer and prototype equipment, SBSL – supervisor for the blood sugar level, SRP – supervisor for the report, SCED – supervisor acting on the composition of computer, prototype equipment and doctor.

Figure 4.15: Flowchart of the strategy used by subject 4 when solving the hospital problem.

of the subject, it seems that they considered their model a part of the system description. In fact, it is not necessary to model the behavior of the patient unless the model is used to specify how the system should be controlled. In some sense, the subject accomplished that by delimiting the desired behavior. However, their approach indicates that they had trouble distinguishing subsystem and specification modules and that they might have had a misunderstanding of how control specifications are given.

After modelling the patient, the subject proceeded by modelling the components of the system. They considered modelling the prototype equipment and the computer together, however, they decided to first model them separately. No explanation was given for this decision but during the modelling, the subject occasionally indicated that they keep in mind their intention to compose the models. For example, they purposefully used the same event names in both models. The design of the module for the equipment did not prove to be an easy task and the subject had to re-examine the problem description on a number of occasions. The model they produced did not describe precisely the situation in the problem. In the model, insulin could be administered without taking a reading from the blood which contradicts the problem description. However, the subject justified this choice by claiming that the description does not explicitly mention that insulin injection and taking a reading always occur together. The design of the computer model also seemed demanding. The subject frequently referred to the problem description. They also made notes to clarify the specifications for the report generated by the computer. In their notes, they referred to underflow and overflow of a “report buffer”. While this is not an explicit reference to the factory problems the subject had had experience with, one could hypothesize that that experience had an impact on the interpretation of the subject. A further indication of previous experience influencing the subject was displayed in their interpretation of the behavior of the prototype equipment. More specifically, they supposed that the software controlling the equipment is not multi-threaded (does not allow concurrent executions of tasks) and this is the reason why the equipment is non-responsive before completing the transmission of a reading. While this guess is reasonable, nothing in the problem description would imply a specific reason for the no-responsiveness of the equipment.

The modelling of the module for the doctor was relatively fast and the subject properly interpreted the problem description: the prescription of medication is irrelevant to the rest of the system; it does not equal the administration of insulin. The subject proceeded by analysing what other modules they had to model. They considered the role of the nurse but decided that they will not make a separate model. This is consistent with the expected solution since the nurse does not influence the behavior of the system in any way. Next, the subject considered the role of the grandmother. They recognized that her visit would result in an increase of the blood sugar level of the patient but such a visit is preventable. The subject thought about incorporating the behavior of the grandmother into the model of the patient but they did not complete this task. Instead, they decided to create the composition of the computer and equipment models. The subject performed the composition correctly, however, they named the operation incorrectly (they referred to “meet”—or intersection—while the operation is called “parallel composition”). During the follow-up interview, they were asked for a clarification of what “meet” means to them and it was discovered that, indeed, the subject referred to the correct operation and they only mixed up the terminology.

After modelling the system modules, the subject reviewed their models and all control requirements in the problem description. Then, they began modelling the modular supervisors, starting with the supervisor which should guarantee that the doctor accesses the reports in the hospital information system correctly. The design of this supervisor also resulted in the subject trying to determine the controllability of the events. Due to the subject's beliefs about the controllability of the events, they discovered that the supervisors for the report and, later on, for the blood sugar level were very simple. A further review of the subject determined that the patient module under the control of the blood sugar level supervisor is identical to the behavior described by the supervisor. The subject then focused their attention on building the model of the equipment, computer and doctor modules under the control of the supervisor for the report. They reasoned that these specific modules need to be considered because they have events which the supervisor uses as well. The subject admitted that doing the composition of the system modules and the application of the supervisor manually is feasible only because the models are small. They did not manage to complete the task, however, since the time for the session expired.

During the modelling, the subject did not make any notes about what events are used in the models besides the occurrences of the events as labels of transitions. The event names were descriptive enough to make their semantics obvious. For example, the event when the child eats a candy and the level of glucose in the blood increases was called “glucose”, the event when the computer transmits the information to the hospital information system was called “transmit report”, the event when the doctor read the report was called “get report”, etc. The controllability of the events was not considered initially. Only after the subject started designing the supervisors did they start considering event controllability. This was also the time when they decided to create a legend with all events used in the models and the corresponding controllability information. The subject did not correctly identify all relevant events in the system, as well as they did not interpret their controllability correctly. For example, they had separate events for the administration of insulin, the start of transmission from the equipment to the computer and the end of transmission. Since the administration of insulin and the transmission of data always happen together and nothing can interrupt this process (the equipment is non-responsive), these events can be amalgamated into a single one. The incorrect interpretation of event controllability also had implications for the control solution proposed by the subject. The solution turned out to be more or less trivial.

The subject did not generally correct or remodel their models during the problem solving. There were only three exceptions, where gradual advancement in the modelling of a module made the subject realize that they need fewer states or fewer transitions—thus necessitating changes to the part of the model they had already created. In the case of the module for the equipment, the subject preferred to simply start over.

During the interview, the subject explained briefly their problem solving. The subject mentioned that in the beginning the problem seemed larger than the factory problem but it turned out to be manageable. They started by trying to model the behavior of different subsystems. Only after did they look back and consider what events were used. After building the list of events, no problems were discovered in the models so there was no need to go back and make modifications. Having the current solution, the subject indicated that as a future step in the problem solving they would like to finish applying the modular supervisors to the system. Then, they would like to merge the controlled system to check for conflicts. They did not expect any conflicts since, to them, the control requirements seemed orthogonal. In case there were conflicts, the subject would prefer to first consult background literature to determine how to fix this problem, however, they hypothesized that changes to the supervisors would be necessary.

4.4.3 Within-subject analysis

For this subject, the factory problem was administered before the hospital problem. The subject did not mention noticing a degree of similarity between the two problems during the interviews, however, there was some indication of knowledge transfer between the two. For example, when interpreting the problem description, the subject referred to the requirements for the report in the hospital information system as the "underflow and overflow" of a "report buffer". This terminology is not normally expected in the context of the problem, but fits well with the terminology of the factory problem.

The subject did not use the provided software at all during problem solving. They performed the required module compositions manually and even designed the supervisors for the system by hand (while the software package offers an automated supervisor construction algorithm). When explaining their decision to work manually, the subject explained that the systems seemed small and manageable. According to them, using the software package requires a lot of preparation, such as inputting all models and typing in all events, so it did not seem worthwhile to do all these steps for the given systems. The subject said that in the past they had used the software only for large systems. However, the subject indicated that they would use the software to check their final solutions.

Roughly speaking, the subject used a similar problem-solving strategy in both problems. After getting acquainted with the problem description, the subject modelled the components of the system, summarized all events and their controllability, and then attempted to model the supervisors to enforce the control requirements. In both cases the subject seemed to be confused about how control specifications should be modelled, instead creating models which are interpreted as parts of the system functionality. In this sense, the subject did not have to distinguish between subsystems and specifications—and they created models in the order of how the models relate and interact.

A small difference in how they solved the two problems is that the subject did not consider marking of states during the hospital problem.

The speed and confidence which the subject manifested was approximately the same during the solving of both problems; and the subject advanced almost equally. However, there are more pronounced differences in the quality of the two solutions. In the factory problem, the models produced by the subject were correct, save for some minor issues. However, the subject did not succeed in formalizing properly all specifications and, subsequently, they experienced great difficulty in designing the supervisors for the system. The design of modular supervisors is not a trivial task and, as the subject correctly observed, in this problem there was some interaction between the supervisors due to the feedback from the testing unit.

In the hospital problem, the subject seemed to advance a bit further, e.g., they completed the design of two supervisors and started applying them to the system. However, it can be argued that the apparent advantage is not due to a lower inherent level of difficulty of the hospital problem. Indeed, the performance of the subject contradicts the expectation since the hospital problem is theoretically of the same difficulty as the factory problem, however, the novel setting and the more ambiguous presentation should make it more difficult to solve. The observed advantage, we believe, stems from the fact that the subject did not model all components correctly and, even more importantly, from the fact that they assigned incorrect controllability to the events in the models. By interpreting some key events as controllable, the supervisory solution was reduced to a triviality. This problem illustrates not only the well-known fact that the correct interpretation of controllability of events is essential to DES problem solving. It shows that making this interpretation from an informal, verbal description of a non-trivial problem can be very challenging. The subject did not make any explicit error of judgment when determining event controllability in the hospital problem; rather, uncertainties and small mistakes led incrementally to an incorrect interpretation of the text and simultaneously to a high confidence in the subject. It seems that the only way to avoid such an escalation is to conduct consultations with the commissioner of a given problem.

Lower-level activities of the subject indicate patterns of similarity. The results of the n-gram analysis show that during modelling the subject was likely to continue working on the same aspect of a model, e.g., when working on transitions they were likely to continue working on transitions. This is visible in the high absolute and relative ratios of the bigrams 'MM', 'TT', 'SS' and 'EE' (see Fig. 4.16). Most pronounced is the effect in the bigram 'EE' whose relative ratio is the highest for both problems. This indicates that the subject tended to specify events in bulk, which is confirmed by the observations. While most of the bigrams have similar ratio distributions for the two problems, one difference stands out. During the factory problem the subject was more likely to work on states in bulk than during the hospital problem. In the hospital problem, the absolute an relative ratios of the bigram 'SS' are lower, while the absolute and relative ratios of 'ST' are higher—indicating that work on states was more likely to precipitate in working on transitions. This may be the result of the subject being less certain of which states a model should consist—and thus considered states incrementally, as opposed to an uninterrupted sequence of state-related actions.

The attention of the subject throughout the problem solving was analysed as described in Section 3.7. During the factory problem, there were a limited number of occasions of perceptually-triggered data discovery, without an apparent pattern. Cognitively-driven data discovery concerned predominantly events and the dynamics of modules. In the cases when the method of discovery was announced, most frequently the intention was to collect data by visual inspection. The subject was most interested in learning about other components when working on the buffers. When working on buffer 2, they were interested in the overall dynamics of the system and the control specifications. When working on buffer 1, they were interested in the testing unit and the control specifications. This cross-context interest is even stronger since during work on the control specifications, the subject was also interested in buffer 1 and the testing unit. From these observations we can conclude that, as could be

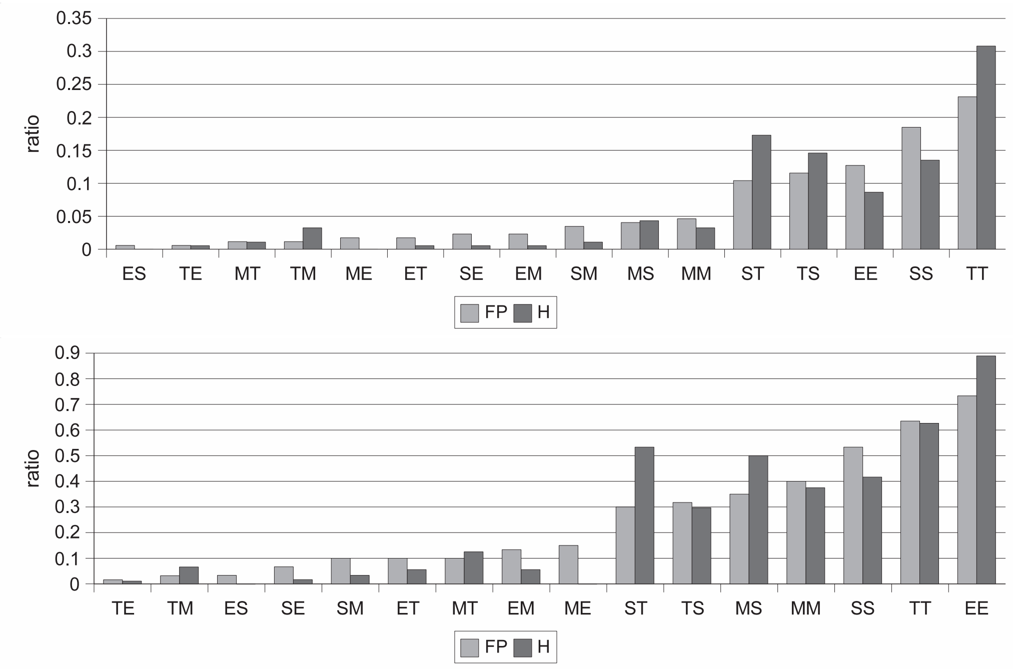


Figure 4.16: The absolute and relative ratios of bigrams for pen-and-paper problem solving during the factory problem (FP) and the hospital problem (H). Data are sorted according to the ratios in the factory problem.

expected, the subject thought of the buffers and the related components as tightly interacting units. During the hospital problem, there were more instances of perception-driven data discovery but not enough to establish any specific patterns. It seems that the subject's attention was attracted by perceptions about the properties of events and about the relations between modules. On occasions, the discovered data would concern parts of the system different from what the subject was working on (e.g., information about the nurse while working on the module of the doctor), however, no stable pattern was discovered. The cognitively-driven data discovery concerned mostly states and events. When counting, the subject counted states, transitions or modules. Again, discovery was targeted most frequently for aspects of the current context. Only when working on the module of the computer did the subject express substantial interest outside the context. More specifically, they sought information about the modules of the equipment and the doctor. Similar to the results from the factory problem, these results are not surprising as these modules are related.

The analysis of the visual attention of the subject revealed the following facts. During the factory problem, the problem description was by far the most common target of visual attention, during the modelling of all modules. The following other patterns emerged. When modifying the models of the specifications for the two buffers, the subject paid attention also to the sheet of paper with the models of the supervisors for the buffers. Conversely, when modelling the supervisors for the two buffers, the subject also paid attention to the sheet with the models of the specifications of the buffers and the models of the machines and the testing unit. During the hospital problem, the subject's attention was, again, attracted most by the problem description, however, not in such a pronounced way as during the factory problem. The other patterns which emerged were expected. When modelling the supervisors for the blood sugar level and for the report, the subject frequently paid attention to the sheet of paper with the models of the patient, equipment, computer and doctor. When modelling the composition of the equipment, computer and doctor modules under the control of the supervisor for the report, they paid attention to the other sheets with all the models and the supervisors.

4.5 Subject 5

At the time of their participation, subject 5 was a PhD student in Computer Science. The subject rated their background, on a scale from 1 (very little) to 5 (very much), as follows:

- Knowledge of some natural science: 5,
- Engineering background: 1,
- Knowledge of DES control theory: 5,
- Experience using software for DES: 4,
- Experience using the IDES software: 3.

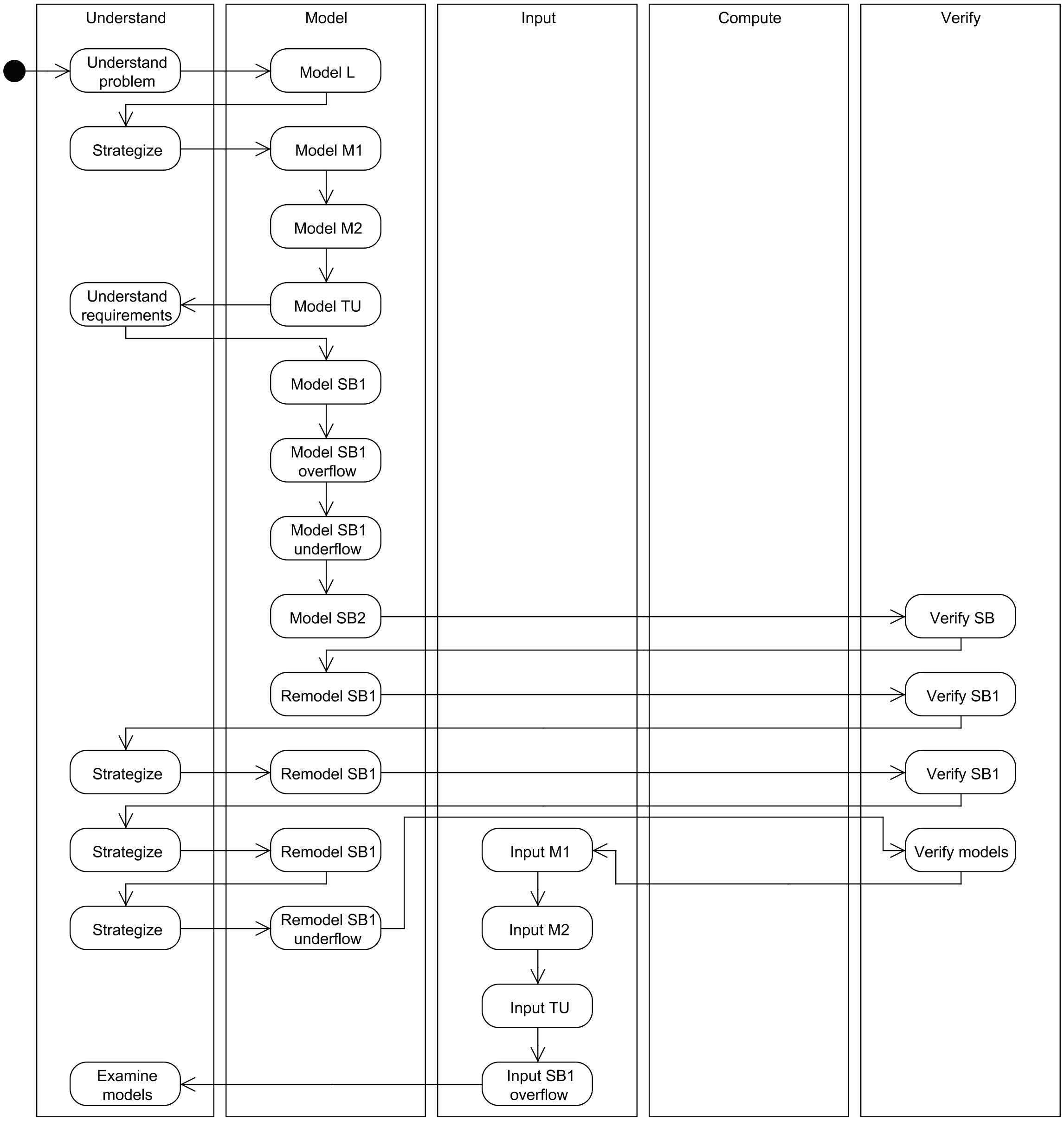
Using the same scale, they rated their comfort using English as 5. At the time of participation, more than one year had passed since the subject was enrolled at the ELEC843 course offered by Dr. Rudie at Queen's University.

The observations of the experimenter were that during the study, the subject did not feel comfortable with thinking aloud. Most of their verbalizations were the pronunciations of elements of thought which already exist in verbal form, such as event names, state labels, etc. During the factory problem, the experimenter reminded them to keep talking 7 times; during the hospital problem—7 times as well. The subject was not talkative, however, the experimenter felt that the interruption inflicted by a reminder was quite high and thus used restraint in reminding the subject to keep talking.

4.5.1 Factory problem

The strategy used to solve the problem can be seen graphically in Fig. 4.17. The subject modelled the system in a modular way and then designed modular supervisors to enforce the control requirements. All modelling was done manually. At the end the subject started inputting their solution into the IDES software to verify it. However, the time for the session expired before they could complete their work.

Upon receiving the description of the problem, the subject read the description and wrote down some events on the supplied diagram. However, they did not spend much time on this task and, soon, started modelling the system. The subject did not mention that they


Understand
Model
Input
Compute
Verify
Understand problem
Model L
Strategize
Model M1
Model M2
Understand requirements
Model TU
Model SB1
Model SB1 overflow
Model SB1 underflow
Model SB2
Verify SB
Remodel SB1
Verify SB1
Strategize
Remodel SB1
Verify SB1
Strategize
Remodel SB1
Input M1
Verify models
Strategize
Remodel SB1 underflow
Input M2
Input TU
Examine models
Input SB1 overflow

The following abbreviations are used in the chart: M1 – machine 1, M2 – machine 2, TU – testing unit, L – monolithic system, SB1 – supervisor for buffer 1, SB2 – supervisor for buffer 2, SB – supervisors for buffers 1 and 2.

Figure 4.17: Flowchart of the strategy used by subject 5 when solving the factory problem.

had recognized any similarity between this problem and the problem they had seen in the ELEC843 course.

The subject used exclusively pen and paper to model the DES modules. The modelling of subsystems proceeded very fast. The subject started by building a model of the complete behavior of the system, however, in midway they announced that creating a monolithic model is not good and, instead, a modular approach has to be taken. The subject used the terms "centralized" and "decentralized" instead of "monolithic" and "modular". Centralization is a property of the method of supervision and not of the design of a system. Since decentralized supervisors are most frequently used with a modular system, apparently the subject confused the two terms.

During the modelling, the subject did not make any notes about what events are used in the models besides the occurrences of the events as labels of transitions. The event names were descriptive enough to make their semantics obvious. For example, the event when machine 1 takes a part for processing was named "M_take", the event when machine 2 puts a part in the output buffer was named "M2_Put_B2", the events when the testing of the quality of a part by the testing unit is positive or negative were named "TU_Pass_Out" and "TU_Fail_B2", respectively. We observed that the subject did not use consistent event names in their models. For example, sometimes they would use "M1_Put_B1" and "TU_Pass_Out" and at other times they would use "M1_Put" and "TU_Pass" instead. We speculate that this might be a result of using event names which are long and carry redundant information. The subject considered the controllability of each event as they introduced it in their models. They used a different graphical notation for controllable events. At one point the subject also considered the observability of events but it was only a brief appreciation that all events are observable.

The subject recognized that the models of machine 1 and 2 have the same structure. After finishing the model of machine 1, they replicated it and only used event names with different indexes. The subject chose to include extra behavior in their models for the machines and the testing unit. In the problem description, there is no mention of machines breaking down or of failing to process a part. However, the subject created models where the machines can fail (or break down) and get fixed. This appears to be an influence from the problem discussed in the ELEC843 course, where machines can break down and, subsequently, fail to process a part. It was observed that similar design decisions were made by another subject as well. The introduction of this functionality in the model was regretted by the subject a number of times during the problem solving as it made the solution much more complicated. However, every deliberation about whether the new functionality was necessary resulted in the subject concluding that, indeed, it is necessary since it does not make sense to have machines which never break down.

After preparing the models of the system components, the subject moved onto the design of supervisors. They did not formalize the control specifications—neither by creating models nor by using equations. The construction of supervisors proceeded relatively fast, however, the subject frequently redesigned their models and changed their approach. Their first attempt at a supervisor for buffer 1 did not work out since they did not consider the breakdown and repair of machines. The subject decided to instead build the supervisor for the interaction between machine 1 and buffer 1 and then for the interaction of machine 2 with buffer 1. While designing the latter supervisor, they discovered that it is very similar to, if not the same as, the first supervisor they had created for buffer 1 and then discarded. They re-evaluated their supervisors and determined that one of them prevents overflow and the other prevents underflow of the buffer. They continued by considering the supervisor for buffer 2. Mid-way the subject realized that their supervisor for the overflow of buffer 1 was incorrect; namely, it did not allow for parts to be taken out once they are deposited. In fixing the problem, they decided to "incorporate" the two supervisors for buffer 1 into a single one. Afterwards, they returned to working on the supervisor for buffer 2. The purpose of the two supervisors—for buffer 1 and for buffer 2—was the same except that they work on different buffers. One can hypothesize that working on the same problem a second time might have given an additional insight which the subject had realized can be generalized and applied to the first solution. The work on buffer 2 soon brought another realization by the subject. They reviewed the dynamics of the testing unit and realized that an unsuccessful test of a part results in the part being deposited into buffer 1, that is, something they had not considered in their solution thus far. The subject did not ponder this fact too long and instead chose to finish the supervisor for buffer 2, announcing that it is simple as it does not deal with feedback.

After the subject finished modelling the supervisor for buffer 2, they again focused their attention on the supervisor for buffer 1. They started building a new model but soon discarded it saying that they need to split the supervisor into more parts as the model had become cluttered. They produced two supervisors for buffer 1, one preventing overflow and the other preventing underflow. However, the subject appeared to have had difficulty deciding how exactly to achieve these goals. There are several indicators of that. First, the specific task of the supervisor for the prevention of underflow seemed to emerge as a result of the other supervisor not solving all problems. Second, the subject needed to remodel one of the supervisors. Third, they did not name the supervisors according to their tasks (prevention of overflow and underflow) but according to a more superficial feature: which event they control (the deposit or removal of a part). A further issue with their solution was the fact that the model for the underflow control was only sketched out, with much detail missing.

An examination of the supervisors created by the subject reveals that they did not use the conventional approach of "implicit" supervision. In implicit supervision, at each state of the supervisor there has to be a transition for each event which is not disabled, even if it does not change the state. In other words, if there is no transition for a given event at a given state, it is assumed that the supervisor disables this event at this state. As a result, all events which the supervisor does not control (i.e., the irrelevant events) appear in self-looped transitions in all states of the supervisor. The subject used a different, "explicit" approach in their supervisor models. They did not create any transitions for irrelevant events. In other words, if there is no transition for a given event at a given state, it is assumed that the supervisor enables the event. In order to indicate that certain events are disabled at certain states, the subject kept a separate list of disablement specifications.

When the models of the system components and the supervisors were ready, the subject briefly reviewed all of them and then proceeded to input them into the software. The subject recognized that a copy operation would be helpful when creating the models for machine 2 and for the testing unit as their structure is identical with that of machine 1. However, the subject did not know if this function is available and they did not want to venture in exploring the software functionality. When inputting the model for the testing unit, the subject forgot to create the event for the case when a part fails the test and the unit deposits it into buffer 1. They discovered this problem later, when they were inputting the model of one of the supervisors. We speculate that this omission was due to the similarity of the event names for a part failing the test and for the breakdown of the testing unit: "TU_Fail_B2" and "TU_Fail", respectively.

The subject had a different approach to naming the states of the models in the software compared to the naming on the paper. The states in the models on paper were labelled with integers, consecutively, starting with 1. Thus, the names of the states did not bear any significance beyond the purpose of distinction. In fact, the subject had to add additional labels on some states to indicate important information (such as "two parts in buffer 1"). On the other hand, the subject labelled the states of the models in the software quite descriptively, e.g., "M2-Working" and "B1-Two" (meaning B1 contains two parts). When modelling with pen and paper, the subject had not considered marking in their solution. However, when they started inputting the model of one of the supervisors, they realized they had forgotten to make any states in the computer models initial or marked, and they went back to fix all input models. Another difference from modelling with pen and paper was the fact that the subject was very conscious of the naming of events. They mentioned that they are trying to come up with a good scheme since they are planning on composing the modules—and having consistent event names is essential for this algorithm. When inputting the supervisor model, they wanted to copy the events from the system models already input. This functionality was not available in the interface, however, it seemed to be very important to the subject since they spent some time examining the online documentation provided with IDES. When they could not discover how to copy events, the subject appeared greatly disappointed. Nevertheless, they evaluated positively the online documentation, noting that the included tutorials were easy to follow. The subject started copying the events manually and at that time the session expired.

The experience of the subject with the software interface was mostly positive. The ability to create events in bulk was important for them, apparently since this is a convenient feature when inputting existing models. They enjoyed the easy creation of new modules and the interface for naming them. As well, they found the labels on the transition aesthetically pleasing. The biggest stumbling block for the subject was the lack of indication in the interface how to start inputting the data for a model once it is created. In fact, the experimenter had to step in and help them out. They also had some trouble locating the interface elements for the creation of events.

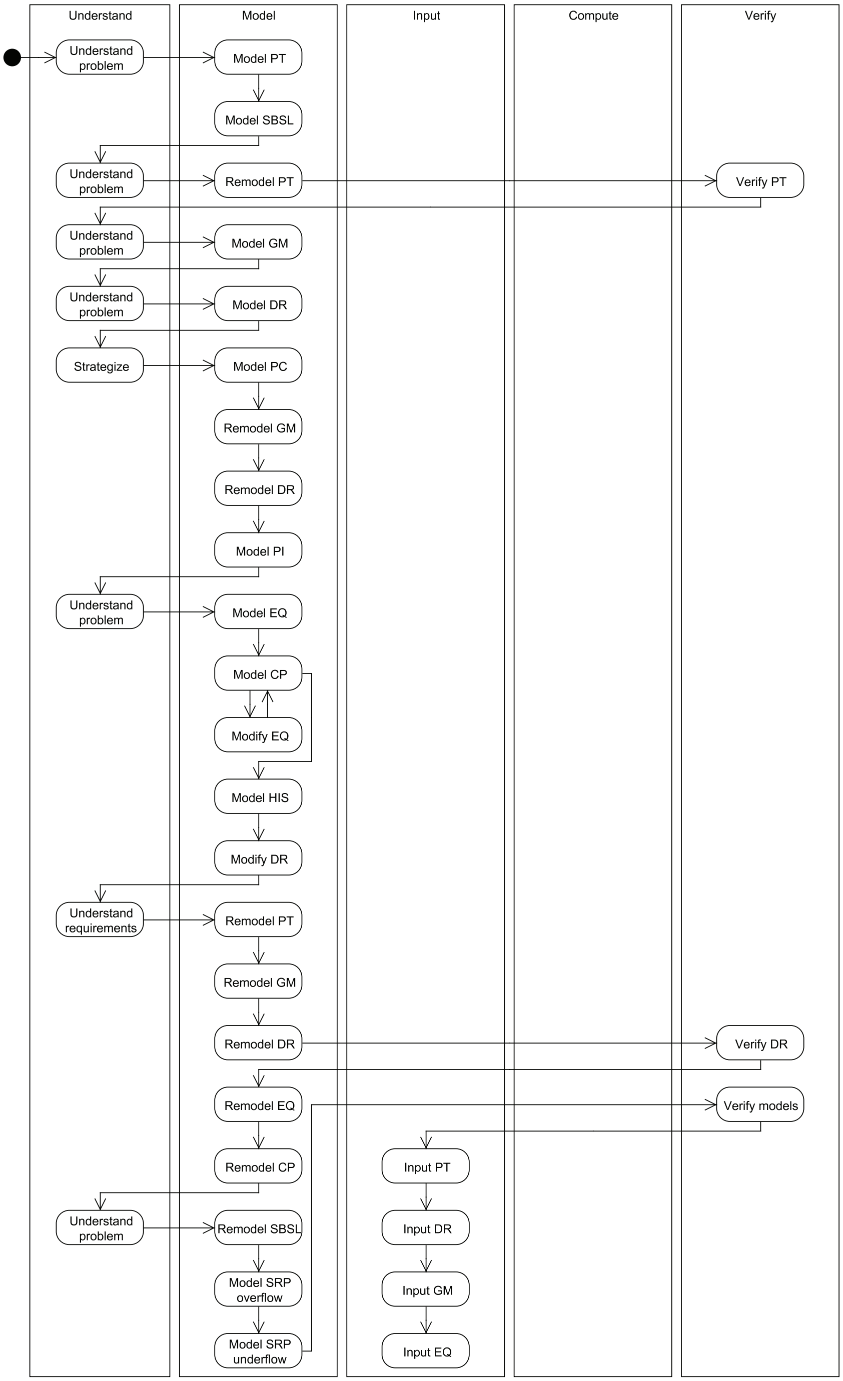
During the interview, the subject explained briefly their problem solving. The subject mentioned that throughout the modelling, they were trying to figure out when it is advantageous to split, or to lump, components of their supervisory solution, thus having a solution with a higher, or lower, degree of modularity. They modelled the system in a modular way as well, which seemed natural to them. Then, they tried to identify the supervision tasks and to model them one by one. The subject noted that they did not start with the easiest task, as the supervision of buffer 2 was easier than that of buffer 1. They recognized that it took significant effort before they were happy with their solution for the control of buffer 1. Having the current solution, the subject indicated that as future steps in the problem solving they would like to finish inputting all models and then verify their solution. More specifically, they would follow a set of steps. First, they would compute the parallel composition all system components to obtain a monolithic model of the system. Then, they would obtain the intersection of all supervisors. Lastly, they would "apply" the supervisor to the system to get model of the supervised system. By "apply", we believe the subject meant obtaining the intersection of the monolithic system and the model of the monolithic supervisor, since this is the standard approach. However, it could also refer to the operation of parallel composition which is the correct one to use in this case, as the supervisors are "explicit" and do not have self-looped irrelevant events. When the subject described how they would intend to verify the final solution, they mentioned that they would inspect the model visually and they would follow traces of events. For example, they would try to determine if machine 2 takes a piece from buffer 1 more than three times in a row. The subject pointed out that the descriptive state labels they used would be of help. For example, if the event for the testing unit picking up a part occurs at a state when the second buffer is empty, that would indicate an error in the supervisors. Overall, the use of DES operations discussed by the subject was correct. However, they designed the supervisors manually and their solution does not come with the guarantee of "correctness by design" provided by the algorithm for automatic supervisor construction. It seems that their hypothetical procedure for verification is not comprehensive; and the subject recognized its drawback if the model is large.

4.5.2 Hospital problem

The strategy used to solve the problem can be seen graphically in Fig. 4.18. The subject modelled the system in a modular way and then designed modular supervisors to enforce the control requirements. All modelling was done manually. At the end the subject started inputting their models into the IDES software to verify their solution. However, the time for the session expired before they could complete their work.

The subject started by reading the description of the problem and and then taking notes, summarizing the points they found important. They focused their attention roughly in the following order: patient, nurse, doctor, candy, equipment, computer, equipment, grandmother, blood sugar level, information system, report.

The subject started modelling by drawing a model for the patient. The first version was unsatisfactory to them. Instead, they drew a model of the specification for the acceptable levels of blood sugar. The subject then became uncertain what to do, noting that there are many entities to model. Their next steps involved creating models for the patient and the grandmother. The subject considered modelling the doctor next, however, they concluded they need to split their models according to two aspects: models dealing with candy and models dealing with insulin. To this end, they remodelled the modules they already had considered and created new models for the rest. Under the heading of models dealing with candy, they created models for the patient, the grandmother and the doctor. Under


Understand
Model
Input
Compute
Verify
Understand problem
Model PT
Model SBSL
Understand problem
Remodel PT
Verify PT
Understand problem
Model GM
Understand problem
Model DR
Strategize
Model PC
Remodel GM
Remodel DR
Model PI
Understand problem
Model EQ
Model CP
Modify EQ
Model HIS
Modify DR
Understand requirements
Remodel PT
Remodel GM
Remodel DR
Verify DR
Remodel EQ
Verify models
Remodel CP
Input PT
Understand problem
Remodel SBSL
Input DR
Model SRP overflow
Input GM
Model SRP underflow
Input EQ

The following abbreviations are used in the chart: GM – grandmother, PT – patient, PC – patient behavior in terms of candy, PI – patient behavior in terms of insulin, DR – doctor, EQ – prototype equipment, CP – computer, HIS – Hospital Information System, SBSL – supervisor for the blood sugar level, SRP – supervisor for the report.

Figure 4.18: Flowchart of the strategy used by subject 5 when solving the hospital problem.

the heading of models dealing with insulin, they created models for the patient, prototype equipment, computer and information system. Indeed, the subject created two models for the patient, however, they were complementary. In the first, the patient's intake of glucose (candy) is modelled; in the second, the patient's intake of insulin is modelled. When the subject started modelling the behavior of the equipment, they experienced difficulty deciding what the role of the nurse is and what the role of the equipment is. Finally, and correctly, they concluded that the nurse need not be modelled as a part of the system. During the modelling of the equipment, the computer and the information system it appeared that the subject was paying specific attention to the event names used in the different models. They frequently went back and forth between the models and adjusted event names. They made sure that the shared events had the same names in all models, e.g., the "connection" event in the equipment and computer models was called the same. The maintenance of compatibility between modules was later confirmed in the interview. An interesting observation was that in the model of the information system the subject included extra behavior. They modelled the event of data becoming garbled and the subsequent fixing of the system. While such behavior is not described in the problem description, it is similar to the breakdown and repair of machines in the factory problem shown in the ELEC843 course. After finishing the models dealing with insulin, the subject tried to discover what the specific role of the prescription of medication is: does it have any impact on the level of sugar in the blood of the patient? They complained that the problem description was vague and incomplete; apparently they did not manage to arrive at an informed decision. The subject made copies of all models and made further adjustments to reconcile the names of shared events. Making copies presumably served as a way to review and verify the existing modules before continuing problem solving.

In the next step, the subject proceeded by modelling the supervisors for the system. They did not formalize the control specifications; they only wrote down remarks which were marginally more succinct than the text in the problem description. The supervisor for the limits to the blood sugar level necessitated reconsideration of the event for the reception of candy by the patient. The subject modified all relevant models to distinguish between candy given by the grandmother and candy given by the doctor. While this distinction is technically necessary, the function and controllability of both events is the same. However, the subject interpreted the problem description in a different way and they believed the two events have different controllability (more specifically, the grandmother giving candy is uncontrollable and the doctor giving candy is controllable). The subject created two supervisors to guarantee that the reports in the hospital information system are dealt with properly: one preventing the generation of multiple reports before the doctor reads any of them, and one preventing an attempt by the doctor to read a report before it is generated. The subject needed to consider event controllability in depth since their first attempt at designing a feasible supervisor was unsuccessful. The last problem the subject tried to resolve before completing the stage of modelling was the impact of the medication on the rest of the system. They examined the dynamics of all components and considered the controllability of events, verbally. At the end, they concluded, correctly, that the prescription of medication is irrelevant to the functioning of the rest of the system.

During the modelling, the subject did not make any notes about what events are used in the models besides the occurrences of the events as labels of transitions. The event names were based on the verbal descriptions of the events, even if at times not very obvious. For example, the event when the child receives a candy from the grandmother was called "gr_candy", the event when the computer transmits the information to the hospital information system was called "info", the event when the doctor read the report was called "report_taken", etc. We observed that initially the subject did not always use completely consistent event names in their models but, as their models matured, the naming scheme stabilized. As the subject discussed in the follow-up interview, they found it particularly difficult to understand the impact of the medication prescribed by the doctor on the system. For example, they wondered if the medication results in the drop of the blood sugar level to zero grams. The problem description does not mention any specific impact and the prescription of medication is more or less irrelevant to the problem. The subject eventually made this conclusion, but not before considering this event on a number of occasions. The subject did not normally consider the controllability of the events during the modelling of system components, however, the design of supervisors required an extensive overview of event controllability. Generally, the subject identified the relevant events and specified their controllability correctly. However there were some problems nevertheless. It seemed that in a number of models the subject used higher than necessary event granularity. For example, they had two events, "connection" and "info", to denote the transmission of information from the prototype equipment to the computer. This separation into two parts is not relevant to the problem and a single event would have been sufficient. Furthermore, a similar situation had larger implications for the solution of the problem. The subject chose to use two events, "result_good" and "dr_candy", to describe what happens if the report lists a good blood reading (and consequently the doctor gives a candy to the child). Using a higher granularity let the subject specify that the event describing the outcome of the reading is uncontrollable, however, the doctor giving candy is controllable. While, from the doctor's perspective, the event is, indeed, controllable, within the context of the problem, it is not. The doctor's expert opinion cannot be superseded by that of an automatic controller. Had the subject used lower granularity for the events, they might have avoided this problem. The correct way to model the reception of candy by the patient was another difficulty experienced by the subject, however, this problem was gradually resolved as the modelling of the system advanced.

The subject created a large number of models, most of them newer versions of existing models. In general, the newer models were evolutionary refinements of older models, rather than complete overhauls. Overall, the models produced by the subject approach the correct solution. The subject made some mistakes with the assignment of controllability of events. As well, there was a lot of redundancy in their models. Besides the unnecessarily high event granularity, more models were created than needed. For example, the subject did not realize that the functionality of the equipment and computer modules can be conveniently described by a single, and much simpler, model. Another redundant model was that of the hospital information system. This system was mentioned in the problem description only for the purpose of providing context, and it is not necessary to model it as a separate entity. Lastly, the model of the patient is also redundant, for the same reasons as the the information system—even more so because the model created by the subject consists of a single state with all events self-looped. In other words, such a model does not contribute any dynamics to the overall system model.

In the last stage of problem solving which was observed, the subject started inputting the models into the software, presumably to verify their solution. The models they input differed a little from the corresponding models on paper. For example, the subject used meaningful state labels, such as “Sending info” and “Good”, as opposed to labelling states with integers in most of their models on paper. Furthermore, the subject made some modifications to the models, which seemed to be driven, at least in part, by the software interface. First, the subject explicitly considered and specified the controllability of all events. Second, during the modelling with pen and paper, they did not consider marking at all. However, in all computer models they specified which states are marked. The specifying of marking coincided with the setting of the initial state of a model. Apparently, the subject was reminded about marking when they set the initial state since the menu they used lists the two commands next to each other. The subject had to stop inputting models and did not manage to complete their work because the time for the session expired.

When using the software, the subject was most upset by the constant reminder that the latest changes to models need to be saved to disk. However, they did not seem to find the software problematic overall.

During the interview, the subject explained briefly their problem solving. The subject mentioned that they started by trying to get an overview of the situation described in the problem. More specifically, they were interested in learning who all the players are, who the supervisors are and what the control requirements are. They also tried to figure out all the actions of the players and what the effect of each action is. The subject elaborated that the system modules, such as the equipment and the doctor, were relatively easy. However, the model for the blood sugar level was difficult. For example, they could not understand what the level of blood sugar was in the beginning. If the patient starts with a level of zero grams, then the injection of insulin is detrimental. The subject explained that, overall, they tried to model what all entities from the problem do when no control is applied. It was important to match all events necessary for the interaction of entities. Afterwards, they created the supervisors. They discovered that the nurse should be viewed as a supervisor since it is the personnel who control the visit of the grandmother and initiate the injection of insulin. The subject did not express significant confidence in their solution, however, they believed that, in the unlikely case that their models of the system are correct, the supervisors would work. As future, hypothetical steps the subject mentioned that they would like to input all models in the software, do the necessary computations and then verify their result, e.g., if the level of blood sugar goes above twenty-four grams. The subject did not elaborate on what the computations would subsume. The subject expected that there could be deadlock in the system since, according to the supervisors, the doctor must read a report before insulin can be administered, and the administration of insulin is required in order to obtain a report. They also noticed that they had not considered the fact that the patient could store candy for later use—apparently, only at the end they noticed the issue with the wording of the problem (described in Section 3.4).

4.5.3 Within-subject analysis

For this subject, the factory problem was administered before the hospital problem. No conscious transfer of knowledge between the two problems was observed and the subject did not mention noticing a degree of similarity between the two during the interviews. However, it seems that solving the factory problem first did, at least minimally, influence the solution of the hospital problem. As discussed, the model of the hospital information system included behavior which was not mentioned in the problem description, namely, the failure and subsequent fixing of the system when too many reports are recorded in the information system. While the problem description mentions that there would be a failure in the system, this information is provided only as context and no fixing is mentioned.

In the interview, the subject expressed a positive attitude towards the study. However, they wished that the problems were smaller so that they could be completed in one hour. In particular, they found the input of models into the software a very tedious and time-consuming process which does not constitute progress in the problem solving. The subject also discussed why they started modelling with pen and paper in the beginning, instead of using the software. They explained that it is their personal preference. Further on, having the stack of paper in front of them at the start of the session was very inviting for them. They did not discuss the reason why they switched to using the software during problem solving. However, given the explanations of their intentions, it can be concluded that they wanted to take advantage of the DES algorithms implemented in the software.

In general, the subject used a similar problem-solving strategy for both problems. First, they modelled all components of the system in a modular way. They did not formalize the control requirements at all, instead proceeding directly to the design of supervisors. The supervisors were also designed in a modular way, where there would be a separate supervisor for every aspect of the requirements. As a conclusion of their work, the subject intended to use the algorithms in the software to help with the verification of their work. They planned to compose the corresponding models into a monolithic system and a monolithic supervisor and then to manually verify that the supervisor satisfies the control requirements when acting on the system. This approach to finding a solution suffers from two major drawbacks. First, the manual verification of a DES solution is, in most cases, infeasible, especially if verification is done by checking traces (or strings) of events. For non-trivial models, there would be more traces to be checked than a person would normally have the patience (and/or time) to check. Second, the solution may be sub-optimal as constructing a the most permissive supervisor manually is a daunting task, similar to that of manual verification. The subject also mentioned a different method of verification, where states with certain properties would be looked up (e.g., where a buffer is empty) and then all outgoing transitions would be examined to see if undesired event occurrences are possible (e.g., a machine attempting to pick up a part when the buffer is empty). This method of manual verification is more feasible (and closer to what DES verification algorithms do). However, it is not sufficient to guarantee the lack of errors in the solution. Furthermore, the optimality of the solution would still have be determined separately.

The speed and confidence which the subject manifested was approximately the same during the solving of both problems; and the subject advanced almost equally. The amount of verbalization was, however, significantly lower during the hospital problem. Furthermore, they seemed to be much more irritable while solving the hospital problem. This could be a result of temporary disposition or a result of the subject experiencing higher cognitive demands. Unfortunately, the subject did not give indication which interpretation is more correct. The quality of the solutions of the two problems was roughly the same, with the exception that the models for the hospital problem had more redundancy. The models of the system components were correct, or close to correct. In the hospital problem the subject misjudged the controllability of some events. The supervisors for the control specifications, unlike the models of the components, were not correct. Some of them had the quality of sketches rather than complete formal models. Also, due to the approach of building "explicit" supervisors, more attention had to be be taken regarding the specification of control decisions. Especially in the hospital problem, the subject seemed to use something in between the "explicit" and "implicit" approaches, where they would assume that the events relevant for the control need to explicitly enabled at the states of a model, while the rest of the events need to be explicitly disabled (which does not happen, since they are irrelevant for the requirements). In this way, the subject avoided both having to create a separate list of control decisions and having to self-loop all irrelevant events at all states in a model. Unfortunately, this mixed approach is not consistent, as it is, with any DES theoretical framework and, even if it were, it would require the explicit enumeration of the events which are "relevant" and the ones which are "irrelevant". Thus, the supervisors which the subject created, cannot be considered formally acceptable. The biggest problem with the supervisors, however, was that the control decisions encoded in the designs were insufficient to satisfy the requirements set forth in the problem description.

The lower-level activities of the subject indicate patterns of similarity and difference in the problem solving. Comparing the two problems, the subject created modules using different steps when working with pen and paper (see Fig. 4.19). However, when using the software, they worked in an almost identical way for each of the two problems (see Fig. 4.20). This may be explained by the fact that the subject used software only to input their models; that is, this was a well-defined procedure. Modelling with pen and paper, however, was a more "creative" process where there are no prescriptions about what specific activities to use and in what order to use them. The results of the n-gram analysis show that when modelling with pen and paper, the subject worked on events without interruptions more frequently in the hospital problem. The relative ratio of the bigram 'EE' is highest of all event-related bigrams, while, in the factory problem, the bigrams 'ES' and 'ET' have higher relative ratios. Examining also the ratios of the bigrams 'SE' and 'TE', one can conclude that during the factory problem, events were considered by the subject during the drawing of modules, while in the hospital problem, the subject considered events in separate "sessions". This corresponds well with the observations made during the problem solving. During the factory problem, the subject made decisions about the controllability of events throughout the process of modelling. During the hospital problem, on the other hand, the subject would interrupt the process of modelling to deliberate over the significance of an event or about the controllability of events. The higher relative ratio of the bigram 'EM' in the hospital problem also indicates that the subject would consider events before starting work on a module, rather than during work on a module. Lastly, the subject considered events much less frequently, overall, during the hospital problem. The absolute ratios of the event-related bigrams are smaller in the hospital problem than in the factory problem. Again, this is consistent with the observations made during the problem solving. It is important to note that even though the subject exercised considerable effort in maintaining compatible event names in the hospital problem, most of the actions were encoded as labelling transitions (i.e., their bigrams contain a 'T' instead of an 'E').

Opposite to the results for the event-related bigrams, the data show that, when considering modules (bigrams containing 'M'), the subject tended to do that without interruption more frequently in the factory problem. The relative ratio of the bigram 'MM' is higher in the factory problem, while, in the hospital problem, the bigram 'MS' has a higher relative ratio. This data is consistent with the observations. During the factory problem, the subject spent most of the time designing the supervisors and they were experiencing difficulty. Thus, they spent more time thinking about how to proceed before starting to draw a model. During the hospital problem, the subject spent most of the time on the system components, remodelling or copying models. Thus, after considering the model they would work on, they simply started modelling by drawing a state.

Aside from the differences in the actions of the subject, there is also a stable pattern of similarity. In both problems, both when using pen and paper and when using the software, the subject was likely to continue working on the same aspect of the model, e.g., when working on transitions they were likely to continue working on transitions. This is visible in the high absolute and relative ratios of the bigrams 'TT', 'SS' and 'MM'—and 'EE', in the case of working with the software. The bigram 'MS' has a very high relative ratio as well, compared to other bigrams. This means that, after considering a module, the subject very reliably proceeded working on the states of a model.

When comparing the results of the of the n-gram analysis for each method of problem-solving, fewer differences stand out (see Figs. 4.21 and 4.22 for the factory problem and the hospital problem, respectively). During the factory problem, as already discussed, the subject considered events throughout modelling with pen and paper. The software interface, on the other hand, encourages the bulk entry of events. For the subject that was not a problem, since they were inputting already existing modules and all events were already known. Thus, naturally, the bigram 'EE' had very high relative and absolute ratios when the subject used the software, at the expense of the ratios for the bigrams 'ES', 'ET' and 'TE'. During the hospital problem, a similar effect can be observed, especially in the absolute ratios of the bigram 'EE'. As discussed, the subject did not consider events very frequently while modelling with pen and paper. However, the inputting of models in the software requires the formal specifications of all events. The specific way in which events are dealt with in the software resulted in another difference in the data as well. In order to label a transition in the software, the relevant events have to have been specified. Thus, a user of the software needs to enter the events in the beginning of modelling, before they start labelling transitions. As a result, when using the software, the ratios of the bigram 'ET' are higher than the ratios of the bigram 'EM', while the opposite is true when using pen and paper.

The attention of the subject throughout the problem solving was analysed as described in

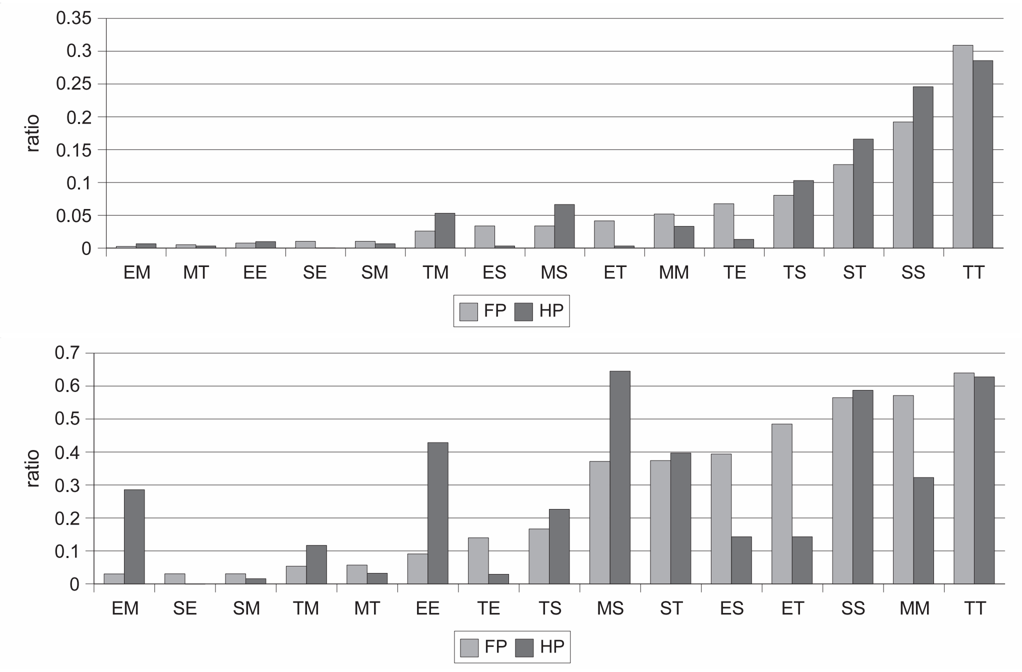


Figure 4.19: The absolute and relative ratios of bigrams for pen-and-paper problem solving during factory problem (FP) and the hospital problem (HP). Data are sorted according to the ratios in the factory problem.

Section 3.7. The verbalizations of this subject were, overall, infrequent and not very elaborate. Thus, it was hard to establish specific patterns of data discovery according to what the subject voiced. During the factory problem, the verbalizations concerning perceptually-triggered data discovery were insufficient to reveal any trend. Cognitively-driven data discovery concerned predominantly the user interface of the software (such as trying to discover how to create events) and the process of problem solving (such as trying to understand why modelling the supervisor for buffer 1 is so hard). In terms of low-level elements, the most common interest for the subject was the events. During the hospital problem, both kinds of data discovery concerned most frequently the dynamics of the modules. Cognitively-driven data discovery also concerned the process of problem solving, the models, and the problem description. Furthermore, the subject indicated on two occasions that they were interested in obtaining the count of modules and of events. The context of cognitively-driven data discovery indicated that, not surprisingly, when working on the modules for the patient and for the doctor, the subject was interested in information concerning the level of blood sugar. When working on the supervisors for the report, the subject consulted the problem description and the dynamics of the module for the computer. Finally, as already discussed, during the modelling of the equipment, the subject considered the purpose of the nurse in the problem. There was little indication of cross-context perceptually-triggered data discovery.

The analysis of the visual attention of the subject revealed the following facts. During the factory problem, the problem description was the most common target of visual attention, especially during the modelling of the system components. The following other patterns

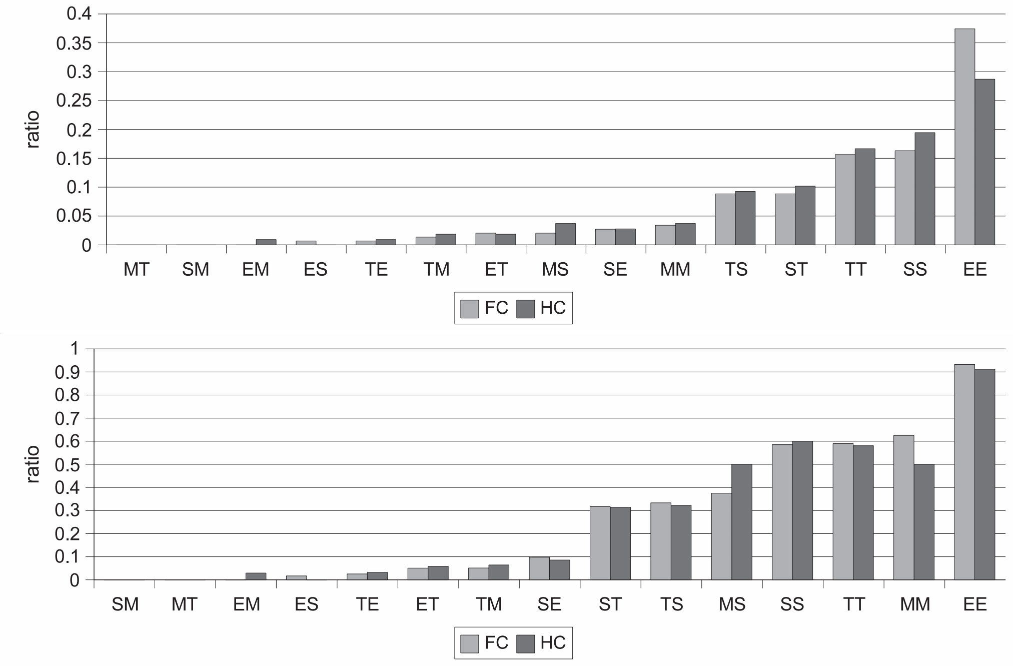


Figure 4.20: The absolute and relative ratios of bigrams for solving using software during the factory problem (FC) and the hospital problem (HC). Data are sorted according to the ratios in the factory problem.

emerged. When modelling the supervisor for the second buffer, the subject paid attention also to the sheet of paper with the models of the supervisors for buffer 1. When modelling the initial supervisor for buffer 1, they paid attention to the sheet with the models of the system components: the two machines and the testing unit. When remodelling the supervisors for the underflow and overflow control of buffer 1, the subject paid mostly attention to the sheets with the supervisor models they had already created. Only in the last iteration of remodelling of the supervisors of buffer 1 did the subject also attend to the sheet with the models of the system components. This could be a result of a decision to verify the supervisor models against all models created up to that point. During the hospital problem, the subject's attention was, again, attracted most by the problem description, while working on all of the models. The following other patterns emerged. When remodelling the module of the patient, the subject frequently also paid attention to the sheet of paper with the initial models of the patient and the blood sugar level specification, the summary of the problem description, and the model of the grandmother. The same sheet of paper was also attended to when remodelling the modules for the doctor and computer, when considering the role of the nurse, and when modelling the supervisor for the blood sugar level and one of the supervisors for the report. When remodelling the computer module, the subject also attended to the sheet with the candy- and insulin-related models (patient, grandmother, doctor, prototype equipment and computer). When modifying the initial model of the doctor, the subject paid attention to the sheet with the model of the hospital information system and, later, when remodelling the module of the doctor, they also paid attention to the sheet with the

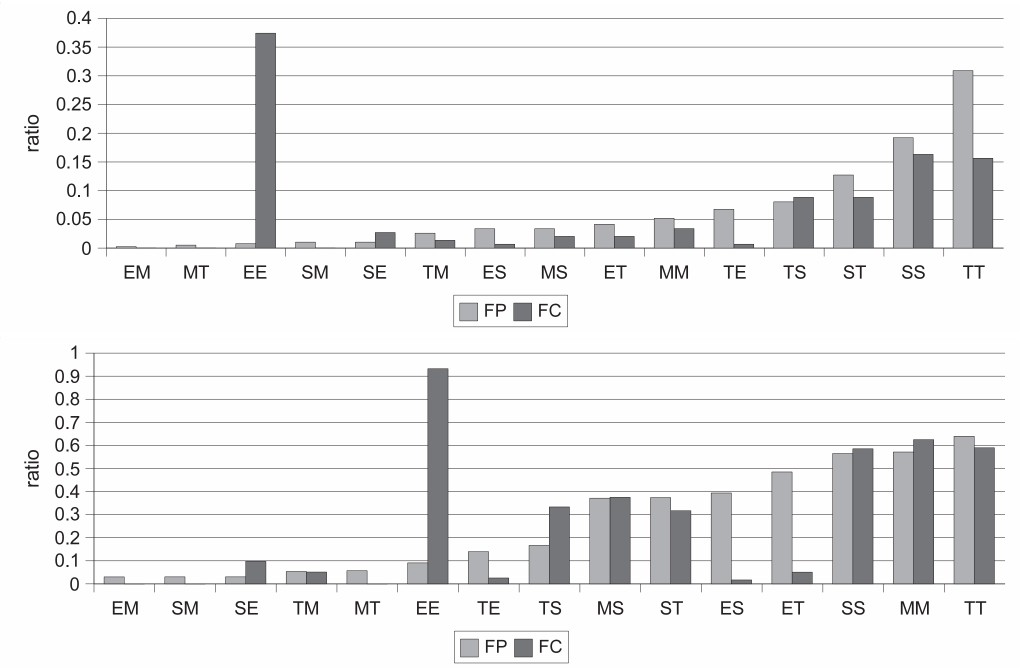


Figure 4.21: The absolute and relative ratios of bigrams for pen-and-paper problem solving (FP) and solving using software (FC) during the factory problem. Data are sorted according to the ratios in pen-and-paper solving.

candy- and insulin-related models (patient, grandmother, doctor, prototype equipment and computer). The same sheet also attracted the subject's attention when modelling the hospital information system. When considering the role of the nurse, the subject paid attention to the sheet with all the proper models of the system components: the hospital information system, patient, grandmother, doctor, equipment and computer. The same observation was made when the subject modelled the supervisors for the blood sugar level and the report—they also looked at the sheet with the proper models of the system components.

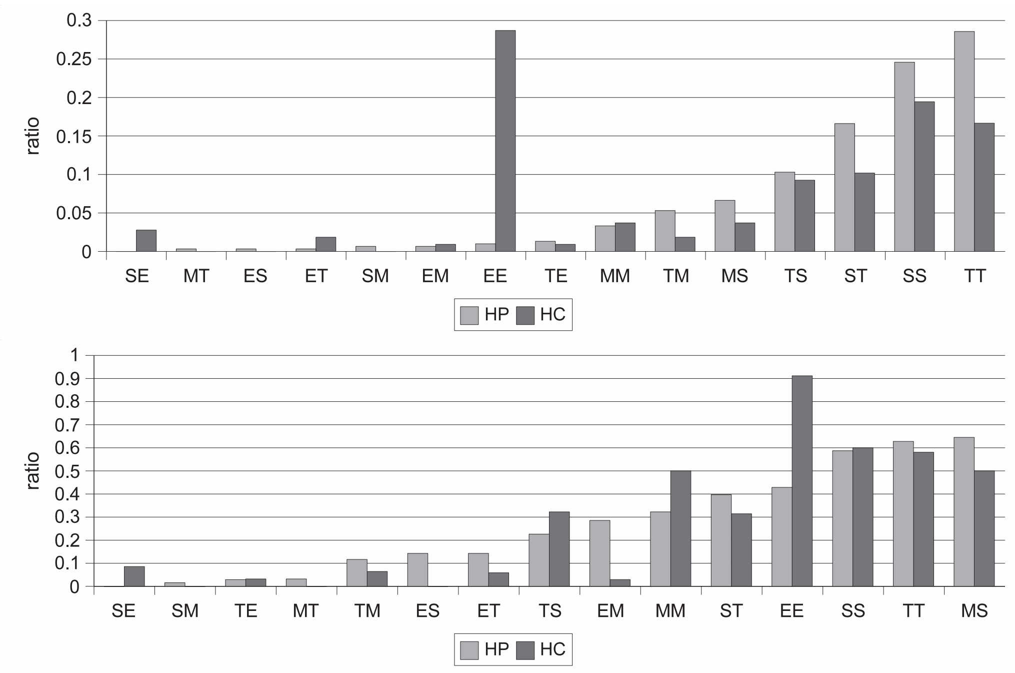


Figure 4.22: The absolute and relative ratios of bigrams for pen-and-paper problem solving (HP) and solving using software (HC) during the hospital problem. Data are sorted according to the ratios in pen-and-paper solving.

Chapter 5

Interviews with experts

Two experts were interviewed about the strategies they use to solve DES control problems. Both interviewees are Control Engineers and have been working in the field of DES education for multiple years. Their insights are valuable in the evaluation of the problem-solving strategies employed by the subjects in our observational study. The experts have different backgrounds and come from different schools of Control Engineering. Thus, collecting one-sided opinion is avoided.

The experts were asked to describe what steps they would follow to solve a hypothetical DES problem. Questions to clarify their exposition were asked.

5.1 Expert 1

The first expert explained their problem-solving approach as follows. The overarching strategy they follow consists of first modelling the given system, then modelling the control specifications and then using software to compute the corresponding supervisors. They would not attempt to produce a solution manually. The theoretical framework they use is based on the classic work by Ramadge and Wonham [45] and on decentralized control, as proposed by Rudie and Wonham [52]. Unless the problem calls for this explicitly, the expert would not consider other frameworks, such as hierarchical or timed DESs.

Initially, the expert reads the problem description and tries to break the problem into modules, separating the system components from the control requirements. They model all system components first, before they move on to the control specifications. They draw the components as finite-state automata, creating new events as needed. The expert uses the finest granularity for the events that they consider, moving to coarser events only if the detail is impractical. The event names they use are descriptive of what they signify. On the other hand usually states are not labelled, except if DES operations will be invoked on the model. The expert refers to the problem description while modelling to get cues about the actions of the system components. If they notice any discrepancy between the model being drawn and the description, they correct the model right away. The expert does not normally maintain a list of the events they use, nor do they use diagrams to guide their understanding of the problem.

When they are satisfied with modelling the system, the expert moves on to the design of the models for the control specifications. They re-read the problem description to remind themselves what the requirements are. During the design, they keep in mind the model of the system—the specifics of the behavior of the system components may be used to simplify the control specifications: it is not necessary to provide control for system behavior which cannot occur. The expert uses the events to link the control specifications to the components where control has to be exercised. During the design of the control specifications is also the time when they would normally consider for the first time the controllability of events and the marking of states (both in the models of the system components and in the models of the control specifications). Since the expert works within the framework of decentralized control, they need to consider which agents exercise control over which events and, respectively, the event observability. When done with the model of the control specifications, the expert would normally go back to the models of the system components and refine them if necessary and/or reduce their complexity if possible. That is, the expert modifies the models according to the requirements of the control specifications.

When all models are ready, the expert proceeds with the invocation of the DES algorithm for automatic construction of supervisors. To that end, they have to first input all models in the software. Since the models in the software have to be formal and rigorous, e.g., controllability of events and marking of states has to be specified, this may be the time when the expert remembers to specify controllability and marking.

If the result of the algorithm is obviously wrong, e.g., empty or too small (compared to the expected size), the expert will check if all marking is correct and if the names of the events in the models are consistent. If the result seems correct at first glance, the expert will verify it using two main approaches. First, they will trace sequences of events starting in the initial state of the result and check if they lead to the expected states (when the sequences should be allowed) or if the occurrence of these strings is at all possible (when the sequences should not be allowed). These event sequences will be chosen arbitrarily, unless the answer to the problem is known. Second, the expert will select sequences from the control specifications and check if they appear in the same way in the result. Such a check would be purely syntactic, without trying to discover if such sequences "make sense" according to the overall goals in the problem. Checking for these sequences, according to the expert, is especially important in modular control where different specifications may block each other. As a result, fewer occurrences of the sequences would be allowed. This may indicate a problem with the composition of models.

If the expert checks multiple arbitrary sequences and all of them are handled correctly by the supervisors, the expert would consider the solution successful. As a last check before submitting the solution, they may draw a diagram of the system components to clarify if the solution makes sense. When submitting their solution, they would hand back all the finite-state automata models, a list of the events used with a description of what they mean, and potentially an informal description of how the functionality of the supervisors maps to the problem.

5.2 Expert 2

The interview with the second expert was not as extensive. Thus, their explanations were not as detailed. The explained that, when solving DES problems, they start by trying to identify the "resources" and the "servers" in problem. For example, in a hospital setting, rooms and beds would be resources, while nurses would be servers. Then, the expert would try to identify all the events in the system. When they have gained some understanding of the problem, they would consider what is part of the system and what is part of the control requirements. In their experience, usually the "servers" are the system components, while the "resources" are the control specifications. As a side note, the expert mentioned that it is a frequent mistake to assume that a control requirement is a part of the system, e.g., the buffers in a factory problem. However, once the supervisor is computed, there is no need to distinguish between the two concepts, as the effect of the control is demonstrated by simply composing all models.

The control specifications, according to the expert, are similar to the "supervisors", in the sense that their states have meaning and it is possible to identify where events are disabled. Thus, the expert believes that control specifications should also be verified, e.g., by tracing sequences of events.

The expert works within the framework of local modular supervision [11], where there is a separate supervisor for each specification. After generating the supervisors with software, to further decrease the complexity of the solution, they normally apply supervisor reduction techniques [63]. Afterwards, they try to understand the result. The two main techniques the expert uses are checking which events are disabled at which states, and tracing sequences of events in the controlled system. If problems are discovered, the expert would normally attempt to correct them by changing the control specifications, leaving the system components intact. They also check the marked states in the result against their expectation of the marking. In the case of discrepancy, they use the event tracing technique to figure out why the states have incorrect marking.

The expert summarized their problem-solving steps as follows:

1. Identify servers and resources and what is the input/output of the system
2. Identify the relevant components (modules)
3. Identify the events—already doing some of point 6
4. Determine what is a component of the system and what is a part of the control specifications (could be done after or during point 5)
5. Create automaton models, first for the system components and then for the control specifications (could be done before or during point 4. The meaning of the states of the automata has to be determined.
6. Determine which events are controllable
7. Compute the monolithic system using parallel composition of all components

8. Compute the supervisors using the DES algorithm for automatic supervisor generation, trying to obtain modular supervisors for the different specifications
9. Apply the reduction algorithm to the supervisors
10. Verify the supervisors, e.g., by looking at disable events at different states and by tracing event sequences
11. If a supervisor blocks all events, check for missing self-looped events in the control specifications
12. If there is a problem with the control, analyse it by tracing targeted event sequences
13. Attempt to correct problems first by modifying the control specifications, only after looking at the system components

At the end, the expert also shared their understanding of what is missing in DES theory and in the software available to DES Control Engineers. According to the expert, in industry there are good programmers to design controllers. People there are interested in using DES theory mainly to verify their solutions, rather than to generate them. If DES is to be used, it should work "undercover", allowing for the use of traditional model representations. To that end, in the expert's experience, it is important to be able to specify control decisions based on variables with thresholds and conditions on them. An example of a condition frequently used in practice, but impossible to model in standard DES theory, is the expiration of a timer. To the expert, the DES solutions—the supervisors—should be available as code or event sequences, to allow for traditional analysis. However, it is most important to let the users simulate the performance of a supervisor visually.

The expert agrees that every new piece of software may be "scary" in the beginning. Thus, a good set of tutorial examples may come a long way in helping users benefit from the advanced functionality. As well, the software should allow users to examine the history of their activities. This would help in the replication of successful strategies. If the software supports scripting, then it would even be possible to encapsulate these strategies in accessible scripts. Taking this even further, it would be nice to be able to encode the experience of users into "templates", or standard parts of problems.

After using the IDES software package, the expert listed a number of suggestions for improvement:

- Let users write Matlab-style scripts (including the nesting of scripts),
- Display the disabled events in each state of a supervisor,
- Facilitate the animation of event sequences,
- Provide a number of algorithms for the automatic layout of models, and
- Allow the selection of multiple input models for DES operations which support this (such as parallel composition).

Chapter 6

Comparative analysis

6.1 Measure of progress in DES problem solving

The progress of a person towards the completion of a DES problem is very hard to evaluate. Such problems fall naturally in the category of ill-structured problems as defined in [54]. Most non-trivial DES problems could have a variety of valid solutions, depending on the interpretations of the problem-solver. Furthermore, a number of different theoretical approaches to the problem solving could be taken. For example, let us consider the factory problem described in Appendix A.1. It could have different solutions depending on whether the problem solver decides to model the potential breakdown of the machines. In this case it could be argued that breakdown is not mentioned in the problem description so it must not be modelled. However, with problems which are harder to describe rigorously (e.g., the interactions between personnerl in a hospital) it becomes infeasible to avoid ambiguity, and thus the multitude of potential solutions. Furthermore, depending on the background and condition of the problem solver, they may decide to model a problem either in a monolithic or a modular way, they may decide to use centralized or decentralized control, they may choose to model supervisors manually or to use the appropriate DES algorithm to generate the supervisors automatically. All these variations and the relative ease with which one could switch between different strategies during the problem solving makes evaluation unreliable. However, despite all the issues mentioned above, it is desirable to be able to give at least some estimate of the progress in DES problem solving.

Looking at the results of problem solving, one has some intuitive feeling about the amount of progress. In order to evaluate the progress of the subjects in our study, we sought to replace this intuitive measurement with a more rigorously defined measure of progress. One natural measure which comes to mind is very simple. If the expected solution involves the models A, B and C, identify if possible the corresponding models A′, B′ and C′ in the solution of the individual and, for each model which is close enough to the expected version, allot 33% worth of progress. However, our belief is that such a method of evaluation is inconsistent with the aforementioned intuitive evaluation. For example, in the factory problem, a subject may only focus on modelling the system components and may produce correct models for the two machines and for the testing unit. Thus, they would achieve 3 units of progress (here, we assume an imaginary measure where each correct model is awarded a unit of progress). Another subject may model the machines and the testing unit, the control specifications for the buffers, and start working on the supervisor for the system. However, if they make a small mistake in the models of the machines and of one of the buffers—a mistake they might discover and correct later on—they would achieve only 2 units of progress (one for the testing unit and one for the correct buffer model). This contradicts our intuitive observation that the second subject advanced much further in problem solving.

In order to address the discrepancy between intuitive evaluation and evaluation using a simple measure of correctly modelled entities, we developed a new measurement methodology suitable for the problems considered in this study. We identified which elements of a solution are essential, accommodating a variety of modelling approaches. A solution is evaluated in the following categories:

- Models at least one system component,
- Models the complete system,
- Models at least one control specification,
- Models all control specifications,
- Specifies event controllability,
- Models the supervisor for at least one control specification,
- Models supervisors enforcing all control specifications,
- Some aspect of the supervisors has been verified, and
- The total control enacted by the supervisors has been verified.

For each of the first five categories, one point will be awarded if the solution satisfies the given condition, and a second point will be awarded if the condition is implemented correctly. For each of the last four categories two points will be awarded, if the solution satisfies the condition. The correctness of the supervisors is not examined separately since it depends very strongly on the correctness of the models of the system components and the control specifications. Furthermore, when supervisors are generated automatically, they are correct by construction (given that the other models are correct). Instead of rewarding the correctness of supervisors by comparing to an expected solution, two points are awarded if some aspect of the supervisory solution has been verified and two more if the complete solution has been verified. The method of verification is intentionally unspecified since different problem-solving approaches may require different methods of verification. Since any aspect of a DES problem may be modelled in a modular way, points are awarded both for modelling at least one entity and for modelling all entities. A centralized (not modular) model would satisfy both conditions and thus not be disadvantaged against modular models.

The proposed model deemphasizes the specific number of correct models produced by the problem solver. For example, if there are five system components, the same level of progress will be noted when only one or when four components are modelled. In DES problems, the exact number of incorrect models has the same impact on the incorrectness of the final solution. On the other hand, the higher diversity of problem-solving activities is rewarded even if no correct models are produced. I.e., points can be collected under each category even if the model is incorrect. Such points are awarded since work in a variety of categories demonstrates that the problem solver is aware of the different aspects which need to be considered to obtain a solution. Two points per category are awarded for work on the supervisors, to counter-balance the points accumulated by work on the system modules and the control specifications. The production of supervisors is the most important aspect of a solution and, at least theoretically, it is possible to design the correct supervisors without modelling any aspect of the system. Thus, the weight of working on supervisors has to be greater.

The application of the proposed measure of progress to the solutions in our observational study resulted in evaluations which were much more consistent with our intuitive understanding of the progress of the subjects. The proposed measure is, to some extent, tailored for the observations in our study, however, the ideas used in its design can be easily applied to the evaluation of solutions in other situations. It is important to note, however, that this is a measure of the *progress* towards a DES solution; it is not a measure of the *correctness* of a solution.

6.2 Rate of errors in incomplete DES problems, perceived mistakes

The determination of error in DES problem solving is, due to reasons similar to the ones discussed above in Section 6.1, very difficult. In our observational study only one subject advanced far enough in the problem solving to perform a more substantial verification of their solution, for only one of the problems. In all other cases, the subjects did not manage to produce a solution before the expiry of the sessions. Thus, for us it was not possible to evaluate solution correctness using the standard approach, namely, via simulations or testing [4]. Furthermore, variations in the interpretation of the problem description created enough variation in the solutions (e.g., the addition of machine failure and repair) to render them impossible to compare directly with the reference solutions (see Section 3.8). In some cases it was relatively easy to classify certain problem-solving steps as incorrect, however, this was not always possible. Furthermore, the subjects did not have the opportunity to complete the problems and thus mistakes early on could not be used as reliable indicators of errors in the final solutions. It could well be that the subjects would have recognized mistakes later on and would have corrected them. Subject 1, for example, discovered a mistake in one of their models when verifying their solution.

It seems that an objective evaluation of the correctness of incomplete solutions is not possible. Thus, in order to evaluate the solutions in our study, we decided to use a method relying on subjective perspectives. We counted the perception of mistakes as demonstrated by the subjects during their problem solving, i.e., the occasions they took corrective measure.

Indeed, this approach cannot be used to estimate the, objective, correctness of an incomplete solution. For example, subject 4 determined incorrectly the controllability of events in the hospital problem but they did not perceive their choices as incorrect. Thus, subjectively, their solution was correct. However, the proposed evaluation of perceived mistakes is useful to estimate the difficulty a subject experiences with the problem solving. The more perceived mistakes they make, the more demanding the problem solving would be subjectively. Furthermore, the count of perceived mistakes gives an insight into the efficiency of problem solving.

In our analysis, we do not examine if the perceived mistakes are real errors or not. We only observe if the subject takes a corrective action (be it necessary or not)—thus using the subject's subjective judgment of errors. We distinguish three types of perceived mistakes, going from trivial to more substantial.

Type 1 These are perceived mistakes indicated by small and brief corrective actions by the subjects. For example, the subject may correct the name of the event on a transition.

Type 2 These are perceived mistakes indicated by more substantial amounts of time spent by the subject correcting a model. For example, the subject may decide to split a single event into finer-grained events and they have to go back and accommodate this change in existing models where the old event is used.

Type 3 These are perceived mistakes indicated by complete overhaul of a model by the subject. For example, the subject may decide that their model of one of the system components is so out of line with the rest of their models that they have to remodel it from scratch.

If a subject is confident and knows what to do, they are less likely to make many subjective mistakes. Inattentive problem solving would result in mostly type 1 mistakes, but mistakes of type 2 and 3 would not be common. On the other hand, subjects who are uncertain in their problem solving would exhibit more type 2 and type 3 mistakes.

6.3 Factory problem

The strategies the subjects employed in solving the factory problem varied in certain aspects, but they were also quite similar. This could be attributed, at least in some extent, to the fact that most subjects had seen the solution to the same problem in the ELEC843 course. Four out of the five subjects explicitly mentioned that they recognized the problem, while the activities of subject 5 indicated that they implicitly used information related to the problem from the course.

All subjects, except one, assumed a modular approach to modelling right from the beginning. The subject who did not take a modular approach recognized fast that creating a centralized model of the system is very difficult and also proceeded by modelling modules separately. Thus, all subjects eventually ended up with models of the system components. Afterwards, different approaches were observed. Subject 3 did not seem to differentiate between system components and control specifications, so they created models indiscriminately.

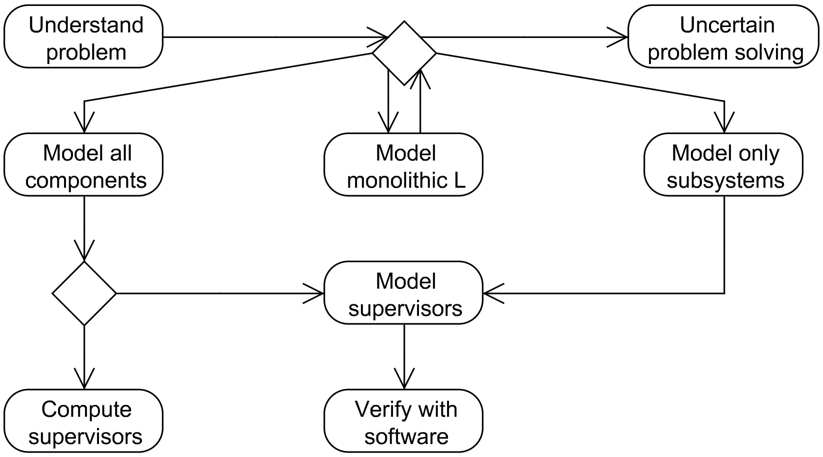


Figure 6.1: Top level of the generalized flowchart of the problem-solving strategies used by subjects during the factory problem. The full flowchart is included in Appendix C.1.

While the ultimate goal of their strategy was not clear, they indicated that eventually they would have liked to compose all modules. Subjects 1 and 4 made distinction between system components and control specifications. They modelled the control specifications in a modular way. Subject 1 modelled all system components first, while subject 4 modelled system components and control specifications approximately in the order parts are transferred in the factory. Subjects 2 and 5 did not model the control specifications at all. After modelling all modules, subject 1 computed the centralized models of the system and control specifications. Then, the subject used the DES algorithm for automatic construction of a supervisor. Subjects 2, 4 and 5 had a different approach. They designed modular supervisors manually, and they planned to verify their supervisors using the software. Subject 2 planned only to compose the system components into a monolithic model and to keep the supervisors separate, while subjects 4 and 5 planned also to compose the supervisors into a monolithic model. The top level of the hierarchical flowchart showing the different problem-solving strategies is shown in Fig. 6.1.

During modelling, all subjects used pen and paper. Only when it was necessary to do computations with the models, such as performing composition, did the subjects choose to use the software. Subject 3 attempted to derive the composition of two modules manually before deciding to use the software. They were also the only subject who, at one point, used both the software and pen and paper simultaneously to work on their models.

The progress of the subjects was different. Only subject 1 managed to complete the problem in the allotted time (one hour). Subjects 2 and 5 started using the software to verify their solution but did not manage to complete the task. Subjects 3 and 4 did not manage to complete the modelling of all components. Subject 3 gave up solving the problem before the allotted time expired. The progress of the subjects is summarized in Table 6.1 according to the evaluation methodology described in Section 6.1.

The difficulties experienced by the subjects while solving the factory problem can be summarized as follows. The most problematic task appeared to be the design of the control specifications and/or the supervisors, especially for buffer 1. That included issues such as

	S1	S2	S3	S4	S5
Models at least one system component	2	2	1	2	2
Models the complete system	2	2	0	2	2
Models at least one control specification	2	0	1	1	0
Models all control specifications	1	0	1	1	0
Specifies event controllability	2	1	0	2	2
Models the supervisor for at least one control specification	2	2	0	2	2
Models supervisors enforcing all control specifications	2	2	0	0	2
Some aspect of the supervisors has been verified	2	2	0	0	0
The total control enacted by the supervisors has been verified	2	0	0	0	0
Total points	17	11	3	10	10

Table 6.1: The progress of subjects 1 to 5 (S1 to S5) in solving the factory problem

how to interpret the buffers, if and how to split the specifications into underflow and overflow prevention, which events to use for the purpose of control, what is the controllability of events, should irrelevant events be self-looped at all states, which states should be marked, what is *deadlock* and how to express it in the DES framework. For subject 5, the work was further complicated due to the additional behavior of the machines (failure and repair) which they modelled. Subject 4 mentioned that they had to pay extra attention to distinguishing the system components from the control requirements. Furthermore, they experienced difficulty designing the supervisor for buffer 1 from the formal control requirements they came up with. The approach subject 3 used for problem-solving did not appear to have the potential to lead to a correct solution and, incidentally, the subject struggled with modelling correctly the modules from the problem. Subjectively, they experienced the greatest difficulty with the modelling of the testing unit and in deciding how to "tie" it with the rest of the system. Except for subject 3, the subjects did not seem to experience significant difficulties modelling the system components (the machines and the testing unit).

The results of the bigram analysis for all subjects were plotted together to compare the patterns of low-level actions across subjects. While it is difficult to interpret the resulting charts, the following trends can be recognized. During work with pen and paper, the different subjects had different patterns of activities (see Fig. 6.2). For example, subjects 1, 3 and 4 worked on events in continuous sessions (see the ratios for the bigram 'EE'). Subjects 2 and 5, on the other hand, did not consider events in bulk. Incidentally, these were the subjects who did not create separate legends of the used events either. Subject 5 worked on events intermittently (high relative ratios of 'ES' and 'ET'). The way subjects drew models also varied. Looking at the data, especially at the relative ratios, it seems that subjects 2 and 3 tended to work on states or transitions in bigger chunks (e.g., drawing all states first and then drawing all transitions). This is indicated by the higher relative ratios of the bigrams 'SS' and 'TT' and the lower relative ratios of 'ST' and 'TS', compared to the ratios when subjects 1, 4 and 5 drew models. Finally, subjects 4 and 5 seem to consider modules for longer before starting to model them—as indicated by the relatively higher relative ratio of 'MM' at the expense of the relative ratio for 'MS'. It seems that all subjects preferred to start drawing a model by creating a state instead of a transition (high ratios of 'MS', lower ratios of 'MT'). This is only natural, since states are the starting points and ending points for transitions.

Contrary to the varied strategies used when modelling using pen and paper, the charts with the results of the N-gram analysis of the work with software show consistent patterns (see Fig. 6.3). There are two noticeable differences in the ratios of the bigrams. The relative ratio of the bigram 'MM' is lower for subject 1 in comparison with the other subjects. This is compensated, however, with the higher relative ratios of the bigrams 'ME' and 'MC'. This means that the subject, after considering the modules, tended to work on events or invoke algorithms. We believe this can be explained by the fact that, unlike all other subjects, subject 1 managed to advance further in the problem solving and they spent relatively more time verifying and trouble-shooting their models. The second noticeable difference in the ratios is when one considers the activities of the subjects after executing a DES operation (the bigrams starting with 'C'). Subject 3 most frequently continued working on the level of modules (higher relative ratio of the bigram 'CM'). This indicates that the results of the operations did not usually, in their opinion, necessitate specific changes to the elements of the models. This is consistent with the general lack of understanding of how to advance in solving the problem which the subject demonstrated. If they worked on elements of the models, they would work on the events (the bigrams 'CE'). On the other hand, subjects 1 and 2 worked on the states ('CS') and the transitions ('CT'), respectively, of the models if necessary after the performance of an operation.

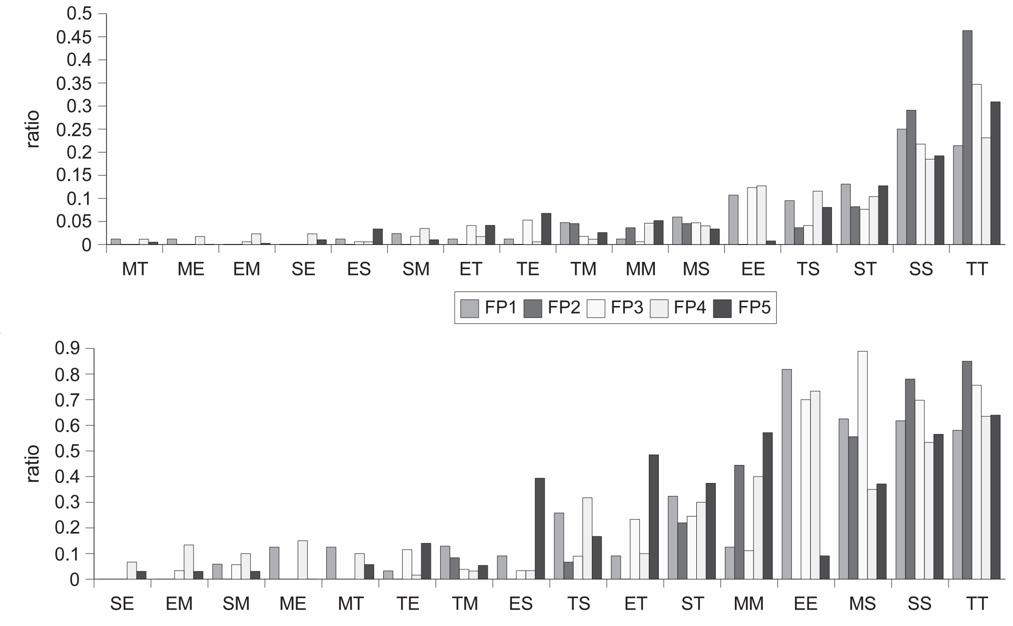


Figure 6.2: The absolute and relative ratios of bigrams for pen-and-paper problem solving during Factory problem for subjects 1 to 5 (FP1 to FP5). Data are sorted according to the average ratios for all subjects.

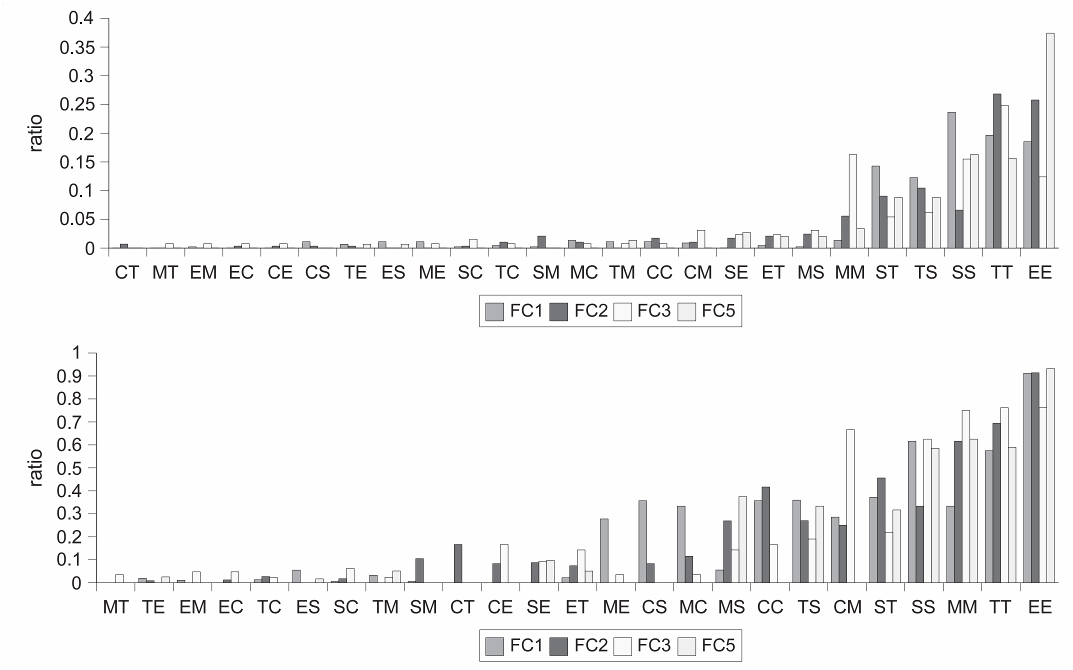


Figure 6.3: The absolute and relative ratios of bigrams for solving using software during the Factory problem for subjects 1, 2, 3 and 5 (FC1, FC2, FC3 and FC5). Subject 4 did not use software during problem solving. Data are sorted according to the average ratios for all subjects.

The analysis described in Section 3.7 did not reveal any consistent patterns of cross-context data discovery. The data discovery verbalized by subject 3 indicates that for them the testing unit and the models of the two buffers formed a unit of interest, where attention would frequently shift between these models. On the other hand, these were the only models which the subject created and, furthermore, the subject was struggling to find a way to advance in the problem solving. Thus, one could speculate that these results do not show any specific association between the given models. The verbalization of data discovery by subject 4, however, also points out to a connection between the models of the buffers, the control requirements and the testing unit.

When the information from the visual context is included in the analysis, however, some patterns start to emerge. In Fig. 6.4 it is clear that the subjects were most engaged with other modules when working on the supervisor(s) for buffer 1. Indeed, the heart of the problem lies in determining the correct control strategy so that a part rejected by the testing unit does not cause an overflow in buffer 1. In the figure, the thickest edges are between the supervisor(s) for buffer 1 and the machines, buffer 1 and the testing unit. While working on the supervision of buffer 1, the subjects also made frequent overviews of all models (ALL) and they referred to the problem description (DESC). The subjects often referred to the problem description also when modelling the machines and the testing unit. Another strong connection exists between buffer 2 and the testing unit. From the graph it can be observed that there is not as much interaction between the system components (machines and testing unit), nor between the control specifications (buffer 1 and buffer 2) and between the supervisors. Reference to previous versions of models were not common since few modules (except the supervisor for buffer 1) were remodelled by the subjects in this problem.

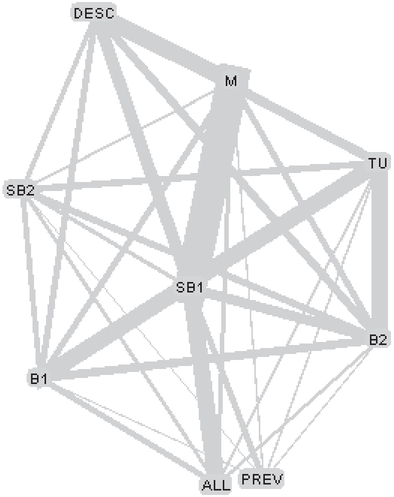


Figure 6.4: A graph of the frequency with which the subjects switched attention between different components of the factory problem. The nodes signify the following components: machines (M), testing unit (TU), buffer 1 (B1), buffer 2 (B2), supervisor for buffer 1 (SB1), supervisor for buffer 2 (SB2), all models (ALL), previous versions of a model (PREV), and problem description (DESC). The width of the edges signifies the relative frequency of switching attention between the given components (thicker line means higher frequency). The frequency is computed from the weighted aggregation of the data for visual attention and for verbally announced data discovery. The frequency was adjusted with an exponential function to allow easier visual differentiation.

6.4 Hospital problem

The strategies the subjects employed in solving the hospital problem were varied, especially in the order in which modules were modelled. No subject expressed explicitly that they recognized the problem as similar to the factory problem or to what they remember from the ELEC843 course. Only subjects 2, 4 and 5 through their activities indicated that they implicitly use information related to the factory problem (or to the problem from the course).

All subjects assumed a modular approach to modelling right from the beginning. First, they modelled the components of the system. Subject 4 started with the model of the patient and then determined the relevant control specifications before continuing with the other system models. Subject 1 modelled the control specifications only after completing the models of the system components. Then, the subject requested the use of the software to continue their work. Subjects 2 and 5 did not model the control specifications at all. Instead, they started modelling the supervisors manually and, when done, requested the use

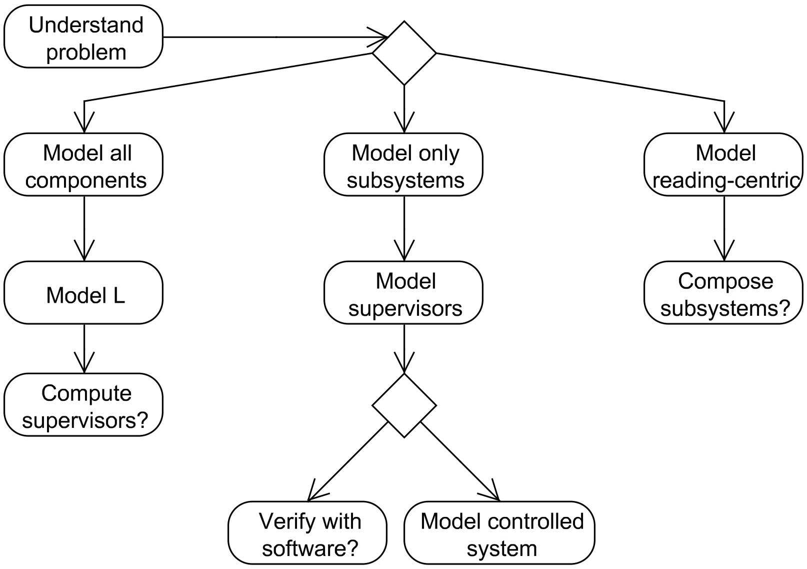


Figure 6.5: Top level of the generalized flowchart of the problem-solving strategies used by subjects during the hospital problem. The full flowchart is included in Appendix C.2.

of the software to verify their solution. Subject 4 also designed the modular supervisors for the control specifications manually. When subjects described what they would plan to do to complete their solutions, consulting with external sources emerged as a common theme. Subjects 1, 3 and 4 mentioned they would like to review their course notes or talk to an expert to verify their approach. Subjects 2 and 4 planned to compose their models and check if the supervisors are controllable and if there is *conflict* between the supervisors (i.e., if there could be a deadlock or a livelock when the supervisors operate concurrently). Subject 3 did not explicitly mention the check for lack of *conflict*, however, they mentioned checking for controllability and lack of deadlock. Subject 5 also referred to composition of the models and then verifying the supervisors, however, they were not specific on the method they planned to use. A flowchart showing the different problem-solving strategies is shown in Fig. 6.5.

During modelling, all subjects used pen and paper. Only when it was necessary to do computations with the models, such as performing composition, did the subjects choose to use the software. Subject 1 attempted to derive the composition of two modules manually before giving up and requesting the use of the software. Subject 4 did not appear to plan on the use of the software at all. They did all composition operations manually and they justified their lack of interest in using the software with the fact that the models are relatively small. However, they did not reject the possibility that they might use the software in the hypothetical future steps in solving the problem.

No subject managed to complete the problem in the alloted time (one hour) but the progress of the subjects was different. Subjects 2, 4 and 5 modelled the supervisors for the system manually, however, they ran out of time to verify their solutions. Subject 1 modelled the control specification and apparently planned to apply the DES algorithm for automatic generation of supervisors, but also ran out of time. Subject 3 gave up solving the problem before the allotted time expired. The progress of the subjects is summarized in Table 6.2

according to the evaluation methodology described in Section 6.1.

	S1	S2	S3	S4	S5
Models at least one system component	2	2	1	2	2
Models the complete system	1	1	0	0	1
Models at least one control specification	2	0	0	2	0
Models all control specifications	1	0	0	0	0
Specifies event controllability	0	1	0	1	1
Models the supervisor for at least one control specification	0	2	0	2	2
Models supervisors enforcing all control specifications	0	0	0	2	2
Some aspect of the supervisors has been verified	0	0	0	0	0
The total control enacted by the supervisors has been verified	0	0	0	0	0
Total points	6	6	1	9	8

Table 6.2: The progress of subjects 1 to 5 (S1 to S5) in solving the hospital problem

The difficulties, experienced by the subjects while solving the factory problem can be summarized as follows. A common theme in the comments of all subjects was that there was too much information in the problem description. Subjects complained about the fact that they had to consider too many entities and detail, and that all information is scattered throughout the description. Subject 3 complained that they even forgot other parts of the description by the time they finished dealing with a specific part. Subjects also pointed out the fact that the information is incomplete, e.g., it is not clear what the effect is of the medicine which the doctor prescribes to the patient and what the initial level of sugar in the blood of the patient is. One of the biggest problems in modelling was caused by the incorrect formulation of the problem (see Section 3.4). Subjects 1 and 2 were confused about how to model the modulations in the level of sugar in the patient's blood. Subject 5 also experienced similar problems but the difficulty did not seem to stem from confusion about the description. The other two difficulties experienced by the subjects involved establishing the interaction between the prototype equipment and the computer system, and modelling of the module for the grandmother. Subjects 1 and 2 eventually realized that there is no need to distinguish between the equipment and the computer, while the other subjects proceeded to model them as components with distinguishable functionality. Subject 5 also had to dedicate more attention to determine the roles of the equipment and the nurse and how these modules interact. Subjects did not find it easy to determine the controllability of events (especially the events pertaining to the increase of the blood sugar level), however, this task posed greatest difficulty to subject 2.

The results of the bigram analysis for all subjects were plotted together to compare the patterns of low-level actions across subjects. The following trends can be recognized. During work with pen and paper, the different subjects had different patterns of activities (see Fig. 6.6). For example, subject 4 worked on events in continuous sessions more consistently than the other subjects (see the high relative ratio for the bigram 'EE'). Subject 1, on the other hand considered events in bulk less frequently than the other subjects (see lower relative ratio of the bigram 'EE' and higher relative ratios of 'ES' and 'ET'). This distribution does not correlate to the creation of legends of events by the subjects. Subject 1 did maintain a legend, however, they entered events in the legend “on the go”. Subjects 2 and 5, on the other hand, did not create legends of the events they used in their models, but the relative ratio of the bigram ‘EE’ is average. Only a look at the absolute ratio of event-related bigrams (bigrams containing ‘E’) reveals that these subjects did not work on events much. The absolute ratios of event-related bigrams for subject 3 are overall higher than for the other subjects. This indicates that subject 3 considered events frequently during problem solving; this is consistent with the observations in the study. The way subjects drew models also varied. Looking at the data, especially at the relative ratios, it seems that subject 2 tended to work on states or transitions in bigger chunks (e.g., drawing all states first and then drawing all transitions). This is indicated by the higher relative ratios of the bigrams ‘SS’ and ‘TT’ and the lower relative ratios of ‘ST’ and ‘TS’. The opposite is true for the subjects 1 and 4, where the tendency to alternate drawing states and transitions is more pronounced. Finally, subject 1 did not seem to consider modules for extended periods before starting to model them—as indicated by the relatively lower relative ratio of ‘MM’. Instead, the subject more frequently launched into modelling immediately after considering a module—as indicated by the higher absolute and relative ratios of the bigrams ‘ME’, ‘MT’ and ‘MS’. It seems that all subjects preferred to start drawing a model by creating a state instead of a transition (high ratios of ‘MS’, lower ratios of ‘MT’). This could be explained by the fact that states are the starting points and ending points for transitions.

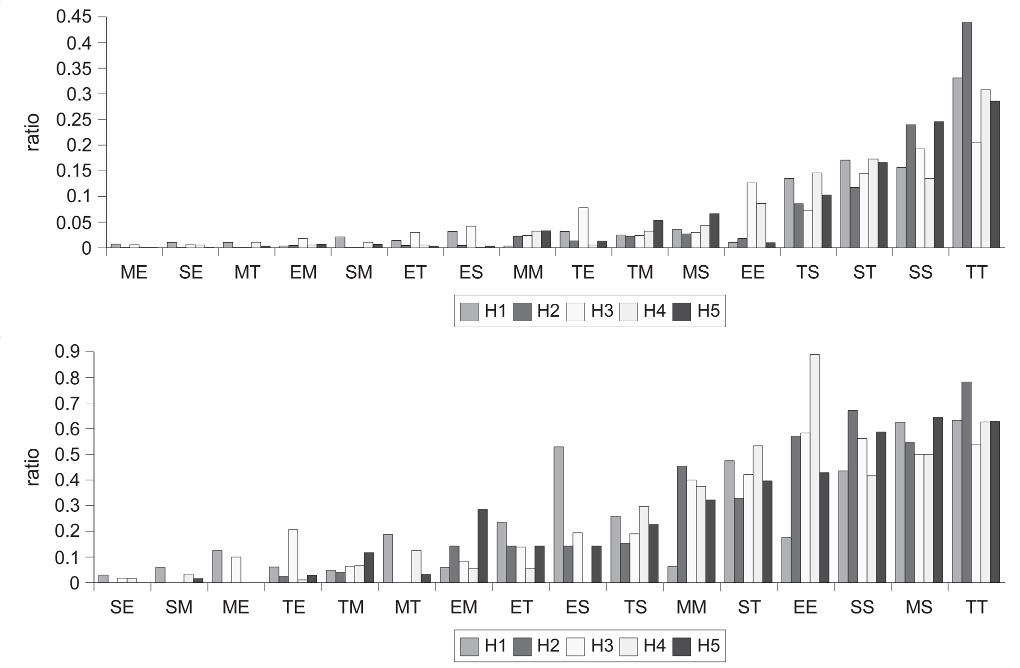


Figure 6.6: The absolute and relative ratios of bigrams for pen-and-paper problem solving during Hospital problem for subjects 1 to 5 (H1 to H5). Data are sorted according to the average ratios for all subjects.

The analysis described in Section 3.7 shows that on a few occasions there was cross-context data discovery. Most cases of data discovery occurred when working on the modules for the patient and the doctor. Besides the general interest in other modules, work on the module for the patient triggered more specific interest in the modules for the grandmother, the doctor, and the blood sugar level. Similarly, work on the module for the doctor triggered more specific interest in the modules for the hospital information system, the nurse, and the level of blood sugar. When working on the module for the computer, discovery concerned the modules for the prototype equipment and the doctor. When working on the control for the report of the blood reading, subject 5 referred also to the the model of the computer and to the problem description. Furthermore, working on the module for the prototype equipment, they also considered the module for the nurse. These results lead to the conjecture that possibly entities representing people are more likely than inanimate entities to serve as the "center nodes" of attention (e.g., the patient and the doctor) in relation to which the behavior of other system components is considered. On the other hand, in the hospital problem most of the entities represent people, thus such a conjecture is very tenuous. Other than this, the results do not seem to indicate any specific pattern of cross-context data discovery, except the natural and expected observation that discovery concerns mutually-related components of the system.

The sequence of modules receiving attention when the subjects formed their understanding of the problem is another source of information on how subjects mentally group the entities. Only two stable units of attention were noticed. The most stable unit is formed by the prototype equipment and the computer. The subjects always discussed these two entities together. The other apparent unit of attention concerned the report generated by the computer, the level of blood sugar, and the hospital information system.

When the information from the visual context is included in the analysis, however, more patterns start to emerge. In Fig. 6.7 it becomes apparent that the module for the prototype equipment (EQ) was most demanding to model. The subjects frequently referred to the problem description (DESC), to previous versions of the model (PREV), and to all other models (ALL). Similarly, it seems that the module of the patient and the supervisor for the report necessitated much reference to the problem description. Two significant groups of components stand out: 1) the patient and the specification for the blood sugar level, and 2) the doctor, nurse, computer, and hospital information system. Surprisingly, the interaction between the modules for the computer and the equipment is not as dominant. A potential explanation could be found in the fact that in most cases the two modules were modelled on the same sheet of paper. Thus, no change of attention could be recorded since the subjects did not need to look at a different sheet. Another surprising observation is the relatively weak connection between the supervisors and the corresponding specifications. However, it should be noted that the subjects had very little time remaining, if any at all, to work on the supervisors. Furthermore, most subjects chose to design either the control specifications (and then compute the supervisors) or the supervisors (and altogether omit modelling the specifications). The requirements for the supervisor for the report seemed to be less clear than for the other supervisor, since the subjects tended to refer to the problem description more frequently. Overall, it is hard to establish an overarching regularity in the graph of attention. There are many interactions and, ultimately, most components are connected to each other.

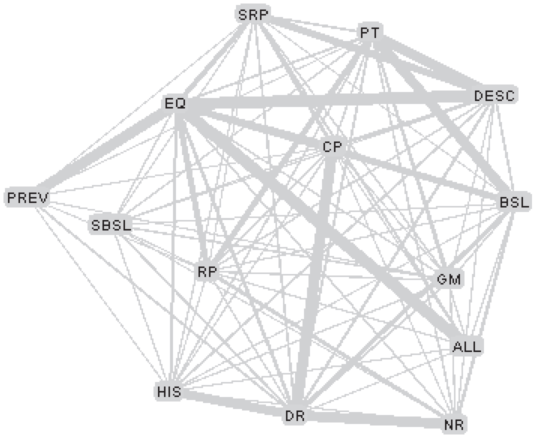


Figure 6.7: A graph of the frequency with which the subjects switched attention between different components of the hospital problem. The nodes signify the following components: patient (PT), doctor (DR), nurse (NR), grandmother (GM), prototype equipment (EQ), computer (CP), hospital information system (HIS), specifications for the blood sugar level (BSL), specifications for the flow of the report through the system (RP), supervisor for the blood sugar level (SBSL), supervisor for the report (SRP), all models (ALL), previous versions of a model (PREV), and problem description (DESC). The width of the edges signifies the relative frequency of switching attention between the given components (thicker line means higher frequency). The frequency is computed from the weighted aggregation of the data for visual attention and for verbally announced data discovery. The frequency was adjusted with an exponential function to allow easier visual differentiation.

6.5 Overall observations

The problem-solving activities of the subjects also have aspects which are not necessarily problem-specific or which can be compared better when considered across problems. These aspects include the way subjects modelled events and states, the use of diagrams, the factors that demonstrate the different levels of difficulty of the problems, the verification techniques the subjects applied, and the experience of the subjects with the software package.

All subjects, except subject 3, seemed to have a good understanding of the significance of events in discrete-event systems. Especially in the factory problem, subjects did not have any trouble deciding on how to extrapolate the relevant events from the description of the system behavior. In the hospital problem, this task seemed a bit more problematic since subjects were occasionally confused about what event granularity they should use (e.g., subject 4 used unnecessarily high event granularity) and which events should be used to synchronize the system components (e.g., subject 2 failed to recognize how the model of the grandmother interacts with the rest of the models).

In addition to choosing what events to use to discretize and model the behavior of the system, one also has to choose the controllability of the events—that is, which events the supervisor(s) for the system will be able to disable or enable as necessary. The distinction between these two tasks (choosing events and deciding on their controllability) is also demonstrated by the activities of the subjects. Only subjects 4 and 5 seemed to consider event controllability concurrently with the extrapolation of events from the problem description; and they did not do this consistently throughout problem solving. Subject 2 discussed event controllability even more sporadically. Subject 1 did not consider event controllability during modelling and in the factory problem, they decided on the controllability only when inputting models into the software where it is more or less unavoidable to specify this property of the events. Subject 3 did not consider event controllability at all. Thus, it is possible to conclude that the considerations of event controllability do not always go hand in hand with modelling of DES modules. In some cases, these decisions are delayed until one starts to consider the supervision of the system.

When naming events and states, the subjects exhibited different approaches. We distinguished four kinds of names (or labels): no name, abstract name, derived name and unabbreviated name. Abstract names are more or less arbitrary and they do not provide any insight into the semantics of the named entity. For example, the event for the administration of insulin may be called “θ” and the label for the state when there is one report in the hospital information system may be “6”. Derived names are labels derived from the description of the names entity and could be interpreted once the naming scheme is known. For example, the event for the administration of insulin may be called “i” and the label for the state when there is one report in the hospital information system may be “1”. Unabbreviated names are names which are descriptive enough to allow recognition of the named entity without knowledge of an encoding scheme. For example, the event for the administration of insulin may be called “admin insulin” and the label for the state when there is one report in the hospital information system may be “1 report”. The types of names used by the subjects are shown in Table 6.3. Only subject 5 used unabbreviated event names in both problems, while the majority of subjects used either abstract or derived event names. The predominant use of non-unabbreviated event names in the factory problem may be influenced also by the previous experience of the subjects since such names are used in the related problem in the ELEC843 course. Subject 4 mentioned that they tried to choose easy-to remember event names in the factory problem. The event names they invented consisted of indexed Greek symbols (such as α_3). Apparently, these scheme was not entirely intuitive to them either, since they explained that they “got used” to the naming scheme half-way through the problem. Subject 3 discussed the type of event names as well. They explained that short (and abstract) event names should be used to reduce clutter in models. However, they recognized arbitrary event names may pose a problem when trying to read the models, so the subject proposed that DES software provides a way to examine the descriptions of events when needed, e.g., by displaying a short description when the mouse cursor hovers above an event name. During problem solving subjects (including subject 3) sometimes maintained a legend of all events used with their description (when non-unabbreviated) and their controllability (when considered). In a sense, this is an implementation of the suggestion of subject

3, albeit using pen and paper. Subject 5 used unabbreviated event names and did not create legends. Subject 4 used unabbreviated event names in the hospital problem and they created a legend, however, the legend listed only the controllability of the events. While on most occasions it did not seem that the use of abstract event names posed any difficulty for the subjects, in the case of subject 3, as mentioned in Section 4.3.3, the event naming scheme may have contributed to their inability to advance in solving the problems.

	S1-F	S1-H	S2-F	S2-H	S3-F	S3-H	S4-F	S4-H	S5-F	S5-H
State names	D	A,D	D	D,U	U	U	N,D	N,U	A,U	A,U
Event names	D	D	D	U	A	A	A	U	U	U
Event legend	D	L	N	N	L	L	L	L	N	N

Table 6.3: The types of names used by subjects 1 to 5 (S1 to S5) in the factory and hospital problems (F and H) for states and events (N – no name, A – abstract name, D – derived name, U – unabbreviated name); and the use of event legends (N – none, D – denoting events on a diagram, L – creating a list of used events)

The naming of states seemed, in general, less important to subjects than the naming of events. Most of the time, states were either not named or given non-unabbreviated labels. Subjects tended to reconsider their state naming scheme, and make it more explanatory, only when faced with the composition of models. Having distinguishable state labels may significantly simplify the verification and understanding of such compositions. Nevertheless, states were usually given unabbreviated labels only when the states could be easily described verbally. For example, machines in the factory problem could be in two states, "idle" and "working". The number of parts in a buffer could easily be used to label the corresponding states (e.g., the state where there are two parts in the buffer could be labelled "2"). States which do not have an obvious short description, such as the state where the doctor examines the results of the readings from the patient and makes a decision about the treatment were usually named using an abstract label.

The creation of diagrams to visualize the structure of a system is a common way to help with the understanding of a system. A simple diagram was provided with the description of the factory problem. However, there was no diagram provided with the description of the hospital problem. We did not provide a diagram for two reasons. First, we did not wish to introduce a bias in the problem solving of the subjects. Second, a diagram may have revealed the similarity of the two problems. During the study, we observed that the the subjects did not, at least explicitly, use much the diagram in the factory problem. Only subjects 1 and 4 made notes on the diagram, while subject 2 looked at it but did not like it. In the hospital problem it was more apparent that the subjects actually used diagrams to help with the understanding of the problem. Subjects 1, 2 and 3 created graphical representations of the entities in the problem and the relations between them. Subjects 1 and 2 made especially elaborate drawings of the entities (see Fig. 6.8). Additionally, subjects 3 and 5 compiled a summary of the information in the problem. Subject 4 did not create separate diagrams or summaries, instead preferring to underline important points in the problem description. The use of extra tools to help understand and reason about a particular problem seemed to be much more important to subjects when they experienced greater difficulty.

The analysis of the shifts of attention of subjects during problem solving is shown in Figs. 6.4 and 6.7 for the factory and hospital problems, respectively. Additional information about the visual attention of subjects is shown in Table 6.4. Even though subjects did not voice all the shifts of attention—the analysis of data discovery (see Section 3.7) did not reveal much—it is clear that the subjects are very active in seeking information. Some subjects shifted their gaze between sheets of paper more than 500 times during the one-hour session! The more sheets of paper subjects used, the higher would be the expectancy of shifts of attention between sheets. Thus, Table 6.4 also contains the normalized data to account for this effect. It seems that most subjects were seeking information more actively during the hospital problem, however, the increase is not very significant and, furthermore, others (e.g., subject 2) seemed less active during the hospital problem. Similarly, no pattern is apparent in the frequency of reference to the problem description between the two problems. However, it is clear that the problem description was one of the most important element subjects referred to. The minimal proportion of attention shift to the problem description was about twenty percent, while the maximum was more than sixty percent. In particular, during the factory problem, more than two thirds of the time that subject 4 looked away from the model they worked on, they referred to the problem description. Obviously, any system designed to help with the problem solving should keep the problem description easily accessible to subjects.

S1-F	S1-H	S2-F	S2-H	S3-F	S3-H	S4-F	S4-H	S5-F	S5-H
Shifts of visual attention context									
139	504	260	147	193	315	490	519	191	295
Number of sheets used									
2	6	2	3	3	5	3	3	4	4
Shifts of context per sheet									
69.5	84	130	49	64.33	63	163.33	173	47.75	73.75
Attention to the problem description									
40%	25%	39%	50%	18%	22%	68%	45%	30%	38%

Table 6.4: Shifts of the context of visual attention by subjects 1 to 5 (S1 to S5) in the factory and hospital problems (F and H). A shift occurs when the subject moves their sight onto a new sheet of paper or when the subject activates a new model in the software.

When examining the performance of subjects individually, it was noticed that all of them except subject 4 found the hospital problem more difficult than the factory problem. (The relative ease of the hospital problem for subject 4 was due to their (faulty) interpretation of event controllability. Considering some key events controllable results in a trivial solution of the problem.) A number of aspects of the subjects' problem solving support this conclusion. The most obvious aspect is the progress of the subjects. Looking at Tables 6.1 and 6.2, it is easy to see that each subject advanced further during the factory problem. However, there are other indicators as well.

It can be argued that the time spent solving a problem is an indicator of the relative difficulty of the problem, given the disposition of the subject. As Simon argues [53], subjects

[redacted]

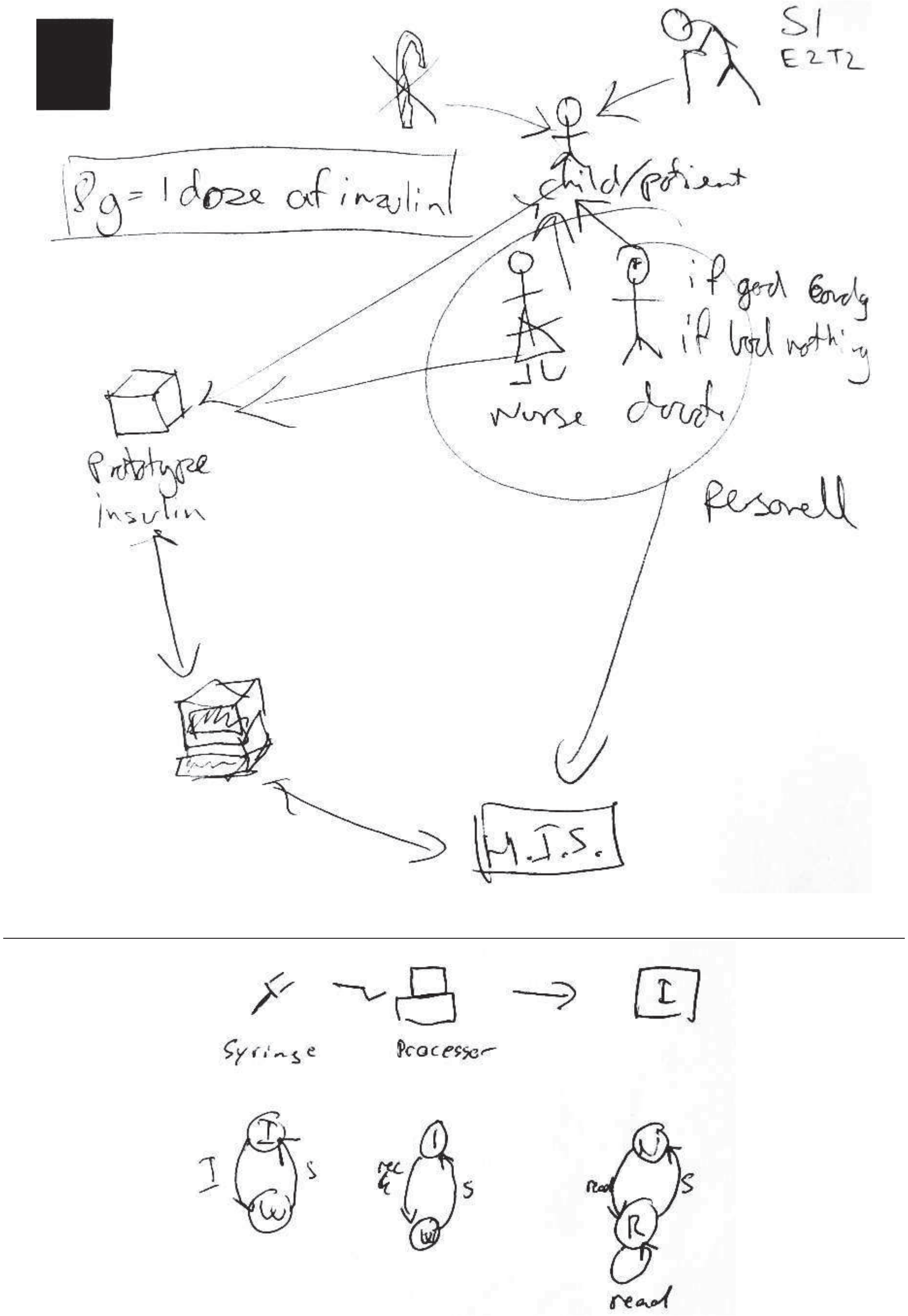

S1
E2T2
8g = 1 doze of inzulin
child/patient
if good Condg
if bad nothing
Nurse
doctor
Resovell
Prototype insulin
M.I.S.
Syringe
Processor
I
I
S
I
rec &
S
W
W
read
S
R
read

Figure 6.8: Reproductions of some of the diagrams created by subjects when solving the hospital problem.

use satisficing when solving problems. That is, a subject would not continue problem solving after their solution reaches a degree of quality. Thus, if a subject has the same disposition, the longer process of problem solving indicates longer time to reach the same degree of quality of their solution, i.e., the problem is more difficult for the subject. In our study, only subject 1 managed to complete their problem solving, and only of the factory problem. In the other cases, subjects ran out of time. Thus we could not look at the total time taken by subjects to complete their problem solving. As most of them, with the exception of subject 3, completed at least the modeling of the system components and the control specifications (where applicable), we decided to examine the time taken to complete this part of the problem solving. In the case of subject 3, we examine the time before the subject gave up problem solving. The results are shown in Table 6.5. All subjects, except subject 3, spent more time modelling the DES modules during the hospital problem. It is true that a part of this increase is due to the higher number of modules the subjects modelled. While the structure underlying the two problems is the same, subjects invariably modelled more modules in the hospital problem. The factor of increase of the number of modules is also shown in Table 6.5. It can be seen that it is lower than the factor of the increase of time for modelling. Thus, even when taking into account the higher number of modules, there was an increase of the time subjects spent modelling. The higher difficulty of the hospital problem is also indicated by the fact that they spent a smaller proportion of the modelling time actually performing modelling activities (such as drawing models, labelling diagrams, creating legends for the events, etc.), as seen in Table 6.5. If we assume that the subjects were completely engaged by the problem, a larger portion of their time was taken by considerations of what to do rather than by doing things. The only exception in these data is subject 3. Their data is contrary to the data for the other subjects. While their performance can be considered an outlier, there is an additional explanation of the result. Subject 3 was administered the hospital problem first and they gave up fast because they did not know how to proceed. When solving the second, factory problem, the subject seemed to put more effort into the solution. They recognized they had already seen the problem and apparently they did not feel comfortable with the thought of failing to solve a second problem, especially since it is a known problem.

S1-F	S1-H	S2-F	S2-H	S3-F	S3-H	S4-F	S4-H	S5-F	S5-H
Total time modelling									
12:16	49:28	15:57	46:59	46:06	34:36	22:42	39:40	8:24	46:09
Factor of increase (time)									
	4.05		2.99		0.75		1.76		5.59
Factor of increase (number of models)									
	2.5		1.8		1.33		1		3
Proportion of time performing actions									
20%	17%	28%	17%	7%	18%	15%	10%	44%	29%

Table 6.5: Data about the DES modelling by subjects 1 to 5 (S1 to S5) in the factory and hospital problems (F and H). Time is in minutes and seconds.

The number of perceived mistakes, as described in Section 6.2, can also be used as an evaluation of the difficulty subjects experience with a problem. The data collected from the observational study are shown in Table 6.6. Surprisingly, type 1 mistakes are much more common during the solving of the, supposedly, easier factory problem. Type 2 mistakes do not form a stable trend. Some subjects made more type 2 mistakes during the factory problem, other made more during the hospital problem, while yet others made the same number of type 2 mistakes in both problems. Type 3 mistakes, on the other hand, were clearly more common during the hospital problem (with subject 4 being the only exception). This fact supports the conclusion that the hospital problem was more difficult for the subjects. However, it also contradicts the results for the type 1 mistakes. Three potential explanations are as follows:

- During the more difficult problem, the subjects focused on higher-level problem solving;
- Since the subjects were more uncertain of the correct solution during the more difficult problem, small mistakes in the models were harder to recognize;
- During the more difficult problem, the incorrect models more frequently required too many corrections and the subjects chose to remodel them instead of applying small corrections.

	S1-F	S1-H	S2-F	S2-H	S3-F	S3-H	S4-F	S4-H	S5-F	S5-H
Type 1	26	19	18	12	28	7	12	5	40	31
Type 2	9	9	4	4	7	0	4	1	4	13
Type 3	0	0	0	5	0	4	1	0	4	9

Table 6.6: Number of perceived mistakes according to type during problem solving by subjects 1 to 5 (S1 to S5) in the factory and hospital problems (F and H). See Section 6.2 for explanation of types.

A more objective examination of the errors in the solutions of the subjects reveals an interesting trend. Most of the errors were caused by incorrect reasoning about events. These errors can be grouped into four categories.

- Incomplete description of the dynamics of a module, missing events;
- Problems with event granularity, either superfluous or insufficient granularity preventing the design of a solution;
- Mismatched events, the interaction of modules is incorrectly modelled due to improper event coupling; and
- Incorrect event controllability, the subject's interpretation of the problem description is incorrect and the controllability of events is not determined properly.

All subjects made one or more of these errors at some point in their problem solving, with the incorrect determination of event controllability being the most frequent one. In fact, only subject 3 did not commit a controllability error, however, they did not consider controllability at all during the problem solving. Other, less common, errors which did not involve events were not so universal. Subject 1 made errors in determining the marking of states and the granularity of states. Subject 3, as described in Section 4.3.3, did not apply the conventional approach to using finite-state automata and their models could not be interpreted within the DES framework. Subject 4 did not model the control specifications correctly. Instead of modelling what the desired system behavior should be, they modelled the physically possible system behavior. The last two errors are indicative more of a lack of knowledge rather than of problems with interpretation. Thus, it is possible to conclude that the greatest cause of incorrect solutions are errors related to events.

Most subjects did not manage to advance far enough with their solutions to go through extensive verification. During the few times the subjects did perform verification or discussed their verification techniques, the following observations were made. The most common technique of verification was tracing of sequences of events in the finite-state automata of the models. The subjects would start at the initial state or at a known state and then follow the transitions in the model sequentially according to a sequence of events they had in mind (the sequence could also be constructed during the tracing). This verification could uncover a number of problems. These include

- The chosen sequence of events should not occur if the model is correct, but there are transitions that make it possible,
- The chosen sequence of events should be possible, but there is no sequence of corresponding transitions, and
- The chosen sequence of events leads to the wrong state in the model.

Two subjects pointed out that visual inspection of the model, and visual tracing is very important to them. For example, it would allow them to see if there is a general pattern in the transitions and if they "flow" in the expected direction. Other verification techniques employed or mentioned by the subject include checking if the transitions exiting different states are consistent with the expectation (which is a variation of tracing), checking the number of states and/or transitions against the expectations, checking the controllability of a supervised system and performing black-box testing. Subject 1 mentioned that, when checking the behavior of a controlled system, it is important to try to relate it to the corresponding real-world system and to answer the question if it makes sense in the real settings. The subject also mentioned that, in their opinion, an exhaustive manual analysis of models with more than thirty states seems infeasible.

Even with the application of the above verification techniques, it is impossible to predict how many errors or what type of errors would be discovered and corrected before the completion of the problem. However, it is reasonable to assume that many of the errors would remain undetected, since they are the result of incorrect interpretations of the problem descriptions. For example, the incorrect determination of event controllability does not result in any specific conditions which would hint at this type of error. A subject could produce a complete solution, all the while not being aware that they have determined the controllability of events incorrectly.

The subjects were able to use the IDES package if they chose to use software for their problem solving. Feedback on the software was not extensive, however, subjects seemed to have an overall positive experience with it. Some features won praise, e.g., the ease of creating and naming models, the ability to drag elements on the drawing canvas, the option to enter events in bulk, and the aesthetics of the rendered labels. On the other hand, there seemed to be also a general reluctance among the subjects to use the software if they could complete the tasks with pen and paper. No subject started problem-solving using the computer. Only subject 3 used pen and paper and the software in a seamless fashion. The subjects who commented on the topic said that they have a personal preference for pen-and-paper modelling and that they consider the use of software only when it is *necessary*, e.g., when the system is too large and complex. On the other hand, the threshold of complexity could be quite low. Subject 2 mentioned that they would consider using software even for a system with only eight states. According to the subjects, other deterrents from the use of software are the facts that it requires preparation, e.g., events have to be input first, and that the input of models is tedious and time consuming. The subjects also mentioned their dissatisfaction with the lack of certain features in this version of IDES. A list of these features is given here:

- Duplication of events across models, or a central repository of events which can be used in multiple models,
- Better automatic layout of models, or a selection of different layout algorithms,
- Duplication of finite-state automaton models (this feature was available, but the subject did not recognize it in the user interface),
- Algorithms for “self-looping” (or inverse projection) and checking two controllers for *conflict*,
- Selection of multiple (more than two) models with multiple-input algorithms, e.g., parallel composition,
- Printing of models,
- Display of detailed information about (e.g., description of) events when requested, and
- Support for diagrams in the software.

One of the concerns the subjects had was the quality of the automatic layout of models. In order to understand the importance of better visual representation of information, we examined the number of times the subjects corrected the automatic layout of the models or the layout of the user interface. The results are shown in Table 6.7. It can be observed that normally subjects did not adjust the visual layout very frequently. Using an estimation of two seconds per adjustment action, it can be computed that subjects spent well below ten percent of the total time of software use adjusting the layout. The only exception is subject 1, who spent about a quarter of the total time—a very significant part, indeed. Through the observation of this subject’s actions, it can be seen that the majority of their layout adjustment occurred during the verification of newly computed models, and most prominently, during the verification of the final supervisory solution. The other subjects did not go through this stage of problem solving. It is important to note, however, that the time spent to make adjustments does not necessarily mean that the subject's attention is exclusively dedicated to this task. As observed in the performance of subject 1, layout adjustment usually occurred concurrently with the process of verification, e.g., with event sequence tracing. Thus, there is some indication that layout adjustment is tied to other tasks. It could be that different kinds of layout are necessary for different verification techniques.

	S1-F	S2-F	S3-F	S5-F	S5-H
Number of corrections	257	38	40	16	17
Time taken by corrections (estimate)	8:34	1:16	1:20	0:32	0:34
Time using software	34:49	28:58	20:11	16:31	9:11
Percent of time taken by corrections	25%	4%	7%	3%	6%

Table 6.7: Data about the corrections of the layout of models and the user interface done by subjects 1 to 5 (S1 to S5) in the factory and hospital problems (F and H). There are data only when the subjects used software in their problem solving. The time taken to make corrections is estimated, assuming each correction took two seconds. Time is in minutes and seconds.

Chapter 7

Discussion

The observational study of problem solving in control of discrete-event systems resulted in large quantities of data which allow the examination of many aspects of the task. The small sample of subjects does not make statistical tests of hypotheses possible. Indeed, this study can be interpreted more as a exploratory study which should provide the information needed to create other, targeted examinations. However, for our purposes, i.e., discovering patterns and strategies of problem solving, the study bore much fruit. In this chapter, we will summarize some of the analysis from the previous chapters and discuss the observations in light of our goals.

7.1 Global strategies

Global strategies are the high-level, overarching approaches subjects take in solving a problem. The global strategies employed by the subjects in this study were varied and did not, in general, follow the strategies described by the experts. Only subject 1 seemed to follow closely the traditional (advocated) sequence of steps: model system, model specifications, apply algorithm for automatic generation of supervisor. All other subjects deviated in one or more ways from this procedure. In the well-known factory problem, the traditional strategy seemed advantageous, i.e., subject 1 was the only subject who managed to complete the problem. However, in the novel hospital problem, this did not seem to be the case. In fact, subjects 4 and 5 advanced further in the problem solving within the allotted time.

All subjects adopted a modular approach to their solution, i.e., they modelled components separately. Some subjects even considered modular supervision, where separate supervisors deal with separate aspects of control. These approaches are in line with the strategies used by the two experts (see Chapter 5). Furthermore, subjects, just as the experts, attempted to make a distinction between system components and components of the control requirements. For example, in the factory problem, the buffers need not be modelled as system components; it is sufficient to model the control requirements for prevention of overflow and underflow.

Beyond the above similarities, the strategies of subjects generally differed from the strategies of experts. Two of the most important differences are in the ways subjects deal with control specifications and with supervisors.

One of the most surprising conclusions from our study is that most subjects did not consider it necessary to formally model the control specifications. Only subject 1 (in both problems) and subject 4 (in the factory problem) modelled the control specifications rigorously. At the heels of this observation comes the fact that most subjects did not attempt to make use of the algorithm for automatic generation of supervisors from models of the system components and models of the control specifications. It is not entirely clear if specifications were not modelled formally because there were no plans to use the algorithm, or if the algorithm was not used because the specifications were not formally modelled. It can be pointed out, however, that subject 4 did not use the algorithm even after they modelled the specifications in the factory problem. Instead of generating supervisors automatically, most subjects designed the supervisors manually. Subject 2 did not even seem to be able to distinguish between models of specifications and models of supervisors. These results contradict what both experts who were interviewed explained as their strategies, namely, the reliance on the automatic generation of supervisors.

The above observation leads to the conclusion that *one of the most central aspects of DES control theory—the existence of an algorithm for the automatic generation of supervisors—was not taken advantage of* by the subjects in our study. This realization makes us ask, "why is this the case?" Were subjects not aware of the existence of this algorithm? This seems highly unlikely as the explanation of the algorithm is an essential part of the ELEC843 course which all subjects had taken. This course includes also assignments which require the application of the algorithm. Furthermore, at the time of the study, a few of the subjects were actively pursuing research interests in the field. Apparently, the reason for this reluctance to use the algorithm lies elsewhere. Based on this study and on a number of other personal observations and experiences, we attempt to provide some plausible hypotheses:

- The problems were relatively small in size and the subjects believed they could construct the correct supervisors manually.

- The subjects were more comfortable using pen and paper rather than software in the process of problem solving and they tried to minimize the time and effort in obtaining a solution (especially as the time of the session was limited).

- The automatic construction of a supervisor results in an "unknown" model where a significant amount of effort has to be invested for understanding it and verifying it. The manual construction of a supervisor results in a model which is already understood (since it was constructed by the subject) and is verified to some extent (since its construction requires cross-referencing with the existing models and comparison against the control requirements).

- The formal modelling of specifications as finite-state automata may be contrived and not natural in some cases, thus subjects were reluctant to expend the effort.

- The phrase "control specification" is ambiguous in the sense that it may either signify the requirements for the operation of a system or the specific control decisions made to satisfy the requirements. Subject 2, in particular, seemed to have this confusion.

- The phrasing of the problem description, namely, the statements "Provide a control solution..." and "You should use pen and paper for your modeling. You may use the IDES software for computationally demanding tasks." (see Appendices A.1 and A.2), might have misled the subjects into believing they must construct the supervisors manually and only use the algorithm when a manual solution is infeasible.

The manual construction of supervisors is not, in itself, a "bad" strategy. After all, this is how most controllers had been designed before automated methods became available to industry. Even now it seems that this is a widely used methodology [14, 10]. The automated construction of DES supervisors, however, provides an advantage over manual work in the fact that it not only guarantees correctness of the supervisors (i.e., that all requirements are satisfied), but also the generation of optimal supervisors (i.e., that all requirements are satisfied in the most permissive way). Normally it is not hard, for small projects, to create correct supervisors manually. However, the construction of optimal supervisors is frequently an infeasible task, given the size of the space of potential solutions. Psychological research, [65], indicates that this task may be even harder for humans. Human decision making seems to be biased towards direct implications and regularly fails to consider the reverse of such implications. Correctness can be naturally determined using direct implications (*modus ponens*), e.g., "if the buffer is full and the machine deposits a new part, then the buffer overflows". On the other hand, optimality frequently requires reverse implications (*modus tollens*), e.g., "if the buffer does not overflow (is not full), then the machine need not be prevented from depositing a new part". The second type of conclusion may be significantly harder for control engineers to make. In our study there was no observation leading us to conclude that a subject was considering the optimality of their manually-designed supervisors. On the other hand, subjects frequently engaged in verifying the correctness of their solutions. Arguably, the use of the algorithm for automated generation of supervisors in DES is preferable not so much because it guarantees correctness but because of the optimality of the solution.

In order to understand which factor (or combination of factors) contributed to the reluctance of subjects to employ automatic generation of supervisors, further studies will be necessary. Nevertheless, some recommendations can be made from our current observations. First, the benefits of the algorithm for automatic construction of supervisors have to be emphasized more in teaching DES control theory. Examples in class should employ the algorithm rather than use substitute solutions by hand. In relation to this, DES software should make the application of the algorithm as simple as possible. Second, strategies to understand and make use of automatically generated supervisor models need to be an essential part of a DES control course. In relation to this, DES software should make generated control solutions more transparent. For example, the representation of a supervisor could make the relation between the supervisor and the system components more obvious, e.g., by highlighting the relevant parts of system components when a specific control decision is examined by the user. Third, the purpose of control specifications has to be explained in detail to practitioners of DES control. The ambiguity of the terminology has to be dispelled. The formal models used for all entities—system models, control specifications and supervisors—are the same (i.e., finite-state automata). Thus, confusion as experienced by subject 2 (mistaking supervisors for control specifications) and subject 4 (assuming control specifications need to represent the unrestricted behavior of a component) is natural. An in-depth examination of the specific purposes of the different kind of entities could help clarify the difference.

7.2 Local strategies

Problem solving does not depend only on global strategies. Simpler strategies applied at individual steps of the process are also very important. These strategies, which we call "local", will be discussed next.

One of the problem-solving strategies we observed can be described as "easy first". Using this strategy, the subject would first concentrate on completing the tasks they know how to complete and only after they would look into more difficult tasks. Based on our observations, we believe that this strategy was employed by most subjects implicitly. It was most apparent, and explicitly mentioned, during the problem solving of subject 2. In the hospital problem, they experienced difficulty modelling the module for the grandmother. They repeatedly postponed this task until they finished modelling all easier components.

Another strategy employed by subjects was "divide and conquer". Using this strategy, the subject would break down hard tasks into easier-to-complete components of the tasks. This was most apparent in the problem solving of subject 5. In the factory problem, they started by trying to create a monolithic model for the whole system. As the model became larger and more difficult to model, they chose instead to model each system component separately. In a similar fashion, after failing to produce a single supervisor for the first buffer, the subject chose to construct two separate supervisors: one for prevention of underflow and one for prevention of overflow. At the same time, they did not break down the supervisor for the second buffer; and it can be argued that its control task is much simpler. A similar observation can be made in the problem solving of subject 2: they constructed two separate supervisors for buffer 1 but a single one for buffer 2. All other subjects employed the "divide and conquer" strategy to some extent as well. Even the basic choice to model a system in a modular rather than a monolithic way is an application of this strategy. It is interesting to note that analogical reasoning seems to be one of the main factors guiding the application of this strategy for the subjects in this study. They were aware of the similarity between the factory problem and the problem they had seen in the ELEC843 course. In breaking down difficult control tasks, they broke them down in the same way that they had seen in class. Furthermore, during the hospital problem some subjects used the same way of breaking down the tasks. They used terminology suitable for the factory problem and which is not, however, natural within the new context.

During problem solving, especially in the hospital problem, subjects complained that they had to deal with too much information. While this was not made explicit, one could hypothesize that the subjects complained when they found that the information they needed to consider was larger than the capacity of their working memory. The strategies subjects used to cope with the amount of information are listed next from least demanding to most demanding:

- Underline (or otherwise mark) specific sections of the problem description. Such markers make it easier to locate relevant information as the visual attention is automatically attracted to the sections deemed important by the subjects.

- Create a summary of all the information given in the problem description. Such summaries are much shorter than the full text. Furthermore, subjects can organize the information in a "personalized" way; as shown in human memory research, [37], different people may use unique cues to remember and recall a given piece of information.
- Draw diagrams representing the entities in the problem and the relations between them. Such diagrams seem to be most helpful since they provide both an overview of the situation in the problem and immediately-accessible information about the context of any component.

All the above approaches show that, usually, the organization, and accessibility of information in a problem description is not suitable for the building of an overall understanding of the problem or for the referencing needed during problem solving. Subject 2 made this most explicit by commenting that they did not like how the information was organized in the description of the hospital problem.

Our recommendations for DES software are as follows. First, it is clear that users must not be forced to follow a specific order of steps in modelling unless the completion of these steps is necessary for the application of an algorithm or otherwise for advancement in the problem solving. For example, a user should be able to compute a supervisor for a part of the system components if all relevant components have been modelled—even if not all components are complete. Second, users should be given the option to customize the organization of information about a problem. The most natural way to achieve this is outside of software, using pen and paper. However, in environments where the use of paper is infeasible, or if documentation about all design steps has to be retained, the software should provide a way for annotations of models and, if feasible, for the creation of free-form diagrams. If lists of items are used, the user should be able to re-order these lists in custom ways.

Our recommendation for teachers of applied DES control theory is to keep in mind the "easy first" and "divide and conquer" strategies, and to make them more explicit in their teaching. Knowing that the successful application of the "divide and conquer" strategy is highly dependent on previous experience, teachers should aim to show a wide range of cases when breaking down hard tasks can be used and to give examples that show how it is done. This will give students a richer set of cases from which to borrow ideas when solving other complex problems.

7.3 Human factors

The specific problem-solving strategy chosen by an individual depends not only on objective properties such as problem size or difficulty. Human factors, such as previous experience and attitude, also play a role.

In the observational study we noticed many occasions where previous knowledge and experience shaped the way subjects solved the problems. As already discussed, knowledge of the problems solved in the ELEC843 course had an effect on the form of the solutions for the factory problem, and even, on some occasions, for the hospital problem. However, there were influences also from previous knowledge which is not as specific. Subjects 1 and 5, for example, decided to introduce into their models of the machines the additional behavior of breakdown. Apparently, this was done because it made sense to them that machines can fail, and not because this behavior was mandated anywhere. Looking at an even larger scale, the unconventional, in DES, interpretation of finite-state models by subject 3 can be explained to some extent by their research background. At the time of the observational study, the subject used in their research the same type of models, however, in a completely different setting. The interpretation they demonstrated seems to be conventional in this other field. Thus, it appears that previous experience played a large role in the failure of that subject's problem solving.

A person's attitude to the problem solving may influence the process as well. In our study we could observe two diametrically different attitudes. Subject 4 was very deliberate. Their problem solving was slow and cautious. It seemed that they wanted to make sure that every action was correct. In the hospital problem, they modelled the smallest number of models (5) and they made the smallest number of corrections to the models (5). Subject 5 was, on the other hand, very prolific. Their problem solving was fast and experimental. It seemed that they want to offset all the mistakes they committed by quick introduction of new variations—leading to a gradual conversion to a correct solution. In the hospital problem, they modelled the largest number of models (20) and they made the second largest number of corrections to the models (31). The latter number is rivaled only by the number of corrections the same subject made during the factory problem. Even though the two approaches were completely different, both subjects advanced at a very similar rate during the sessions (see Tables 6.1 and 6.2). Thus, based on our observations, there is no ground to recommend one of the approaches over the other.

Human factors can vary along a very large scale, thus, it is hard, if at all possible, to recommend anything specific in order to accommodate different personalities and levels of experience. To teachers of DES modelling, we recommend training students to recognize when design ideas come from their personal experiences and when the ideas are based on the hard information in the description of a problem. Students must not be discouraged from using previous experiences, however, we believe the ability to differentiate between background knowledge and hard information may lead to better designs with less confusion about design decisions. To developers of DES software, we recommend keeping in mind that every user may have a unique approach to problem solving. The software should be flexible in order to accommodate this variation. Furthermore, different users may have different levels of DES knowledge. Scalable degree of simplicity in the interface, with simple interaction for novices, and up to full representation of the complexity for experts, is a desirable property of software systems [12].

7.4 Events

Events, one could argue, form the essence of discrete-event models. The choice of events when modelling (i.e., granularity, controllability, synchronization, naming, etc.), thus, is critical for the success of a supervisory solution. As discussed in [68], the event set has immense influence on the form of a solution. Furthermore, direct observations during this study showed that just as systematic errors may lead to a lack of success (subject 3, poor synchronization of modules), a single incorrect choice could throw the solution off track (subject 4, incorrect choice of event controllability). It is thus unfortunate that, in our observations, problem solvers experience the most difficulty with the determination of what events to use and how to use them. When working with events, the major problems experienced can be summarized as problems with the determination of

- Granularity of events,
- Synchronization of models, and
- Controllability of events.

The granularity of events stands for the level of detail used in the modelling of the behavior of a system. For example, the work of the computer in the hospital problem could be described as the acquisition of data from the prototype equipment and the subsequent preparation of a report, or simply as a single event combining the acquisition of data and preparation of a report. In general, the greater granularity, i.e., the more detailed description, is not detrimental to the correct solution of a problem. As well, some experts advocate a detailed description (see Section 5.1). However, such detailed descriptions are not always needed. The models become much larger and more difficult to understand and manipulate. In some cases they may even cause the formation of computationally infeasible solutions. Subject 1 mentioned that they tried to use the smallest models possible specifically for this reason. Greater granularity also implies that more choices will have to be made about the controllability of events and the correct synchronization of DES components. This creates more opportunities for making mistakes.

Events are used also to accomplish the synchronization of DES models. For example, the output of a part by a machine in the factory problem has to be synchronized with the acceptance of a part by the corresponding output buffer. In order to achieve this synchronization, the models of the machine and the buffer have to use the same event for the same "action" (that is, the names of the events have to be identical). Then, the DES algorithms will be able to recognize properly how the modules interact. Subject 3 did not realize this essential proposition and their models were not usable with the available mathematical tools. Other subjects knew about this fact, however, they occasionally struggled to determine which events had to be used for the purpose of synchronization. To some extent, this problem is related to the problem of choosing the right granularity of events. Detailed component models do not contribute to a correct solution unless the detail lets one discern component states (or event sequences) at which different control decisions have to be made or unless it allows the correct establishment of synchronization between models. The greater the choice for synchronization points, the more opportunities for making a mistake.

Our recommendation is that models be modelled using the lowest possible granularity first, with subsequent granularity refinement as needed. Each model may be viewed as a black box which has a number of input/output events, or events which are used to interact (synchronize) with the environment and the rest of the models. The model itself, then, only needs to specify in what sequences these events may occur. For example, a machine in the factory problem interacts with the rest of the world only through the input of an unprocessed part and the output of a processed part. The sequence of such events need only describe that an output cannot happen unless there is a corresponding input and, once there is an input, another one cannot occur until there in an output. In this case, it is not necessary to describe exactly what operations the machine performs to convert an unprocessed part into a processed part. In some sense, this approach is similar to that of hierarchical DESs, e.g., [64]. Thus, it could find potential use also when modelling systems for different levels of control.

The situation with the controllability of events is not that simple. In the initial proposal for DES control, [45], it was assumed that events are inherently controllable or uncontrollable. When centralized control is used (a single omnipotent supervisor), this is a valid assumption. Controllable events would be the ones which the supervisor would be able to disable (i.e., prevent from occurring). All other events would be uncontrollable. This is also the way in which DES control is introduced in teaching—tying controllability to the events. However, this assumption is fundamentally wrong when one considers a system controlled by a number of entities. For example, even at the most trivial level, most machines have an "emergency stop" button accessible to human operators. This button can override all functionality (all events) of a system and prevent the further progress of operation. However, the microcontroller of a system usually does not have a similar capability. When one considers all microcontrollers in a given system, a similar observation holds. The microcontroller for machine 1 may not be able control the operation of machine 2, while all activites of machine 2 may be perfectly "controllable" for its own dedicated controller. Thus, the controllability property is not an inherent property of events. Rather, it depends on the agent who performs the control [52]. In this sense, the modelling of the behavior of a system and the relevant specifications does not require the determination of the controllability of events; this is an independent process. Only when the proper supervisor will be designed, is it necessary to determine which events will be controllable by the supervisor. The performance of the subjects in our study shows that for many of them the determination of controllability of events is a separate process from the modelling of behavior. However, in other cases, this was not so. Especially when using the software, the interface for event entry urges the users to associate the controllability with a model rather than with the supervisory solution. Experts also seem to occasionally consider event controllability concurrently with the modelling of components.

Our recommendation is to emphasize the fact that event controllability is not an inherent property of events and that it completely depends on the point of view of the entity exercising control. This will be helpful especially in situations when event controllability is not apparent—such as in verbal descriptions of complex systems. In the hospital problem, subjects occasionally determined controllability of events based on the point of view of the entity described with the events. For example, subject 4 determined that the events prescribing candy or medicine to the patient are controllable since the doctor has control over which decision they make. However, from the point of view of a computerized supervisor, these events are uncontrollable. In a hospital, the doctor has ultimate authority (and responsibility) for the treatment of a patient. Thus, a supervisor must not have the ability to prevent or enforce an alternative treatment method. The mistake of subject 4 (and the subsequent erroneous problem solution) could have been avoided if the proper point of view were assumed. Software interfaces should decouple controllability of events from models and associate it with the corresponding supervisors.

The naming of events is, theoretically, not relevant to the construction of DES solutions. As long as different events have different identifiers, and the same events have identical identifiers, the form of the identifier is irrelevant. However, practical applications of the theory require the choice of a specific naming strategy. The subjects demonstrated three different naming strategies, as discussed in Section 6.5. The observations show that, in general, using arbitrary event names did not cause confusion in the process of problem solving. Subject 4 mentioned that they tried to choose a scheme which "made sense" but their naming scheme was apparently arbitrary. They also said that it took them some time before they became comfortable with the event names. Thus, it may be conjectured that arbitrary naming schemes will be disadvantaged compared to a naming scheme where event names reflect in some way the meaning of events. Descriptive event names did not cause confusion either, however, it was observed that very elaborate names tended not to be replicated identically in all instances. A further conjecture, then, is that very elaborate names may lead to naming inconsistencies (as observed with subject 5 on one occasion). The subjects did not always maintain a list of the events that were used—and it may not be necessary for such a small problem. Expert 1 also does not normally use events lists. However, based on personal observations, more complex systems place extra demand on the creation of meaningful naming schemes. Keeping all event names in memory may not be possible, and further complication occurs when models have to be interpreted by individuals different from the model creators. Lastly, as discussed previously, synchronization of models and specifications is done through identical events. Inconsistencies in the naming of events may result in hard-to-find errors in the synchronization of components. This problem is exacerbated with the growing number of events to be considered in complex problems.

Our recommendation is to use concise names of events, derived from the meaning of the events, whenever possible. A list of events used with their descriptions should be maintained. This will help in cases when a sizable number of events is used and/or when models have to be interpreted by other people. This idea can be advanced further using the appropriate software interfaces. Research in information visualization [55] advocates the use of an overview+detail paradigm. Design software for DES may easily support such a visualization. A list of all events used in a model or a number of models can provide the overview—and it can be always current if the software maintains it dynamically while users work. The details about events (such as a verbal description), as per subject 3's recommendations, could appear when the user chooses to see it. This could be both in the event list and within usage context, where the event appears in a model. Further support can be offered for the synchronization of models. If the user has access to the list of events from all models, they could choose a specific event for use in their current model without having to recall the exact name. Also, when the synchronization of different events from existing models is necessary, manual event renaming can be avoided. Instead, an automated event transformation could be available to the user.

7.5 Control specifications

As discussed in the earlier sections, the subjects in our study experienced confusion about the use of control specifications in problem solving. In some cases this was apparently due to the lack of sufficient knowledge. However, arguably the form of control specifications proposed in the seminal work of Ramadge and Wonham, [45], has many drawbacks.

In systems control, controllers have to guarantee two types of requirements: the controlled system must not exhibit undesired behavior, and it must exhibit desired behavior. In our opinion, DES control theory is very well suited for the first type of requirement. In DES, control is exercised by *preventing* the occurrence of of events, i.e., by preventing the exhibition of undesired behavior. However, the theory is not suitable for the second type of requirements. It is not instrumented to effect a specific behavior—event occurrences are purely spontaneous. The only tool control engineers have to specify desired behavior is the use of *marked* states. When states are marked, any acceptable supervisor must ensure that no matter what sequence of events occurs in the system, the system is able to return to a marked state. In this sense, states signifying the completion of tasks can be marked and, consequently, imply control which guarantees the completion of tasks. Theoretically, this is not a significant drawback. However, the practical application of the theory is quite demanding. As seen in this study, and in our personal experiences, it is difficult to decide which states exactly need to be marked to “enforce” the desired behavior. Real systems usually include both spontaneous events (e.g., the completion of the processing of a part) and enforced events (e.g., the command for the start of part processing). Standard DES control specifications require a very significant shift of the control paradigm: from an active/reactive paradigm to a passive paradigm. Not only does this contradict the traditional way of thinking in control engineering, but it also leads to the issue of thinking with direct versus reverse implications, i.e., specifying what should happen by using prevention (see Section 7.1 and [65]). Naturally then, in DES, designing specifications to prevent undesired behavior is much simpler than designing specifications to ensure desired behavior. It is a frequent experience of practitioners (and experienced both by the subjects in the study and by the interviewed experts) that initially the marking of states is incorrect or altogether forgotten.

The mathematical construct for formal modelling of control specifications—the finite-state automaton—is not always easily used in practice. This fact came to light more prominently during the hospital problem. A number of subjects summarized the control requirements from the verbal description using inequalities, giving the limits on the number of occurrences of certain events. This way of expression is much simpler than the use of FSAs and, at the same time, can be just as rigorous. In the factory problem, subject 4 actually attempted to define the requirements formally using inequalities rather than FSA models. Furthermore, DES researchers sometimes also use this method, e.g., in [8]. Our personal experiences show as well that in many cases inequalities are a very easy tool to define formally the control requirements for a system. Research in other areas, such as formal verification [30], regularly makes use of temporal logic to formally define specifications, especially ones relating to the

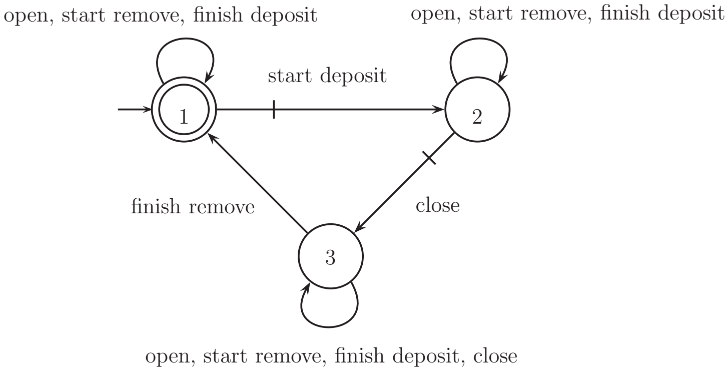

open, start remove, finish deposit
open, start remove, finish deposit
start deposit
1
2
finish remove
close
3
open, start remove, finish deposit, close

Figure 7.1: Specification for the process flow in a factory cell with a robotic arm and a press. Notice the large number of self-looped transitions with all the events irrelevant for the process flow.

liveness of a system (or the making sure that a system accomplishes something). Use of temporal logic is not common in DES even if the idea has been entertained, [61, 32], since it is not as convenient as FSAs for the definition of *safety* specifications (remember that in DES supervisors exercise control by preventing the execution of events) and because checking certain properties is more difficult in temporal logic. However, it seems that temporal logic may provide a complementary power when used together with FSAs to define control specifications.

Our recommendation is to examine alternative formal methods of defining control specifications. In many cases the use of finite-state automata is convenient and justified. However, this is not always the most natural method of expressing specifications. When the number of occurrences of events has to be maintained with a certain range, inequalities may be more suitable. When the desired behavior of a system has to be defined, temporal logic may be more suitable. Teachers of DES should explain the rationale behind using FSAs to model control requirements and should give examples of how other formal methods can be used instead. Software for DES should also be designed so that alternative formal methods can be used by the users. For this purpose, it may be necessary to develop new algorithms for checking controllability and for supervisor generation. However, the basic theory for DES control does not preclude the use of alternative models. For example, in [1] Extended Finite Automata can be used in the software interface. In this case, however, there is no need for extensions to compute supervisors as the models are automatically converted to standard FSAs before applying computational algorithms.

The use of FSAs for the definition of control specifications, as proposed in [45], suffers one other drawback. Within the classical framework, specifications need to be explicit. I.e., all events allowed to occur in a given situation have to be mentioned explicitly, otherwise they are assumed to be prevented from occurring. As a result, FSA models of specifications usually contain a large number of what are referred to as “self-loops” (see Fig. 7.1). These self-loops list at every state the events which are irrelevant to the specification (i.e., events about which the specification does not care and which it enables by default). The observations from this study show that, when modelling control specifications, subjects rarely included these self-loops in the models. For example, subject 1 modelled only the "essential" part of the control specifications (i.e., using only the relevant events) and then simply wrote

+ selfloops.

next to the model to indicate that irrelevant events have to be included as self-loops. Furthermore, subjects occasionally forgot to include these self-loops even in the rigorous DES models in the software—leading to incorrect computations. This behavior is not unique to the subjects in our study. Teachers of DES theory report similar observations about their students [19].

Our recommendation, based on a suggestion of Fabian [19], is to consider the use of implicit specifications. In these specifications, events which are not mentioned in the FSA model are assumed to be permitted. This new kind of specifications can be incorporated very easily in the existing DES framework. The only difference is in the operation for the combination of specifications. Instead of using intersection, it is necessary to use synchronous product. If the use of implicit specifications is assumed in the teaching of DES control theory, the students will find the application of the theory much simpler. Furthermore, many potential errors due to missing or incorrect self-loops can be avoided. Software for DES should be designed to either work with implicit specifications directly (i.e., to use synchronous product for the composition of specifications), or to automatically convert implicit specifications into explicit ones (i.e., to introduce all necessary self-loops automatically).

7.6 Supervisors

The subjects in this study usually chose to model supervisors manually. Thus, the exact form of the models is not dictated by the algorithm used for automatic generation of supervisors. It was interesting to observe the approaches the subjects took when designing the supervisors. Similar to the control specifications, two main alternatives where identified: implicit and explicit supervisors. Implicit supervisors are supervisors where all events which are not prevented from occurring at a certain state have to be assigned to transitions at the state (even if only as self-looping transitions). In other words, the events disabled at a state are specified implicitly—by omission. The drawback of this approach is that numerous transitions may need to be modelled at each state and the clarity of the model deteriorates, while its complexity increases. The benefit of the approach is that all information relevant for the control is included with the model and it can be used in DES operations directly. Explicit supervisors are supervisors where the disabled events at every state have to be listed explicitly, in a separate data structure. The drawback of this approach is that a separate data structure has to be maintained and the models are not directly compatible with the DES algorithms. The advantage is in the clarity of the models and in the greater ease in creating them.

It seems that implicit supervisors are more suited for algorithmic manipulation while explicit supervisors are more suitable for human understanding. Thus, neither type can be said to have a definite advantage over the other one. While observing the work of subject 5 during the hospital problem, it became apparent that they were using a "mixed" approach to supervisor construction. With this approach, events are divided into two types: relevant and irrelevant for control. Relevant events are the events which are disabled at at least one state of the supervisor. Irrelevant events are the rest, i.e., the events which are always enabled by the supervisor. In the model of the supervisor, it is necessary to explicitly enable only the relevant events. I.e., if at a state there is no transition with a relevant event, it is assumed that the event is disabled at the state. On the other hand, all irrelevant events are assumed to be always enabled. If at a state there is no transition with an irrelevant event, it is assumed that the occurrence of the event does not change the state (i.e., there is an imaginary self-loop with the events). This approach seems to lead to very clear (and "intuitive") models. However, it also requires the maintenance of extra information about the partitioning of events into relevant and irrelevant.

From our study it becomes apparent that different styles of supervisor modelling are used by subjects. Our recommendation is that flexible DES software support both implicit and explicit supervisor models. Furthermore, since the relation between the two is trivial and well-defined mathematically, we recommend that software offer users the option to view a single supervisor model using either of the two alternatives. Further research is necessary to reveal if there are significant benefits to using the "mixed" style of supervisors in modelling. However, we believe that a simplified version of this style could be beneficial. Instead of requiring a separate partitioning of events into relevant and irrelevant, it is easier to assume that the existing partitioning into controllable and uncontrollable is sufficient. Uncontrollable events are always enabled by the supervisors, thus all uncontrollable events are also irrelevant by definition. This simplification may still lead to more clear models (all self-loops with uncontrollable events may be omitted), while at the same time removes the necessity for additional data structures. Furthermore, the implementation in software should not be very difficult. Including this option for the way a supervisor is represented may help subjects understand supervisor models faster and better: clutter is reduced, control decisions stand out, and there is no need to reference other information.

7.7 Computations

In our observational study there were very few occasions where subjects applied the DES algorithms from the IDES software package. In most of the sessions the subjects either did not advance far enough in their problem solving to need to apply the algorithms or they decided to model the supervisors manually and thus did not need to apply the algorithms. From the few cases when algorithms were applied, the following conclusions were made.

First, the user interface for algorithm invocation did not seem to pose a problem to the subjects. The only registered complaint was that theoretically some DES operations support unlimited number of inputs, however, the interface limited the number of inputs to only two at a time. This limitation was only perceived. In fact, it is possible to use more than two inputs at a time, however, the interface does not make it obvious how to accomplish this. Our recommendation for DES software is not only to allow users to apply relevant operations (such as "synchronous product") on multiple inputs at a time but also to make the procedure obvious in the user interface. One potential solution could be to let the user select the inputs for an operation from a list of all available models. During the interview with expert 2 (see Section 5.2), they proposed an alternative to the graphical user interface available in IDES. Instead of invoking DES algorithms graphically, they proposed the introduction of a text-based command line, similar to what is available in the Matlab® environment. Users would then be able to type in which operations they would like to invoke. If the command line supports nesting of operational expressions and the use of user-written command scripts, DES problem solving may be simplified (and accelerated) significantly. For example, in the factory problem, after modelling the two machines, the testing unit and the specifications for the two buffers, one could create the supervisor by issuing a single command,

```
supcon(sync(M1,M2,TU),intersect(B1,B2)),
```

instead of selecting each operation separately using the graphical interface. Furthermore, if this command is saved as a user script named, e.g., “factory”, after a modification to one of the models, it would be sufficient to type only

```
factory
```

in the command line to repeat all operations necessary to regenerate the supervisor. It could be argued that the context of use of Matlab® is very different from the context of DES. For example, in DES it is not sufficient to produce only numerical results; graphical models (FSAs) are used extensively. However, similar user interaction is also available in highly-visual CAD environments such Autodesk AutoCAD®. The result of a text-based command can be immediately reflected in the visual representation of a model. E.g., the supervisor generated by the invocation of operations from the command line can be displayed visually in the same fashion as when the user invokes the operations through the graphical interface. The key advantage of user scripts is, in our opinion, in the support of analogical problem solving [60, 29]. If the user determines a successful strategy for the solution of a problem and saves the sequence of operations as a script, they (or their colleagues) will be able to use the same script to solve analogous problems. In the case of minor differences between the problems, it might be simpler to modify the existing script rather to reinvent the global strategy for solving the problem. Similar are the considerations motivating the introduction of case-based reasoning in other fields [34]. Lastly, the pre-packaging of successful strategies, e.g., in the forms of “wizards” where the user only needs to select basic input parameters, may simplify the application of DES by non-experts.

The second observation in our study is that, in the case of an incorrect (or unexpected) result of an operation, the subjects engaged in what we call a “recursive” search for the reason of the inconsistency. Usually they would first invoke the algorithm again, presumably to discard the possibility that they had performed the wrong action by mistake. Then, they would check the inputs to the operation, and then the inputs to the operations producing these inputs, etc. For example, in the factory problem, if the output of the supervisor construction algorithm is unexpected, the subject might first invoke the same algorithm again. If the result is the same, they would examine the monolithic models of the system and the control specifications (which serve as inputs to the algorithm). Then, in case that the monolithic model of the system seems to have an error, they would invoke the DES operation for the composition of the system modules. If the error persists, the subject would proceed to examine each of the models for the machines and the testing unit. If they find an error in one of the models, they would correct it and back-track by re-composing the modules and then re-computing the supervisor. It is clear that such a process can benefit from two things:

- Information on what operations were performed. If the subject can verify which operations they had performed and which inputs they had used, they would not need to repeat operations to discard the possibility of incorrect operation invocations.

- Ability to repeat the sequence of operations automatically after a change to one of the inputs. If the subject identifies an error in one of the system modules, for example, and they correct it, they then need to manually recompute all aspects of their solution which rely on the given module. If the software remembers what sequence of operations are applied to obtain the solution, it would be possible to automatically repeat them after the correction of the given module.

Our recommendation is that DES software include a journal (or history) of the sequence of operations and their inputs, similar to what is available in Adobe Photoshop®. This should help not only with the recall and verification of which operations have been invoked, but also allow the "replay" of the operations in the case of a modification in a model. A further improvement could be the implementation of automatic maintenance of consistency for computed models. For example, a supervisor may be generated from a system model and a specifications model. The specifications model may be computed by composing a number of smaller specifications. Then, a change to one of the smaller specifications would automatically trigger the update to the composed specification to reflect the change. This, in turn, would trigger the re-generation of the supervisor to make it consistent with the new models. Such updates would not only speed-up problem solving when modifications to the basic models are needed. It would also provide support for prototyping of solutions, as users would be able to see the changes to their solutions introduced by any modification they consider.

Lastly, during the n-gram analysis of the activities of the subjects (see Section 3.6) it became apparent that subjects most frequently apply computational algorithms while working at the high level of modules. In Fig. 6.3 it can be seen that the bigrams 'CM', 'CC' and 'MC' have the highest absolute and relative ratios from all bigrams which involve computations (i.e., which contain the code 'C'). This implies that, usually, subjects do not need to consider low-level details of models, such as specific transitions, when applying algorithms. Thus, the interface of DES software need not offer detailed displays of models when the user wishes to apply algorithmic computations. Instead, it should focus on displaying the relations between modules.

7.8 Verification

In this study, on very few occasions did subjects engage in elaborate verification of their models and solutions. This is mostly due to the fact that they could not advance far enough in the process of problem solving within the limited time, and thus their solution was not developed enough to merit extensive verification. Using the performance of subject 1 during the factory problem, the comments of subjects 3 and 5, and the comments of the experts, it is possible nevertheless to conclude that the most likely technique subjects would use for verification is event tracing (see Section 6.5 for a discussion of tracing). Some subjects mentioned explicitly that visual examination and tracing is very important for them. However, in large and complex solutions it might be even infeasible to display visually the computed models (e.g., models with tens of thousands of states and hundreds of thousands of transitions).

Our recommendation is to provide a facility in DES software to pose questions relevant to tracing, e.g.,

1. Is this sequence of events possible starting at the initial state? Which state does it lead to?
2. Is this sequence of events possible starting at a given state? Which state does it lead to?
3. Is this sequence of events possible starting at any state? In how many cases is the sequence possible? Which are the starting states and which states does it lead to?

Such a facility would make it possible to perform verification even when models are too large for visual inspection. Answers to the last question would most probably allow verification which is infeasible in most non-trivial cases, if it has to be done manually. Questions of the second type will require an additional facility; the user must be able to select a state of interest. Thus, even in large models, the interface must provide a way to select states, e.g., by showing a list of all states in a model. In conjunction with state selection, DES software should also display detailed information about states if requested. However common, tracing is not the only verification technique used by the subjects and the experts. Other techniques involve, for example, checking which events are possible at a given state or what the control decisions at a given state are. This information could easily be displayed when the user selects a state from the list of states in a model. In the case where the model is displayed graphically, control decisions can be visualized as well.

Visual inspection of models was mentioned a number of times as important in verification tasks. Though not always possible due to model size, sometimes it is feasible to generate graphical representations of computed models. Unfortunately, computed models do not come with a pre-determined graphical layout. For finite-state machines, it is necessary to apply a graph layout algorithm to the structural description of the model to obtain a description suitable for visualization. A number of layout algorithms exist but the consensus is that there is no algorithm which produces "nice" layouts for all possible inputs (graph structures). In our study, it was observed that subjects usually adjusted manually the automatic layout of computed models. It seems that this will not be avoidable in the general case, however, there are two features which, if implemented in DES software, we believe will contribute to easier visual inspections: the availability of a deterministic layout algorithm, and the availability of a number of different layout algorithms. A deterministic layout algorithm will be helpful in cases when certain computations are repeated—the user will not have to hunt for the initial state each time, for example. However, the true significance of deterministic layouts is that they allow the direct comparison between model dynamics. I.e., if two models appear the same, this means that they have the same structure. Users would also benefit from the availability of different layout algorithms since different layouts may emphasize different visual aspects of a model (e.g., symmetry, modularity, etc.) Furthermore, if a given layout algorithm performs particularly poorly for a given input model, the user would be able to select a different layout algorithm for the model.

If solving of a DES problem follows the traditional steps, verification should focus most on the basic models from which a solution is derived (as supervisors are generated automatically using an algorithm). However, as already discussed in Section 7.1, subjects in this study frequently chose to design supervisors manually. Thus, the main focus in the process of verification becomes, naturally, the models of the supervisors. Of course, the control policies of manual models can (and should) be verified using all the strategies employed with generated models—but there are additional checks that need to be performed since the correctness and optimality of the solutions is not guaranteed when they are designed by hand. Unfortunately, only subject 2 advanced far enough in their problem solving to start the verification of their manually designed supervisors. However, from the short interviews at the end of each session it was possible to collect information about the potential strategies of other subjects as well. The most basic test is for the controllability of the supervisor model, in other words, if the control policy enforced by the supervisor is feasible. If this test fails, the control policy includes an attempt to block the occurrence of an uncontrollable event and thus has to be corrected. The other tests are more advanced and are necessary when more than one supervisor will concurrently control a system (this was the approach taken by subjects who designed supervisors manually). If the subject decides to compose all supervisors into a monolithic solution, then they can test the deadlock/livelock properties of the resulting model (simply by checking if the model is *trim*). If the subject decides to keep the supervisors separately, they can verify the same properties by checking if the supervisors are *non-conflicting*.

Usually, DES software is designed with the idea that supervisors will be automatically generated. Thus, an automated test suite for "unknown" supervisors is not available even if all individual operations are. (As a side note, in our study we used a version of IDES where the check for *conflict* was not available.) In view of the fact that solutions to DES problems may not always be derived algorithmically, our recommendation is to include a comprehensive package of tests for supervisors introduced manually. Such tests should allow for verification of single supervisors as well as modular supervisors, both when all participating supervisors are designed manually and when some of them have been generated automatically. Such a package becomes especially important if the observations of expert 2 are correct. According to them, the main interest in industry is in the verification of controllers rather than in the construction of controllers.

A common comment of subjects when discussing their verification strategies was that they would like to consult external literature for information on specific algorithms and strategies. This indicates that their proficiency with the theoretical background was not fluent. In such cases, it would be helpful if a short reference guide is included with the DES software. It is clear that experts would rarely need to refer to such a guide. However, the learnability of the software and the DES methodologies will be improved in this way, simplifying the transition of users from novices to experts.

7.9 Low-level modelling

Even though our study focused mostly on the high-level aspects of DES problem solving, we also briefly looked at the low-level actions of users when modelling. We performed n-gram analysis of the collected data, as well as analysis of the visual attention and the mistakes committed by subjects.

The n-gram analysis revealed that there is a dominant trend in modelling to perform separate kinds of activity in chunks rather than to alternate between activities. For example, once a subject starts work on the states of a model, they tend to continue working on the states before switching to transitions or events. However, the degree to which this trend is manifested varies between subjects. Some subjects prefer to work in larger chunks, while others switch between working on states and transitions more frequently. Such differences are most pronounced when modelling with pen and paper is considered. When subjects use the software to input their models, the differences all but disappear. The version of the IDES package used in the study mandates a more rigid modelling environment. For example, events have to be defined in order to be used in a model. Such rigidity, we believe, contributed to the greater similarity in the modelling activities of subjects. However, one must also keep in mind that, in the study, subjects did not use the software to design their models. The primary function of the software was to perform computations. Thus, subjects used the modelling environment only to input models which already exist. As a result, one cannot make a direct comparison between activities during the modelling with pen and paper and the modelling with the software.

Our recommendation is that DES software support a wide variety of modelling styles. In many software packages there are separate tools for the creation of states and transitions. Such environments make it difficult to alternate between the two main elements of FSAs. We believe that the interface in the IDES package is more flexible since a single "creation" tool is used to create all types of entities [51]. The user can use the tool more naturally, similar to pen and paper, and apply any style of input with the same ease.

The analysis of attention during the problem solving allowed us to make the following conclusions:

- The problem description is referenced very frequently. It attracted the most attention, overall.
- The amount of attention shift between models appears to form groups of highly related modules, while attention does not shift as much between these groups.
- Previous versions of a model are referenced more frequently when the user experiences more difficulty with the design of the model.

Based on these observations, we recommend that the following features be implemented in DES software. First, in case the problem description is available electronically, it should be easily accessible from the software. For the case when this is feasible, it should be viewable concurrently with the workspace of the user—so as to allow direct comparisons between requirements and models. Second, referencing other models while working on a given model should be simple. Many graphical DES software packages, including IDES, allow for the simultaneous work on a number of models. However, only one model can be "active" at a time, i.e., be displayed in the software environment and allow the user to modify it. Quick referencing does not require the full activation of another model since the referenced model will not be modified. Thus, we believe it may be sufficient to include a feature in the interface where a model may be selected for a brief read-only preview. For example, moving the mouse over an icon next to the name of a model may cause the model to appear in the main workspace and, subsequently to disappear when the mouse is moved away from the icon. In such an interaction, the user will be minimally distracted from their workflow. The third feature we recommend for DES software is the ability to store multiple versions of models. Such a feature may be simulated to some extent via the use of the undo/redo facility found in most software, however, this process is not only inconvenient, but also infeasible in some situations. Multiple model versions would allow users to experiment with their designs as they will be able to take "snapshots" and then easily return to them if they are not satisfied with the additional modifications they make. Furthermore, even if users are not satisfied with old versions, as seen in our study, referencing older versions is a common activity when experiencing difficulty with modelling.

The mistakes committed by subjects during problem solving were only infrequently the result of poorly executed actions. Furthermore, most of the time subjects used pen and paper modelling, thus there was less opportunity to observe mistakes in the use of the software. In general, most of the observed mistakes were made due to poor understanding of what needs to be done (e.g., poor design choice, or incorrect understanding of the problem description). Thus, not much can be done in DES software to prevent these mistakes, except to gear the features of the interface towards making understanding of the problem and the models simpler. The only opportunity we see is in the prevention of a specific kind of mistake, frequently committed by subjects. We observed in many cases that subjects forgot to denote the initial and marked sates of models when inputting them into software. It is possible to introduce a very simple automatic test which would warn the users if a model does not have initial and marked states. However, the application of this test should be very judgmental as users may be experimenting and/or intentionally creating incomplete design. Such a test should be applied only when absolutely needed (e.g., before calling a DES operation) or as a non-intrusive reminder, in the background.

7.10 Software for DES

The subjects in our study occasionally used the IDES software package during problem solving. They made explicit comments regarding their experience with the software and they expressed some wishes about adding extra functionality. These requests are summarized in Section 6.5. The two most important features missing from (or hard to find in) the version of IDES used in the study are, according to our opinion, the duplication of models and the printing facility.

The ability to duplicate models is very important because it supports analogical problem solving. If a successful model has been constructed for a component of the solution, then a similar component could be modelled by using the existing model as a base and making only the necessary changes. For example, in the factory problem the only difference between the models of the two machines lies in the events used; the structure of the models is the same. Similarly, the model for the testing unit can be derived from the model of a machine with minor modifications.

The lack of printing facility was the primary concern of the subject who complained about it. However, we feel this is indicative of a larger problem shared by most DES software: the lack of facility for interaction between the software and the outside environment. Discrete-event system control solutions have very little value if they must remain within the confinement of the software which computes them. The least DES software should allow is the use of the solutions in other software tools (e.g., simulators, code generators, or programming environments). Users of DES software may want to print their models, include their models in manuscripts, reports, or presentations, simulate the behavior of their models, or apply the control solutions for the control of a real system. None of this can be achieved if the software package they use only supports displaying models on the screen and storing them in a proprietary format. Conversely, users may also wish to use models coming from external sources (e.g., a different modelling tool). If the software package does not allow the import of external models, the usefulness for the user would be reduced. Thus, our recommendations are as follows. Software for DES should be designed with interoperability in mind. Especially in a research environment, normally there would not be any trade secrets which necessitate a lock-in of the users. Different software distributions inevitably have different strengths and the ability to reuse their work across different packages would simplify the work of users. Teachers of applied DES should not only ask students to compute supervisors with the software used in the course. Assignments should require also the reuse of the solutions in realistic contexts, e.g., in the preparation of a report about the performance of the solutions or in the control of a real hardware system. In this way, students will gain understanding about methods for using computed solutions outside the environment of the DES software.

Chapter 8

Conclusions

The observational study which we described was an initial effort in understanding the cognitive aspects and human factors in solving DES control problems. The study was designed as a primarily qualitative investigation, however, numerical data were used to support some conclusions. The study is the first part of larger research with the ultimate goal of improving the methodologies and software used in the area of DES control. As such, there were a number of specific questions which motivated the development of this study (see Chapter 1).

Five subjects were asked to solve two DES problems each. The problems were crafted to be of similar structure and complexity. The subjects were asked to voice their thoughts as they proceed in problem solving. Their performance was recorded with a video camera. Additionally, two experts were interviewed about their problem-solving strategies.

The amount of data collected was big and thus required the judicious use of different techniques of analysis. For this purpose, we developed a typology which can be used to encode activities in DES problem solving. Furthermore, we defined novel methods for the measurement of progress and of error rate in DES problem solving. We used both low-level data (such as individual actions of subjects) and high-level data (such as the overall problem-solving strategies of subjects). We compared different aspects of the data, such as the duration and count of selected activities or processes. We also used N-gram analysis to discover patterns of activity.

The results of the analysis were very disparate, motivating a number of recommendations for the design of DES software and for the teaching of DES theory. Not all questions which motivated this study could be answered with the desired depth and certainty, however, the study served its main purpose: to give insights into the cognitive aspects of DES problem solving and provide the basis for further investigation on this topic. In light of the discussion in the introduction of this work, our observations can be summarized as follows.

There is a difference between problem solving using pen and paper and using software. Pen and paper gives more flexibility in expression, while software is more rigid. None of the subjects in our study preferred to use the software when designing their models. They appeared reluctant to use the software at all and used it only for computational tasks which were too difficult to do manually given the size of the problem. Thus, the modelling environment in the software was used predominantly simply as a tool to input the models designed on paper.

The strategies used in solving DES problems were varied. There were many alternative approaches and different subjects made different choices. It was observed that only one subject followed the approach advocated by the experts. Most subjects chose to design supervisors manually instead of using the algorithm for automatic generation of supervisors. In this way, they forfeited the guarantee of correctness and optimality of the solution. On the other hand, they did not need to create formal models of the control requirements, and the verification of the solution was cognitively less demanding.

The exact information produced and consumed by subjects was not reliably qualified. However, patterns of shifts in attention were discovered, and the type of mistakes committed were described. It was determined that the most frequently referenced item is the problem description. Furthermore, both other models and other versions of the same model are referenced during problem solving. The mistakes committed by subjects were mostly the result of incorrect understanding or reasoning, rather than due to incorrect execution of actions. It was discovered that even a simple error may render the solution completely incorrect.

These observations, and many others, served to recommend many changes to the way DES software is designed traditionally. Similarly, we believe that the way DES theory is taught may be improved following some suggestions. Most importantly, it is necessary revisit

- the role of control specifications and supervisors,
- the selection of events and the significance of controllability,
- the representation of DES elements and relations in the interface of DES software,
- the flexibility necessary for different styles of problem solving, and
- the techniques for verification of solutions.

Advanced research in DES software design may include topics such as trying to implement software which has the desirable properties of pen and paper, and the development of adaptive interfaces. Different users have different preferences and different styles of interaction with a software system. If the software system is able to identify these styles and preferences automatically, e.g., through the analysis of the actions of users, it would be able to adapt automatically and improve the experience of the user. For this purpose, further in-depth investigation of the low-level actions of subjects is necessary.

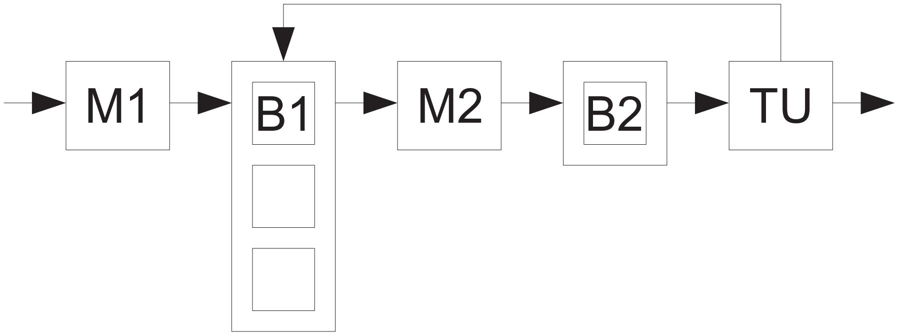
Future investigation of human problem-solving in DES may use this study as a base and target specific aspects of the process. We would nevertheless recommend that a mini-pilot study always be done before the real observational study is administered. This would help in discovering minor glitches which may end up having a more significant impact on the problem solving of the subjects. In our study, we discovered that the wording of the problems we gave to the subjects was confusing in some cases. The angle of the video camera was not optimal and some of the activities of the subjects were not captured. We also recommend that experimenters weigh carefully the amount of structure they build into the study design. The more structured the study is, the more reliable data can be collected. However, this comes at the price of less freedom in the activities of the subjects and, ultimately, to a reduction in the validity of the data.

Appendix A

Problem definitions

A.1 Problem 1: Factory problem

Provide a discrete-event control solution to the problem of "Two machines and a test unit".



The problem is described as follows. There is a factory that needs to be controlled. A diagram of the factory is displayed above. There are two machines, M1 and M2, which process parts. M1 takes a part from an input bin and delivers it to a 3-slot buffer, B1. M2 picks up a part from B1 and when it is done processing, delivers it to a 1-slot buffer, B2. At the end of the processing line there is a testing unit, TU, which takes a part from B2 and tests if it meets the quality standards. If the test is positive, the part is delivered to an output bin. Otherwise, the part is delivered to B1 for reprocessing by M2. There is a microcontroller in each of M1, M2 and TU which can control when a part is picked up for processing by the machine/testing unit. However, once the part is taken, the rest of the process is automatic and there is no control over when and if the part is delivered to its destination (buffer or output).

Model the system described above as a discrete-event system (DES) and use standard supervisory control theory (Ramdage&Wonham) to provide a control solution which guarantees that the buffers do not overflow or underflow.

You should use pen and paper for your modeling. You may use the IDES software for computationally demanding tasks.

A.2 Problem 2: Hospital problem

Provide a discrete-event control solution to the problem of "Sick patient".

The problem is described as follows. There is a child patient at a hospital. He suffers from a disease related to diabetes. Unfortunately, the child likes candies very much. There is a nurse and a doctor who take care of the young patient. The nurse is responsible for the administration of insulin into the patients bloodstream to keep the level of glucose down. The standardized dose is supposed to complement the intake of 8 g of sugar (approximately as much as that found in a candy). The equipment used to inject the insulin is a research prototype which also automatically takes readings from the blood and sends them wirelessly to a computer. The computer processes the data and then sends it to the hospital information system. While the data is processed, the computer does not accept more data from the insulin injection equipment. The equipment, due to a flaw in the initial design, remains non-responding while it waits for a connection with the computer, thus allowing the administration of more insulin only after the last reading has been processed. The doctor can use the hospital information system to access and review the latest data collected with the research equipment. If the results in the report are not good, the doctor will prescribe medication for the child. However, if the results are good enough, the doctor will give the child a candy to make him happier because it is safe to do so. The child doesn't miss his candies that much, since, when the hospital personnel allows it, his grandmother comes for a visit and she always manages to sneak in a candy for him which he sometimes eats right away, or saves for later, depending on his mood. Fortunately, the hospital personnel are aware of the cheating grandmother and they keep this fact in mind during the treatment.

Your job is to model the described situation as a discrete-event system (DES). Then, use standard supervisory control theory (Ramadge&Wonham) to provide a control solution which ensures safety for the patient. More specifically, the level of glucose in the bloodstream should be controlled. It is dangerous if there is intake of more than 25 g of sugar without any insulin to counter the effect. As well, insulin must not be administered unless there is a buildup of glucose in the blood (e.g., after consuming sugar). Furthermore, a problem has been identified in interface between the research computer and the hospital information system. If the blood reading report is not retrieved for review before another report is generated by the computer, the data in the system becomes garbled. It is important to guarantee that this does not occur. As well, the doctor should not request a report from the system prematurely (before a new report has been generated) since this causes her terminal to get stuck and she cannot use it for other purposes.

You should use pen and paper for your modeling. You may use the IDES software for computationally demanding tasks.

Appendix B

Encoding typology

Encoding of visual attention The events in this "stream" were encoded with the prefix '**L**'. The following suffixes were used to describe where attention is directed:

O The sheet of paper with the description of the problem,

P A specific sheet of paper used for problem solving. All sheets of paper handed to the subjects were marked with unique graphical symbols to allow identification from the video footage. The symbol for the sheet which attracts the subject's visual attention is given as a parameter in the code. As the sheets of papers which the subjects used were retained, it is possible to deduce which parts of their models they were attending to.

Q One of the sheets of paper used for problem solving. This code is used when it is not possible to determine which specific sheet the subject is attending to.

Y The computer display. The software has two views: one showing the graph of a DES module and the other showing a table of the events used in a DES module. The view which is active when the subject looks at the display is given as the first parameter of the code: '**G**' and '**E**' for the graph end events, respectively. The second parameter specifies which DES module is active in the software (i.e., whose graph or events are being displayed).

K The keyboard,

A Away (none of the other listed targets). This code is used for events such as the subject looking away from all tools, e.g., looking at the experimenter or staring at the wall.

This "stream" was chosen (arbitrarily) to also include information about interruptions in the process of problem solving. For example, when a bug in the software package manifested itself, it was necessary to temporarily pause the observational study in order to resolve the issue. The code used for such events consists solely of the prefix '**I**'. There is no code for "resuming" the observational session. Instead, the first encoded event following an interruption event is assumed to signify the end of the interruption.

Examples of event encoding are listed below.

shifting gaze to the sheet marked with a square	LP(SQR)
looking at the screen where the graph the 'Machine 2' module is shown	LY(G,M2)
reaching for one's cup of coffee and shifting gaze towards it	LA
shifting gaze to the keyboard to type up a label	LK
the software stops responding and intervention is required	I
spreading three sheets of paper on the table and observing them	LQ

Encoding of physical activity The events in this "stream" were encoded with the following two prefixes: '**P**' and '**C**'. These two prefixes stand for, respectively, actions involving the use of pen and paper and actions involving the use of the software. Suffixes were used to describe each action in more detail. Each suffix starts with the type of entity providing the context of action, namely,

O Description of the problem,

G DES control theory,

N Solution or solution approach,

F Figure/drawing,

R Relationship between modules/parts of the system,

D Dynamics of the system,

M DES module,

S State in an FSA,

T Transition in an FSA,

E DES event,

C DES computational algorithm,

I Functionality of the software,

U Software interface and

B FSA layout.

A letter was appended to encode the specific action as follows:

C Create,
N Name (create a label),
P Create copy (e.g., using the "copy" command in the software),

R Remove. If using pen and paper, this means either erasing or crossing-out. If using software, this means deleting the entity.

I Modify the "initial" property of an FSA state,

M Modify the "marked" property of an FSA state,

O Change controllability of a DES event,

B Modify layout of an FSA (when using software),

E Write down an explanatory note,

S Write down a (more formal) specification,

T Mark an entity (circle, underline, etc.) This does not refer to making a state in an FSA "marked".

D Draw a figure.

Additionally, codes can be appended with '**F**' to signify that the action was performed to correct (fix) a similar previous action. For example, "CSN" is used when the subject names an FSA state in the software for the first time (even if they use "backspace" while typing the name). If they decide to change the label at a later, separate time, "CSNF" is used.

Parameters are used to specify within the context of which DES module the subject performs each activity. For example, "CSN(M1)" is used when the subject names a state of the FSA model of 'Machine 1' in the computer. As each subject created and worked with a unique set of modules, with each data file we included a separate legend of all the module IDs used to encode the data in the file. However, some standard parameters, common to all subjects were also used. They are listed next.

PREV Previous knowledge or experience, including knowledge and experience acquired during the problem solving,

***** All modules,

- Nothing,

O Problem description,

L The plant (controlled system). This symbol may have a more specific meaning in the encoding for some subjects.

K The legal language (control specifications). This symbol may have a more specific meaning in the encoding for some subjects.

H The IDES help site.

Additional standardized parameters are used when a subject executes a DES operation (codes starting with "CC"). The first parameter signifies the algorithm used. The algorithms are encoded as follows:

TR Trim,

MT Intersection (meet),

SP Synchronous product,

CC Check controllability of a specification (second parameter) with respect to a system (third parameter),

SU Compute supremal controllable sublanguage of a specification (second parameter) with respect to a system (third parameter).

During the encoding, the following conventions were observed. When subjects created a new DES event implicitly by using it to label a transition, the action was encoded as "PTN" (in the software, implicit event creation is not possible). The code "PEC" was used only when a subject created a DES event by explicitly listing it separately from the FSA model. In that case, the code "PEN" was not used as the event creation actually encompasses its naming. When using the software, event creation and event naming requires two separate actions, thus the codes "CEC" and "CEN" were both used. The control decision to enable or disable a transition in a supervisor (when modelling supervisors manually) was encoded with "PDS". In the software, the enablement or disablement of a transition is done implicitly and thus there is no activity to be encoded. The addition of a new event to a transition constitutes, strictly speaking, a modification to the labelling of the transition. However, the '**F**' suffix is used only if there is reasonable grounds to assume that the subject had to do so in order to correct their model (as opposed to being in the stage of creating the model and labeling transitions incrementally). The movement of the label of a transition when using the software is encoded as "CTB".

Examples of event encoding are listed below.

underline part of text in problem description	POT
write down specification of supervisor	PDS(S)
draw a diagram of different parts of the system and how they connect	PRD
write down the title for the 'Testing unit' module	PMN(TU)
erase the marking of a state in the 'Machine 1' module	PSMF(M1)
cross out the model for the 'Doctor' module	PMR(DR)
rename an event in the 'Computer' module	PENF(CP)
using the software, create a new state in the 'Buffer 1' module	CSC(B1)
adjust the width of the left pane in the software	CUB
make an event controllable in the computer model for the 'Report' module	CEO(RP)
apply the "check controllability" operation to the 'Report' module and the system	CC(CC,RP,L)
invoke the "copy" command in the software to copy the 'Machine 1' module	CMP(M1)

Encoding of verbalization In this "stream" the verbalizations of the experimenter and the subjects were encoded. The encoding includes an interpretation of the cognitive activities of the subjects, based on their outward behavior (e.g., verbalizations, physical activities, etc.)

The verbalizations of the experimenter were encoded with the prefix '**E**'. The suffixes used were

J the reminder "Keep talking" and

H the answer to a question or concern, or the provision of information.

No parametrization of the codes was used.

The verbalizations of a subject were encoded with the prefix '**A**'. The suffixes and parameters were the same as the ones used to encode physical activities. For example, "AD(M1)" is used when the subject discusses the dynamics of the 'Machine 1' module. Two additional suffixes were introduced.

Z irrelevant verbalization, e.g., "okay", "wow", etc.

X unintelligible verbalization

The following conventions were used. When the subject comments on the specification of the behavior for a DES supervisor (e.g., 'S'), the code suffix used is "DS(S)". For example, this will be used if the subject says "so we have to prevent the event 'take' from happening". If, instead, the subject comments on the natural behavior of the supervisor, e.g., "we can't disable any of these events here", the suffix used is "D(S)". When the subject mentions the controllability of a language, e.g., for 'Buffer 2', the suffix used is "MO(B2)". When the subject talks about actions unrelated to the problem solving (e.g., about choosing a folder to save a file), the code used was "AZ". As well, characterizations of elements of the problem with no significance to the process (such as sarcastic remarks) were encoded as "AZ". Non-word verbalizations were not encoded.

Finally, interpretations of the subjects' cognitive activities were encoded with the prefix '**X**'. The suffixes are listed and explained next.

D Data collection (cognition-driven perception). E.g., "let me see…". The following codes may be appended:

  W Read written text (usually that is the problem definition),

  V Perceive visually,

  T Ask experimenter,

  D Determine using software,

  U Count. The type of entity counted is appended (see entities for encoding physical activities). For example, when counting states, '**S**' will be appended and the code will be "XDU-S".

P Perception (visual). E.g., "apparently…", "it seems…", "so…", etc.

- **R** Recollection. The following codes may be appended:
  - **K** Background knowledge or previous experience. E.g., “I’m thinking…” (trying to remember).
  - **A** Analogy. The parameters identify which entities are analogous. This code focuses on recognizing similarity based on memories. However, encoders may confuse it with the code “XCL” below. Thus, a more reliable typology may consolidate the two codes.
- **C** Classification. The following codes may be appended:
  - **L** Determine similarity or relation between entities. The parameters identify which entities are compared. This code focuses on recognizing similarity based on perception. However, encoders may confuse it with the code “XRA”. Thus, a more reliable typology may consolidate the two codes.
  - **R** Rate entities,
    - **P** Positively,
    - **N** Negatively,
    - **L** As large/many,
    - **F** As small/few,
    - **C** As complex/difficult,
    - **E** As simple/easy,
    - **S** According to a specific property of the entity.
- **H** Hypothesis. The following codes may be appended:
  - **B** Belief. E.g., “I think…”, “must be…”, “should be…”, “…I guess…”, etc. This code focuses on the expression of some understanding. However, encoders may confuse it with the code “XHE” below. Thus, a more reliable typology may consolidate the two codes.
  - **Q** Formation of a question or a hypothesis to test. E.g., “I wonder…”, “I’m not sure…”, “I don’t know…”, “could be…”.
  - **S** Resolution or outcome of testing.
  - **E** Expectation. E.g., “probably…”. This code focuses on the expression of prediction. However, encoders may confuse it with the code “XHB”. Thus, a more reliable typology may consolidate the two codes.
- **I** Induction. E.g., “so…”.
- **J** Justification or rationalization. E.g., “since…”, “as…”, “because…”, etc. The following codes may be appended:
  - **K** Using background knowledge or previous experience. E.g., “it’s always…”.

  - **F** Using currently determined facts.

- **G** Planning (considering goals). The following codes may be appended:

  - **O** Ordering of goals. E.g., "first…", "…then…", "…later…", "I'll start with…", etc.
  - **I** Intention. E.g., "I will…", "let's…", "I'm going to…", "I want to…", etc.
  - **Y** Plan analysis/evaluation. E.g., "I need to…", "I should…", "I'd better…", "I have to…", "once [something]…", etc.

  The codes are extended with the code for the planned activity. The activity may be either a physical activity (encoded as described earlier) or a cognitive activity (encoded with the respective "X…" code). For example, When the subject says that first they have to model the 'Doctor' module, the code will be "XGO-MC(DR)". When the subject justifies their decision to model 'Machine 1' and 'Machine 2' in the same way because they are similar, the code will be "XJF-XCL(M1,M2)".

The context of the cognitive activities, when clear, is given as parameters. For example, when counting the number of states in the 'Buffer 1' model, the code will be "XDUS(B1)". When there are two codes, "X…" and "A…", with the same suffix, e.g., "XGI-SI(B1)" and "ASI(B1)", only the "X…" code is used to avoid redundancy—the code for the cognitive activity is derived from the verbalization.

One common feature of verbal discourse, especially prominent in thinking aloud, is the fact that one semantic unit may be spread across a number of verbalizations with potentially non-trivial gaps between them. For example, a subject may say: "To see the graph freshly laid out by the software instead of…more or less destroyed by myself," where there is a five-second pause in place of the ellipses. To capture such situations, we introduced ellipses as a special code. If this code is used, it indicates that the encoded verbalization is a continuation of the thought expressed in the verbalization encoded immediately preceding the current one. For example, assuming the discourse shown above refers to the graph of the module 'M', it will be encoded with the two (consecutive) codes "XDV" and "..." in between them.

Examples of verbalization encoding are listed below.

keep talking [pronounced by the experimenter]	EJ
ahem. . . yeah, well	AZ
takes a part from the input buffer and delivers it to. . .	AD
so we have state “0”, initial state	ASI
then the administration of insulin. . . from state “8” to state “0” [module ‘CH’ is the context]	XGO-D(CH) & ...,AS(CH)
so we have, I’ll start, this may be more transitions than we need [module ’EQ’ is the context]	XI-T(EQ)
oh I see, that problem we solved in class	XRA(O,PREV)
oh, I guess I kind of model that twice then [modules ‘B1U’ and ‘B1O’ are the context]	XCL-N(B1U,B1O)
so that’s the main, sort of, constraint	XCRS(K)
and now I’m just going to look at the three different systems I sort of devised, sorry automatons, and see if they actually do relate to one another	XGI-XDV(*),XHQ-R(*)
have I considered buffer underflow or overflow?	XHQ-N(K)
apparently, the kid’s only behavior is eating candy [model ’CH’ is the context]	XP-D(CH)
OK, these, the way I’ve set it up with two supervisors is not gonna work because these’s no way to get back, you put like one thing in the buffer and. . . that’s it [modules ‘S’ and ’B’ are the context]	XCRN-N(S),XJF-D(B) & ...
I should just probably mark these two [modules ‘M’ and ‘TU’ are the context]	XGY-SM(M,TU)

Appendix C

Flowcharts of problem-solving strategies

This appendix contains the generalized flowcharts for problem solving during the Factory and Hospital problems. See Section 3.5.2 for more explanation.

C.1 Factory problem

The abbreviations used in the following flowcharts are as follows: M – machines 1 and 2, TU – testing unit, L – monolithic system, B1 – buffer 1, B2 – buffer 2, K – monolithic control specification, SB1 – supervisor for buffer 1, SB2 – supervisor for buffer 2, S – monolithic supervisor, SL – composition of testing unit with monolithic control specification.

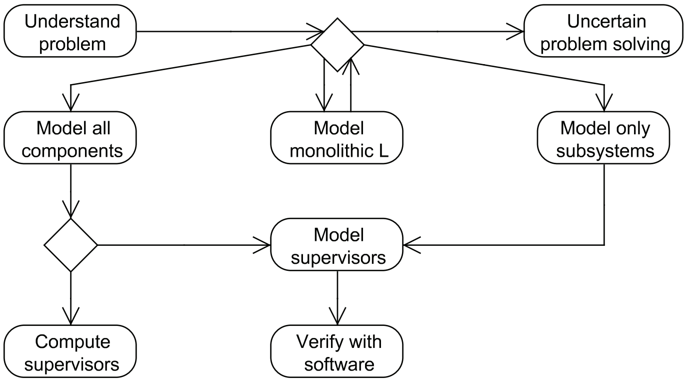

Understand problem
Uncertain problem solving
Model all components
Model monolithic L
Model only subsystems
Model supervisors
Compute supervisors
Verify with software

Figure C.1: The top-level flowchart for the Factory problem.

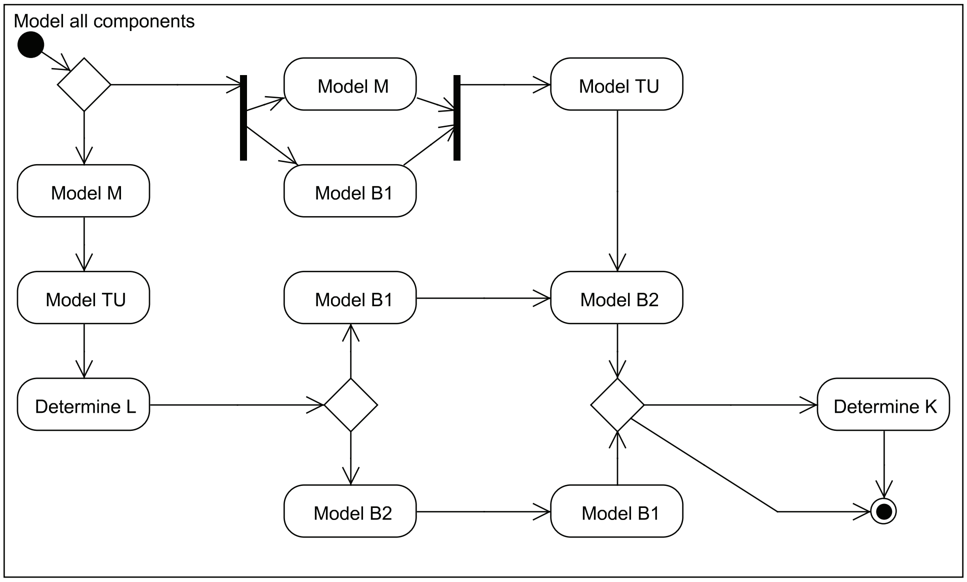

Model all components
Model M
Model TU
Model M
Model B1
Model TU
Model B1
Model B2
Determine L
Determine K
Model B2
Model B1

Figure C.2: Flowchart for the "Model all components" box.

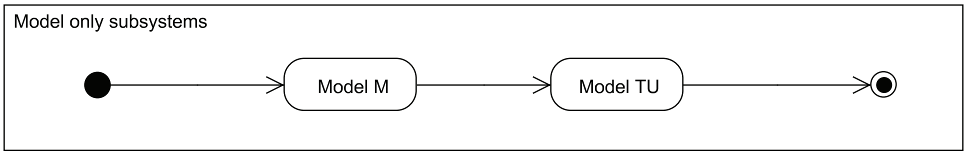

Model only subsystems
Model M
Model TU

Figure C.3: Flowchart for the "Model only subsystems" box.

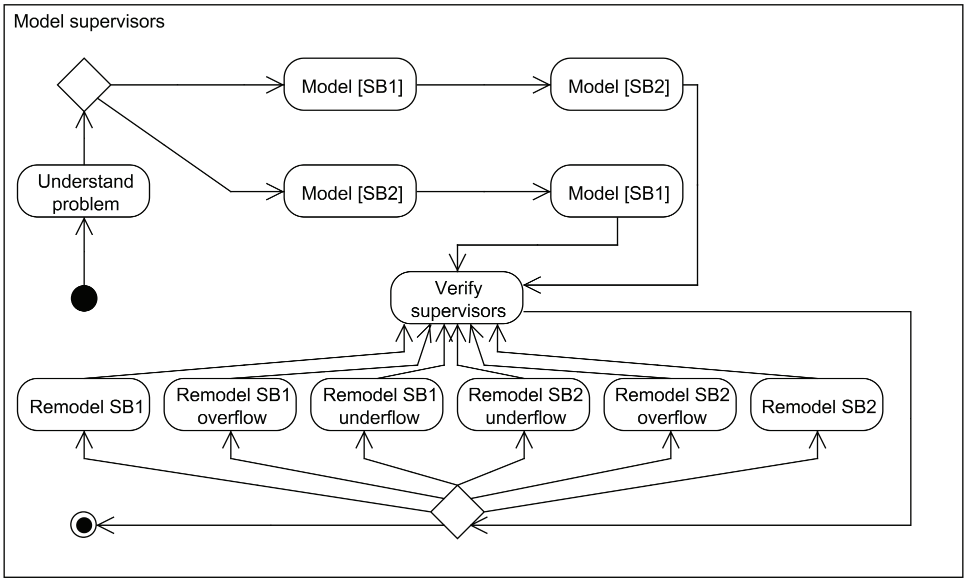

Model supervisors
Model [SB1]
Model [SB2]
Understand problem
Model [SB2]
Model [SB1]
Verify supervisors
Remodel SB1
Remodel SB1 overflow
Remodel SB1 underflow
Remodel SB2 underflow
Remodel SB2 overflow
Remodel SB2

Figure C.4: Flowchart for the "Model supervisors" box.

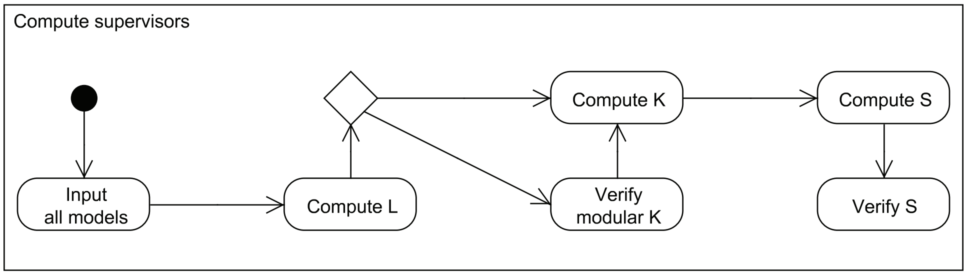

Compute supervisors
Compute K
Compute S
Input
all models
Compute L
Verify
modular K
Verify S

Figure C.5: Flowchart for the “Compute supervisors” box.

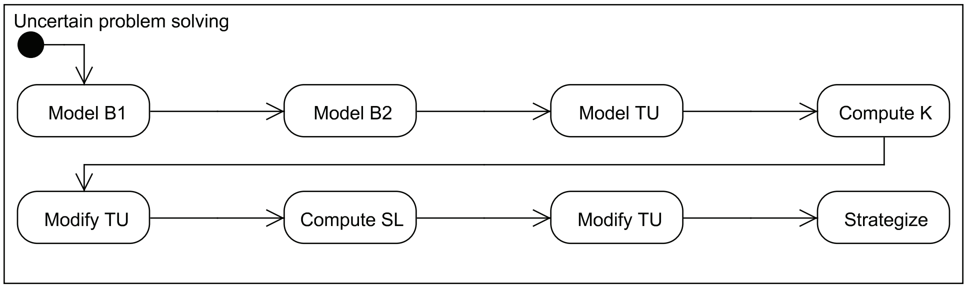

Uncertain problem solving
Model B1
Model B2
Model TU
Compute K
Modify TU
Compute SL
Modify TU
Strategize

Figure C.6: Flowchart for the “Uncertain problem solving” box.

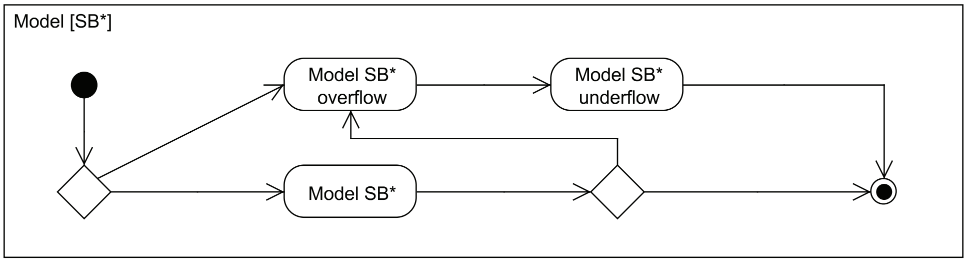

Model [SB*]
Model SB*
overflow
Model SB*
underflow
Model SB*

Figure C.7: Flowchart for the “Model [SB*]” boxes. Here the star can be replaced by either ‘1’ or ‘2’.

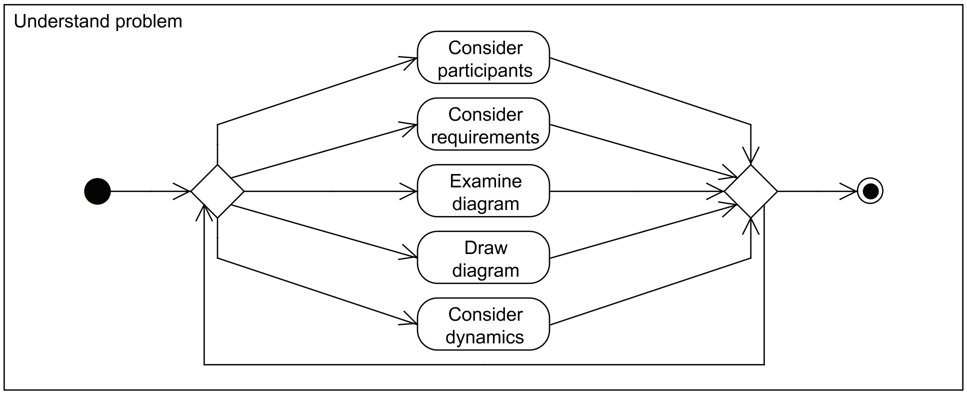

Understand problem
Consider
participants
Consider
requirements
Examine
diagram
Draw
diagram
Consider
dynamics

Figure C.8: Flowchart for the “Understand problem” boxes.

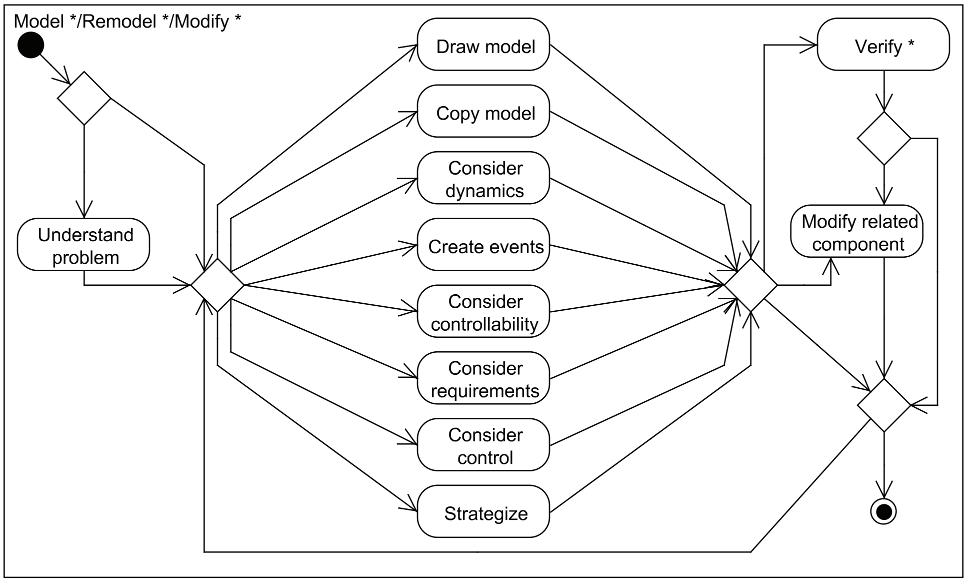

Model */Remodel */Modify *
Draw model
Copy model
Consider dynamics
Create events
Consider controllability
Consider requirements
Consider control
Strategize
Understand problem
Verify *
Modify related component

Figure C.9: Flowchart for the “Model *”, “Remodel *” and “Modify *” boxes. Here the star can be replaced by any module name.

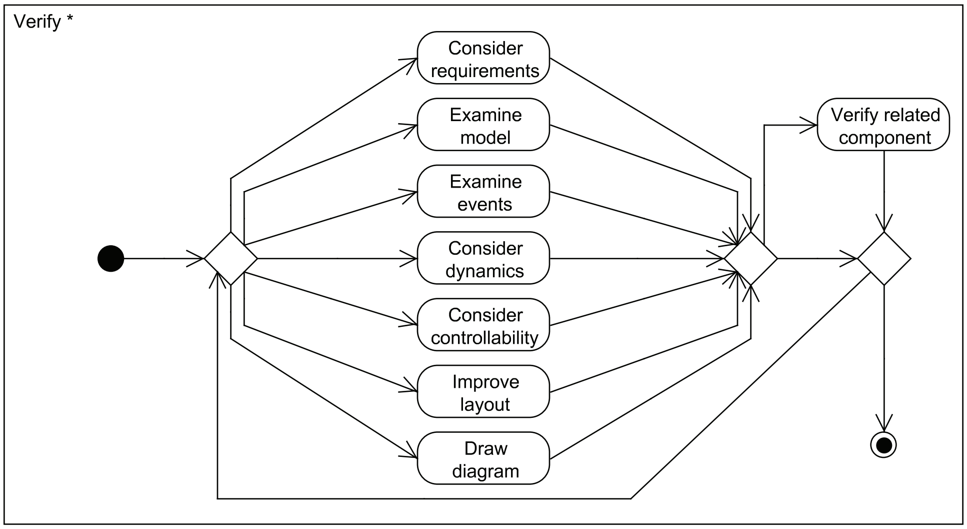

Verify *
Consider requirements
Examine model
Examine events
Consider dynamics
Consider controllability
Improve layout
Draw diagram
Verify related component

Figure C.10: Flowchart for the “Verify *” boxes. Here the star can be replaced by any module name.

C.2 Hospital problem

The abbreviations used in the following flowcharts are as follows: GM – grandmother, PT – patient, PC – patient behavior in terms of candy, PI – patient behavior in terms of insulin, NR – nurse, DR – doctor, DN – personnel (doctor and nurse), EQ – prototype equipment, CP – computer, HIS – Hospital Information System, CE – composition of computer and prototype equipment, BSL – blood sugar level, RP – report, SBSL – supervisor for the blood sugar level, SRP – supervisor for the report.

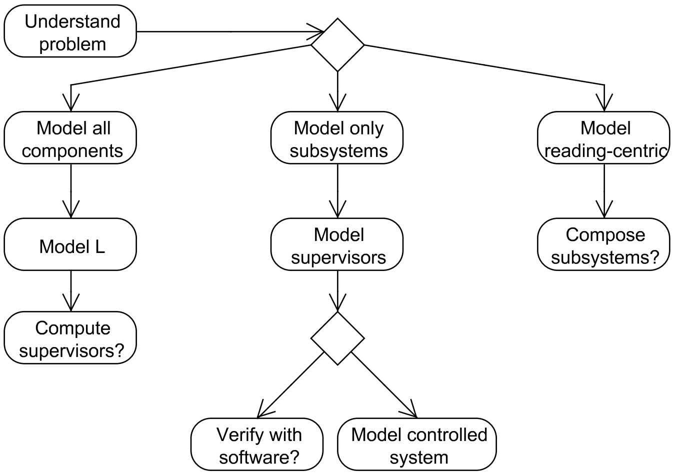

Understand problem
Model all components
Model only subsystems
Model reading-centric
Model L
Model supervisors
Compose subsystems?
Compute supervisors?
Verify with software?
Model controlled system

Figure C.11: The top-level flowchart for the Hospital problem.

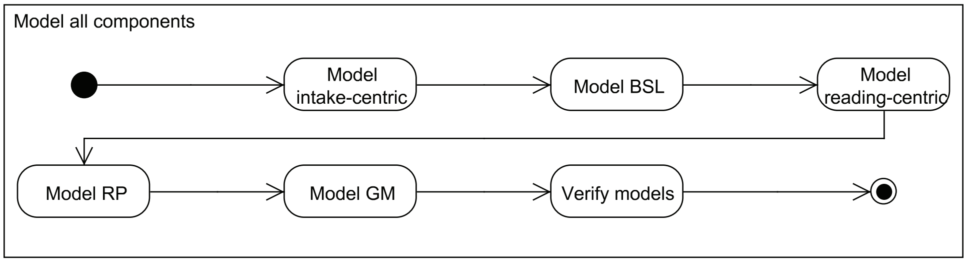

Model all components
Model intake-centric
Model BSL
Model reading-centric
Model RP
Model GM
Verify models

Figure C.12: Flowchart for the “Model all components” box.

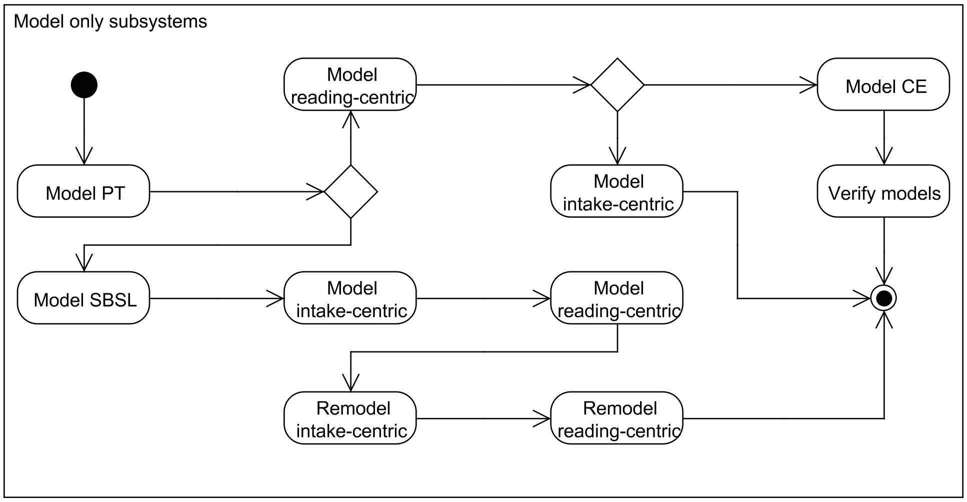

Model only subsystems
Model reading-centric
Model CE
Model PT
Model intake-centric
Verify models
Model SBSL
Model intake-centric
Model reading-centric
Remodel intake-centric
Remodel reading-centric

Figure C.13: Flowchart for the "Model only subsystems" box.

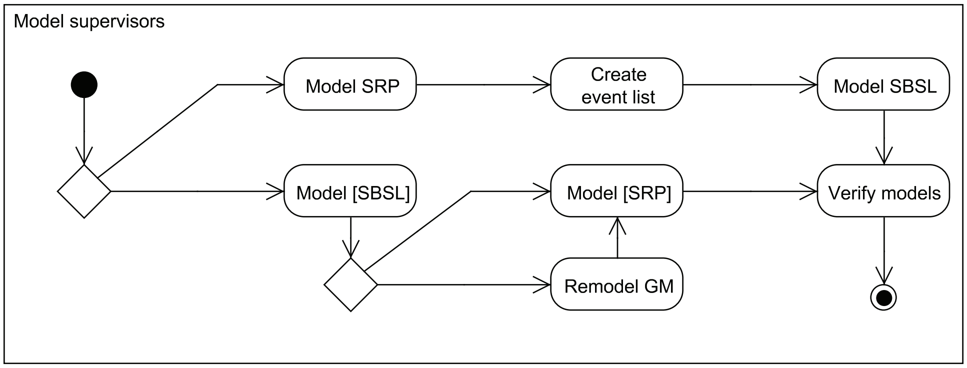

Model supervisors
Model SRP
Create event list
Model SBSL
Model [SBSL]
Model [SRP]
Verify models
Remodel GM

Figure C.14: Flowchart for the "Model supervisors" box.

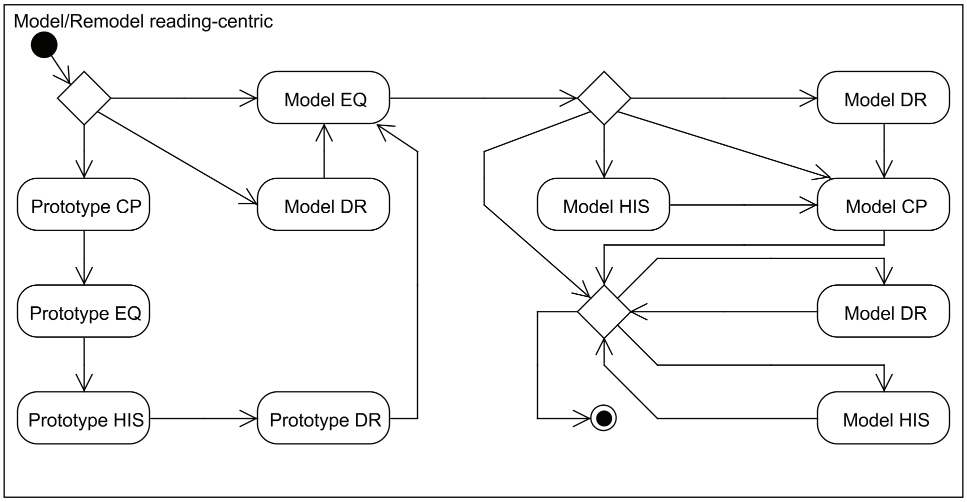

Model/Remodel reading-centric
Model EQ
Model DR
Prototype CP
Model DR
Model HIS
Model CP
Prototype EQ
Model DR
Prototype HIS
Prototype DR
Model HIS

Figure C.15: Flowchart for the "Model reading-centric" and "Remodel reading-centric" boxes.

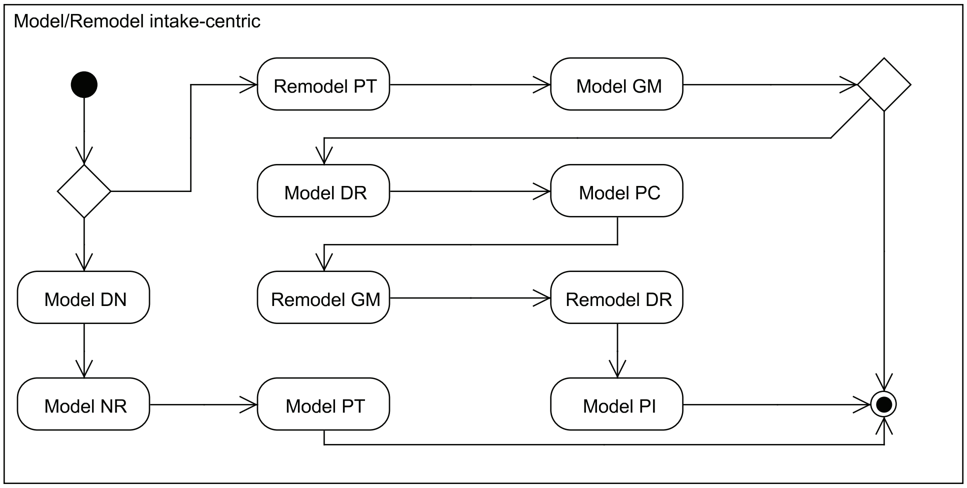

Model/Remodel intake-centric
Remodel PT
Model GM
Model DR
Model PC
Model DN
Remodel GM
Remodel DR
Model NR
Model PT
Model PI

Figure C.16: Flowchart for the “Model intake-centric” and “Remodel intake-centric” boxes.

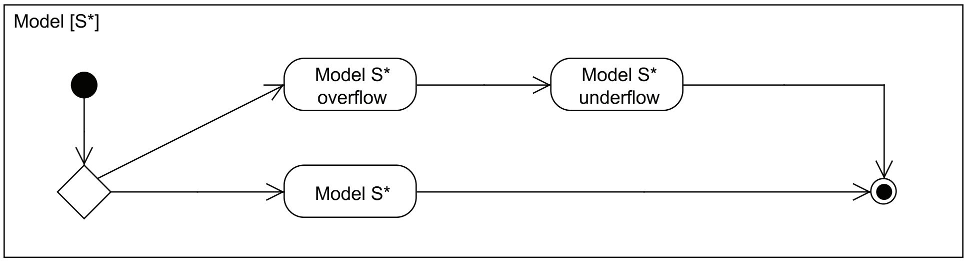

Model [S*]
Model S* overflow
Model S* underflow
Model S*

Figure C.17: Flowchart for the “Model [S*]” boxes. Here the star can be replaced by either ‘BSL’ or ‘RP’.

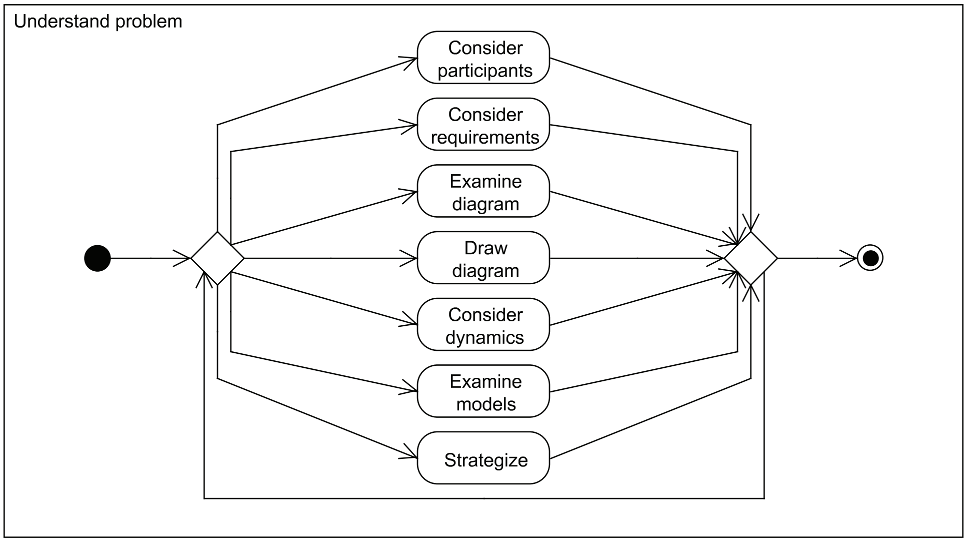

Understand problem
Consider participants
Consider requirements
Examine diagram
Draw diagram
Consider dynamics
Examine models
Strategize

Figure C.18: Flowchart for the “Understand problem” boxes.

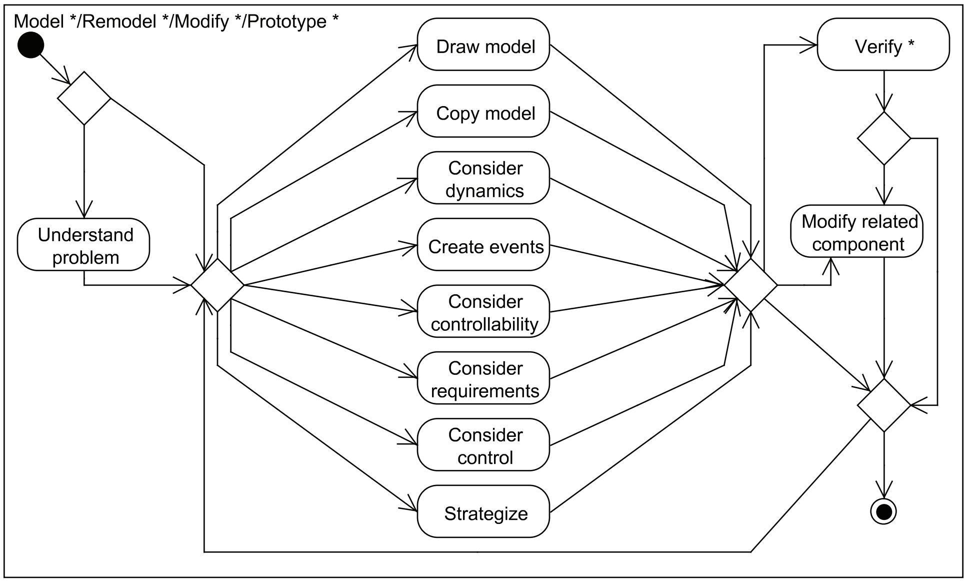

Model */Remodel */Modify */Prototype *
Understand problem
Draw model
Copy model
Consider dynamics
Create events
Consider controllability
Consider requirements
Consider control
Strategize
Verify *
Modify related component

Figure C.19: Flowchart for the "Model *", "Remodel *", "Modify *", and "Prototype *", boxes. Here the star can be replaced by any module name.

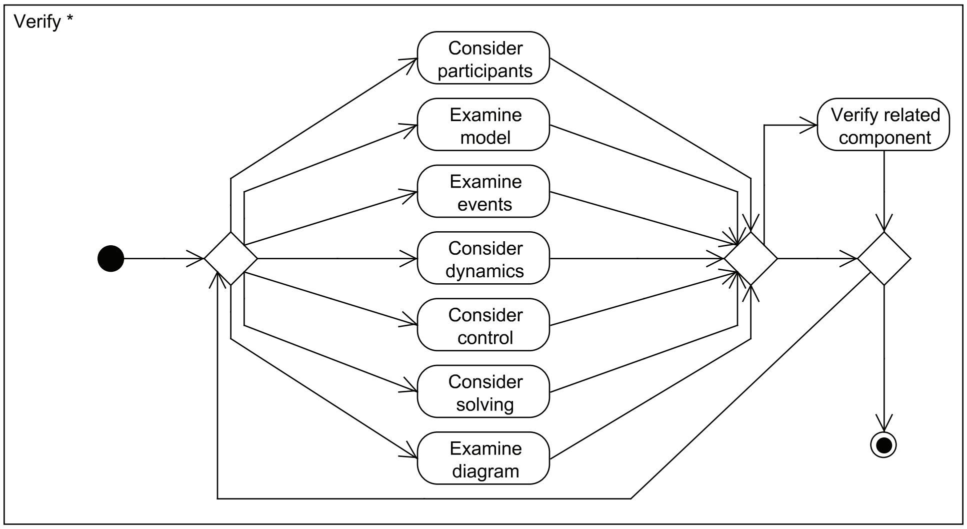

Verify *
Consider participants
Examine model
Examine events
Consider dynamics
Consider control
Consider solving
Examine diagram
Verify related component

Figure C.20: Flowchart for the "Verify *" boxes. Here the star can be replaced by any module name.

Bibliography

[1] K. Åkesson, M. Fabian, H. Flordal, and R. Malik. Supremica – an integrated environment for verification, synthesis and simulation of discrete event systems. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 384–385, Ann Arbor, MI, USA, July 2006.

[2] K. Åkesson, M. Fabian, H. Flordal, and A. Vahidi. Supremica – a tool for verification and synthesis of discrete event supervisors. In *Proceedings of the 11th Mediterranean Conference on Control and Automation*, Rhodos, Greece, 2003.

[3] C. J. Atman, J. R. Chimka, K. M. Bursic, and H. L. Nachtmann. A comparison of freshman and senior engineering design processes. *Design Studies*, 20(2):131–152, 1999.

[4] J. Banks, J. C. II, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Pearson Prentice Hall, Upper Saddle River, NJ, USA, fourth edition, 2005.

[5] M. T. Boren and J. Ramey. Thinking aloud: Reconciling theory and practice. *IEEE Transactions on Professional Communication*, 43(3):261–278, 2000.

[6] X.-R. Cao, G. Cohen, A. Giua, W. M. Wonham, and J. H. van Schuppen. Unity in diversity, diversity in unity: Retrospective and prospective views on control of discrete event systems. *Discrete Event Dynamic Systems*, 12(3):253–264, 2002.

[7] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1999.

[8] S.-L. Chung, S. Lafortune, and F. Lin. Limited lookahead policies in supervisory control of discrete event systems. *IEEE Transactions on Automatic Control*, 37(12):1921–1935, 1992.

[9] CTCT software. Department of Electrical and Computer Engineering, University of Toronto, Canada. Available at http://www.control.toronto.edu/DES/.

[10] A. E. C. da Cunha. private communication, March 2007.

[11] M. H. de Queiroz and J. E. R. Cury. Modular control of composed systems. In *Proceedings of the 2000 American Control Conference*, volume 6, pages 4051–4055, June 2000.

[12] A. J. Dix, J. E. Finlay, G. D. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice Hall Europe, second edition, 1998.

[13] K. Duncker. *On Problem Solving*. Number 58 in Psychological Monographs. American Psychological Association, 1945.

[14] G. Ekberg and B. H. Krogh. Programming discrete control systems using state machine templates. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 194–200, Ann Arbor, MI, USA, July 2006.

[15] L. ellin Datta. A pragmatic basis for mixed-method designs. In *Advances in Mixed-Method Evaluation: The Challenges and Benefits of Integrating Diverse Paradigms*, number 74 in New Directions for Evaluation, pages 33–46. Jossey-Bass Publishers, 1997.

[16] C. M. Enright and M. Barbeau. An evaluation of the TCT tool for the synthesis of controllers of discrete event systems. In *Canadian Conference on Electrical and Computer Engineering*, volume 1, pages 241–244, Vancouver, BC, Canada, September 1993.

[17] K. A. Ericsson and H. A. Simon. *Protocol Analysis*. The MIT Press, Cambridge, Massachusetts, USA, revised edition, 1993.

[18] K. A. Ericsson and H. A. Simon. How to study thinking in everyday life: Contrasting think-aloud protocols with descriptions and explanations of thinking. *Mind, Culture, and Activity*, 5(3):178–186, 1998.

[19] M. Fabian. private communication, March 2008.

[20] M. Fabian and A. Hellgren. Desco – a tool for education and control of discrete event systems. In *Discrete Event Systems: Analysis and Control (Proceedings of the 5th Workshop on Discrete Event Systems)*, pages 471–472, Ghent, Belgium, August 2000.

[21] J. Flochová, R. Lipták, and P. Bachratý. An on line course for supervisory control teaching. In *Proceedings of the 6th IFAC Symposium on Advances in Control Education*, Oulu, Finland, June 2003.

[22] M. E. Fonteyn, B. Kuipers, and S. J. Grobe. A description of think aloud method and protocol analysis. *Qualitative Health Research*, 3(4):430–441, 1993.

[23] J. S. Gero and T. Mc Neill. An approach to the analysis of design protocols. *Design Studies*, 19(1):21–61, 1998.

[24] F. Gobet and I. Oliver. A simulation of memory for computer programs. Technical Report 74, ESRC Centre for Research in Development, Instruction and Training, School of Psychology, University of Nottingham, Nottingham, United Kingdom, 2002.

[25] P. Gray. *Psychology*. Worth Publishers, New York, NY, USA, fourth edition, 2001.

[26] J. C. Greene and V. J. Caracelli. Defining and describing the paradigm issue in mixed-method evaluation. In *Advances in Mixed-Method Evaluation: The Challenges and Benefits of Integrating Diverse Paradigms*, number 74 in New Directions for Evaluation, pages 5–17. Jossey-Bass Publishers, 1997.

[27] L. Grigorov and K. Rudie. Problem solving in control of discrete-event systems. In *Proceedings of the European Control Conference 2007*, pages 5500–5507, Kos, Greece, July 2007.

[28] R. Guindon. Knowledge exploited by experts during software system design. *International Journal of Man-Machine Studies*, 33(3):279–304, 1990.

[29] D. Hofstadter and the Fluid Analogies Research Group. *Fluid Concepts and Creative Analogies*. Basic Books, New York, USA, 1995.

[30] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, second edition, 2004.

[31] IDES software. Department of Electrical and Computer Engineering, Queen's University, Canada. Available at http://www.ece.queensu.ca/directory/faculty/Rudie.html.

[32] S. Jiang and R. Kumar. Supervisory control of discrete event systems with CTL* temporal logic specications. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 5, pages 4122–4127, Orlando, Florida, USA, December 2001.

[33] A. Kalnins, J. Barzdins, and E. Celms. UML business modeling profile. In *Proceedings of the 13th International Conference on Information Systems Development*, pages 182–194, Vilnius, Lithuania, September 2004.

[34] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann Publishers, Inc., San Mateo, California, USA, 1993.

[35] F. Lin and W. M. Wonham. On observability of discrete-event systems. *Information Sciences*, 44(2):173–198, 1988.

[36] A. S. Luchins and E. H. Luchins. *Rigidity of Behavior: A Variational Approach to the Effect of Einstellung*. University of Oregon Books, Eugene, Oregon, USA, 1959.

[37] T. Mäntylä. Optimizing cue effectiveness: Recall of 500 and 600 incidentally learned words. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 12:66–71, 1986.

[38] J. L. McClelland and D. E. Rumelhart. An interactive model of context effects in letter perception: I. An account of basic findings. *Psychological Review*, 88:375–407, 1981.

[39] A. Newell and H. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. MIT Press, Cambridge, MA, USA, 1963.

[40] D. A. Norman. *The Design of Everyday Things*. Currency, New York, New York, USA, 1990.

[41] *OMG Unified Modeling Language: Superstructure*, chapter Activities. Object Management Group, Inc., http://www.uml.org/, 2007.

[42] S. Owen, P. Brereton, and D. Budgen. Protocol analysis: a neglected practice. *Communications of the ACM*, 49(2):117–122, 2006.

[43] M. Q. Patton. *Qualitative Research and Evaluation Methods*. Sage Publications, Inc., Thousand Oaks, CA, USA, 3 edition, 2002.

[44] Y. Qin and H. Simon. Imagery and mental models. In *Diagrammatic Reasoning*, pages 403–434. The AAAI Press and MIT Press, 1995.

[45] P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

[46] E. Rogers. A cognitive theory of visual interaction. In *Diagrammatic Reasoning*, pages 481–500. The AAAI Press and MIT Press, 1995.

[47] E. Rogers. VIA-RAD: a blackboard-based system for diagnostic radiology. *Artificial Intelligence in Medicine*, 7:343–360, 1995.

[48] E. Rogers and R. C. Arkin. Visual interaction: A link between perception and problem solving. In *Proceedings of the 1991 IEEE International Conference on Systems, Man, and Cybernetics, 'Decision Aiding for Complex Systems'*, volume 2, pages 1265–1270, Charlottesville, VA, USA, October 1991.

[49] E. Rogers, R. C. Arkin, and M. Baron. Visual interaction in diagnostic radiology. In *Proceedings of the Fourth Annual IEEE Symposium on Computer-Based Medical Systems*, pages 170–177, Baltimore, Maryland, USA, May 1991.

[50] E. Rogers, R. C. Arkin, M. Baron, N. Ezquerra, and E. Garcia. Visual protocol collection for the enhancement of the radiological diagnostic process. In *Proceedings of the First Conference on Visualization in Biomedical Computing*, pages 208–215, Atlanta, Georgia, USA, May 1990.

[51] K. Rudie. The integrated discrete-event systems tool. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 394–395, Ann Arbor, MI, USA, July 2006.

[52] K. Rudie and W. M. Wonham. Think globally, act locally: decentralized supervisory control. *IEEE Transactions on Automatic Control*, 37(11):1692–1708, 1992.

[53] H. A. Simon. *Models of Man: Social and Rational*. John Wiley and Sons, New York, NY, USA, 1957.

[54] H. A. Simon. The structure of ill structured problems. *Artificial Intelligence*, 4(3–4):181–201, 1973.

[55] R. Spence. *Information Visualization*. Addison-Wesley, 2000.

[56] Subtitle workshop. URUWorks. Available at http://www.urusoft.net/.

[57] C. Y. Suen. N-gram statistics for natural language understanding and text processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):164–172, 1979.

[58] H. J. M. Tabachneck-Schijf, A. M. Leonardo, and H. A. Simon. CaMeRa: A computational model of multiple representations. *Cognitive Science*, 21(3):305–350, 1997.

[59] H. J. M. Tabachneck-Schijf and H. A. Simon. Alternative representations of instructional material. In D. Peterson, editor, *Forms of representation*, pages 28–46. Intellect Books, Exeter EX2 6AS, UK, 1996.

[60] P. Thagard. *Mind: Introduction to Cognitive Science*. The MIT Press, Cambridge, Massachusetts, USA, second edition, 2005.

[61] J. G. Thistle and W. M. Wonham. Control problems in a temporal logic framework. *International Journal of Control*, 44(4):943–976, 1986.

[62] UMDES software library. Department of Electrical Engineering and Computer Science, University of Michigan, USA. Available at http://www.eecs.umich.edu/umdes/.

[63] A. F. Vaz and W. M. Wonham. On supervisor reduction in discrete-event systems. *International Journal of Control*, 44(2):475–491, 1986.

[64] B. Wang. Top-down design for RW supervisory control. Master's thesis, Department of Electrical and Computer Engineering, University of Toronto, 1995.

[65] P. C. Wason and P. N. Johnson-Laird. *Psychology of Reasoning: Structure and Content*. Harvard University Press, Cambridge, MA, USA, 1972.

[66] W. M. Wonham. Notes on supervisory control of discrete-event systems. Available at http://www.control.toronto.edu/DES/, July 2004.

[67] W. M. Wonham and P. J. Ramadge. Modular supervisory control of discrete-event systems. *Mathematics of Control, Signals, and Systems*, 1:13–30, 1988.

[68] M. M. Wood. Application, implementation and integration of discrete-event systems control theory. Master's thesis, Department of Electrical and Computer Engineering, Queen's University, 2005.