# Conceptual design of discrete-event systems using templates

by

Lenko Grigorov Grigorov

A thesis submitted to the
School of Computing
in conformity with the requirements for
the degree of Doctor of Philosophy

Queen's University
Kingston, Ontario, Canada
August 2009

# Abstract

This work describes the research conducted in the quest for designing better software for discrete-event system (DES) control. The think-aloud data from an exploratory observational study of solving DES control problems contributed to the formulation of a list of recommendations on how to design and improve DES software. These observations, together with other relevant research, led to the proposal of a novel approach to DES problem solving, namely, the *template design* methodology. This methodology does not require the introduction of new control theory; it is rather an reinterpretation of the existing modelling framework. Software supporting this methodology was implemented and subsequently evaluated using twelve subjects. Significant improvements in the speed of problem solving as well as positive evaluations by the subjects were observed. The usability data do not show any drawbacks to applying the methodology.

# Acknowledgments

I would like to thank foremost my supervisor, Dr. Karen Rudie, for her dedication as a supervisor, for the encouragements I received, and for her cheerfulness and sense of humor which helped me in the completion of this work.

I would also like to thank for the professional help of everyone who helped shape this work. My supervisory committee included Dr. Brian Butler, Dr. Janice Glasgow and Dr. Kai Salomaa from Queen's University, Canada. The advices of all of them were very valuable in making some critical decisions. The examination committee at my defence included Dr. Martin Fabian from Chalmers University of Technology, Sweden and Dr. Roger Browse, Dr. William Egnatoff, Dr. Juliana Ramsay and Dr. Robert Tennent from Queen's University, Canada. To them I thank for all the suggestions which helped improved the quality of this work. Other people who contributed with advice or otherwise include: Dr. Knut Åkesson from Chalmers University of Technology, Sweden, Dr. Dorothea Blostein from Queen's University, Canada, Dr. Eduardo Carrilho from the Military Institute of Engineering, Brazil, Daniel Cownden from Queen's University, Canada, Dr. José Cury from the Federal University of Santa Catarina, Brazil, Steffi Klinge from Otto-von-Guericke University, Germany, Guilherme Lise and Luis Marques from the Federal University of Santa Catarina, Brazil, Dr. Kathleen Norman from Queen's University, Canada, Dr. Greg Phillips from the Royal Military College, Canada, and Dr. Max de Queiroz and Francisco da Silva from the Federal University of Santa Catarina, Brazil. I also acknowledge the help of my colleagues Anthony Auer, Christopher Dragert, Sarah-Jane Whittaker and Creag

Winacott with whom we had fruitful discussions, and the help of all volunteers who participated in the studies which form an essential part of this work.

Last but not least, I would like to thank my family and friends without whom the Ph.D. life would have been much harder.

# Statement of Originality

I hereby declare that I am the sole author of this thesis and the research described herein. Parts of this research have been conducted under the supervision of Karen Rudie, Queen's University, Kingston, Ontario, Canada and José E. R. Cury, Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil. I hereby also acknowledge that the data from the studies described in the thesis were collected from the participants in these studies.

Parts of this thesis have been published elsewhere, as indicated where relevant.

# Preface

The donkey swung its tail and looked at the owl once more.

"Hmm... I still don't understand. You say you manage the forest?"

"Look, it's simple," answered the owl. "From up there, in the sky, I can see everything. I can see where the little lake is, where the patch of oak trees is... I can even see the path from your house to the candy store."

"Really?!"

"So, then it's easy to see where I need to plant more trees," continued the owl. "Do you remember the storm last week that broke all the young pines on the East side of the hill?"

The donkey nodded.

"Well, yesterday I took some pine cones from my closet, flew over to the place, and planted them. In a few years, we'll have new pine trees—exactly the same as before."

"Not that I understand anything about pine cones..." murmured the donkey, hoping that the owl would not hear him.

The owl continued excitedly. "Yes, from up there you can see everything! I even noticed that the turtle has to go around the hill to get to the store. Me and the lion planned a different route—straight through the meadow—which will save the turtle a lot of time."

As the owl talked with glistering eyes, the donkey looked to the berry bushes and thought. "Will the owl be able to plant some more of these?"

# Table of Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

The computerization and digitization of all aspects of our lives is an undeniable result of the technological advances during the last few decades. Unfortunately, as researchers in Human-Computer Interaction (HCI) point out [18, 78], our understanding of the human factors in computer use and in information visualization is very rudimentary. There are no central theories which reliably and robustly describe human thinking and performance when dealing with a complex and interactive device such as a computer. This leaves most computer interface designers with the only option of relying on their intuition of what is appropriate design. This frequently leads to problematic interface designs because of insufficient understanding of the needs of the users [83]. Unfortunately, the use of a computer is unavoidable for the solution of certain problems—even if the interfaces of the available software packages are very contrived.

An example of a field of work where computers are indispensable is the field of Control of Discrete-Event Systems (DESs) [10]. Many systems can be modelled as DESs, including manufacturing equipment, avionics, network protocols and logistic operations. Using the tools in the DES supervisory control theory then allows for the automatic construction of correct and optimal controllers. Unfortunately, the problems which need to be solved may involve computations over enormous discrete sets with more than $10^6$ elements [36, 57]. These computations are clearly outside the capabilities of human agents (if one is to keep the expenses reasonable). In the field of DES control there has been a long-recognized need for better software tools [9]. In most DES software, the central role is assumed by the implementation of different computational algorithms; such is the case with the TCT [14] and UMDES [86] tools, for example. However, the implementation only of algorithms proves to be insufficient to aid in real-world problem solving. In [21], 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 [23] and later the Supremica software [1] were developed to address issues surrounding the application of the DES supervision.

The IDES software developed at Rudie's research laboratory, [73], 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 than for 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 not sufficient to resolve 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, it is necessary to understand the origins of the difficulties humans experience when working on DES control problems.

Previous to our work, [35, 32], 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 [24], 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 [1], 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 [97], 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. However, these insights do not provide enough information on the DES problem-solving strategies.

At the onset of the research described in this work, it seemed to us that a more thorough investigation of problem solving in DES has been long overdue. In order to develop the next generation of DES software which goes beyond a simple collection of algorithms (with or without a graphical modelling environment), it was necessary to understand how people deal with DES control problems. Our ultimate goal was to make use of this knowledge to guide the extension of the IDES software package so that using it makes DES problem solving simpler, faster, and more reliable. To this end, we conducted an exploratory observational study of DES problem solving and we analysed the collected data [32].

Based on our observations in [32] and inspired by the work of Santos *et al.* [75], here we propose a new approach to DES problem solving within the standard supervisory control framework, namely, the *template design* methodology. There are two main ingredients in the methodology: high-level conceptual modelling and the availability of templates. The high-level design consists of entities and connections between them. Entities are simply finites-state models which, as in the classical framework, can describe either components of the system or components of the control specifications. We call the system components *modules* and the specification components *channels*, as the control specifications serve to define the protocols of interaction between system components. Unlike the classical framework, synchronization between DES entities is not done directly via event name equivalences. Rather, separate event name maps are created, and the connections between modules and channels serve as the embodiment of these maps. The use of connections makes the reconfiguration of high-level designs easy and fast. This not only simplifies the sharing and reuse of models, but also enables the introduction of templates. Templates are designated finite-state models which describe the behavior of commonly used components. For example, if a factory has a number of the same robots, a template can be created describing the generic behavior of this type of robot. During modelling, templates can be *instantiated*, i.e., copies of the template models can be made. For example, in order to model all the similar robots on the factory floor, it will be sufficient to instantiate the robot template the corresponding number of times. The availability of templates simplifies the process of modelling and reduces the opportunities of making errors. Furthermore, the users of the software no longer need to be experts in creating low-level finite-state models in order to design control solutions.

We implemented the proposed methodology as an extension of IDES, in the form of a plugin. In order to evaluate the usability of the new tool, and of the methodology, we performed an experiment involving twelve participants. The participants were asked to solve two DES problems each, where one of the problems had to be solved using the classical approach (using IDES without the template design plugin) and the other problem had to be solved using the template design methodology (using IDES with the template design plugin). Different measures of usability were collected via questionnaires. Three measures—speed, experiential ease of use, and System Usability Scale scores [7]—showed improvement when the template design methodology was used. The evaluation did not discover any negative impacts in using the proposed methodology in comparison to the classical approach.

The main contributions of our work include:

- Description of recommendations for the implementation of DES software based on observations of DES problem solving,
- Proposal of a novel method, called template design, for the design of DES control solutions,
- Implementation and evaluation of the proposed method in order to validate it.

The rest of this dissertation is organized as follows. Chapter 2 contains a review of relevant background information. A list of recommendations for IDES, and DES software in general, based on the observational study in [32] is given in Chapter 3. As well, in Chapter 3 we discuss the motivation behind these recommendations and behind the proposal of the template design methodology. The template design methodology itself is described in Chapter 4, while the subsequent implementation is described in Chapter 5. Then, in Chapter 6, we discuss the results of the usability evaluation of the template design software tool. We conclude with Chapter 7. The appendices at the end list the problems and questionnaires administered during the evaluation from Chapter 6.

Some parts of this work have already been published in [35, 33, 34]. Additional details of the work can be found in [32, 31].

# Chapter 2

# Literature Review

## 2.1 Discrete-Event Systems

The process of computerization and digitalization of devices and protocols implies, at a low level, the discretization of the process or device which will be controlled. While Classical Control Theory deals with the control of continuous systems, a new approach is necessary for the control of discrete systems. In the 1980s, Ramadge and Wonham published a number of very influential articles [67, 95, 68]. They proposed a framework for the control of a class of discrete systems, called Discrete-Event Systems (DESs).

Discrete-Event Systems are systems where events (changes of state) occur sequentially and asynchronously. For example, an elevator in a building can be modeled as a DES. The events would be the opening and closing of the doors, the pressing of the buttons, the arrival at a floor, etc. The states of the system would include information about requests for a stop at different floors, the position of the elevator, the direction in which it moves, etc. Events occur sequentially: the model does not allow the simultaneous pressing of two buttons (in reality, the low-level event processing unit ensures the sequencing of events). Also, events are asynchronous because, for the purpose of the model, it does not matter how much time passes between two events. Even though an elevator is a much more complicated system, in reality, the electronic control of elevators uses a model similar to the one described. There are many ways to model DESs, including Petri nets [61], fuzzy matrices [54], and temporal logic [82]. However, the most commonly used and natural model is that of an automaton and, for practical purposes, a finite-state automaton (FSA). The latter is also used in the Ramadge and Wonham framework. Synchronous languages such as Esterel [5] and Signal [37] may also be suitable for the description of DESs. These languages allow the description of reactive systems, where input signals trigger the synchronous computation of output signals; in essence input signals can be viewed as event occurrences. In the case of Esterel, the source code can be efficiently translated into an FSA. However, such an FSA would describe only an abstraction of the corresponding program. Namely, Esterel uses integers, valued signals, and *execution* statements (to call functions external to the language)—all of which preclude complete encoding in a finite structure. As a consequence, the FSA abstractions of Esterel programs are not suitable for the synthesis of correct and optimal supervisors as done in the Ramadge and Wonham framework. For example, there are properties of supervisory control solutions (such as *non-blocking*, which is described later in this section) which require checking whether the system can advance from a particular state. In the general case, it is not possible to determine if an integer signal will have a specific value under runtime conditions; thus, in the FSA abstraction, it is not possible to predict if the system will be able to advance from a state which has a single outgoing transition, where the transition is conditioned on an integer signal. The problem of applying supervisory control to reactive systems is overcome, to some extent, by Marchand *et al.* who succeed in introducing controller synthesis techniques to Signal [59]. They take advantage of the polynomial dynamical system abstractions of Signal programs to which they attach additional control conditions. With this methodology, it is possible to synthesize controllers for invariance, reachability and attractivity over the states of the system. The main problem of abstracting away potentially significant behavior, however, remains. As the authors explain, for the numerical functional behavior of Signal programs, only the synchronization properties of the signals can be part of the control specifications. In the rest of this work we will focus on the Ramadge and Wonham framework for DES control as the theoretical research done within it is one of the most extensive and comprehensive.

An FSA is a tuple $G = (\Sigma, Q, \delta, q_0, Q_m)$, where $\Sigma$ is a finite set of symbols (also referred to as the "alphabet"), $Q$ is a finite set of states, $\delta : Q \times \Sigma \to Q$ is a partial transition function, $q_0$ is the initial state and $Q_m \subseteq Q$ is the set of marked states. The "empty symbol" $\epsilon$, which is not in $\Sigma$, is used to denote a string of symbols with length zero. The notation $\Sigma^*$ stands for the set of all finite strings of symbols from $\Sigma$ and $\epsilon$. The transition function $\delta$ can be naturally extended to $Q \times \Sigma^* \to Q$. Such an FSA can be interpreted as a DES if the states are considered to be states of the system and the symbols from $\Sigma$ to be the events which can occur in the system. Thus, strings of symbols would describe sequences of events.

The language $L(G)$ is defined to be the set of all possible sequences of events in the system. The FSA $G$ is said to generate $L(G)$. The language $L_m(G)$ is defined to be the set of all sequences of events which lead to a marked state. The FSA $G$ is said to accept $L_m(G)$. More formally,

$$L(G) = \{s \mid s \in \Sigma^*, \delta(q_0, s) \text{ is defined}\},$$

$$L_m(G) = \{s \mid s \in \Sigma^*, \delta(q_0, s) \text{ is defined}, \delta(q_0, s) \in Q_m\}.$$

The language $L(G)$ can be viewed as the unrestricted behavior of a DES and $L_m(G)$ as the sequences of events that accomplish a task, also called *marked strings*.

The string $t$ is called a prefix of the string $s$, denoted $t \leq s$, if $\exists u \in \Sigma^*, s = tu$. The empty string $\epsilon$ is a prefix of all strings. The *prefix-closure* of a language is defined to be the set of all prefixes of strings in the language:

$$\overline{L} = \{t \mid t \in \Sigma^*, \exists s \in L, t \leq s\}$$

An FSA $G$ is called *non-blocking* if $L(G) = \overline{L_m(G)}$. In other words, all string prefixes it can generate eventually lead to a marked state. The non-blocking property is important because, when it is not satisfied, the DES may get "stuck" during runtime, i.e., reach a state from which a marked state is not reachable.

An example of a DES is the simplified model of a customer at a store [36] (Fig. 2.1). The customer can enter the store, pick something to buy, pay with cash or a credit card, and leave at any time. Here $\Sigma = \{$"enter", "pick", "pay_cash", "pay_cc", "leave"$\}$. The set of states is $Q = \{q_0, q_1, q_2, q_3\}$. The transition function can be determined from the diagram in Fig. 2.1, e.g., $\delta(q_1, \text{pick}) = q_2$. The initial state is $q_0$. The set of marked states is $Q_m = \{q_0\}$. Examples of event sequences are "enter, leave" or "enter, pick, pay_cc". The second sequence is not "complete"—it does not belong to $L_m(G)$. However, it belongs to $\overline{L_m(G)}$, since it is a prefix of the

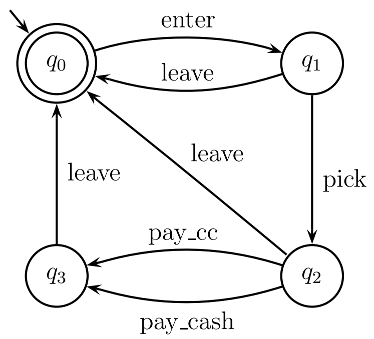


Figure 2.1: DES model of a customer in a store.

sequence “enter, pick, pay_cc, leave”, which is in $L_m(G)$. This particular example is very simple, but one can easily imagine the application of DESs in factory processes, computer protocols, and other areas.

After having defined a DES, one of the questions of greatest interest is how one would be able to influence its unrestricted behavior. In other words, what restrictions would one use so that certain specifications on the behavior are met? The largest body of research on DESs deals with this specific problem: the control of DESs.

The basic FSA model does not provide any means of control. Thus, Ramadge and Wonham [67] extend it by distinguishing between *controllable* and *uncontrollable* events. Controllable events are events which can be “disabled”, or prevented from occurring, and “enabled”. Uncontrollable events remain enabled all the time. The sets of all controllable and uncontrollable events are denoted $\Sigma_c$ and $\Sigma_{uc}$, respectively. Thus,

$$\Sigma = \Sigma_c \cup \Sigma_{uc},\ \Sigma_c \cap \Sigma_{uc} = \emptyset.$$

A specification for the desired behavior of a DES $G$ is given as a language $K \subseteq L(G)$. The restriction of the complete behavior is done by disabling the controllable events when needed. This can be formalized by the construction of an FSA $S = (\Sigma, Q^S, \delta^S, q_0^S, Q_m^S)$, such that $K = L(S)$. Consequently, the controlled behavior of the DES, $L(S/G)$, can be obtained by intersecting the two languages: $L(S/G) = L(S) \cap L(G)$. This method is called *supervisory control* and $S$ is termed a *supervisor*.

Unfortunately, control of DESs is not such a trivial issue. Sometimes, the specification $K$ may contain a string $t$ such that an uncontrollable event can follow in the system while the specification does not permit it, i.e., $\exists s \in L(G), s = t\sigma, \sigma \in \Sigma_{uc}, t\sigma \notin K$. Let us consider the example in Fig. 2.1. The only controllable events are $\Sigma_c = \{$"pay_cash", "pay_cc"$\}$ (paying with cash or a credit card, respectively). Imagine that the credit card reader is broken. Then, the desirable behavior from a customer would be $K = \{$"enter, leave", "enter, pick, pay_cash, leave"$\}$. This specification, however, cannot be implemented using supervisory control: the event "leave" is uncontrollable, thus, it cannot be disabled after the string "enter, pick". Despite our best intentions to prevent theft from the store, the underlying system does not have the necessary capability. This discussion leads to the following definition: a language $K$ is called *controllable with respect to a system* $G$ if and only if

$$\{s\sigma \mid s \in \overline{K}, \sigma \in \Sigma_{uc}, s\sigma \in L(G)\} \subseteq \overline{K}.$$

Controllability of a specification language is important because only in such a case can the required restrictions be implemented via supervisory control. In [95], the authors show that the class of all controllable sublanguages with respect to a DES, $\mathcal{C}(K, G) = \{L \mid L \subseteq K, L \text{ is controllable with respect to } G\}$, is a complete semilattice with respect to set union and has a supremal element. The largest controllable sublanguage of $K$ with respect to $G$, $\sup \mathcal{C}(K, G)$, can be computed in polynomial time in terms of the number of states. In the example from Fig. 2.1, $\sup \mathcal{C} = \emptyset$ since, once we let a customer in the store, we cannot prevent theft.

In many cases, a complete system is composed of separate modules which interact. This fact can be utilized when the system is modeled as a DES. *Modular control* of DESs, [96], uses the operation called *synchronous product* (also known as *parallel composition*) and denoted $||$ to compose DES systems (modules) into supersystems. For two systems, $G_1 = (\Sigma_1, Q_1, \delta_1, q_{01}, Q_{f1})$ and $G_2 = (\Sigma_2, Q_2, \delta_2, q_{02}, Q_{f2})$, the synchronous product is defined to be the automaton $G_1 \| G_2 = (\Sigma_1 \cup \Sigma_2, Q_1 \times Q_2, \delta, (q_{01}, q_{02}), Q_{f1} \times Q_{f2})$, where the states are elements of the Cartesian product of the sets of states of the two automata, the transition function $\delta$ is defined as $\delta : (Q_1 \times Q_2) \times (\Sigma_1 \cup \Sigma_2) \to Q_1 \times Q_2$,

$$\delta((q_1, q_2), \sigma) = \begin{cases} (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma)) & \text{if both } \delta_1(q_1, \sigma) \text{ and } \delta_2(q_2, \sigma) \text{ are defined,} \\ (\delta_1(q_1, \sigma), q_2) & \text{if only } \delta_1(q_1, \sigma) \text{ is defined and } \sigma \notin \Sigma_2, \\ (q_1, \delta_2(q_2, \sigma)) & \text{if only } \delta_2(q_2, \sigma) \text{ is defined and } \sigma \notin \Sigma_1, \\ \text{undefined} & \text{otherwise.} \end{cases}$$

In other words, the modules interact, and are synchronized, through their common events. The parallel composition can be defined equivalently in linguistic terms. Let $i = 1, 2$ and $P_i : (\Sigma_1 \cup \Sigma_2)^* \to \Sigma_i^*$ be the natural projections of strings from the combined alphabets to $\Sigma_i^*$. Then, $L(G_1) \| L(G_2) = P_1^{-1}(L(G_1)) \cap P_2^{-1}(L(G_2))$. Modular DES design is an elegant and convenient architectural approach, however, it has a significant drawback. In general, the state space of a supersystem may grow exponentially with the number of modules. Thus, the computation of a supervisor for the complete system becomes intractable even for moderately-sized real systems.

One positive result, however, is that, under certain conditions, *local supervisors* can be constructed for each module such that their combined use ensures the global specification is met [96, 16].

Further information on discrete-event systems and their control can be found in [10, 49, 94].

The main purpose of software packages for the solution of DES control problems is to implement the algorithms necessary to check controllability of languages, to compose modules, compute supremal controllable sublanguages, and perform other similar operations. Usually, the user interfaces are not designed to support many other activities. In extreme examples, such as CTCT [14], the interface consists of a menu where different algorithms can be called. However, as discussed next, human cognition is a very complex process. It cannot be supported efficiently with the simple provision of computational algorithms in isolation from everything else.

## 2.2 Human Problem-Solving

During the last half-century, research in Cognitive Science [81] and Cognitive Psychology [2] in particular has resulted in many advancements in the understanding of the human cognitive processes. Such information can be used successfully in the design of novel interfaces for software systems [18, 71].

The act of problem solving is essential to the cognitive function of human beings, some claim [2]. Whether one takes such a stance or not, it cannot be denied that a large portion of our conscious mental activity involves problem solving. Many of the tools built by people are meant to assist in problem solving. The computer may be viewed as the most versatile tool to assist people in this task. There are two ways the computer can be used: it may *replace* or *aid* a human solving a given problem. In this section we will discuss previous work regarding general problem solving and argue that for DES control problems, it may be impossible to create software which will replace experts. Instead, better software tools can be built through a better understanding of how people solve DES problems.

Given a problem, humans will rely on general background knowledge, problem-related knowledge and past experiences to understand the task and make the correct inferences. Also, people have an array of strategies that can be used to attempt to solve a problem. These include random trial-and-error, systematic trial-and-error, heuristics, analogy, etc. [92]. As Thagard [81] argues, analogy is one of the most powerful problem solving strategies. Past problem-solving experiences create certain structures which relate entities and concepts according to their roles in the solution. In encountering a similar situation, the person would recall this structure and then simply map the new entities and concepts onto the structure to automatically obtain the necessary relations. In the terminology of object-oriented programming, experiences are "abstracted" into "classes" of objects which are "instantiated" for new problems. Analogical problem-solving requires very little mental effort and this is the reason why people have a preference for it. Sometimes they might even try to "force-map" a new problem onto the structure of an old problem even though the two problems are incompatible. This leads to an easily-obtainable solution, however, the solution might be sub-optimal or even incorrect. The analogy method of solving problems is related to the Einstellung phenomenon discussed later in this section. While it may seem that there is a great risk of erring associated with the use of analogies to solve problems, human activity as we know it would not be possible without "taking shortcuts" through analogies. Imagine, for example, what it would be like if every door one needed to open appeared as completely unrelated (non-analogous) to other doors one has encountered in one's past experience. Then, one would need to spend time and mental effort trying to figure out how to open every new door.

Unfortunately, it is not always possible to use analogies. How would one proceed to solve a novel problem which has not been encountered yet? This depends on the type of problem being solved. Johnson-Laird [46] differentiates between two categories of problems: those that involve gradual advancement towards the goal and those that involve *insight* to reach the solution. The first category usually involves problems that are familiar to the problem-solver. For example, given a particular quadratic equation, a student who knows the formula for finding the roots would apply a familiar methodology and gradually work their way to solving the problem. In another example, somebody who is acquainted with the general method of solving the Tower of Hanoi puzzle [55] would be able to solve any instance of the puzzle (with any number of disks) by systematically applying the learned rules. The second category of problems involves problems where the solver cannot complete the solution without gaining *insight*—a qualitatively different view of the problem or strategy. For example, if a student knows only the formula for finding the area of a parallelogram and they need to find the area of a triangle, they may solve the problem by gaining the insight that the triangle may be viewed as half of a parallelogram.

Any problem can be seen as an insight problem if one considers each step in a gradually advancing solution to be a small-scale insight. However, as Johnson-Laird points out, the practical experience in the solution of an insight problem is completely different from the experience in gradual problem-solving [46]. Furthermore, he suggests that following a gradual problem-solving strategy in an insight problem usually precludes reaching a successful solution [60].

How do people gain the necessary insight to a problem? The research on this topic has been largely inconclusive [46]. However, the outward representation of the behavior can be described as follows [88]. First, there is the stage of *preparation*, that is, when one builds a knowledge base and attempts to solve the problem using consciously directed activity. If there is an *impasse*, when all problem-solving strategies employed seem to no longer offer new inferences that bring one closer to the goal, one can stop directing any conscious activity towards finding a solution. The process enters the *incubation* period. The gain of insight happens suddenly, without prior warning, and results in a qualitatively different view of the problem. This is called an *illumination.* In the last stage, the person resumes conscious problem-solving activity to *verify* that the new inferences will lead to the solution.

Of course, the gain of insight is not white magic in the working: one cannot solve a problem “unconsciously” if one does not have the necessary background knowledge or experiences needed to solve the problem. For example, it makes no sense to ask a life-long inhabitant of the Amazon jungle to intuitively choose the better of two pairs of snowshoes since they do not have any experience with snow and thus no rational basis to make one choice over another. In fact, any problem solving strategy would be unsuccessful without knowledge relevant to the problem [92]. It is important to point out, however, that the knowledge needed to solve a problem through insight need not be obviously related to the problem. For example, a bridge construction engineer may be able to design the construction of a tower because of their more general expertise with, or “intuitive feel” about, the construction material (steel). On the other hand, there are situations when a person has to make a decision without having a sufficient amount of background knowledge. For example, managers frequently have to make decisions without deep understanding of the matter—either because it requires extensive studying (as in engineering problems) or because it involves a large number of disciplines (as in the design of a spacesuit). Is there a way to enhance the performance of “intuitive” problem-solving and increase the chances of getting the right insight? In [65], Osborn formalized a method for “creative problem-solving” which he calls *brainstorming*. It involves the participation of a group of people focused on solving a problem. People propose ideas in turns and the following guidelines are used:

- Criticism of ideas is suspended until the end.
- The wildest ideas are welcome.
- The more ideas proposed, the better.
- Combination and improvement of ideas is encouraged.

In essence, the method is designed to help with the reformulation of the problem in a way that the insight necessary for the solution is reached. The only drawback of this process is that it requires multiple participants. Thus, it is not applicable to solitary problem solving. On the other hand, it makes a strong case for interface designs that allow easy sharing of information and exchange of feedback.

The preceding discussions described problem-solving as a process that starts off by considering a problem and, after some mental effort—be it conscious or unconscious—produces a solution to the problem. Such observations can be made in settings where the problems are well defined and there are strict constraints on the solution to be obtained. Problems handed out in the laboratory of a psychologist or problems on a math exam in school are good examples of this kind. However, in real life people most frequently face problems which allow much greater freedom of interpretation and have more relaxed constraints on the solution. As described below, it has been observed that in many activities, people formulate the problem concurrently with solving it. In other words, the process consists of an iteration where the performance of activities leading to the solution results in a modification of the problem. Thus, the problem is dependent on the process of solving—and the problem solved at the end may not bear close resemblance to the initial problem. In looking for a house to buy [78], for example, the person may set off looking for a small house within a 5 km radius of the downtown core of a city but, after checking the price range, settle for a large house at the outskirts. The person may not even think about a big house in the beginning of the search. In the context of supervisory control of DESs, there is a similar situation that may occur. In [97], on page 43, Wood points out that in DES systems which are tightly coupled with the control objectives,

> . . . the modeling of the [system], legal specification and supervisor is a more arbitrary process where each affects the other's design.

This is especially pertinent to situations where the system is not yet built, or where the system is implemented using programmable circuitry. In the latter case, there is great freedom in designing the "unrestricted" behavior of the system, since a large part of the behavior is actually generated by the control circuitry. As well, some "uncontrollable" input may be ignored or the design altered so that it becomes controllable. For example, in a vending machine, the uncontrollable event of a person inserting a coin can be rendered controllable by equipping the machine with a switch that can block the path to the bank, thus flushing the coin to the coin exit. The design of a supervisor might necessitate changes to the system which, in turn, might necessitate changes to the control specifications and, in turn, require a redesign of the supervisor. At the end of the process, the supervisory solution might be for a system quite different from the original system. There are no known algorithmic rules for making the correct redesign choices, thus, the above iterative process cannot be automated.

The reader might at this point wonder: if people are solving vaguely defined problems or problems that need insight, what can be expected as the outcome of the problem-solving activity? Is it always the case that either people find the correct solution or they recognize their inability to successfully complete the task? Surely, the outcome may simply be a solution which is incorrect. In other words, the problem-solver makes an error. Some common patterns of erring are discussed next.

The most famous work on human factors in problem-solving is the work of Wason and Johnson-Laird on the so-called *selection task* [90]. The selection task can be described as follows. There are four cards where each card has a number on one side and a letter on the other. The cards are laid down on the table so that the sides that face the subject read, for example, “E, K, 4, 7”. The subject is given the rule: “If there is a vowel on one side, there is an even number on the other side”. Then, the task of the subject is to choose which cards have to be flipped over to verify the rule. Since the rule is a simple implication, according to mathematical logic one needs to flip the card showing “E” (to verify that there is an even number on the other side) *and* the card showing “7” (to verify that there is no vowel on the other side). Surprisingly, or maybe not surprisingly at all, only 4% of the subjects gave the correct answer. The vast majority, 79%, chose either only the card with "E" or the cards with "E" and "4". This experiment has also been given to a class of mathematics students and the results again show that the majority of subjects fail to give the correct answer. On the heels of the selection task comes further research which elaborates on the observed results [89]. It has been found that people have a *verification bias* in confirming hypotheses. When there is a hypothesis to be tested, people try to generate as many examples as possible that confirm the hypothesis, while little effort, if any, is directed at trying to find examples that disprove the hypothesis. In other words, given the hypothesis $A \rightarrow B$, people will generate many examples of $B$ and try to see if, indeed, they are valid. However, usually no examples of $\neg B$ will be generated to see if they are valid (thus, disproving the hypothesis). For example, in an experiment [89], subjects were asked to determine what rule is used to generate the sequence of numbers "2, 4, 6" (where the rule was "a sequence of increasing numbers"). The subjects could generate a different sequences of numbers and ask if they can be generated by the same rule. In order to test if the rule is "add two to the previous number", subjects would test sequences such as "11, 13, 15" (conform to the hypothesis) instead of sequences such as "11, 12, 13" (do not conform to the hypothesis). Twenty three of the twenty nine subjects did not manage to discover the rule with their first hypothesis. When testing their hypotheses, the mean ratio of non-conforming to conforming sequences was only 0.24, i.e., subjects generally did not attempt to disprove their hypotheses. The selection task and verification bias have stirred much debate and much research has been done on different aspects of the phenomenon. The theories proposed so far have not been conclusive, however, they give rise to some ideas that can be summarized as follows.

People do not use logical rules in thinking. The selection task and similar experiments demonstrate quite convincingly that people generally do not employ the rules of mathematical logic when problem-solving. This conclusion, however, still leaves many questions unanswered. For example, Legrenzi and Legrenzi [48] have modified the selection task in a minor (from the logical point of view) way to obtain a significant change in the performance of subjects. The task was rephrased to involve checking if envelopes have sufficient postage (again, involving the same simple implication rule) and suddenly almost all tested subjects gave the right answer. In an attempt to explain such performances, Cosmides [13] proposes a theory based on evolution. She claims that that human mind has adapted in a way that lets people detect cheaters—and which explains why people do not perform well on abstract tasks but are good at tasks that involve dealing with regulations and restrictions pertaining to real-life situations.

People build mental models when reasoning instead of using mathematical proof systems. This thesis has been advanced by Johnson-Laird *et al.* [45, 47] and is based on many observations of how people solve mathematical problems. Instead of using mathematically sound inferences which lead to the proof or disproof of an argument, people attempt to create a model that conforms to the argument. If they succeed, they continue building alternative models until they exhaust all possibilities and then they accept the argument. If at any point they fail to build a model, they reject the argument. The mental modeling thesis is supported by numerous experiments. For example, in Johnson-Laird *et al.* [47], subjects were asked to answer the following question. *There is a box in which there is a black marble, or a red marble, or both.*

*What is the probability that, upon opening the box, one would discover a black marble with or without another marble?* The vast majority of subjects answered 67%. (In fact, the given information is insufficient to solve the problem.) This indicates that a mental model of having three different scenarios is built (only black marble, only red marble, both black and red marbles). The model theory also explains other factors in human problem-solving. For example, general knowledge has a significant impact on human judgment. In [52], subjects were observed to reject the logically valid argument:

> Wars are prosperous.
>
> Prosperity is desirable.
>
> Thus, wars are desirable.

while accepting the logically invalid argument:

> All communists are radicals.
>
> All labor leaders are radicals.
>
> Thus, all labor leaders are communists.

If mental models are created, then they employ any knowledge that the individual might have and this knowledge may influence the problem-solving activity. An argument against the model theory is that, in some cases, it is impossible to exhaustively build all model variants either due to the nature of the problem or due to the large number of possibilities. Then, the theory predicts that people would get "stuck" creating models forever. The answer can be found in [77], where the authors argue that people use bounded rationality, a phenomenon termed *satisficing*. In other words, they continue building models until they feel they are confident enough in the answer (the solution is “good enough”). Since for different problems different people have different motivation, the performance of subjects may widely vary depending on the point at which they subjectively evaluate the solution as “good enough”.

How does the above discussion relate to the solving of DES problems? In our opinion, the major implication it has for the field is that researchers, while acquainted in detail with the mathematical theories behind control of DESs, most likely continue to build approximate models of the systems in their minds and their thinking is colored by any real-life knowledge they may have. In the observational study described in [32], we noticed that subjects usually started modeling the buffer in a factory system as a separate module of the system. This is consistent with the real-life notion of a buffer as a separate entity in a factory. However, theoretically, the prevention of overflow of a buffer is only a constraint (or a specification) that has to be met by the supervisor that controls the machines in the factory. Subjects were well aware of the fact that a buffer should be a part of the specifications through previous examples in their research career.

Another factor in human problem-solving was explored in a study by Luchins and Luchins [56]. In an experiment, subjects were asked to solve a series of simple “water jar” problems. These are problems where a set of hypothetical jars with certain volumes are given and the subject is asked to measure a precise amount of water by filling and emptying the jars as needed. In the particular series of problems given to the subjects, the solutions of the first few problems are the same (i.e., require the same sequence of pourings of water). Then, for the last couple of problems the same solution is not optimal or an altogether different sequence is required. The subjects consistently failed to see the optimal solutions for the last problems and in general had trouble solving them when they were preceded by the series of initial problems. Subjects did not experience these difficulties if the last problems were given first. This interference effect was termed *Einstellung* or the mechanization of thought. In other words, if one encounters the same solution a number of times, one has difficulty seeing alternatives.

We believe that, unfortunately, almost any field of research suffers from a form of the above phenomenon. Once a researcher becomes an expert in their area of research, they tend to see everything as an instance of an issue in their field of expertise. The same holds for research in the field of DES control. For example, we have observed that researchers trained in the modeling using FSAs tend to consider their approach “superior”, or at least more convenient, than an alternative approach using Petri nets. The converse is true for researchers trained in the use of Petri nets. However, it is likely that the most successful solutions will be the ones that take advantage of the results of a larger body of research rather than staying within the cast of the specific modeling paradigm. Brainstorming is a suitable tool to address this issue, since different researchers can bring different backgrounds to the table and the environment is conducive for an exchange of ideas. From our personal experience, a similar setting led to the rapid generation of the ideas discussed in [19].

As a conclusion to problem solving the following can be noted. The human cognitive processes involved in problem solving are not yet completely understood. Much of the research should be viewed only as propositions that offer only tentative explanations. There are a number of biases and other factors, such as prior knowledge, that influence the human performance in problem solving. People may decide to use short-cuts such as solution by analogy. Furthermore, problem solving is bounded by satisficing, i.e., reaching “good enough” solutions. The general activity of problem solving is not necessarily a straightforward process which starts with a well-defined problem and proceeds systematically towards the solution. Many times people use the process of solving to actually define the problem. All of the above apply also to problem solving in the area of DES control. As pointed out in places throughout this section, it is not possible to completely automate DES problem solving due to a variety of reasons. The iterative process of adjusting specifications and verifying the resulting model is not yet understood. The act of creating initial models for a DES is also not covered in current DES research and this activity is left entirely to the “designer’s intuition” [72]. Current software designed for DES researchers focuses exclusively on providing computational support for the generation of a solution (i.e., in generating supervisors). This alleviates a great chunk of the activities necessary to solve a DES control problem. However, there are other stages involved in DES problem solving: modeling the system, verifying the proposed solutions, modifying different constraints, etc. If software is to be designed to assist in problem solving, the human cognitive processes have to be understood and taken into consideration so that they can be augmented effectively.

## 2.3 User Interfaces

As was explored in the previous section, there are many factor that may determine the suitability of a particular machine interface for use by humans. Unfortunately, there are no detailed rules a designer can follow to deterministically obtain a better or worse interface. There are, however, ideas that are helpful during the design process. Some of them are discussed next.

In [83] and [64], the authors make an important point: the designer and the user of a tool typically are not one and the same person. It is crucial that the designer keeps in mind the fact that what seems to be the best solution to him or her may not be the best solution to the end user(s). One type of difference is in the general attitude or personality. A brief study done by Tognazzini [83] at a software company shows that, according to the Myers Briggs type indicator [62], software engineers are predominantly of the "intuitive" type. In other words, they are able to easily build abstract models of non-visible but well-defined interfaces (such as the command line), and prefer such interfaces if they provide greater flexibility and control. On the other hand, previous research shows that in the general population, the "sensory" type is predominant [62]. In other words, most people prefer to work with highly visible interfaces that allow direct manipulation of objects. Thus, interfaces for the general public have to be built using a different paradigm than what the software developers feel is most intuitive. The same information, however, points to the fact that in the context of DES control problem-solving, the users are expected to be predominantly of the intuitive type.

Even when the interface designer chooses the appropriate interface paradigm, there are many variations of the design that may improve or hinder the success of the product. The usability of a product is the actual performance that can be achieved by the user. This is different from functionality since some of the functionality may be unusable due to problematic design. As a classic example, most VCR models have a clock but the vast majority of VCR owners do not use it since the procedure to set it is too complex. In classical books on usability, e.g. [64], and newer books geared toward software interfaces, e.g. [18], the authors point out that there are no particular methodologies of design that guarantee that the product will be successful in terms of usability. The two (complementary) approaches recommended are

- follow general guidelines for successful design and
- perform user testing.

Some of the general guidelines for usable design are summarized in [18, 64, 83]:

- The design should be learnable. The designer has to choose a paradigm, a guiding concept, and use it to create a consistent design. The design has to be predictable so that the user can come up with a reliable mental model of the product. Instrumental in this is also taking advantage of any previous knowledge the user might have so that fewer operations or relationships have to be learned. For example, if there is a vertical slider to control the temperature setting of a thermostat, it is desirable to design it so that "cool" is at the bottom and "warm" is at the top. This would be consistent with another common tool, the mercury thermometer, which traditionally has this design.
- The interface should be flexible. As already noted in previous sections, everyone has a different "cognitive approach". While good designs work well for most users, customization might improve usability for each individual user. This is especially relevant to software interfaces since customizable designs are relatively easier to implement for computers than for other tools.
- The interface should be robust. It is human to make mistakes; as Norman puts it in [64],

  > If an error is possible, someone will make it.

> Unfortunately, not all designs are created with the above maxim in mind. The design has to be simple, and the state of the system and the available actions need to be visible to the user. The performance of operations needs to be prompt. Adequate feedback has to be provided. The user needs to be able to discriminate between desirable and undesirable states. All of the above will help in minimizing the probability of an error occurring. Nevertheless, errors may eventually occur. Thus, the design should afford corrective actions whenever possible. In computer interfaces, the "undo" command is essential.

Unfortunately, the above guidelines offer no guarantee that the resulting product will be usable—to the intended users. It is very hard, if not impossible, to escape any subjective judgment when a product is being designed. The simple solution, points out Tognazzini [83], is to perform user testing. In an anecdotal recollection, the author describes how the design team for a piece of software needed to rephrase six times a simple question in the interface before they started getting consistently correct responses from the users. Such issues can be discovered only during user testing since the designers are well aware of their own design and thus will not run into confusion. Furthermore, the designers must witness the user testing:

> ...People must see their users in action... Any attempt...to verbalize the results of testing will automatically filter away intuitive information important to design team members. [83]

Another reason for the importance of user testing is the fact that when a new product is developed, the process is gradual and spreads over a long period of time. The developers and designers have time to accommodate to any new aspect of the design.

However, the end users are faced with the complete product all at once. They may not be prepared to handle all aspects in the way the more experienced developers do.

The user interface design principles are derived from practice and common sense more than from rigorous research. However, they form a good complement to the results from (cognitive) psychology research (e.g., from Section 2.2) when developing a product to be used by humans, such as a system that will assist in the solution of DES control problems.

# Chapter 3

# Recommendations for Improvement of DES Software

The improvements of DES software proposed in this chapter and the development of the template design methodology described in Chapter 4 were motivated greatly by our investigation of how people solve DES problems. The complete report of our investigation of DES problem solving, which included an observational study, can be found in [32]. Here we will provide a brief overview of the study and summarize some of the observations which motivated our work. Then we will present a list of specific changes which were recommended for implementation in the IDES software package [73] and discuss the implementation of a subset of these changes.

## 3.1 Motivation

### 3.1.1 Study of DES problem solving

As a precursor to the work in this dissertation, we conducted an exploratory observational study to acquire a deeper insight into how people familiar with DES supervisory theory solve DES problems [32]. The study was set up so that we could observe and record the performance of subjects.

We recruited five subjects in total. Each subject had at least one university semester of exposure to supervisory control theory for DESs. Each subject was administered two problems in the field of DES control. One of the problems is an adaptation of a classic problem in the field, the “Transfer Line” [94]. The problem asks for the design of a controller for a system of factory machines and buffers between them. Each subject had seen this problem being solved in a university course. 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. 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 [96]) was recommended. Furthermore, subjects were provided with a pen, sufficient amount of paper, and a computer running version 2 of the IDES software developed at Karen Rudie’s research laboratory at the Department of Electrical and Computer Engineering, Queen’s University, Canada [43]. 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.

The subjects in the study were asked to *think aloud* while solving the problems in order to collect data for a subsequent protocol analysis (as per the recommendations in [22]). 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.

In addition to conducting observational sessions with subjects, 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 many years.

### 3.1.2 Discussion

#### Unconstrained design

We believe that the design activity when solving a DES problem should not be restricted to a prescribed order or constrained by requirements for model consistency. This belief is substantiated not only by the findings of other authors; e.g., in [97] Wood explains that the process of modelling a DES solution may be arbitrary due to mutual influence of the components of the solution. Our direct observations during the study of problem solving provide many examples of the diversity of the process, even with a small number of subjects. The participants in the observational study used varying approaches in solving the given problems. We noticed a variety not only at the low level (e.g., in the way subjects draw models), but also at the high level of problem-solving strategies. Of the five subjects, only one used the approach recommended by experts, when solving the problem the subject was acquainted with (the factory problem). When solving a novel problem (the hospital problem) subjects in general used different strategies as, in our opinion, the prescribed strategy is not suitable for working with ill-structured problems where it is not immediately clear what the sub-components of the problem are and how these sub-components interact. One of the subjects created numerous versions of their models while experimenting with the interaction between them—most of the time working with formally inconsistent models. Conversely, another subject preferred to "get it right" in the first try, by studying the problem description longer and by constructing their models slowly and carefully. However, at the end of the session, both subjects had advanced comparably. In this sense, we feel that no specific approach can be identified as advantageous.

Most subjects attempted to construct supervisory solutions manually, even though they had access to software which can synthesize supervisors that are correct and optimal. According to us, there are two likely reasons for choosing to work manually. First, manual construction does not require formal models of the control specifications; these are required if the algorithm is to be used. Second, the verification of the correctness of the manually constructed supervisor is simpler since the subjects participate in its construction (as opposed to receiving an automatically synthesized model). The drawback to the manual construction is that, in the general case, it is hard to find the optimal supervisory solution. Indeed, during modelling, subjects frequently verified the correctness of their solution, however, they almost never examined the optimality, i.e., if the supervisor is as permissive as possible. Psychologically, this can be explained with the reluctance of humans to explore reverse implications (required to check optimality), as noted by Wason and Johnson-Laird in [90]. We think that, overall, the use of the algorithm for automatic synthesis of supervisory solutions should be strongly encouraged, however, DES software should not prevent the use of manually constructed supervisors. In some cases, such supervisors may be preferred by the users of the software. Especially in industry, as interviewed DES experts remark, there is more interest in the verification of manual solutions and less in the synthesis of solutions.

### Modularity and hierarchy

In our opinion, DES problem solving is essentially modular. Research so far has focused on the theoretical foundations of modular DESs for the purpose of reducing the computational complexity. However, it can be argued that for the same purpose people prefer solving DES problems in a modular way. That is, relatively independent components are identified and modelled separately. In the study on problem solving, we observed that all subjects used this approach, including the subject with the least sound overall strategy. On one occasion, one of the subjects started modelling the factory problem as a single monolithic solution, however, they soon decided to remodel everything in a modular way to handle the growing complexity.

The modular approach to problem solving results in a number of separate components, each one much simpler and better understood compared to a monolithic solution. However, these components are not entirely independent and the separate models do not necessarily make the patterns of interaction between the components obvious. It seems that a separate model is needed in order to explain these interactions. Indeed, in the study we observed that subjects created high-level diagrams to better understand the relationships between components, especially in the (more difficult) hospital problem. See Fig. 3.1 for examples of the diagrams. These separate diagrams seem to serve as an overview of the complete solution. In a sense, a hierarchical model is created where the individual modular components serve as the low level of the high-level diagram. While the subjects in the study used the diagrams only to help themselves in clarifying the solution requirements, we see no reason why such diagrams cannot be used in a formal way. The use of high-level conceptual modelling is one of the main ingredients of the methodology proposed in Chapter 4.

### Automation and replication

Discrete-event systems are, inherently, unforgiving to errors. There is no graceful degradation of performance with faulty models; even a simple mistake can render the whole solution incorrect. This was observed on a number of occasions during the study of DES problem solving. Notably, in one case the incorrect choice of the controllability of a single event reduced the solution to the hospital problem to an (incorrect) triviality. Due to the complexity of DES systems, it may be hard, if at all possible, to spot the presence of an error. In the observational study, it was rare that subjects recognized they had made a mistake. Thus, it is essential that DES software reduces the opportunities for making errors during modelling.

According to us, based on the observations from the study of problem solving, there is the need for greater automation of the mundane and error-prone modelling of synchronization between DES modules. The synchronization of modules via common

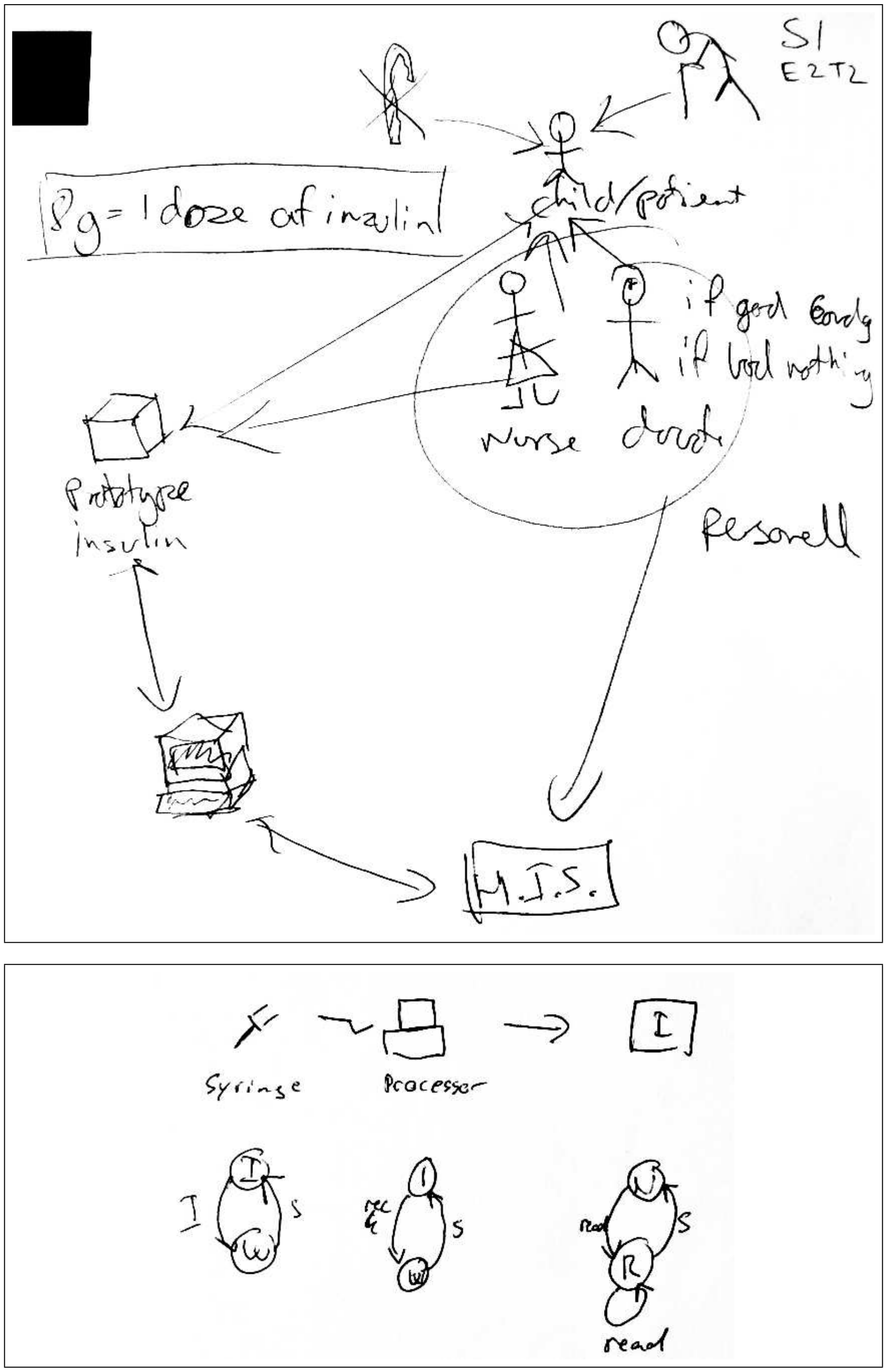

[redacted]

Figure 3.1: Reproductions of some of the diagrams created by subjects when solving the hospital problem.

events proved to be one of the most challenging tasks. First, in order to synchronize two modules, the common events have to be named in exactly the same way. Setting aside the possibility of spelling errors, it is cognitively demanding to keep track of all the events in a big system. Furthermore, it may be necessary to introduce artificial modifications to event names in order to avoid synchronization where it is not desired. For example, if the models of two robots have a “start” event, it is necessary to name the events for example “start1” and “start2” to avoid the synchronous start of the robots. In the observational study, some subjects had difficulty remembering the exact labels for all events in the system and frequently had to refer to the events in different parts of the model. Second, according to the framework for supervisory control proposed by Ramadge and Wonham (described for example in [94]), the models of specifications need to list explicitly for each state the events enabled at that state. In practice, this frequently leads to the proliferation of self-looped events at all states. Not only do models become cumbersome and hard to read but also there are plenty of additional opportunities to make errors when modelling, such as events omitted in the self-loops or incorrectly self-looped events. In the observational study, subjects often decided against modelling the specifications (and manually constructing the supervisors instead) or, if they were modelling the specifications, they created informal models where the self-loops were not specified. The self-looping of events need not present a difficulty, however. It can be completely automated by software, and performed only at the time of need, i.e., before the computation of the supervisory solution. The synchronization of events can also be simplified if a mapping between events is used instead of direct name equivalence. In such a case, the events in different modules will be independent (even if they share the same name). Synchronization will be achieved via a separate structure, a mapping between the synchronized events. We believe this will lead to clearer models, reduce the likelihood of chance synchronization, and, most importantly, allow the easy substitution and reuse of models by keeping the event names independent across components.

Finally, the opportunities for making errors can be greatly reduced if there is no need to create models. Of course, it is not possible to avoid modelling completely, however, in practice, often the same or similar models are used for different components of a modular design. For example, on a factory floor there can be independent workstations that take in parts for processing and output the parts when the processing is finished. If the workstations operate independently, it is not necessary to know what specific actions they perform. For the purposes of flow control of parts, the model for each workstation can be the same, a simple alternation of "start" and "finish" events. In our experience, the majority of published DES problems contain parts which are modelled in a similar way, i.e., their models are equivalent or contain minor modifications. In the study of problem solving this was also observed, especially in the factory problem. Since the structure of different components might be the same, it should not be necessary to re-create the same models over and over. Instead, a mechanism for copying should be available to replicate desired models, even across projects. This observation serves as the motivation for the second main ingredient of the methodology proposed in Chapter 4, the availability of model templates.

## 3.2 Recommendations for IDES

In this section, we summarize a list of recommendations for changes or new features to be introduced in the next generation of IDES. The list not only includes recommendations inspired directly from the study on problem solving, but also recommendations inspired by conversations with experts and DES students, and by observations during the work on Template Design at the Department for Automation and Systems, Federal University of Santa Catarina, Brazil [31].

For each recommendation, we specify which part of DES problem solving will benefit most. We consider the following categories, based on the problem-solving taxonomy discussed in [35, 32] and on understanding of general problem solving:

**Understand** Support for gaining understanding of the problem and the existing situation.

**Model** Support for the creation of a representation of the problem solver's understanding of the problem and for the proposal of a solution.

- **Granularity** Support for the use of representations at different levels of abstraction.
- **Analogy** Support for the use of analogy in problem solving.
- **Robustness** Support for the creation of robust solutions.
- **Speed** Support for faster problem solving.
- **Other** Other modelling support.

**Verify** Support for the verification that the solution is correct.

- **Visualization** Support for the visualization of the solution.

**Other** Other verification support.

In addition to the above general categories, we introduce a new one, geared specifically to the implementation in software:

**Usability** Support for a functional, efficient, effective and pleasant process of problem solving while using the software.

In Table 3.1, the specific recommendations are listed. More detailed descriptions of the recommendations follow next.

| | | Model | | | | | Verify | | |
|---|---|---|---|---|---|---|---|---|---|
| **Recommendation** | Understand | Granularity | Analogy | Robustness | Speed | Other | Visualize | Other | Usability |
| Online access to problem description | ● | | | | | | | ● | |
| DES theory reference | ● | | ● | ● | | | | ● | |
| Conceptual design | ● | ● | | ● | | | ● | | |
| Templates | | ● | ● | ● | ● | | | | |
| Model/environment interaction | | | | | | ● | | | |
| Copy models | | | ● | ● | ● | | | | |
| Copy events | | | ● | ● | ● | | | | |
| Display event lists | | | ● | ● | | | | | |
| Event annotation | | | | ● | | | | | ● |
| Model annotation | | | | | | ● | | | ● |
| Wizards | | ● | ● | ● | ● | | | | |
| Organize models in groups | | | | | ● | ● | | | |
| History of models (checkpoints) | | | ● | | | | | ● | |
| Derivation of models (auto-update) | | | ● | ● | ● | | | | |
| History of commands/actions | | | ● | | | | | ● | |
| Text entry of commands (command line) | | | | | ● | | | | |
| Command scripts | | | ● | ● | ● | | | | |
| Unbounded inputs for algorithms | | | | | ● | | | | |
| Deterministic layout | | | ● | | | | ● | ● | |
| Multiple layout algorithms | | | | | | | ● | | |
| State-browsing for large models | | | | | | | ● | ● | |
| Implicit/explicit display of control decisions | | | | | | | ● | ● | |
| Event sequence tracing | | | | | | | | ● | |
| Simulation | | | | | | | | ● | |
| Verification of manually created supervisors | | | | | | | | ● | |
| Auto-check for initial/marked states | | | | ● | | | | | |
| Auto-check for controllability of supervisors | | | | ● | | ● | | | |
| Split/merge supervisors | | | ● | | ● | ● | | | |
| Implicit specifications | | | | ● | ● | | ● | ● | |
| Specifications as inequalities | | ● | | | ● | | | | |
| Glance at models | | | ● | | ● | ● | | ● | |
| Import/export to common formats | | | | | | ● | | | ● |
| PLC export | | | | | | | | | ● |

Table 3.1: Recommendations for changes or new features in IDES.

**Online access to problem description** A feature which will provide quick access to the (informal) description of the problem from within the software. This is especially important when the description is available only in electronic form. This recommendation is similar to the recommendation in [38] to provide access to domain knowledge throughout the design process.

**DES theory reference** The inclusion of a reference for DES theory with the software package. This document should include the main theoretical results, as well as descriptions of the available DES algorithms and a guide about their usage.

**Conceptual design** The software should support the creation of conceptual designs of the control solution, such as the display of a high-level abstraction of the design elements. This feature will help users attain an overview of their solution which may not be possible otherwise, e.g., when many low-level components are used. Conceptual designs may offer a big advantage over paper diagrams. Depending on their implementation, such designs can be updated automatically to reflect the current low-level designs. Furthermore, conceptual designs may allow for faster updates of the model and thus increase the flexibility of designers to explore different design variations. This recommendation is partially related to the recommendation in [38] to allow the exploration of alternative solutions.

**Templates** A repository of common DES modules which can be "instantiated" (duplicated) when needed. This is especially valuable when the design of controllers for similar systems is done, or when a system consists of a number of identical components (e.g., an assembly line populated with identical robots). The availability of reusable templates will result in a reduction of the errors made during DES design as each template can be verified before inclusion in the repository. Duplicates then will have the same degree of correctness as the original template. This recommendation is similar to the recommendation in [38] to provide a library of design schemata or pieces of expert advice.

**Model/environment interaction** Support for the modelling of the interaction between the system and the environment. In all classical DES system models, this is already done implicitly with the assumption that the environmental forces regulate the occurrence of events, i.e., the occurrences of events are spontaneous. However, sometimes it may be useful to distinguish between causal forces internal to the system (e.g., what causes a workstation to finish processing a part after it has started processing it) and causal forces external to the system (e.g., what causes the demand for an additional batch of parts to be produced). The ability to model such "external" interactions explicitly will help in identifying the inputs and outputs of a system and allow for a more "natural" way of modelling.

**Copy models** Support for the duplication of DES models. This could take the form of a copy/paste operation.

**Copy events** Support for the duplication of events across models. This could take the form of a copy/paste operation. The duplication of events is especially important when different modules need to be synchronized via events. Exact copies of events are needed in such cases, and manual copying is prone to errors.

**Display event lists** The display of an aggregation of all events in a project, across all models. The ability to see all events can help in the recognition of undesired event synchronization, or of incorrect event naming. Furthermore, combined with an event duplication facility, it will be possible to duplicate events from different modules at once. This is useful in the modelling of control specifications as specifications may need to be synchronized with a number of modules.

**Event annotation** Support for the annotation of events. This will not only help with the sharing of information when models are exchanged between different parties. When it is necessary to use short (or coded) event names, it will be possible to annotate events with a full description.

**Model annotation** Support for the annotation of models. Annotations can be used to store remarks about a model for use at a later time, or to help share information when models are exchanged between different parties.

**Wizards** A feature which will guide users in solving typical DES control problems, similar to the wizards available in other software. In DES, most problems vary in terms of the specifics of the model, however, the structure of the solution—the sequence of operations applied—remains the same. Thus, wizards for typical problem structures can be designed, where the user will only need to specify which models have to be used to compute a control solution.

**Organize models in groups** Support for the custom grouping of models. Some DES software, including IDES, allows the simultaneous work on a number of models (e.g., through a "project"). However, the user should be able to organize the models in a project into smaller subgroups according to their mental model of the system. Some users may wish to group system modules and control specifications separately, while others may prefer to form groups according to the relevant subsystems. Support for custom grouping need not be complex. It may be as simple as support for custom ordering of the loaded models.

**History of models (checkpoints)** The software should maintain the history of changes in each model, e.g., through checkpoints. It should be easy to browse the records. For example, each model may have an associated time-line which allows the user to view a given model at a selected interval in the past. Being able to view a previous version of a model may help in problem solving, e.g., by supporting *comparative statics* [80]. Subjects in the observational study frequently referred to previous versions. The model history may simultaneously serve as an undo/redo mechanism. An advanced version of the same facility may be implemented to maintain a history of the whole workspace/project—to allow for review of past work, synchronized among all models, and to offer access to models which may have been deleted.

**Derivation of models (auto-update)** For models which have not been designed manually, the software should keep track of how they have been computed, i.e., which algorithm was used and which were the source models. This information should be available to the user as a reminder. Furthermore, the software may use the information to update (recompute) models automatically when the source models change. Such updates can be carried out recursively when the user makes a change, starting with the models which depend on the manually designed models.

**History of commands/actions** The software should maintain a record of the history of all commands or DES operations the user has invoked. This history can be used by users as a reminder or as a repository which can be examined *post factum* for problem solving strategies. The history can be integrated with other facilities such as the auto-update feature for computed models or the creation of command scripts. In some sense, this recommendation will provide some of the support recommended in [38] to document design decisions in the transition from informal to formal models.

**Text entry of commands (command line)** Support for the entry of text commands, e.g., a command line, within the software. The users will be able to type the names of the algorithms they want to invoke and specify their parameters. In some cases this may be faster than using the graphical interface, particularly for expert users. This feature will be especially useful for the automation of activities, or for sharing problem-solving know-how between parties. If nesting of commands is supported, there may be further gains in the user productivity as it will not be necessary to deal with intermediate models.

**Command scripts** The facility to allow repeated invocation of a selected sequence of commands/operations, i.e., a script. In conjunction with the text entry of commands, this will offer a tool for the easy encapsulation and reuse of problem-solving know-how. Parametrized scripts will make such know-how simple to apply in a variety of situations. Scripts may serve additionally as a mechanism for the creation of custom wizards.

**Unbounded inputs for algorithms** Support for the selection of an unbounded number of inputs to relevant (commutative and associative) operations. Examples of operations which can be extended to support an unbounded number of inputs are *intersection* and *parallel composition*. The speed of problem solving will increase as the user will no longer have to create intermediate models in order to apply such operations to more than two inputs.

**Deterministic layout** Implementation of an algorithm for the deterministic layout of finite-state automata (FSAs). Such an algorithm will produce identical layouts for identical FSAs. This will provide an important advantage in verifying DES solutions, specifically by facilitating the recognition of (dis)similarity between software-generated FSAs. By extension, a built-in minimization of FSAs before layout will allow the comparison of the *behavior* represented by such automata—as all automata will be laid out in their canonical form. It is important to remember, however, that minimization should be optional. In certain situations, when solving DES problems, the structure of an FSA may provide the problem solvers with more clues than the minimized version of the same FSA.

**Multiple layout algorithms** Implementation of a number of different layout algorithms for FSAs. Different layout algorithms may emphasize visually different features of an FSA structure. Thus, users will have a wider selection of tools to help them make sense of software-generated models.

**State-browsing for large models** A facility to examine the structure of very large FSA models where a graphical visualization is impractical or infeasible. A standard approach involves the listing of all states and transitions of a model in a table. State browsing is an extension of this idea, where the user is able to select transitions leading into, or out of, states to explore the states these transitions come from, or lead to, respectively. In essence, this interaction style is similar to browsing web pages with links.

**Implicit/explicit display of control decisions** Support for the display of control decisions in supervisors (i.e., the enablement or disablement of events) in either implicit or explicit form. The implicit form is when, at each state of the supervisor, the events on the transitions that lead out of the state give the set of all enabled events at this state. This is the visualization supported by most DES software; it does not require any features beyond the ability to display FSAs. The explicit form is when each state of the supervisor is augmented with a list of all events which are disabled at the state. The user should be able to switch between the two modes of display.

**Event sequence tracing** A facility enabling the tracing of event sequences in a model. It should be able to answer questions such as "Does a given event sequence belong to the language generated by the model?" or "Starting at state $q$, is it possible to generate the given event sequence?" Further improvements include highlighting in the graphical model the states and transitions through which an event sequence passes.

**Simulation** A feature which allows the simulation of the performance of a (possibly controlled) DES system, similar to what existing methods of discrete-event simulation offer [3]. This should include both an event generator module where events to drive the simulation are generated and an analysis module where the performance of the system is analysed—e.g., number of occurrences of a given event, proportion of occurrences of a given substring, number of times a state was reached, etc. The feature should allow repeated simulation, as well as provide a way to specify conditions for the termination of a simulation run. This recommendation is similar to the recommendation in [38] to allow simulation of complex solutions.

**Verification of manually created supervisors** The software should provide comprehensive support for the verification of manually created supervisors. This should include automatic checks such as language containment, controllability or checking if a number of supervisors are *conflicting*. However, checks for trivial errors should be included as well, such as checking if the event sets of the supervisors and the controlled systems are equal (e.g., to uncover typos).

**Auto-check for initial/marked states** A feature which will verify the correctness of FSA models online, i.e., while the user designs the model. The absence of initial and marked states is a very common error in modelling and is very easy to diagnose. The indication of such an error should be very judicious so as not to interfere with the design process. For example, there can be a flag in the interface which will be raised until the user creates initial and marked states. Alternatively, the correctness of models can be checked just before they are used in DES operations. This recommendation is similar to the recommendation in [38] to perform automatic heuristic evaluation of models during the process of problem solving, in order to provide the user with helpful hints.

**Auto-check for controllability of supervisors** A feature where the controllability of computed or manually created supervisors will be checked automatically. It is a feature which will automate the most commonly applied test during the verification of supervisory solutions.

**Split/merge supervisors** A facility to allow the segregation or aggregation of control responsibilities between supervisors as needed. The splitting of supervisors into separate modular supervisors will support problem solving strategies which rely on the breaking down of hard problems. The merging of selected supervisors will speed up the problem solving process when modular supervisors are in *conflict*; such conflicts can be resolved by combining the conflicting supervisors into a monolithic supervisor. The implementation of supervisor merging is trivial, as it is only necessary to intersect the control strategies of the supervisors. More research needs to be done on how it makes sense to split supervisors. Such an operation will in all likelihood require intervention from the users.

**Implicit specifications** The software should support the use of implicit control specifications alongside with explicit specifications (see [32] for more details). mplicit specifications are specifications where the events irrelevant to the given specification can be omitted (instead of having to be "self-looped" in each state). Such specifications are easier to read by humans; however, they require the explicit definition of all events which participate in the model. Implicit and explicit specifications have equivalent expressive power and it is easy to compute one from the other. Thus, the software should allow the users to switch between the two views as needed.

**Specifications as inequalities** Support for the entry of control specifications in the form of inequalities, rather than FSA models. Many specifications are much more convenient to express as inequalities in terms of the count of events or substrings. For example, one may have a specification which says that there should not be more than ten cars simultaneously on a section of a bridge. Such a feature will require the implementation of a facility for the conversion of inequalities into FSAs. This, in itself, will necessitate that a number of restrictions be placed on the form of the inequalities—otherwise such a conversion will be infeasible.

**Glance at models** A feature of the software interface where the user will be able to examine loaded models without fully activating them for editing. This could be implemented as a response to a hover of the mouse cursor over a non-active model name or icon.

**Import/export to common formats** The software should support the import from and export to common file formats. This will enable the interoperability of the software and allow easier exchange of models between parties. Furthermore, it will enhance the appeal of the product since users will be able to use seamlessly other software for operations and features which are missing from this product.

**PLC export** A facility to export abstract supervisory control decisions into Programmable Logic Controller (PLC) code. There are international standards for PLC code, thus such code will be usable for a wide variety of PLCs. Research has been done on how to implement such a facility and shows that it is indeed feasible [17, 4, 1].

Table 3.1 shows which aspects of problem solving will be positively affected by the implementation of the listed recommendations. However, it is important to consider that some of these recommendations may also have a negative impact on problem solving. Notably, the automation of modelling or supervisor generation may result in models and solutions which are non-transparent and not well understood by the problem solvers. Recommendations which fall in this category include: templates, wizards, automatic updating of models and command scripts. All of the above take control away from the problem solver—which in cases may lead to confusing or unexpected results during problem solving.

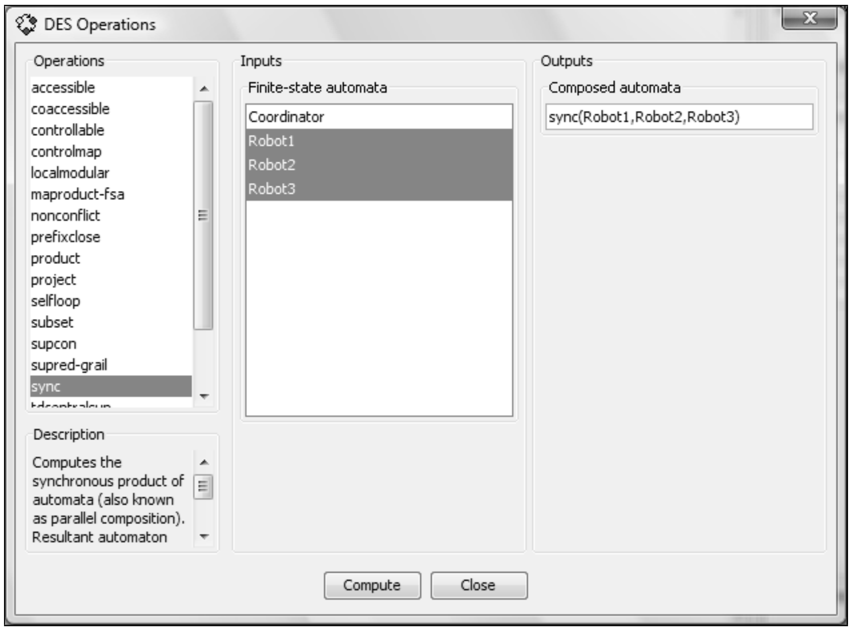
## 3.3 Implementation

The work on the most recent major revision of IDES, version 3, focused on two major aspects: implementation of the infrastructure necessary for the development of plugins (i.e., the API), and the development of a plugin for conceptual and template design of DESs. The theory behind template design and the specifics of its implementation are discussed in the following two chapters. Here it is worth noting that the template design methodology encompasses or supersedes some of the recommendations listed in Table 3.1. Template design is described in Chapter 4.

The recommendations which fall outside the scope of the template design methodology did not have a high priority in the development process. Thus, only a small subset was implemented. The rest of the recommendations will be included in future releases gradually, when project resources allow.

**Model annotation** The annotation of models was implemented in a way which not only supports the custom annotation of models by the user, but also introduces some aspects of the recommendation for information on the *derivation of models*. There is a text-entry area associated with each model loaded in IDES. The users can type arbitrary text to annotate the given model. Models which are generated by the invocation of a DES operation are automatically annotated with the description of the operation and its input arguments. For example, the result of the *product* operation invoked for the models "Robot1" and "Robot2" will be annotated with "product(Robot1,Robot2): Composed automata". This automatic annotations will carry over information about how a given model was computed and users can refer to it if they forget or get confused about the history of a model. Furthermore, automatic annotations can be removed or replaced if the user so desires.

**Unbounded inputs for algorithms** The interface for the invocation of DES operations was extended to support an unbounded number of inputs for operations which support it. First, the relevant operations were identified. These include *intersection* (or *product*) of FSAs, *synchronous product* (or *parallel composition*) of FSAs, *local modularity* check for FSAs, and *multi-agent product* of FSAs. All of these operations are commutative and associative, with the exception of *local modularity* and *multi-agent product* which cannot be performed incrementally, i.e., they are not associative. The implementation of the operations was modified to allow an unbounded number of inputs (a list of inputs whose size is determined at runtime). Furthermore, these operations now include a flag which announces that unbounded inputs are accepted. The interface was then modified to display, upon the choice of such an operation, a list with all compatible inputs (i.e., FSAs) available in the workspace. The user is then able to make selection of as many of these inputs as desired. The new interface



Operations: accessible, coaccessible, controllable, controlmap, localmodular, maproduct-fsa, nonconflict, prefixclose, product, project, selfloop, subset, supcon, supred-grail, sync

Description: Computes the synchronous product of automata (also known as parallel composition). Resultant automaton

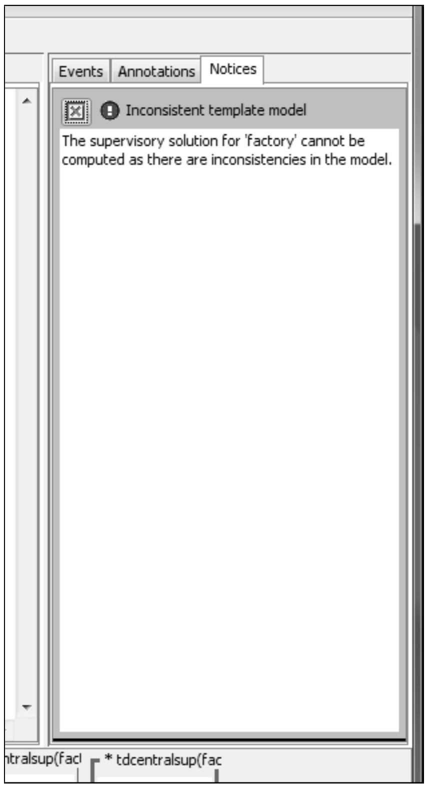
Finite-state automata: Coordinator, Robot1, Robot2, Robot3

Composed automata: sync(Robot1,Robot2,Robot3)

Figure 3.2: The user interface in IDES 3 for the selection of an unbounded number of inputs for operations which support it. In this case, three inputs are selected for the *synchronous product* operation.

is shown in Fig. 3.2. With this change, users no longer need to perform such operations incrementally, e.g., by performing *intersection* three times to intersect four FSAs. Furthermore, in the case of *local modularity* and *multi-agent product*, it is now possible to perform the operations on more than two inputs.

**Auto-check for initial/marked states** The automatic checking of model correctness was partially implemented. More specifically, the infrastructure for such auto-checking was built. Subroutines in the software now have access to a "notice board" where messages may be posted and removed as needed. The notice board occupies a part of the main interface (as shown in Fig. 3.3), however, the user may choose to use this space for the display of other relevant information. The posting



Inconsistent template model

The supervisory solution for 'factory' cannot be computed as there are inconsistencies in the model.

Figure 3.3: The tab with notices.

of new messages is announced through a small pop-up window which does not interfere with the main activity of the user (as shown in Fig. 3.4). The pop-up window disappears automatically after a preset period. As well, clicking on the pop-up window brings the notice board to the foreground to allow the inspection of the new messages. This approach to notifying users seems to be unobtrusive during the use of IDES. However, further research needs to be done on whether it will be effective for auto-check messages. A similar notice board is used by the Temlate Design plugin discussed in Section 5.3.2 to display any consistency issues with template design models.

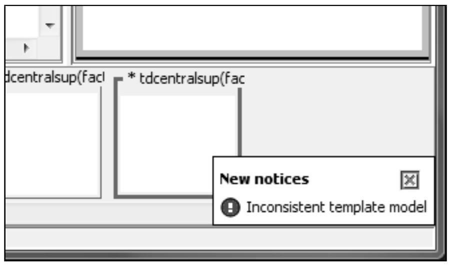


Figure 3.4: The pop-up window in the lower-right corner of the main window, notifying of new notices.

**Import/export to common formats** The IO module of IDES was redesigned to allow the introduction of plugins for the import and export of custom file formats. All internal import and export filters were re-implemented as (built-in) plugins. With the newly introduced filters, IDES is capable of importing models from the *TCT* and *Grail+* software packages, and of exporting to the *TCT*, *Grail+*, *LaTeX*, *EPS*, *JPEG* and *PNG* file formats. The *TCT* and *Grail+* packages are used in many research laboratories in Canada and Brazil, [14, 28]. The export to a variety of graphics formats simplifies the inclusion of models in publications.

**PLC export** Export of supervisors to PLC code was implemented as a proof-of-concept feature in an internal release of IDES (described in Section 5.2). This functionality relies on the *BAJ* experimental library developed by Francisco da Silva at the Federal University of Santa Catarina, Florianopolis, Santa Catarina, Brazil. As the future of the *BAJ* library is unclear, it was decided that the introduction of this feature in a public release of IDES will be withheld.

Preliminary work on a number of other features based on the recommendations from [32] has also been done. However, we hope that our main focus, the development of the infrastructure for plugins, will bear the most fruit as it will allow the independent implementation of the recommended improvements. An example of such work is the development of a plugin for conceptual and template design of DESs.

# Chapter 4

# Template Design Methodology

In this chapter, we propose a new methodology for the design of DES control which we termed *template design*. The reader can also refer to [33, 34]. The methodology is strongly motivated by the observations on solving DES problems and by the recommendations for DES software in Chapter 3. Our ideas are also inspired by concepts proposed by other researchers, e.g., in [74, 75, 20].

Here we provide a theoretical description of the template design of DESs. This description must not, however, overshadow the key reason why the methodology was developed—that is, it was conceived in order to make the application of DES control simpler. Our goals include making the design of systems faster, helping produce more robust designs, and automating repetitive tasks.

## 4.1 Preliminaries

The theoretical framework proposed by Ramadge and Wonham [67] allows the modelling of system behavior as a set of sequences of discrete events. Practical implementations of this theory, however, have run into a number of problems. The most significant problem is what is called “state-space explosion”. The state complexity of a system model may grow exponentially with the number of participating subsystems. Another problem for the use of the theory in practice is the fact that modeling a system and verifying the end result are difficult and non-transparent for the users. Further complications arise from the fact that the usability of software packages for DES control is generally unsatisfactory and that generally there is little support for the use of a computed supervisor in the control of a real system.

While there does not seem to be an easy solution to this complex set of issues, the use of predefined DES units by engineers may lead to a much easier application of supervisory control. In [20], the authors describe an approach where the controlled behavior of a discrete-event system is designed using a set of very simple specifications. Each specification is built from a prototype structure, a *template*, and exercises control over a single aspect of the system—such as the operation of a gripper. All specifications are executed in parallel and thus, simultaneously, provide control for the whole system. The benefits pointed out by the authors include significant reduction of the time needed to design controllers, (e.g., one hour versus 12 hours), lower cost of the project (the approach encourages the substitution of software complexity with cheap hardware sensors) and more robust handling of failures (no need for complex reset procedures). However, this approach also has some disadvantages. It is assumed that almost all system behavior can be described as the concurrent execution of simple units without much interaction. This is not suitable for the definition of global specifications, such as the control for nonblocking. The suggested templates seem too simple to express more complex requirements. Furthermore, the methodology is not cast within the supervisory control framework and it cannot take advantage of the algorithms therein. The main contribution of the research in [20], in our opinion, is the demonstration of the use of templates in the design of discrete-event controllers. The same idea plays a central role in the methodology we propose later in this chapter.

The work of Holloway *et al.* on condition systems, e.g., [41], also promises to alleviate much of the burden of designing controllers. Condition systems in this framework are modelled as Petri nets where the firing of transitions happens if specified conditions are met (similar to input signals) and the marking of states defines what conditions are satisfied (similar to output signals). A system is described as the composition of simple and independent condition models. Control specifications are also given as such a condition model, defining the start marking and the desired end marking of the Petri net. Then, synthesis algorithms exist to produce automatically the required *task blocks* (condition models to drive the evolution of the system), and to transform the specification model into a form which will ensure the correct control of the underlying system. Under this approach it is possible to reuse the independent condition models for the system behavior; and a library of the frequently used models can be maintained. It is also possible to combine task blocks hierarchically so as to accomplish more complex tasks. Furthermore, the synthesis of the controller is fully automated and, at the end of the process, code may be generated in C++ to control real hardware. A software tool with a graphical interface is available for the design and verification of condition models [76]. Among the disadvantages of the condition systems approach is the fact that there are too many restrictions on the class of systems which can be feasibly and effectively controlled. Interdependence of task blocks is limited to tree-form hierarchical structures and thus it is not clear how it would be possible to specify tasks including non-sequential interactions between system components. The authors remark that more research is needed in order to develop analysis techniques for unwanted task interactions (such as contradictory requests). The template design methodology presented later in this chapter has some similarity to the condition systems framework, in terms of the independence of system components, the high-level specification of the control requirements, and the automation of all the steps which do not need human intervention.

Other attempts to resolve some of the problems in the application of supervisory control, especially the state-space explosion, include *modular* or *hierarchical* supervision and dealing with systems *incrementally*. A relevant discussion of these topics can be found in [96], [51] and [6], respectively. Of the methods mentioned, modular supervision seems to be most mature. The system is modeled as a set of separate modules or subsystems which may interact. Usually, control specifications can then be given in a modular fashion as well—concerning only a subset of all the modules. The reduction of complexity is a result of being able to compute separate, smaller, supervisors for each separate specification. Incremental approaches to DES control usually also rely on having a modular system model. Then, compositions of modules are constructed only as needed in order to determine a given property of the system. In hierarchical control, the base system is usually abstracted in a specific fashion and then supervisors can be computed for only the simpler high-level model of the system. Unfortunately, the research done on hierarchical supervision is more disparate and a unifying theme is lacking [30]. Modular control is not without problems either. When separate supervisors are constructed for each specification, it is not possible to predict what the net effect will be of the simultaneous application of all supervisors. Sometimes, due to some interdependence between the different control policies, the system may block. Thus, after the separate supervisors are constructed, it is necessary to check if the simultaneous application of these supervisors will lead to blocking. For this purpose, all supervisors have to be composed, which in some sense forfeits the benefit that is achieved by constructing separate supervisors. However, since blocking is a global property, in the general case there is no way to avoid the global check. Despite the fact that modular supervision does not resolve all problems, it is certainly beneficial when managing complex systems and solutions. As discussed in Section 3.1.2, modularity seems to be an essential tool when solving DES problems. The template methodology described next takes advantage of modular design techniques.

## 4.2 Template Design of DESs

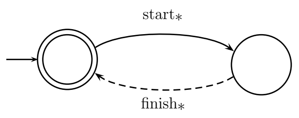
One of our observations during the study discussed in Section 3.1.1 is the following. When faced with a new problem, subjects frequently engaged in drawing a simple diagram of interactions between parts of the system which needed to be modeled. It appeared that the subjects liked to isolate different aspects of a system before they proceeded with the low-level modeling. Thus, we wanted to develop an approach to modelling where control engineers can focus on assembling blocks of subsystems and specifications instead of worrying about every little detail of the system. Furthermore, our goal was to develop the approach *within* the framework of supervisory control, rather than overhaul existing theoretical results and ask experts to acquire additional background.

### 4.2.1 Framework

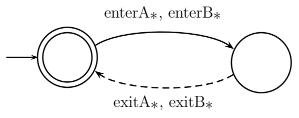
Before we proceed with the theoretical aspects of our work, we will describe a simple system (a part of the system from Section 5.2). It will be used to illustrate the steps of the new methodology. We will consider three system modules: a rotating table, a robotic arm and a drill. In this subsystem, there has to be mutual exclusion between the table and each of the other components so that the table does not rotate while another module performs an operation. Thus, we will use two specifications: one for the table and the arm, and one for the table and the drill. The system modules and the specifications are shown in Fig. 4.1.

The framework for template design is largely based on the work of Santos *et al.* [74, 75]. The authors propose a methodology for conceptual design of DESs using *entities* and *channels*. Entities are the active parts of the system (e.g., workstations). Channels are passive parts of the system which facilitate the transfer of matter and energy between entities (e.g., conveyor belts). This framework is suitable for the modeling of complex systems since it allows the simultaneous definition of both structure and functionality.

In our framework we decided to keep all the basic propositions of [75], however, we decided to cast the whole idea purely in DES terms. A system model consists of a set of modules (subsystems), a set of channels (specifications), and links between



(a) Modules $G_*$: rotating table (substitute '$T$' for '*'), robotic arm (substitute '$R$') and drill (substitute '$D$').



(b) Specifications $E_*$: mutual exclusion between table and arm (substitute '1' for '*') and mutual exclusion between table and drill (substitute '2').

Figure 4.1: The modules and specifications used to illustrate the template design methodology.

the modules and channels. Modules and channels as we use them here are similar to the subplants and local specifications in [16]. Finite-state automata are used for the models. Let $I$ and $J$ be index sets such that $|I|, |J| \in \mathbf{N}$ and $I \cap J = \emptyset$. The set of modules is

$$M = \{G_i = (\Sigma_i, Q_i, \delta_i, q_{0i}, Q_{mi}) \mid i \in I\}$$

and the set of channels is

$$N = \{G_j = (\Sigma_j, Q_j, \delta_j, q_{0j}, Q_{mj}) \mid j \in J\}.$$

Furthermore, all modules and channels have to be asynchronous, i.e.,

$$\forall i \neq j, G_i, G_j \in M : \Sigma_i \cap \Sigma_j = \emptyset$$

$$\forall i \neq j, G_i, G_j \in N : \Sigma_i \cap \Sigma_j = \emptyset$$

$$\forall G_i \in M, G_j \in N : \Sigma_i \cap \Sigma_j = \emptyset.$$

The requirement that modules be asynchronous is not a stringent restriction as discussed in [16]. The benefit of having asynchronous modules is mainly in being able to make more uniform assumptions about the system. If some modules are not asynchronous, they can be composed until there are no dependencies between modules. The channels have to be asynchronous because they describe generic specifications. It is only with the help of links that the specifications are synchronized with the given system. In our example, $M = \{G_T, G_R, G_D\}$ and $N = \{E_1, E_2\}$ (as shown in Fig. 4.1).

In order to relate modules and channels, and determine what specifications should be enforced on the different subsystems, one would link the appropriate events. Let $\Sigma_M = \bigcup_{G_i \in M} \Sigma_i$ be the set of all events in the modules and $\Sigma_N = \bigcup_{G_j \in N} \Sigma_j$ be the set of all events in the channels. Then, the links in the system model will be given by the function

$$C : \Sigma_N \to \Sigma_M.$$

In other words, the function defines links between events of channels and events of modules. The interpretation of the link $C(\tau) = \sigma$ is that the event $\tau$ in the given channel should be considered equivalent to the event $\sigma$ of the given module—thus relating the generic specification to the given system. Synchronization between the modules and channels is established, in effect defining the protocols for the transfer of information between parts of the system. For all $G_j \in N$, the restrictions of the function,

$$C|_{G_j} : \Sigma_j \to \Sigma_M,$$

have to be injective to ensure the consistency of the model. The function

$$C^{-1} : \Sigma_M \to 2^{\Sigma_N}$$

is the inverse of $C$ and, given $G_j \in N$, the restriction of $C^{-1}$ to $G_j$ is

$$C^{-1}|_{G_j} : \Sigma_M \to \Sigma_j,$$

where $C^{-1}|_{G_j}(\sigma)$ equals the only element of $C^{-1}(\sigma) \cap \Sigma_j$ if it exists, and is undefined otherwise.

In our example, we need to link channel $E_1$ to the table, $G_T$, and the robotic arm, $G_R$. Similarly, we need to link $E_2$ to the table and the drill, $G_D$. The channel events marked with "A" will be linked to events of the table, while the events marked with "B" will be linked to the arm (in $E_1$) and the drill (in $E_2$). Thus, we define the function $C$ as follows:

$$C(\text{enterA}_1) = \text{start}_T; C(\text{exitA}_1) = \text{finish}_T;$$
$$C(\text{enterB}_1) = \text{start}_R; C(\text{exitB}_1) = \text{finish}_R;$$
$$C(\text{enterA}_2) = \text{start}_T; C(\text{exitA}_2) = \text{finish}_T;$$
$$C(\text{enterB}_2) = \text{start}_D; C(\text{exitB}_2) = \text{finish}_D.$$

As a result, for example, $C^{-1}(\text{finish}_T) = \{\text{exitA}_1, \text{exitA}_2\}$ and $C^{-1}|_{E_2}(\text{finish}_T) = \text{exitA}_2$.

After a system is modeled in the proposed framework, modular control can be applied to obtain supervisors for the separate specifications. This is possible since,

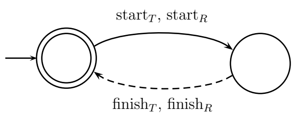


Figure 4.2: The synchronized version of $E_1$.

under the right interpretation, the model is equivalent to that of a regular modular system. In our work we propose the use of an optimized version of modular control, namely local modular control [16]. The precondition for the application of this method is satisfied, i.e., the participating modules are asynchronous. All modules which are linked to a channel participate in the subsystem influenced by the specification determined by the channel. Let $G = (\Sigma, Q, \delta, q_0, Q_m) \in N$ be a channel. Then define $G' = (\Sigma', Q_E, \delta', q_0, Q_m)$ as the synchronized channel $G$ where all channel events have been replaced with their corresponding module events, i.e.,

$$\Sigma' = \{\sigma \mid \exists \tau \in \Sigma, C(\tau) = \sigma\},$$
$$\delta'(q, \sigma) = \delta(q, C^{-1}|_G(\sigma)).$$

Furthermore, define

$$C(G) = \{G_i \mid G_i \in M, \Sigma_i \cap \Sigma' \neq \emptyset\},$$

the set of modules influenced by $G$.

In our example, in order to synchronize the channel $E_1$, the events are replaced as specified by the function $C$ (as shown in Fig. 4.2). Channel $E_2$ is synchronized in a similar way. Furthermore, $C(E_1) = \{G_T, G_R\}$ and $C(E_2) = \{G_T, G_D\}$.

For every channel $G_j \in N$, all the modules influenced by it are composed via synchronous product.

$$G^j_{sys} = (\Sigma^j_{sys}, Q^j_{sys}, \delta^j_{sys}, q^j_{0sys}, Q^j_{msys}) = \|_{C(G_j)} G_i.$$

Then all events in the subsystem which do not appear in the synchronized channel are applied as self-loops to all states in the synchronized channel, i.e., the channel has no influence on the occurrence of these events.

$$G^j_{spec} = \mathrm{selfloop}(G'_j, \Sigma^j_{sys} \setminus \Sigma'_j)$$

Finally, the algorithm from [67] for the construction of the supremal controllable sublanguage of the synchronized channel with respect to the relevant subsystem is invoked.

$$S_j = \mathrm{supcon}(G^j_{sys}, G^j_{spec}).$$

As a result, local supervisors for each channel are constructed.

In our example, $G^1_{sys} = G_T \| G_R$ and $G^2_{sys} = G_T \| G_D$. All events in each subsystem are linked to the corresponding channel, e.g., the events in $G^1_{sys}$ are start$_T$, finish$_T$, start$_R$ and finish$_R$—and all of them are used in the synchronized channel $E'_1$ (shown in Fig. 4.2). Thus, no self-loops are introduced into the channels, i.e., $G^1_{spec} = E'_1$ and $G^2_{spec} = E'_2$. The supervisor $S_1$ obtained for $G^1_{spec}$ with respect to $G^1_{sys}$ is shown on Fig. 4.3. It is easy to see that the simultaneous operation of the table and the arm is avoided. The supervisor for $G^2_{spec}$ is analogous.

The last step involves checking whether the supervised system is nonblocking, as

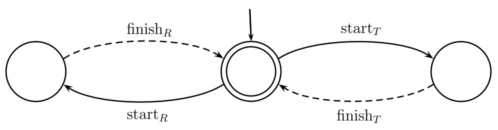


Figure 4.3: The supervisor for $G^1_{spec}$ with respect to $G^1_{sys}$.

defined in [16]. As long as the supervisors are nonconflicting, i.e.,

$$\|_{G_j}\overline{S_j} = \overline{\|_{G_j} S_j},$$

the nonblocking property is satisfied and, furthermore, the concurrent operation of the modular supervisors is optimal (i.e., equivalent to a monolithic solution). In our example, the two supervisors for channels $E'_1$ and $E'_2$ are nonconflicting.

### 4.2.2 Templates

The next advantage of our methodology is that it allows the use of templates. A template is simply a model of some discrete-event behavior. In the supervisory control setting, the model would be an FSA. In other words, any FSA can be a template. The idea behind templates is that if they define some frequently used behavior, one need not manually create a separate FSA each time this behavior is needed. Instead, the software can make a copy of the template, or *instantiate* the template.

Let $G = (\Sigma, Q, \delta, q_0, Q_m)$ be a template. The instance with index $p$ is defined as $Ins(G, p) = (\Sigma_p, Q, \delta_p, q_0, Q_m)$, where the events of $G$ are indexed with $p$. I.e.,

$$\Sigma_p = \{\sigma_p \mid \sigma \in \Sigma\},$$

$$\delta_p(q, \sigma_p) = \delta(q, \sigma).$$

Thus, for example, creating the DES modules for ten workstations would be reduced to instantiating the corresponding template with ten different indexes. Since the copies can be made automatically, the process is both faster and less error-prone. Furthermore, if the templates have been designed by experts and thoroughly tested, any user can use them with the same degree of reliability.

Since templates can describe both system behavior (i.e., modules) and restrictions on behavior (i.e., channels), the use of templates within our framework is very natural. Suppose there is a library of templates $Lib = \{G_k \mid k \in K\}$, where $K$ is an index set such that $|K| \in \mathbf{N}, K \cap I = \emptyset = K \cap J$. Then, the set of modules, $M$, participating in a design can be created by instantiating the required templates, i.e., $\forall G_i \in M$ (where $i \in I$), $\exists G_k \in Lib : G_i = Ins(G_k, i)$. Since the events of every template instance are named in a unique way, all modules will be asynchronous as required. Similarly, the set of channels, $N$, can be created by instantiating templates.

The example we used in Section 4.2.1 is an ample illustration of this idea. All system modules—rotating table, robotic arm and drill—share the same basic behavior, as shown in Fig. 4.1(a). The mutual exclusion specifications also share the same behavior, as shown in Fig. 4.1(b). Thus, if templates are used, the system modules can be instantiations of a generic "workstation" template, while the channels can be instantiations of a generic "mutual exclusion" template. If one looks again at the caption of Fig. 4.1, something very similar is described verbally.

### 4.2.3 Parametrization

A further improvement to the template design methodology can be made by considering parametrization of the template behavior. For example, if one would like to create templates for buffers, a separate template has to be constructed for all buffer capacities that need to be considered (e.g., buffer with two slots, buffer with three slots, etc.) However, it can be easily seen that the basic workings of a buffer are the same regardless of capacity. It would be much more convenient if there were a single "buffer" template which is parametrized in terms of capacity—and then at instantiation one would be able to choose the specific capacity to be used.

One possible approach to the parametrization of FSAs is described in [15]. There, a regular FSA is augmented with a *data collection*. The data collection is a vector of scalars which can range over some set. A vector of unary functions is associated with each transition in the FSA. For example, a buffer can be modeled as a single state with two self-looped transitions, "insert" and "remove", and a single integer in the data collection to keep track of the number of items in the buffer. Then, the functions "$+1$" and "$-1$" will be applied to the integer when "insert" and "remove", respectively, occur. In such a system, control can be based on predicates about the current state of the system and on the current value of the data collection. The authors propose a method to compute the supremal controllable sublanguage of a system by incrementally backtracking with the predicates until the control decisions do not attempt control of uncontrollable events. Unfortunately, the use of this model may easily result in non-regular behaviors and specifications. This is the reason why the model cannot be readily applied in the template framework proposed in this work. A potential solution would be to restrict the type of data collections that can be used. For example, each scalar in a data collection could be restricted to belong to a closed integer interval. However, even in this case it is necessary to find an efficient transformation from the parametrized model into a "simple" FSA.

## 4.3 Summary

We introduce the notion of DES templates within the framework of supervisory control. Typical behaviors for both DES modules and specifications are represented in an abstract way. The control engineer creates instances of these abstractions and then needs only to specify the way the instances interact. System modeling and the design of specifications occur simultaneously. The computation of the supervisory solution can be automated. Through the use of this methodology, we expect the following advantages:

- Faster design of systems. The use of pre-built templates not only reduces the time to mechanically input new FSAs but also the time to mentally consider low-level details of FSA implementations.

- More robust designs. Fewer errors may be made during the design since it is not necessary to manually copy FSAs and to keep track of the names of events in different modules and specifications.

- Easier design. Instead of considering the FSAs which underly every template, the designer can focus their creative effort only on the important task of determining which modules and channels are to be used and how to link them. The creation of supervisors is completely automated.

In the following chapters, we discuss the implementation of the template design methodology (Chapter 5) and we evaluate its usability (Chapter 6).

# Chapter 5

# Implementation of the Template Design Methodology

In this chapter, the implementation of the proposed template design methodology for modelling DES problems will be discussed. Instead of designing new software from scratch, we decided to take advantage of the extensibility of the IDES package already being developed at Dr. Rudie's research laboratory.

## 5.1 IDES Software

The IDES software has been under development since 2003, starting with a proof-of-concept design for a graphical tool for DES. During the early days, an interaction style for "natural" drawing of finite-state automata was developed. Also, a subsystem integrating $\LaTeX$ rendering into the graphical environment was implemented.

In 2005, a new team of developers implemented a number of FSA algorithms such as checking for language containment and intersection, and some DES-specific algorithms such as parallel composition and the computation of the supremal controllable sublanguage. Additionally, the IDES interface was redesigned to allow simultaneous work on a number of DES models. This software was released as IDES version 2. This version was also used in the observational study described in [32].

The first two versions of IDES allowed us to collect feedback and to understand better what are the technical and user requirements for the project. In 2006, a complete overhaul of the code was made. We proposed a newly designed, extensible architecture and guided the reimplementation of IDES. Similar to the previous versions, the software was developed using Java. The new architecture emphasized the separation of model and presentation. Furthermore, it prepared the ground for the use of different types of DES models in addition to FSAs. Interoperability with other DES software was improved through a redesigned, model-independent IO subsystem. On the surface, all key aspects of the interactions style were preserved, with the incorporation of many improvements such as the unique "film-strip" workspace. The reimplementation was released publicly as IDES version 2.1. By the end of 2008, five iterations of this version were released. In addition to bug fixes, new features were introduced, such as the "undo" capability.

The study of DES problem solving served as the jumping board for the development of the next major version of IDES, version 3. The new version shares much of the code with IDES version 2.1. However, some architectural changes were made to accommodate the development of external plugins which can extend the package by introducing new model types, algorithms or IO filters.

## 5.2 Prototype Tool for Template Design

The work on the implementation of the template design methodology started in 2007, as a part of a research project at the Federal University of Santa Catarina, Brazil. We implemented a prototype of the user interface for the methodology and conducted a test application for the control of a robotic testbed. In this section, we will highlight the most important parts of our experience. The full report can be found in [31].

The prototype was implemented as an extension of IDES version 2.1, which resulted in full access to all FSA functionality available in the original software. The implementation included a graphical interface where the user of the software can create and manipulate the design elements using the mouse cursor. A screenshot of the interface is shown in Fig. 5.1.

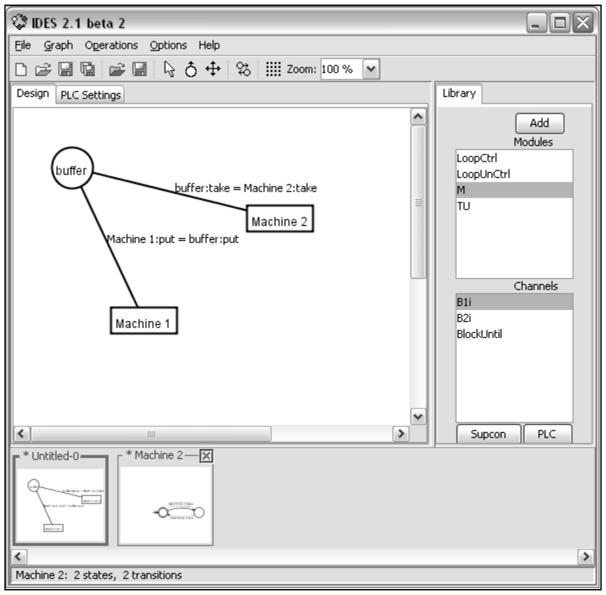


Figure 5.1: The interface of the prototype template design software.

Since the purpose of template design is to make the application of DES theory easier, we decided to try to streamline the complete process of application: from modeling to control of the real hardware. In many cases the real system is controlled by a Programmable Logic Controller (PLC); this was the case in the robotic testbed as well. Thus, we considered the generation of PLC code from the template design. There are many ways to convert FSAs into code, however, the method proposed in [17] seems to be suitable for two reasons: it converts FSAs directly into PLC code, and it is designed with modular control in mind. Since this approach is generic, the users still need to make manual modifications to insert hardware-specific instructions. In our software, for each event in the template design the user can specify a snippet of PLC code. Then, during PLC code generation, this code will be incorporated into the automatically produced code.

The prototype system for template design was used to design a controller for a robotic testbed at the Department of Automation and Systems, Federal University of Santa Catarina, Brazil. The functionality of the system, shown in Fig. 5.2, is to retrieve parts from an input buffer, perform operations on the parts and test if the operations were successful. Depending on the outcome of the test, the given part is output into one of a number of buffers (such as "accepted", "reprocess", etc.) The system is controlled via a Siemens S7-200 series PLC unit.

The application of the template design methodology to a real project, even though very small, brought some interesting insights from the participating engineering students. Surprisingly, the biggest advantage of the design methodology does not seem to be the ability to use templates *per se*. According to the feedback from the users of the software, the biggest benefit of the proposed methodology comes from the fact

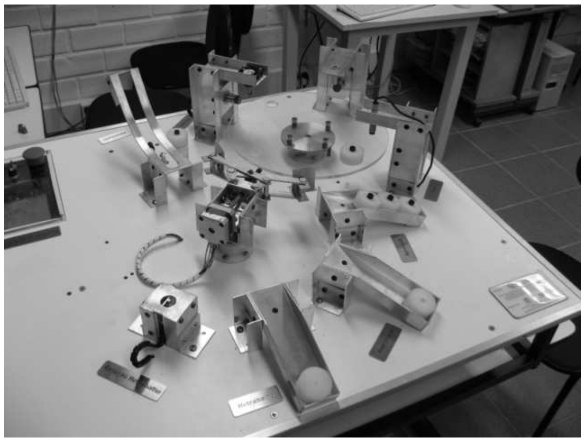


Figure 5.2: The robotic testbed where template design was applied.

that the template design environment makes it very easy to model and remodel systems, i.e., to create prototypes in the initial stages of system design. It is simple to replace modules and channels and then generate the corresponding supervisors to see what happens. The users no longer have to keep track of event name consistency between modules and between specifications. Synchronization is not achieved by naming events consistently but rather by visually linking them. Then, it is easy to try different synchronization strategies and it is possible to use a single template instance in a number of ways without having to always rename events. This property seemed to be especially liberating since renaming events is laborious and error-prone. In our project it was necessary to go through a large number of iterations where the system was simplified with different approaches. This rapid prototyping would not have been feasible if all operations had to be called manually and if event names had to be changed for every new approach.

From the observations made during the application of the template design methodology, it becomes clear that future work should focus on the usefulness for rapid prototyping. For example, it is desirable to allow the creation of conceptual designs without having to instantiate specific templates, i.e., by creating "placeholder" modules and channels. The user will be able to delay the assignment of templates to these placeholders until more of the overall design is ready.

The prototype implementation and the test application of the methodology served as a great motivation for the following, comprehensive implementation which we discuss next.

## 5.3 Implementation

The comprehensive implementation of the template design methodology required the introduction of a completely new modelling environment in IDES. Thus, in line with the original design goals for the IDES project, it was decided to first refine the architecture of the program to accommodate externally-developed plugins. Then, the template design modelling environment was implemented as a plugin. Besides the most obvious goal of reducing development effort, this decision was beneficial in the sense that the API for plugins was immediately verified through the newly created plugin. In fact, the development of the plugin API and the template design plugin (TD plugin) proceeded largely concurrently. Last but not least, the source code for the TD plugin will be made available publicly, thus serving as a reference implementation for IDES plugins.

### 5.3.1 Plugin architecture

Since version 2.1, IDES has been internally modularized with the expectation of the creation of an API for plugins. Thus, it was not hard to refactor the code and expose this functionality so that independently developed software can take advantage of it.

There are two parts of the IDES API. The first part is a collection of services available to plugins (or any other IDES code). The second part is a set of programming interfaces for plugins.

#### IDES services

The services available to plugins include:

**Core** Access to the user interface, the workspace manager, IO subsystem, settings interface, resource manager, utilities, and the *annotation* mechanism.

**Undo** Access to the undo manager.

**Notice** Access to the subsystem for managing warning and error notices.

**Latex** Access to the LaTeX rendering subsystem.

**Cache** Access to the persistent caching subsystem.

**FSA Model** The interface for FSA models as implemented in IDES.

The *annotation* mechanism merits some explanation as it underlines the main design philosophy undertaken in IDES—that of simplicity. Since the beginning, it was anticipated that model elements may need to "evolve" and eventually contain attributes that were not originally planned. For example, events in the classical

DES approach have only one attribute—that describing their controllability. Subsequent research introduced other attributes, such as observability or enforceability. Similarly, some research involving hierarchical DESs calls for the ability to associate (lower-level) FSAs with the states of a (higher-level) FSA. Thus, it is not feasible to decide *a priori* what features should be supported by each element of a DES model. There are a number of ways to introduce flexibility in a software system, such as by using subclassing in object-oriented languages. In IDES, however, we opted for a much more simple mechanism where various information can be added to objects during run time—similar to the way objects can be viewed as associative arrays in the JavaScript language [44]. Each entity (or class) of interest can be *annotated* with arbitrary attributes of arbitrary type. To this end, we introduced an *Annotable* interface which allows the association of any Java *Object* with any key of type *String*. The mechanism can be used to dynamically add information to entities, as needed. The only implicit requirement of this approach is that software must be able to handle missing annotations gracefully. As shown in practice during the development of IDES, the drawback of a less rigid, and thus more messy, approach was far out-weighed by the simplicity of implementing new features. The annotation principle is not only available to plugins, but is also used within the IDES code. The graphical layout of FSA models is implemented simply as a set of annotations of the states and transitions of the model. The file format used by IDES also relies on a description of the basic model, annotated with various additional information.

### Interfaces for plugins

Plugins for IDES need to implement one or more of the interfaces published in the API. Each interface determines what type of functionality the plugin implements. There can be a number of different types of plugins.

**Model** A plugin which implements a new type of model. Such a plugin is responsible solely for the maintenance of the consistency of the model and needs to implement only a limited number of methods, such as access to the model name. Upon initialization, the plugin must register with the repository of available model types.

**Presentation** Each model type requires one or more *Presentation*s to display the model. A plugin can register a *Toolset* which specifies which presentations will be used to display the model. The presentations usually need only provide the user interface element with which the user will interact. The toolset needs to provide a detailed descriptor of the presentations, menus and toolbars to be initialized when a model is loaded.

**IO** Plugins which can save and/or load models need to register with the IO subsystem. When loading or saving a model is required, the subsystem will automatically select the appropriate plugin for the given model type. Plugins can register to process the main data of a model (e.g., the mathematical description) or to process meta-data (e.g., annotations). There can be more than one meta-data section in a file, allowing for the annotation of models by different plugins for different purposes.

**Import and export** Besides loading and saving of native models, plugins can register to import from or export in non-native file formats. For example, the JPEG-export functionality in IDES is internally implemented through the plugin interface.

**Operation** A plugin which implements a DES operation. Such a plugin needs to specify what kind of inputs it needs, what outputs it produces, and implement the *perform* method. *Filter* operations, where the operations act directly on the inputs, are also supported. Upon registering with the operations manager, an operation becomes automatically available in the Operations dialog in IDES.

### 5.3.2 Template design plugin

The TD plugin introduces a new model type, the template design, to IDES. As a result, it was necessary to implement all interfaces provided in the plugin API. Design decisions needed to be made not only about the software architecture but also about the user interface of the plugin.

#### User interface

One of the biggest motivations for the template design methodology was the idea to offer the ability to create conceptual designs when solving DES problems, and to allow for rapid prototyping. Naturally, this led to the choice of a graphical interface for the modelling environment. Some insights about the user interface were also obtained in the initial proof-of-concept implementation (discussed in Section 5.2).

The TD plugin interface is shown in Fig. 5.3. It consists of three main parts: the modelling area, the template library and the consistency validator.

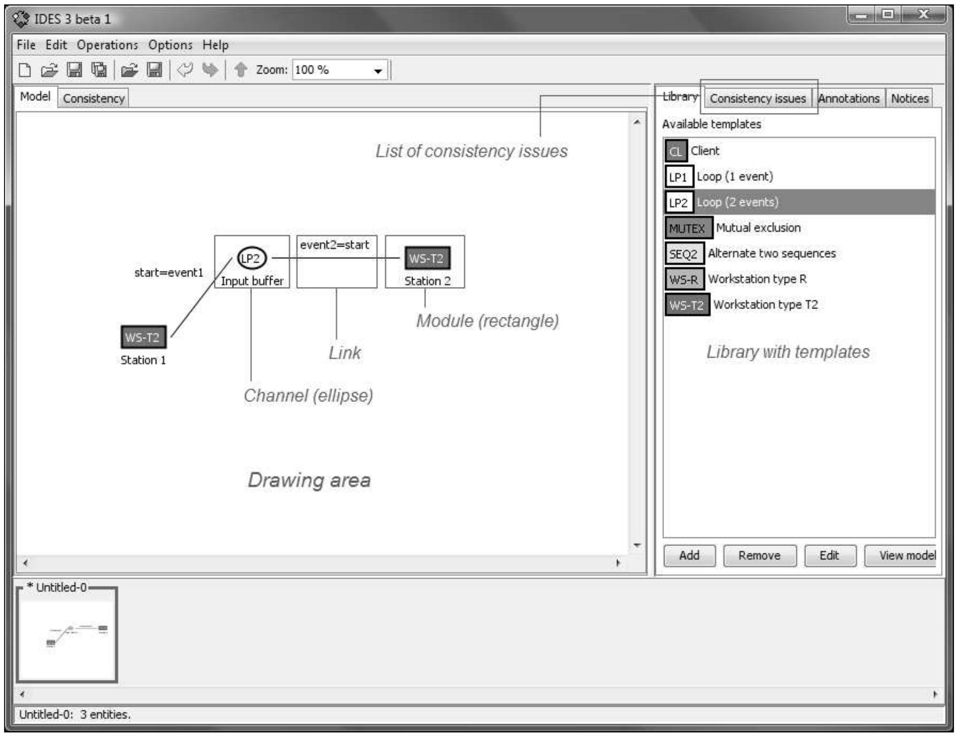


Figure 5.3: The user interface of the template design plugin.

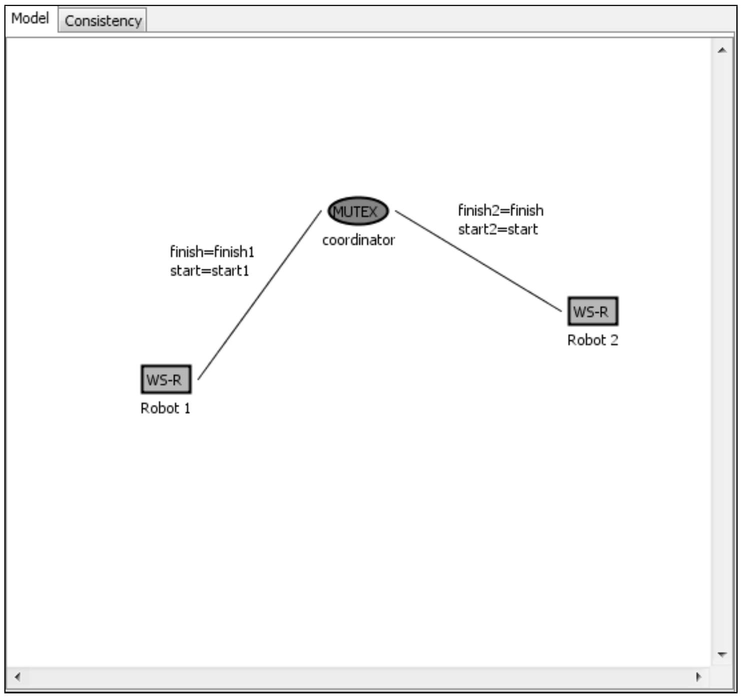


Figure 5.4: The template design modelling area.

**Modelling area** The modelling area, see Fig. 5.4, is where the conceptual design of a DES solution is built. Users can create modules and channels and establish links between them. The interaction is mouse-based. The model is presented graphically, as a diagram. Modules are represented by rectangular icons. Channels are represented by elliptic icons. This allows for the immediate visual recognition of the two classes of entities. Links are represented as lines connecting modules and channels. If multiple events in two entities are linked, only one line is drawn between the entities, labelled with all corresponding event pairs. The mouse, and pop-up menus are used to create or remove design elements and to modify the design as needed.

The following approaches were assumed in the design of the visualization and interaction style.

**Context-centric** The interaction with the design is context-centric. All elements of a design can be manipulated locally by clicking, dragging, or by invoking a (context-sensitive) pop-up menu. Furthermore, some operations become available upon the mouse entering the context of an element—such as the appearance of “connectors” (small circles) which can be used to directly link entities. In essence, the goal was to make all relevant interactions available immediately from the context of an element. Inspiration was also drawn from the “pie menu” where menu actions are available from a circular shape around the point of interest [8]. The following operations are available by directly manipulating modules or channels:

- Relocate
- Open underlying FSA model
- Relabel
- Link to another entity
- Compute local supervisor (channels only)

All other operations are available from the context-specific pop-up menus of the elements.

**Modeless** The use of different modes in user interfaces is sometimes necessary, however, it may lead to confusion and may reduce the efficiency and desirability of the interface, [69]. Furthermore, remembering which mode of interaction one uses could put an additional strain on the limited capacity of the human working memory. The interface for the modelling environment was designed so that the user interaction is modeless. In particular, it is not necessary to switch between different “tools” when modelling. This is possible, in part, by having a context-centric approach where the palette of available actions is naturally restricted by the affordances of the manipulated object.

**Unconstrained** Without the establishment of many restrictions, it is easy to create inconsistent template designs. For example, linking the events of two modules or two channels breaks the requirement that FSAs in the model be asynchronous. Similarly, linking a single event from a channel to multiple module events leads to an inconsistent design. One way to tackle this issue is to prevent the user from creating inconsistent models, by constraining the available operations when needed. For example, Norman argues that constraints can be used in product design to prevent undesired use [64]. Indeed, constraints are already used in the FSA-drawing interface of IDES, e.g., when the user draws an edge to an empty space, a new node is created there automatically, preventing the creation of an inconsistent model (containing an edge without a target node). In the case of the template design interface, however, we decided not to constrain the user actions and to allow the design of inconsistent models. This decision was motivated by two factors. First, the constraints would have to be numerous, dynamic, and complex. Under such conditions, it is very likely that the users would not be able to form the correct mental model of the interface. More specifically, it would be hard to obtain a mental model with a sufficiently strong predictive power—a key property according to Norman, [63]. Thus, it is likely that users would experience unexpected program behavior and the usability of the software would be greatly reduced. The second factor in making the decision is that the anticipated use of the software will be the rapid prototyping of control solutions. As already discussed, solving complex problems may involve several iterations, where the requirements are refined, or even replaced, as the solution takes shape (e.g., see [97]). For example, it may not be clear from the very beginning which parts of a model should be the modules and which ones should be the channels. However, the user may wish to establish links between them to denote some sort of dependency. The construction of consistent designs requires one to make many decisions for which information may not be available in the very beginning. Not constraining the user interaction lets users explore the solution space more freely, gradually refining an initial "sketch" to a consistent formal model.

**Flexible** The interface was designed to be flexible and accommodate a number of interaction styles. For example, it is possible to link entities either by clicking, dragging, or though a menu. Furthermore, most operations which can be accomplished by direct action (such as labelling an entity), can also be accessed through a pop-up menu. Flexibility is intertwined also with the lack of constraints in the interface. Different sequences of actions are acceptable in the creation of a template design. For example, it is possible to link all entities first and then decide which ones are modules and channels, or to decide first which entities are modules and channels and then link them.

**Consistent** Consistency is an essential property of usable systems [18, 64]. The interaction style when linking entities in the modelling area and when linking individual events in the event linking dialog is the same. Similarly, there is no essential difference, in terms of user interaction, between modules and channels in a design. As the template design environment was implemented as a plugin for IDES, it was also important to make sure that it integrates well with the rest of the interface. Thus, all common interface elements were shared between the standard IDES environment (FSA modelling) and the template design environment. For example, the dialog box for labelling nodes and labelling entities appears and behaves in exactly the same way in both environments. Furthermore, instead of introducing a new environment for the FSA models of entities, when the user wishes to work on a FSA model, it is loaded into IDES in the same way that a regular FSA model would.

**Zoomable** Zoomable interfaces are interfaces where the user can select the level of detail they want to use for different elements [25]. While the interface of the template design environment does not subscribe completely to the zoomable interface paradigm, the template designs are essentially conceptual, or higher-level views of a DES. Thus, users can "zoom into" each entity to explore it in more detail, i.e., examine the lower-level FSA model. Similarly, it is possible to "zoom out" of lower-level FSA models to see the higher-level template design.

The interface offers some other features on which less emphasis was put. For example, the user is able to customize the appearance of entities by using colors. Thus, it is possible to encode different aspects of a model to allow for quick visual segregation.

**Template library** The template library is the second important part of the user interface (shown in Fig. 5.5). While the modelling area allows for the creation of conceptual designs, it is the template library that allows for the use of templates in the design.

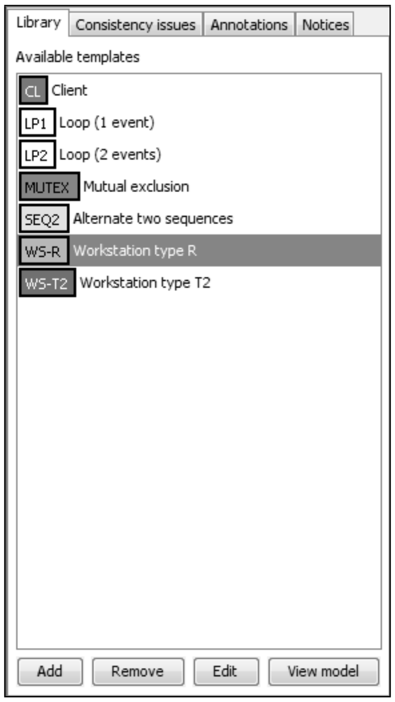


Figure 5.5: The template library.

The template library is simply a repository of FSA models which can be instantiated, or copied, into the design. It is possible to add any FSA model to the library, both from a template design and from the regular FSA models loaded in IDES. Each template (i.e., FSA model in the library) has an icon to represent it. Currently, icons can have different colors and different IDs—short descriptions, a few characters long. As well, each template has to have a longer description. Once added to the template library, templates can be modified. Not only is it possible to change the appearance of the icon and the description, but also changes to the FSA model can be made. Templates can be removed from the library as well.

The user can make use of the templates in a very simple way. A template can be dragged from the library to the modelling area to create an instance of it, i.e., to create an entity whose underlying model is the given FSA. The entities will be

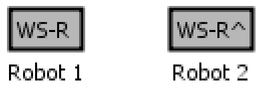


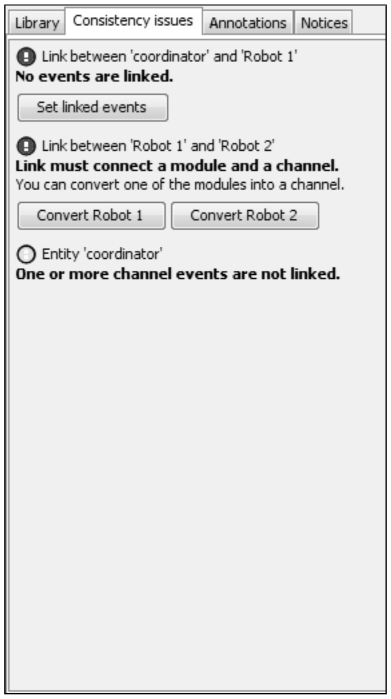
Figure 5.6: Two instances of the same template. The model of the instance on the right has been modified after the instantiation; this is denoted by the addition of a marker in the icon.

represented with the icon of the instantiated template. It is possible also to drag a template onto an existing entity in the design, to replace the FSA model of the entity. Lastly, in line with the context-centric approach of the interface, the pop-up menu of each entity allows the replacement of the FSA model with a specified template, or the addition of the existing FSA model to the template library.

In order to differentiate between entities which contain the original version of a template model, and entities whose template model has been altered, the icons of entities with modified models are augmented with a small symbol (as shown in Fig. 5.6).

**Consistency validator** The consistency validator, as seen in Fig. 5.7, is the last major part of the user interface. It visualizes the inconsistencies, if any, in a template design.

One of the design choices for the interface was not to constrain the modelling process of the user. As a result, it is possible to create inconsistent template designs. However, the ultimate goal of most users is to model a DES correctly and to obtain the supervisory solution. This implies that, at some point, most users would like to arrive to a consistent design. It was already discussed that the rules for consistent designs are multiple and not necessarily obvious. The proposed solution involves automatic consistency evaluation. The main goals in its introduction were to allow



Link between 'coordinator' and 'Robot 1'
**No events are linked.**

Link between 'Robot 1' and 'Robot 2'
**Link must connect a module and a channel.**
You can convert one of the modules into a channel.

Entity 'coordinator'
**One or more channel events are not linked.**

Figure 5.7: The list of consistency issues for template designs. In this case, a few sample issues are listed.

the users to assess the consistency of a template design, but not to impose such an assessment.

First, all potential inconsistencies in a template design were enumerated and described. In total, eight types of errors were identified (such as a link between two channels) and three types of potential issues (such as a module which is not linked at all). Each issue was described in plain language. For example, if there are channel events which are not linked, the description will be "One or more channel events are not linked."

Second, a validator was implemented which checks the consistency of the design continuously (after each modification of the design). The output of this validator is displayed in an unobtrusive way, so as not to interfere with the design process. The output is available in three places in three different forms:

1. The status bar of IDES displays a summary of how many inconsistencies are found in the current design. This information is always available to the user (e.g., for a quick reference), however, it is very unobtrusive and easy to ignore if so desired.

2. There is an alternative view of the modelling area, entitled "Consistency". In this view, all inconsistent elements of the design are highlighted in orange. Otherwise, the modelling interface is not affected at all and the user can work with either view (regular or highlighting) in the same way, and can switch between them at will. With this view, the user can obtain a more immediate sense of the inconsistencies in the model. However, the workflow need not be altered, as the modelling interface behaves in the same way as when the regular view is used.

3. The full list of consistency issues in a model is available in a separate tab, entitled “Consistency issues”. Each item in the list contains the description of the issue and specifies which elements are affected. In some cases, it also offers shortcuts for solving the issue. For example, the “Link must connect a module and a channel” has shortcuts to convert one of the linked modules into a channel (or, conversely, to convert one of the linked channels into a module). Clicking on a issue highlights the affected elements in the “Consistency” view of the modelling area. Through the full list of issues, users can explore in detail all issues and learn about the causes for the issues. It is anticipated that this list will also help users construct the correct mental models of what the causes of issues are and, eventually, learn how to create designs without consistency issues.

By offering different visualizations of the consistency issues, the users are able to select the level of detail they want to see. Novice users can take advantage of the detailed list of issues, while expert users may not need to refer to any of the available information. All visualizations are unobtrusive and do not restrict or interfere with the design process.

### Implementation

In order to implement all the required functionality of the TD plugin, it was necessary to take advantage of all plugin interfaces.

**Model** The base of the TD plugin is the implementation of the template design model. Much like the implementation of the FSA model in IDES, the mathematical description of the model structure is independent from the information necessary to display the structure.

The mathematical description includes elements for the modules, channels, and event links. The implementation is very simple, with only three classes. In order to allow for incomplete models, consistency is not enforced at the model level. For example, it is possible to have a module without an associated low-level FSA description, or to have a link between two channels. Furthermore, the links do not point to the event objects which they link, but rather refer to the events by name. Thus, it is possible to remove the linked events from the FSA models of modules or channels without affecting the links. The possibility for such inconsistency is desired in order to allow for experimentation at the FSA model level without worrying about the immediate impact on the higher-level template design.

The model layout information is encoded with the use of visual elements, mirroring the elements from the mathematical description. Modules and channels are represented by entities, while links are represented by connectors. Entities carry additional information about location, size, icon and label. Connectors visualize the links between events from different entities. If there is more than one event link between the same entities, these links are automatically grouped into the same connector. Entities are maintained as annotations of the corresponding modules and channels. Connectors do not carry any specific layout data and thus are computed when needed from the layout of the entities. The only exception are the connectors which do not link specific events. This will be the situation when the user links two entities but does not yet specify which events from these entities should be linked. In this case, there will be no underlying links in the mathematical description of the model. Such “empty” connectors are added as layout annotations to the model.

**Presentation** The presentation interface describes any user interface fragment to be employed when visualizing a model. The four presentations which are part of the TD plugin subsume the user interface parts identified in Section 5.3.2. The modelling area has two views, “regular” and “consistency highlighting”, each one implemented as a separate presentation. The highlighting view is a subclass of the regular view. The other two presentations are the template library and the list of consistency issues. These presentations are described and packaged together by the toolset for the template design model. Thus, when the user loads a template design model, the toolset will provide a descriptor of what presentations to use in the interface for the given model.

All these presentations can be used together, concurrently. A change to the model, initiated via any of them, will be reflected in the other ones immediately and automatically. This functionality is possible because of a messaging mechanism in the template design model. Presentations for a model can (but do not need to) “hook” onto the model and receive a message any time there is a change to the model. The messaging mechanism is synchronous and uses the same approach as the event-passing in the GUI library in Java, Swing. It is necessary to implement a set of callback methods and then register as a “listener” with the model. Presentations which no longer wish to track changes in a model can unregister. In the TD plugin, the modelling area views and the consistency validator register with the model to track changes, while the template library presentation does not register as it is not affected by any changes to the model.

**IO** The TD plugin also contains a class responsible for the storing and loading of template design models. The plugin reads and writes not only the mathematical description of a model, but also the annotations with the layout information. The latter are saved in a meta-data section of the file. The IO for a template design, however, has some specifics which are not common to other types of models. As a template design is only a high-level, conceptual description of a system of low-level FSA models, it is also necessary to include IO for these models when saving or loading the template design. As IDES already has support for the IO of FSAs, we decided to make use of it instead of creating a custom solution. Thus, the FSA models are not saved as part of the template design file. Rather, they are saved as independent, regular FSA model files and the template design file contains only links to them. The drawback of this approach is that users have to manage a number of files for a single template design model. However, there is no need to develop a new IO solution for FSA models which has to be maintained up to date with the rest of IDES, and users are able to access the models from a template design independently.

The template library also requires an IO implementation, however, it is independent from the IO interfaces in the plugin API. The only way it interacts with the IDES IO subsystem is through a small class which registers as a meta-data loader/saver for FSA models. When the system saves or loads the FSA model of a template, this class steps in and processes the extra data for the template, e.g., the description of the template icon.

**Import and export** The import and export interface in the plugin API was implemented to provide JPEG and PNG export for template designs. The FSA models from such a design can be exported to all file formats which IDES supports. No support for import is provided; to our best knowledge, there is no other implementation of the template design methodology.

**Operation** Three operations for template designs were included with the TD plugin. Each operation is just an encapsulation of a number of standard DES operations, together with algorithms for the renaming of events according to the event links in a template design. The three operations are the following:

**tdchannelsup** This operation is used to compute the local supervisor for a channel. It renames all events in the models involved in the computation, as necessary, composes all modules connected to the channel, and computes the supervisor for the channel with respect to the composed modules.

**tdcentralsup** This operation is used to compute the centralized supervisor for the whole template design. It renames all events as necessary, composes all modules into a monolithic system model, composes all channels into a monolithic specification, and computes the supervisor for the specification with respect to the system.

**tdmodularsup** This operation is used to compute the modular supervisory solution for the whole template design. It invokes the *tdchannelsup* operation for every channel in the template design, and then checks if the resulting supervisors are locally modular, i.e., if the solution is non-blocking and optimal.

With the exception of the algorithm for the renaming of events according to the event links, no new algorithm is introduced with the above operations. The template design operations rely on the DES operations already available in IDES to perform the necessary computations.

Overall, the implementation of the TD plugin for IDES has more than 130 classes and more than 7000 lines of method code. The software is entirely written by the author and is released under the BSD license.

# Chapter 6

# Evaluation of the Template Design Methodology

One of the main results of the investigation of problem solving in the area of DES control problems, [32], was the proposal of the template design methodology (Chapter 4). The software to support this methodology of DES problem solving was implemented as a plugin for the IDES package (Chapter 5). The last part of this work describes the evaluation of the usability of this implementation. It shows that, indeed, the proposed methodology of DES problem solving is advantageous in comparison to the classical approach.

## 6.1 Method

The template design methodology was developed in order to provide a more efficient way to model DES problems and to speed up the process for obtaining control solutions for such problems. The informal evaluation of the prototype implementation of this methodology (discussed in Section 5.2 and [31]) showed that the methodology could be useful in solving DES control problems, especially when rapid development and prototyping of a supervisory solution is needed.

After the implementation of the template design methodology as a plugin for the IDES software package, it was necessary to perform a more formal evaluation of the approach. As there are no other implementations of the methodology, it is unavoidable that any evaluation will test the combined effect of both the approach and the specific software implementation. Thus, in the rest of this chapter, we will refer to the methodology and the implementation interchangeably. This is done with the understanding that, in fact, we refer to the combined effect.

### 6.1.1 Test conditions

The main focus of the evaluation is the usability of the implementation. As the proposed methodology for DES problem solving is new and there are no established results for the performance using such a methodology, it is necessary to use comparative evaluation, where the performance is contrasted with other methodologies. The software for template design was implemented as a plugin for IDES version 3 which originally supports only traditional methods of solving DES problems, using FSA models as proposed in the Ramadge and Wonham framework [94]. Thus, the most natural form of evaluation was to contrast the performances when IDES version 3 is used with and without the TD plugin. In this way, any difference in the performance would be attributable mostly to the impact of the plugin. The overall interface of the software remains the same. The condition when IDES is used without the TD plugin is called the "classical approach", while the condition when IDES is used with the

TD plugin is called the “template design approach”.

There are two main types of experiment designs which describe how test conditions are assigned to subjects. In the *between-subjects design*, one condition is assigned per subject. In the *within-subjects design*, all conditions are assigned to every subject (Chapter 4 in [39]). The main advantage of the between-subjects design is that no learning can be transferred between earlier and later tasks, as only one task is performed by a given subject. This design, however, also has higher demands on the size of the subject pool. If there are two tasks and ten data points are needed per task, in total twenty subjects will have to participate. In comparison, for within-subjects designs only ten subjects will have to participate since each subject will perform all tasks. Recruiting subjects is a very challenging task for the evaluation of DES problem solving; a similar problem was faced in [32]. The expected low recruitment rates confined our study to using a within-subjects design.

Preventing learning in within-subjects experimental designs is a big challenge. Subjects unavoidably learn during the performance of a task and this could affect their performance in consequent tasks. In some cases, due to the nature of the tasks, the effect of learning can be negligible. For example, in tasks where subjects use different pointing devices, [58], little, if any, learning can be transferred from one device to another. However, in tasks which involve problem solving, solving a problem once has a huge impact on the performance when the problem is solved again. In *one-off* problems (involving insight), gaining the right insight is equivalent to solving the problem [46]. Then consecutive solutions of the same problem involve simply recalling the insight. In problems where successive steps need to be taken, the mechanization of thought can lead to significant gains in the speed of problem solving [56]. Consecutive solutions to the same problem would involve recalling and re-enacting the sequence of steps which lead to the solution.

A traditional method to counter the effect of the transfer of learning is to assign randomly and to counterbalance the order of tasks which each subject performs (Chapters 3 and 4 in [39]). Thus, with two tasks, half of the subjects will perform the first task first, while the other half will perform the second task first. With counterbalancing, the results are equally affected by learning transferred from the first task to the second and from the second task to the first, avoiding ordering bias. However, this approach, when used with problem solving, will not avoid the issues already described. Namely, the results from the tasks performed second may be meaningless.

Another method to counter the carry-over effects in a study is to allow for a substantial amount of time to pass between the performance of tasks [29]. This method was used in the initial study of DES problem solving in order to facilitate forgetting [32]. However, this method may lose effectiveness if subjects are asked to solve identical problems. Furthermore, due to expected difficulty in subject recruitment, it was not possible to plan for experimental design which calls for a long-term commitment from the subjects.

As a result of the above considerations, the design of the evaluation study needed to be more complex. It became obvious that it would not make sense to ask each participant to solve the same DES problem under the two conditions (classical and template design). Instead, two different DES problems were designed.

The particular DES problems used in the study were influenced by two factors. First, it was important to reduce the impact of the cognitive load when solving a problem. Second, the problems had to be solvable in a short amount of time. The cognitive effort in solving the problems had to be reduced because the goal is to evaluate the usability of the template design methodology, and a big cognitive effort component can “overpower” the effects of any specific interface. In other words, it has to be “obvious” how to solve the problems so that subjects do not spend too much time deliberating on the essence of the solution, with potentially unpredictable impact on the collected metrics. Furthermore, the problems had to be short in order to increase the likelihood of recruiting subjects. Our goal was to keep the total time for each task under 30 minutes. After investigation of existing DES toy problems (e.g., [10, 94]) and using the experience from the study of DES problem solving (in [32]) it became apparent that such problems are cognitively too demanding for our purposes and/or take too long to model and solve. The natural approach, then, was to design problems where a part of the solution is already modelled. The subjects only need to modify a part and add some extensions. Not only are such problems fast to solve, but also significantly reduce the cognitive effort needed for the problem solving. An additional, and important, benefit which comes with this type of problem is that they lend themselves well for the study of the template design methodology. First, the preliminary study of the proposed methodology (described in Section 5.2) showed that it is suitable for prototyping (i.e., experimenting with and modifying solutions). Second, the advantage of using templates can be demonstrated if the solution calls for the replication of some part of the provided partial model. It must be noted that selecting partially-solved problems for the experiment in essence favors the template design methodology. In our opinion, however, this is not an issue as our goal is to evaluate the proposed methodology for problems where its application makes sense. Furthermore, we conjecture that many of the real DES problems do not require modelling from scratch.

The problems designed for the evaluation are shown in Appendices A.2 and A.3. The first problem describes an electronics factory (the “factory problem”) and the second one describes a manager of network devices (the “spooler problem”). Both problems use simple and concrete language in order to reduce the cognitive load needed to understand them. The problems come from different areas of automation (manufacturing and network management) and the control solutions are different. Thus, the amount of leaning which can be transferred between the problems is reduced. However, both problems require roughly the same number and type of actions in order to solve them. Thus, the performance when solving the two problems using the same methodology should be comparable.

After the two problems were designed, four potential conditions emerged:

- Factory problem, classical approach.
- Factory problem, template design approach.
- Spooler problem, classical approach.
- Spooler problem, template design approach.

Each subject only had to complete two tasks, one with the classical approach and one with the template design approach. The order of the two approaches was randomized and balanced. Similarly, one of the tasks had to involve solving the factory problem, while the other one had to involve solving the spooler problem. Again, the order of the two problems was randomized and balanced. Thus, each subject started with one of the four tasks listed above, while their second task was the complementary task from the list. This experimental setup is called *mixed design* (Chapter 12 in

[27]). A part of the testing condition is assigned within-subjects, i.e., each subject used both the classical and the template design approaches. The other part of the testing condition is assigned between-subjects, i.e., each subject solved the factory problem using either the classical approach or the template design approach (and, consequently, they used the complementary approach for the spooler problem).

The wording of the problems for both approaches was identical. However, the subjects under the template design condition had a potential advantage in that the partial model (template design) they received also serves as a conceptual diagram of the existing solution. To compensate for this advantage, such a conceptual diagram, in printed form, was included with the problem description for the classical approach conditions (as seen in Appendices A.2 and A.3).

### 6.1.2 Metrics

There are many aspects of usability which can be measured. The three main categories of measures are *efficiency*, *effectiveness*, and *satisfaction* [42]. Furthermore, there are objective (physically measurable) aspects and subjective (experiential) aspects of usability. The method of evaluation which was chosen for this study was greatly influenced by the book *Measuring the User Experience* [84]. The authors not only describe different methods of evaluation but also discuss the applicability of the methods based on the professional experience of the authors. For our evaluation, the following measures of usability were selected:

- Rate of task completion,
- Time for task completion,

- Error rate,
- Subjective evaluation of the experience and
- the System Usability Scale.

**Rate of task completion** This basic measure of effectiveness expresses the proportion of subjects who managed to complete a given task. In this case, the rate of completion is compared for tasks involving the classical approach and tasks involving the template design approach. There is no limit on how long subjects can work on a task, thus a task is considered incomplete (or failed) only if the subject announces that they wish to give up solving the task. It is expected that, due to the simplicity of the tasks, very few tasks, if any, will be incomplete.

**Time for task completion** The time for task completion is another basic measure of the usability of a system, more specifically, of its time efficiency [84]. It is assumed that if the same task is performed using two methods (or systems, interfaces, etc.), the method where the task is completed faster is more efficient. In this case, the speed of task completion is compared for tasks involving the classical approach and tasks involving the template design approach. The timing only of tasks which are completed is included. In order to allow a more detailed investigation of the performance of subjects, two time intervals are measured: time to supervisor computation and time to completion. The time to supervisor computation is defined as the length of time since starting work until the supervisor computation algorithm is invoked (locally or globally) for the first time. The time to completion is the length of time since starting work until announcing completion of the task. If the supervisor computation algorithm is never invoked, the two times are identical. Why does it make sense to collect these two times? From previous observations (e.g., in [32]), the first invocation of the supervisor computation algorithm is roughly the time when a subject transitions from the "modelling" stage of problem solving to the "verification" stage. Recording this time together with the total time for task completion allows for a more granular investigation of the task performance.

**Error rate** The error rate, or how many errors subjects make during the execution of a task, is another commonly used metric in usability testing [84]. It is assumed that a higher number of errors corresponds to a less effective, more difficult-to-use system. In this case, the error rates observed in the classical approach tasks and the template design tasks are compared. Error rate information is collected only for complete tasks, where the subject did not give up. Unlike the first study of DES problem solving, [32], subjects produce complete solutions for the problems used in the evaluation. Furthermore, since the problems are very simple, much smaller variation in the potential acceptable solutions are expected. Thus, for the purposes of this evaluation, it is not necessary to use techniques similar to the counting of perceived mistakes employed in the initial study [32]. It is possible to examine the models and supervisory solutions obtained from the subjects and compare them with the expected solutions. As practice has shown, however, it is always necessary to apply a degree of lenience towards the "correctness" of solutions as there can be multiple interpretations of the same textual description of a problem. For the purposes of this evaluation, the following errors rubric was developed.

| | |
|---|---|
| **Model of newly introduced plant component** (same as existing models) | |
| correct event set | the model contains the correct number of events, with the correct controllability property |
| correct model dynamics | the FSA model generates the correct event sequences |
| **Modified model of existing plant component** (under valid interpretation) | |
| correct event set | the model contains the correct number of events, with the correct controllability property |
| correct model dynamics | the FSA model generates the correct event sequences |
| **Newly introduced control specification** (under valid interpretation) | |
| correct model dynamics | the FSA model generates the desired event sequences; the new specification may be introduced as a separate model, or incorporated into the existing specification model |
| correct synchronization | the specification is synchronized correctly with the plant through the use of events; in the classical approach, the event names in the plant and specification models have to be identical and in the case of the template design approach, the correct events have to be linked |
| **Supervisory solution** | |
| correct inputs | the correct models (plant and specification) are used as inputs to the supervisor computation algorithm, even if the models themselves are not correct |
| valid approach | the computed supervisory solution will enforce the desired control, given that the input models are correct; in the case of modular supervision (with local supervisors), the combined operation of all supervisors must be equivalent to the operation of the optimal centralized supervisor |

For each article from the rubric, one penalty point is added to the score of a solution if it does not satisfy the given article. Overall, a solution can receive a maximum of eight penalty points. As both problems developed for this evaluation, the factory and spooler problems, have the same structure and call for the same type of problem solving steps, the rubric can be applied to both of them. Similarly, the rubric can be applied uniformly to solutions with both the classical and the template design approaches. A template design approach is only a higher-level wrapper for the same basic solution elements. The low-level models and the computation algorithms are identical to the classical approach.

**Subjective evaluation of the experience** As discussed in [18, 84], many aspects, especially the subjective experience of users, can be investigated through the administration of questionnaires. The selection of questions for such questionnaires greatly depends on what information is of interest. Based on previous experience with DES problem solving (e.g., in [32]), we decided to focus on a few aspects which could help compare the template design methodology with the classical approach. The most important aspects were the following:

- user confidence and
- subjective learnability.

As discovered in the initial study of problem solving and as other researchers point out [93], there is significant lack of transparency in the automatically generated supervisory solutions to DES problems. This leads to the lack of confidence by the users in the solutions they obtain. Thus, it is of interest to see how the proposed methodology compares to the classical approach in terms of confidence. Another important aspect of usability is the learnability of a system, or how fast a user can learn to operate the system. Learnability requires a longitudinal study, where the performance of users is studied over time [18, 84]. As the limitations of this evaluation does not allow for such a study, we decided to instead collect information about the subjective opinion of the subjects on their experience with learning the system. It is important to keep in mind, however, that the subjective perception of learnability does not necessarily correlate to the objective measure [42]. Information about confidence and subjective learnability was collected by asking users to make selections from a Likert-style scale from one to five [53]. Likert-style scales allow the ordinal comparison of hard to quantify data (such as "liking"), however, they do not provide a deeper insight into the experiences of users. Thus, we also included open-ended questions where subjects could write what was easy and difficult for them when completing a task with a given methodology. In order to collect data immediately after the experience (as recommended in [84]), two questionnaires were administered to every subject—one after each task completed by the subject—and each questionnaire asks for feedback related to the given task. The questionnaire administered after the second (and last) task, also asks the subject directly about their personal preference of DES problem solving method. Finally, the second questionnaire contains an open-ended question where the subject is encouraged to describe what they see as the greatest contribution of the template design methodology. This information allows a comparison of the envisioned advantages of the methodology and the perceived advantages by the users. The two questionnaires can be seen in Appendix B.

**System Usability Scale** The overall subjective usability evaluation of systems is a challenging task and many questionnaires have been developed to collect such data. These include Computer System Usability Questionnaire, Questionnaire for User Interface Satisfaction, System Usability Scale, etc [84]. The use of standardized questionnaires is strongly advocated by Hornbæk, [42]. In our evaluation we also wanted to use a standardized questionnaire in order to compare the overall usability of the TD plugin to that of established numbers. Furthermore, it was necessary to also choose a short and simple questionnaire to keep the time commitment of subjects low. The System Usability Scale (SUS) [7], satisfied both conditions. The questionnaire contains only ten questions which are answered using a Likert-style scale. Despite its simplicity, the SUS correlates well with other measures of usability [85]. Furthermore, the SUS has been used for the evaluation of many software systems, thus standardized scores have been accumulated [84, 79].

### 6.1.3 Subjects and experimental procedure

For this experiment, in total twelve subjects were recruited. All subjects had knowledge of DES control theory through taking a graduate-level course on the topic. All subjects were administered a preliminary questionnaire, asking about their experience with DES software in general and IDES in particular. The data is tabulated in Table 6.1. As can be seen, the majority of subjects had some experience with DES software (e.g., IDES), however, very few had much experience with template design. Unfortunately, as explained earlier, it was not feasible to design a longitudinal study where subjects are asked to become experts on template design before evaluating the tool. In order to mitigate, as much as possible, this lack of experience, the experimental procedure was designed to introduce the subjects to the template design in a short time-frame.

Before the beginning of the study, the subjects are asked to complete two tutorials available online. Both tutorials use a version of the popular "Transfer line" problem

| | Engineering | Knowledge of | Experience using | | |
|---|---|---|---|---|---|
| Subject | background | DES theory | Any DES software | IDES | IDES+TD plugin |
| A | 3 | 5 | 4 | 4 | 2 |
| B | 5 | 5 | 5 | 2 | 2 |
| C | 5 | 3 | 4 | 5 | 1 |
| D | 4 | 4 | 4 | 4 | 3 |
| E | 5 | 4 | 3 | 3 | 3 |
| F | 2 | 5 | 5 | 5 | 3 |
| G | 5 | 4 | 4 | 2 | 2 |
| H | 1 | 4 | 5 | 5 | 5 |
| I | 5 | 5 | 5 | 5 | 2 |
| J | 5 | 5 | 5 | 1 | 1 |
| K | 5 | 5 | 3 | 3 | 2 |
| L | 5 | 5 | 4 | 2 | 2 |

Table 6.1: Self-reported background information about the subjects. The scale used is from one (very little) to five (very much).

[94] to teach how to use IDES (first tutorial) and how to use the TD plugin in IDES (second tutorial). Each tutorial was designed to take not more than one hour and subjects can complete these tutorials at their leisure.

When the subject arrives for the study, they are first asked to solve a simple, but somewhat linguistically ambiguous DES problem (reproduced in Appendix A.1), using both the classical and the template design approaches. The two solutions normally take less than forty-five minutes to produce. The subject is encouraged to seek consultations with the experiment conductor if they experience problems or have questions. The tutorials are also available for reference. The goals of this practice problem are the following:

- The subject is given a chance to learn IDES and the TD plugin in case they failed to complete the tutorials.

- The subject is reminded how to use IDES and the TD plugin immediately before the test.

- The subject can interact with the conductor to clarify any questions or misunderstandings they have about the software before the test.
- The subject's mind is stimulated to solve a DES problem which is more challenging than the real problems in the study. Thus, the first problem in the study will not require a "cold start" of the DES problem-solving skills of the subject. In a sense, the practice problem serves a the stimulus which primes the brain to "activate" the learned structures for DES problem-solving activities.

After the practice problem, there is a short break (about ten to fifteen minutes) and then the subject is asked to complete the first task of the experiment. Their performance is timed and the models they produce are retained for analysis of the error rate. The subject is asked to complete the first feedback questionnaire and, if the first task involved the template design methodology, they are also asked to complete the SUS questionnaire.

After the first task, there is another break which lasts ten to fifteen minutes. The subject then is asked to complete the second task for the experiment, where their performance is timed and the models they produce are retained. After completing the task, the subject fills out the second feedback questionnaire and, if the second task involved the template design methodology, they are also asked to complete the SUS questionnaire. With that, the participation of the subject ends.

### 6.1.4 Hypotheses

The hypotheses tested by this evaluation are the following:

- The total time for task completion is shorter using the template design methodology in comparison to the classical approach.
- The time for modelling during a task is shorter using the template design methodology in comparison to the classical approach.
- Fewer errors are made using the template design methodology in comparison to the classical approach.
- The template design methodology results in higher confidence in the models users produce, in comparison to the classical approach.
- The template design methodology results in lower confidence in the supervisory solution users obtain, in comparison to the classical approach.
- The template design methodology is (experientially) easier to use in comparison to the classical approach.
- The template design methodology is (experientially) easier to learn in comparison to the classical approach.
- Users have preference for the template design methodology over the classical approach.
- The average SUS score for the TD plugin is not lower than the average SUS score reported in the survey [84, 79].

## 6.2 Results

Data from all twelve subjects were collected. Before the start of analysis, it was necessary to identify outliers in the data as these can have marked detrimental impact on the power of statistical tests [39, 66]. It is common to identify outliers by examining the distance of data points from the mean in terms of standard deviation units, i.e., the *$z$-scores* of data points. If a data point is too far from the mean, i.e., the $z$-score is too large, it is considered an outlier. Commonly, a cutoff level of $z = 3$ is used [66]. However, in [87] the authors discuss the impact of different cutoff levels for data of small sample size. When a non-recursive cutoff procedure is used, for sample size of twelve, a cutoff level of $z = 2.246$ is recommended.

The identification procedure revealed outliers in the data for two subjects. One of the subjects spent a relatively long time completing their solution using the classical approach, with $z = 2.284 > 2.246$. Their time for modelling under the classical approach condition was also relatively long, with $z = 1.931$. At the end of their participation, the subject commented that they were very slow because they were not very comfortable with the DES theory required for the experiment, and that their performance was not slowed down by the software. The other subject spent a relatively long time completing their solution using the template design approach, with $z = 2.476 > 2.246$. This subject misinterpreted the factory problem. They assumed that robots 2 and 3 must not operate concurrently and struggled to find a satisfactory solution. The solution hinted at in the problem description does not guarantee the lack of concurrent operation of these robots. The subject was trying to accommodate the additional requirement within the expected solution.

There are three options for what to do with outliers in data [39, 66]:

- transform the data,
- recode the outliers, or
- remove the outliers.

Transformation of the data means that the value of each data point is transformed using a function, such as square root, logarithm, or inversion. The rationale for such transformations is that they can "pull in" outliers closer to the mean. However, transformations have the negative effect of changing the relative distances between data points, i.e., may effectively convert ratio variables into ordinal variables where the degree of effect can no longer be quantified. Recoding of outliers consists of changing the value of each outlier to the value of the closest non-outlier. This is a simple procedure, however, its applicability in within-subject designs seems dubious as the performances under different conditions are related and the independent recoding of only one performance will alter the nature of the relation. It is then difficult to determine which data point is "closest" as sets of related data points have to be compared for each subject. The removal of outliers is the simplest procedure, and the most applicable to complex experimental designs. When the sample size is not very small, it improves the power of the statistical tests as the main effects become more pronounced.

In the discussion that follows, only the data from the ten subjects without outlier data points are considered.

### 6.2.1 Rate of task completion

The rate of task completion was 100% for all subjects for all tasks. That is, no subject announced that they had given up solving a task and, furthermore, all subjects reported in the questionnaires that they completed the solutions to their satisfaction.

### 6.2.2 Time for task completion

As discussed in Section 6.1.2, two separate measurements were collected: time to supervisor computation and time to completion. The times for different subjects are shown in Table 6.2. Subject I was the only subject who was faster using the classical approach. They explained that they had extensive experience using the IDES software without the TD plugin, while the template design interface was very new to them.

In order to assess the difference between means and to analyse the interaction of the conditions, the mixed factorial (one within-subjects, one between-subjects) ANOVA was employed. One of the assumptions of ANOVA is that the data are normally distributed [27]. Using the Shapiro-Wilk test of normality [70], there is significant probability that, under the classical approach condition, the values for the time to supervisor computation and the total time are not sampled from a normal distribution. However, no sufficient evidence against normality is found for the values under the template design approach. Given that both processes (DES problem solving using the classical approach and using template design) are inherently similar, and given the general robustness of the ANOVA test under violations of the normality assumption [26], we decided that this test is indeed applicable in our case. Last but not least, it seems that a distribution-free (non-parametric) test is not available for mixed-design experiments (e.g., no test is suggested in [27], [39] and various other

| Subject | Problem in TD condition | Classical | | Template design | | Difference | |
|---|---|---|---|---|---|---|---|
| | | Model | Total | Model | Total | Model | Total |
| A | Spooler | 649 | 809 | 189 | 209 | 460 | 600 |
| B | Spooler | 541 | 788 | 261 | 480 | 280 | 308 |
| C | Factory | 672 | 726 | 444 | 511 | 228 | 215 |
| D | Factory | 615 | 935 | 446 | 545 | 169 | 390 |
| E | Factory | 754 | 834 | 506 | 579 | 248 | 255 |
| F | Spooler | 676 | 840 | 488 | 692 | 188 | 148 |
| G | Spooler | 1031 | 2035 | 590 | 708 | 441 | 1327 |
| H | Spooler | 795 | 819 | 660 | 735 | 135 | 84 |
| I | Factory | 426 | 582 | 782 | 851 | -356 | -269 |
| J | Spooler | 1483 | 1486 | 1263 | 1267 | 220 | 219 |
| | Mean | 764.2 | 985.4 | 562.9 | 657.7 | 201.3 | 327.7 |
| | Std | 299.06 | 438.01 | 301.09 | 277.21 | 223.61 | 415.61 |

Table 6.2: Times taken by subjects (in seconds): time to supervisor computation (*Model*) and total time (*Total*), under the classical approach condition (*Classical*) and the template design condition (*Template design*). The second column shows which problem was administered under the template design condition (the complementary problem was administered under the classical approach condition). The last two columns show the differences between the times for the two conditions: *Difference-Model* = *Classical-Model* − *TemplateDesign-Model*; *Difference-Total* = *Classical-Total* − *TemplateDesign-Total*. Positive differences indicate shorter time under the template design condition. The mean and standard deviation for each column is displayed at the bottom. The data are sorted according to the total time under the template design condition.

sources).

The results of the mixed factorial (one within-subjects, one between-subjects) ANOVA test are summarized in Tables 6.3 and 6.4. In the tables, of interest are the rows labelled 'Problem', 'Method' and 'Method' $\times$ 'Problem' and the columns labelled $p$ and $\eta^2$. The row labelled 'Problem' describes the effect on the data due to which problem was assigned for the template design condition (factory or spooler). The row labelled 'Method' describes the effect due to the problem solving approach (classical or template design). Finally, the row labelled 'Method' $\times$ 'Problem' describes the

| Source | *SS* | *df* | *MS* | *F* | *p* | $\eta^2$ |
|---|---|---|---|---|---|---|
| **Between-subjects** | | | | | | |
| ‘Problem’ | 91687.408 | 1 | 91687.408 | 0.562 | 0.4748 | 0.050 |
| Subjects within groups | 1304124.042 | 8 | 163015.505 | | | |
| **Within-subjects** | | | | | | |
| ‘Method’ | 202608.450 | 1 | 202608.450 | 9.563 | 0.0148 | 0.111 |
| ‘Method’ $\times$ ‘Problem’ | 55513.008 | 1 | 55513.008 | 2.620 | 0.1442 | 0.030 |
| ‘Method’ $\times$ Subjects within groups | 169496.042 | 8 | 21187.005 | | | |
| **Total** | 1823428.950 | 19 | | | | |

Table 6.3: Results of the mixed factorial (one within-subjects, one between-subjects) ANOVA test of the time to supervisor computation. The (between-subjects) ‘Problem’ factor denotes which problem, factory or spooler, was solved under the template design condition (the corresponding complementary problem was solved under the classical approach condition). The (within-subjects) ‘Method’ factor denotes the used approach, classical or template design.

effect due to the interaction between problem and approach. The $p$ value gives the probability that the measured difference is due to chance and the $\eta^2$ value gives the *effect size*, i.e., how much of the variability in the data is due to the given effect, or what is the impact of the effect. According to [12], if $0.01 \leq \eta^2 < 0.06$ the effect is small, if $0.06 \leq \eta^2 < 0.14$ there is medium effect, and large effect is observed if $0.14 < \eta^2$. A more complete description of the ANOVA table of results can be found for example in [39].

The results shown in Tables 6.3 and 6.4 allow us to make the following conclusions.

- There is significant difference between the times to supervisor computation ($p = 0.0148 < 0.05$) due to the effect of the problem solving approach. Namely, the time to supervisor computation can be expected to be shorter when the template design is used. The size of this effect is medium in comparison to all variability

| Source | $SS$ | $df$ | $MS$ | $F$ | $p$ | $\eta^2$ |
|---|---|---|---|---|---|---|
| **Between-subjects** | | | | | | |
| 'Problem' | 212268.408 | 1 | 212268.408 | 1.189 | 0.3074 | 0.072 |
| Subjects within groups | 1428730.042 | 8 | 178591.255 | | | |
| **Within-subjects** | | | | | | |
| 'Method' | 536936.450 | 1 | 536936.450 | 6.418 | 0.0351 | 0.182 |
| 'Method' × 'Problem' | 107940.008 | 1 | 107940.008 | 1.290 | 0.2889 | 0.037 |
| 'Method' × Subjects within groups | 669336.042 | 8 | 83667.005 | | | |
| **Total** | 2955210.950 | 19 | | | | |

Table 6.4: Results of the mixed factorial (one within-subjects, one between-subjects) ANOVA test of the time to completion. The (between-subjects) 'Problem' factor denotes which problem, factory or spooler, was solved under the template design condition (the corresponding complementary problem was solved under the classical approach condition). The (within-subjects) 'Method' factor denotes the used approach, classical or template design.

in the data ($\eta^2 = 0.111$).

- There is significant difference between the times to completion ($p = 0.0351 < 0.05$) due to the effect of the problem solving approach. Namely, the time to completion can be expected to be shorter when the template design is used. The size of this effect is large in comparison to all variability in the data ($\eta^2 = 0.182$).

- The variability in the data due to the problem assignment, i.e., which problem was solved using which approach, does not reach significant levels ($p > 0.05$ for both the time to supervisor computation and the time to completion). Thus, it is not possible to reject the hypothesis that the observed difference is due to chance. In other words, it seems that the measured differences in performance were not affected by which problem was assigned for the template design condition (and, consequently, for the classical approach condition).

- The variability in the data due to the interaction between problem assignment and approach (row 'Method' × 'Problem'), does not reach significant levels ($p > 0.05$ for both the time to supervisor computation and the time to completion). Thus, it is not possible to reject the hypothesis that the observed difference is due to chance. In other words, it seems that the measured differences in performance did not depend on which problem was solved.

### 6.2.3 Error rate

The solutions of each subject were examined and the error rates were computed using the rubric from Section 6.1.2. Each solution could receive from zero to eight points, where zero points denotes no errors in the solution according to the rubric and eight is the maximal number of penalty points. The results are summarized in Table 6.5.

| | Error rate | |
|---|---|---|
| Subject | Classical approach | Template design |
| A | 3 | 1 |
| B | 1 | 0 |
| C | 0 | 0 |
| D | 0 | 0 |
| E | 0 | 1 |
| F | 0 | 0 |
| G | 1 | 0 |
| H | 0 | 0 |
| I | 0 | 1 |
| J | 3 | 1 |
| Sum | 8 | 4 |

Table 6.5: The error rates for the solutions of the subjects. Zero points denotes no errors; each solution can collect a maximum of eight penalty points.

It is important to note that the nature of this metric is *ordinal*. In other words, the error rate can only be used to order the solutions according to the selected measure of correctness. The amount of difference in correctness between equidistant levels of the measure need not be constant, e.g., the difference in correctness between solutions with error rates 2 and 3 need not be the same as the difference in correctness between solutions with error rates 0 and 1. The ordinality of the data precludes the use of parametric statistical tests such as the $t$-test. Thus, we used the Wilcoxon signed-rank test, a distribution-free test recommended for within-subjects data [27].

The results of the Wilcoxon signed-rank test are $T(6) = 5.00$, $p > 0.05$. Here, the $p$ value gives the probability that the observed difference in the data is due to chance. As this probability is not significantly low, it seems that the use of different approaches to DES problem solving did not produce an observable effect.

### 6.2.4 Subjective evaluation of the experience

The data from the questionnaires in Appendix B are summarized in Table 6.6. The table does not include the answers to the open-ended questions. All subjects reported that they completed both solutions to their satisfaction. As well, all subjects reported that they preferred the template design methodology over the classical approach.

The Wilcoxon signed-rank test reveals that there is significant difference ($p < 0.05$) in the answers only to the question "How easy was it to *apply* the problem solving methodology which you used?". Furthermore, the effect of the difference in approaches on the ease of application is large, as measured by $r$. According to [12], if $0.1 \leq r < 0.3$ the effect is small, if $0.3 \leq r < 0.5$ there is medium effect, and large effect is observed if $0.5 < r$.

The open-ended questions cannot be analysed directly using similar statistical

| | Confidence in model | | Confidence in supervisor | | Easy to learn approach | | Easy to apply approach | |
|---|---|---|---|---|---|---|---|---|
| Subject | C | TD | C | TD | C | TD | C | TD |
| A | 3 | 4 | 2 | 4 | 5 | 5 | 2 | 5 |
| B | 4 | 4 | 4 | 4 | 5 | 5 | 4 | 5 |
| C | 5 | 5 | 5 | 5 | 4 | 4 | 3 | 4 |
| D | 5 | 4 | 5 | 4 | 4 | 5 | 4 | 5 |
| E | 3 | 3 | 5 | 5 | 5 | 5 | 4 | 4 |
| F | 4 | 5 | 4 | 5 | 5 | 4 | 4 | 5 |
| G | 3 | 5 | 5 | 5 | 4 | 5 | 5 | 5 |
| H | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 5 |
| I | 4 | 4 | 4 | 4 | 4 | 4 | 3 | 5 |
| J | 4 | 4 | 4 | 4 | 4 | 5 | 2 | 5 |
| Sum | 39 | 42 | 42 | 44 | 44 | 46 | 34 | 48 |
| Wilcoxon signed-rank test | | | | | | | | |
| | T(4)=2.00 $p > 0.05$ | | T(3)=1.50 $p > 0.05$ | | T(4)=2.50 $p > 0.05$ | | T(8)=0 $p = 0.0039$ $r = 0.5949$ | |

Table 6.6: The answers of the subjects to the scaled questions from the questionnaires in Appendix B, after the classical approach condition (C) and the template design condition (TD). The scale used is from one (very little) to five (very much).

tools. However, we aggregated similar responses into corresponding categories, in order to present the answers in a compact form. The summary of responses is shown in Table 6.7. The relative difference included in the table gives an idea of which features of the TD plugin were rated relatively favorably (high value) and relatively unfavorably (low value). If the Wilcoxon signed-rank test is used to analyse the responses, the difference of ease for the two approaches does not reach a significant level ($p > 0.05$). Thus, the data can be used only as indicators. According to the subjects, the biggest advantages of template design are the way event synchronization and specification self-loops are treated, the creation of new models, copying of models, and the fact that the design provides a better overview of the situation. The single,

| Category | Classical **E**asy, **D**ifficult | Template design **E**asy, **D**ifficult | Relative difference (in favor of TD) |
|---|---|---|---|
| Call operations | EEEE | DD | -6 |
| Copy models | D | EEEEED | 5 |
| Create supervisors | EE | EE | 0 |
| Event synchronization | DDDDD | EE | 7 |
| Find errors | D | D | 0 |
| Model new models | D | EEEEE | 6 |
| Model models | E | | -1 |
| Modify models | E | EE | 1 |
| Overview of the situation | DD | EEE | 5 |
| Self-loops | DDDDD | EE | 7 |
| User interface | ED | ED | 0 |
| Wilcoxon signed-rank test | | | T(8)=7.00 $p = 0.0742$ |

Table 6.7: Aggregated responses to the two open-ended questions from the questionnaires in Appendix B. For each response to what was difficult, a 'D' is added to the corresponding categories. For each response to what was easy, an 'E' is added to the corresponding categories. In the last column, the relative difference in favor of the template design approach is computed: $\#(E, \text{Template-desing}) - \#(D, \text{Template-design}) - \#(E, \text{Classical}) + \#(D, \text{Classical})$, where $\#(X, Y)$ gives the count of the letter $X$ under the $Y$ condition.

big disadvantage of the TD plugin is that subjects found it difficult to call or interpret the new DES operations.

Finally, the responses of the subjects regarding the contribution of the template design methodology were aggregated under a number of features derived from the answers. The summary is shown in Table 6.8. It can be seen that the most important contribution (according to number of subjects who mention it) is the introduction of high-level structure to the overall model. The other features valued by most subjects are the automatic handling of self-loops in specifications and the ability to use templates. Generic features such as automation, speed of modelling, reduced risk of

| Feature | Count |
|---:|:---:|
| High-level structure | 7 |
| Handling of self-loops in specifications | 5 |
| Templates | 3 |
| Automation of modelling | 2 |
| Modelling is less error-prone | 2 |
| Speed of modelling | 2 |
| Convenient user interface | 2 |

Table 6.8: Counts of how many subjects mentioned a given feature as a contribution of the template design methodology.

making errors and user interface are not referred to as frequently.

### 6.2.5 System Usability Scale

Each subject was administered the System Usability Scale questionnaire after completing the task under the template design approach. The results are summarized in Table 6.9.

| Subject | SUS score |
|:---:|:---:|
| A | 85 |
| B | 95 |
| C | 92.5 |
| D | 77.5 |
| E | 82.5 |
| F | 92.5 |
| G | 92.5 |
| H | 65 |
| I | 82.5 |
| J | 92.5 |
| Mean | 85.75 |
| Std | 9.36 |

Table 6.9: SUS scores for IDES with the TD plugin.

The SUS results were compared with the data from [79] (mean=66.41, std=12.97) using the two-tailed $t$-test with the Welch adjustment for unequal variances [91]. The difference in means is significant, $t(14.46) = 5.315$, $p = 0.0006 < 0.05$, $r = 0.813$. Thus, it is possible to conclude that the usability of IDES with the TD plugin, as measured via SUS, is higher than the average usability in the various software packages from the survey [79].

## 6.3 Discussion and Conclusions

The collected experimental data and its analysis allowed us to evaluate more rigorously the usability of the implementation of the template design methodology. Going back to the hypotheses from Section 6.1.4, we can now say the following.

- There is significant evidence that the total time for task completion is shorter using the template design methodology in comparison to the classical approach. Furthermore, the observed effect is large.
- There is significant evidence that the time for modelling during a task (i.e., "time to supervisor") is shorter using the template design methodology in comparison to the classical approach. The observed effect is of medium size.
- There is no significant evidence that fewer errors are made using the template design methodology in comparison to the classical approach.
- There is no significant evidence that the template design methodology results in higher confidence in the models users produce, in comparison to the classical approach.

- There is no significant evidence that the template design methodology results in lower confidence in the supervisory solution users obtain, in comparison to the classical approach.
- There is significant evidence that the template design methodology is (experientially) easier to use in comparison to the classical approach. The observed effect is large.
- There is no significant evidence that the template design methodology is (experientially) easier to learn in comparison to the classical approach.
- All subjects in the study showed preference for the template design methodology over the classical approach.
- The average SUS score for the TD plugin is not lower than the average SUS score reported in the survey [84, 79]. Furthermore, there is significant evidence that the subjects rated the usability (as measured with SUS) of IDES with the TD plugin higher than the average usability of the pool of software in the survey.

Anecdotal analysis of the errors which subjects committed reveals that the majority of mistakes were committed while modifying the existing models (e.g., the model for robot 1 in the factory problem)—7 out of 12. Similarly, most mistakes involved setting incorrect controllability—6 out of 12. In the experience of the experiment conductor, the subjects appeared to forget about setting controllability, rather than explicitly making the wrong choice. Two of the errors committed involved designing specifications to alternate robots 2 and 3 in the factory problem, rather than robots 1 and 3. Again, this points to lack of attention to the problem description, rather than an inherent lack of understanding of how to solve the problem. It is possible to expect that all the errors discussed here will have similar impact regardless of the problem solving approach used. Thus, it should not be a surprise that no significant difference in error rate was found between the two approaches. A different and more focused experiment may be needed to reveal any effects of the approaches on the error rate.

As mentioned in the beginning of this chapter, one of the major problems regarding the setup of the experiment was the lack of experience of the subjects. As indicated, the vast majority of subjects had very little prior experience with the template design methodology, while they had relatively more experience with other DES software and with IDES without the TD plugin. Put another way, subjects were more experienced with the classical approach than with the template design methodology. In light of this, it may not be possible to compare directly the scores of experiential learnability reported by the subjects for the two approaches. The same issue has to be considered also for the other measures. The better performances recorded and the higher ratings given under the template design condition become much more impressive if one considers that many users had only a few hours of experience with the TD plugin, solving one or two problems, prior to their participation in the study.

In this evaluation, the data from two subjects were not considered because the times they took to produce their solutions were judged outliers. It is worth discussing in more detail the activity of one of these subjects. This subject was assigned the factory problem under the template design condition. As already mentioned, they decided that it was not sufficient to guarantee the alternation of the operation of robots 1 and 2 and robots 1 and 3. In addition to this, the subject thought that it was necessary to prevent the concurrent operation of robots 2 and 3 since they would conflict when placing chips on the same circuit board. This is clearly a misinterpretation of the problem description where it is explicit that "it is sufficient to add one more coordinator to guarantee that the outputs of robots 1 and 3 alternate as well". The subject spent much time and effort trying to figure out how to accomplish the additional goal by modifying the two coordinators. This issue was resolved when the subject realized that the template library available to them contains the "MUTEX" template (a template which can be used to specify mutual exclusion). This template was included in the library because it is used in the spooler problem. The subject quickly instantiated the template, connected it to the modules for robots 2 and 3 and, upon computing the supervisory solution, immediately achieved the goal they had in mind. This development was not foreseen in the design of the study, and the subject was not originally aware that such a template would be available to them. A big motivating factor for the development of the template design methodology was to allow the encapsulation of useful system or specification behavior, and enable the easy reuse across projects. Even though the described incident could not be included in the planned analysis of usability, we believe that it demonstrates clearly the envisioned advantage of the proposed methodology.

In summary, it is possible to conclude that, according to this evaluation, the overall usability of the template design implementation is good—as demonstrated by the preference of the subjects, the rating of the experiential ease of use, and the higher-than average SUS scores. We observed one objective advantage of the template design methodology, that is, the increase of speed (both for task completion and for modelling). The increase of speed does not seem to come at a cost to the users, i.e., there is no significant difference in the error rate, confidence in the solution, or the experience of learning how to use it. This is in contrast to other attempts to design software according to observations of problem solving, e.g., in [71] the authors record improvement of the objective measures of performance, but the experiential ratings the users report deteriorate.

In most of the aspects of usability measured for this evaluation, no significant difference was found between the classical and template design approaches. However, in all measures, on average there is at least a small advantage for the template design methodology. Altogether, this is a indicator that the template design approach is a better tool than the classical approach. Further experiments are needed to investigate the advantages and disadvantages of the methodology. Most importantly, it is necessary to investigate the (objective) learnability of the implementation, and compare the performances of experts in both approaches.

Lastly, looking at the contributions of the template design methodology mentioned by the subjects, in Table 6.8, one can see that the subjects have recognized features which correspond to the main goals of the methodology. Subjects mention automation, robustness, speed, and convenience. Furthermore, as already hinted at in Section 5.2 and [31], it seems that more subjects see value in the high-level structure in the template design, rather than the availability of templates. As the problems used for this study were very simple, the question remains open if under real applications templates will prove to be as beneficial as having a structured environment, or will remain a convenience factor. The anecdotal evidence regarding the unexpected use of the “MUTEX” template suggests that, indeed, the availability of templates could be of more aid than demonstrated in this evaluation.

# Chapter 7

# Conclusions

This work described the research conducted in the quest for designing better DES software. The exploratory observational study of solving DES control problems, [32], served as a precursor. The think-aloud data collected from the subjects helped us develop a list of recommendations on how to design and improve DES software. These observations, together with other relevant research, led to the proposal of the template design methodology for DES problem solving. This methodology does not require the introduction of new control theory; it is rather a reinterpretation of the existing modelling framework. Software supporting this methodology was implemented and subsequently evaluated using twelve subjects. Significant improvements in the speed of problem solving as well as positive evaluations by the subjects were observed. The usability data do not show any drawbacks to applying the methodology. According to the subjects, the biggest benefit of template design is the support of conceptual modelling. There is some indication that the encapsulation of DES behavior, in the form of templates, could also prove advantageous in certain circumstances. In summary, we managed to accomplish the goals motivating this work. The insights gained from the observation of DES problem solving helped us design better DES software. We believe that practitioners of DES supervisory control will benefit from using the new methodology.

## 7.1 Other Lessons

Beyond the conclusions drawn explicitly in the previous chapters, we mention next a few other lessons learned from this research.

The review of literature from the fields of both cognitive psychology and human-computer interaction revealed that often usability research is conducted without sufficient understanding of human cognitive processes, and without the rigor of psychological research. Acquiring at least an overview of cognitive psychology research, e.g., by reading Anderson's book [2], might prove to be beneficial for HCI studies. Naturally, usability research focuses predominantly on human behavior, however, knowledge of the cognitive processes resulting in observed behavior will make it easier to predict the impact of interface design decisions. On the other hand, we found it hard to establish the link between theoretical research on cognition and its practical application, such as the investigation of solving complex, ill-defined problems. The relevant research we found tends to focus on the examination of very small aspects of cognition which do not necessarily lead to a better understanding of the overall processes or experiences. Authors normally do not discuss how the investigated cognitive processes manifest in "real life". Furthermore, many procedures employed in such research remain vaguely described or seem applicable only within the particular setup of an experiment. For example, most descriptions of the protocol analysis technique focus on how to conduct think-aloud studies, but are very brief on how to analyse the data afterwards.

No standardized approach is discussed which could be useful for usability research.

On numerous occasions authors in usability have pointed out that observations of the real users of a system are indispensable during the design of the system, e.g., [83]. During the interactions observed between subjects and the IDES software, we have been repeatedly surprised by the innovative ways which subjects found to complete their tasks. These observations both confirm the maxim that someone is bound to make a mistake if it is possible (as all possible ways to interact with a product actually get explored), and call for the design of software which gives the maximal degree of freedom in the form of interaction (as the approaches of different users vary markedly). We recommend that designers of software never forgo, if at all possible, usability testing with real users.

The development of the IDES software, and the template design plugin, gave us much experience with the design and architecture of software. With the risk of being challenged, we will extol the virtues of simple interfaces between modules even if such interfaces seem inadequate for every conceivable situation. Complex interfaces not only become cumbersome; they offer more chances to make errors, they render the programs hard to understand, and they are much more challenging to learn. Historically, a similar lesson can be gained from the failures of CORBA [40]. We believe that the flexibility of the interfaces may be improved with approaches such as the object annotation mechanism in IDES (as detailed in Section 5.3.1), instead of using complex interfaces. Indeed, the annotation mechanism may result in “messy” programs, however, it increases the extensibility of software and inspires creativity. Especially in research applications, such approaches should be promoted rather than stifled in the pursuit of stability or security.

## 7.2 Future Work

There are many limitations to the studies and analyses presented in this work. To gain a better and more detailed understanding of the cognitive processes in DES problem solving, it is necessary to conduct many more observational studies, focusing on different aspects of the task. For example, in [32], the verification stage of problem solving, i.e., when subjects make sure that their solutions are correct, was not explored in depth. Similarly, our usability evaluation did not examine the learnability of the template design implementation, nor did the experimental setup allow for discerning if there is a difference in the error rate when using the new methodology. We believe that a longitudinal study of template design would bear the most fruit, e.g., the comparison between the template design and the classical approach in a semester-long undergraduate project.

As pointed out in Section 4.2.3, the template design methodology can be improved by the incorporation of a mechanism to parametrize templates. This is only one specific way to extend the methodology. More generally, one can consider the fact that the template design approach makes very few assumptions about the underlying low-level framework (which, in our case, used finite-state automata). We believe that it should not pose a problem to employ the approach using a different low-level framework, e.g., that of Petri nets. In template design, it is only assumed that there is a way to define discrete-event behavior formally (i.e., the models that underly the module and channel entities in the design), and an algorithm to compute supervisory control according to the channel specifications. It is conceivable that even mixed-framework designs could be used where modules and channels may be modelled using a variety of techniques. In order to enable the use of such approaches, it is necessary to develop the supervisory control algorithms which will act upon heterogeneous models.

Finally, the template design methodology addresses only some of the recommendations stemming from the observational study on problem solving. To us, a number of other recommendations seem equally important for the development of usable (and useful) DES software. The key areas where we currently see deficiencies are the following:

- Tools for the verification of solutions. Better visualizations are needed, such as specialized layout algorithms when the models are small, or means to explore models of thousands of states. Better automation is needed, such as automated string tracing or simulation environments that are oriented to DES control.

- Application of theoretical solutions. The ability to compute optimal supervisors is worthless if one cannot use the results for some other purposes, e.g., for publications, presentations, group work and, most importantly, the control of systems. We are bewildered that for most DES software, obtaining the supervisor in a form suitable for the control of real systems is non-trivial at best, and not possible at all at worst. Here we should acknowledge that, unfortunately, the template design implementation falls in the latter category. The generation of output such as Programmable Logic Controller code has not been implemented yet.

- Openness to and integration with other approaches to DES control. Most research in DES supervisory control has been focused on the framework proposed by Ramadge and Wonham [67]; however, observation of the work of subjects reveals that the finites-state automaton is not always the most suitable type of model for giving specifications or for supervision. The use of inequalities and temporal logic for specifications are among the alternatives which most easily come to mind. Thinking further one can consider that, in the modern world, systems often operate in dynamic and uncertain environments. It seems that other approaches to control, such as online control [11], or stochastic control [50] may be more suitable. Finally, there has been a staunch reluctance in the supervisory control community to give programmatic control (i.e., where the specifications are given in the form of an algorithm whose output is to be determined at runtime) a place within the DES universe. Especially in online control, at each step, small control programs can take advantage of up-to-date data to make control decisions. Programmatic control can also employ heuristic evaluations or machine learning algorithms. Naturally, accepting programmatic control will undoubtedly render many theoretical results in DES inadequate or inappropriate, as most of the theories assume static systems, precomputed control, or specifications from a restricted class of linguistic complexity (e.g., only regular or context-free specifications). However, it is our belief that programmatic control is unavoidable in practice, especially as the controllers in most new equipment are in the form of embedded computers. Thus, programmatic control has to be embraced and its theoretical implications investigated.

Looking at all the potential venues in which this work can be extended, we feel most of all excited. There are countless ways in which to make DES theory more applicable and useful, and only a few of these paths have been explored.

# Bibliography

[1] 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.

[2] J. R. Anderson. *Cognitive Psychology and Its Implications.* Worth Publishers, New York, New York, USA, sixth edition, 2005.

[3] 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.

[4] F. Basile and P. Chiacchio. On the implementation of supervised control of discrete event systems. *IEEE Transactions on Control Systems Technology*, 15(4):725–739, 2007.

[5] G. Berry and G. Gonthier. The Esterel synchronous programming language: Design, semantics, implementation. *Science of Computer Programming*, 19(2):87–152, 1992.

[6] B. A. Brandin, R. Malik, and P. Malik. Incremental verification and synthesis of discrete-event systems guided by counter examples. *IEEE Transactions on Control Systems Technology*, 12(3):387–401, May 2004.

[7] J. Brooke. *Usability Evaluation in Industry*, chapter SUS: a 'quick and dirty' usability scale, pages 189–194. Taylor and Francis, 1996.

[8] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. An empirical comparison of pie vs. linear menus. In *Proceedings of the ACM CHI Conference on Human Factors in Computing Systems*, pages 95–100, 1988.

[9] 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.

[10] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Springer, second edition, 2007.

[11] 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.

[12] J. Cohen. *Statistical power analysis for the behavioral sciences.* Lawrence Erlbaum Associates, second edition, 1988.

[13] L. Cosmides. The logic of social exchange: Has natural selection shaped how humans reason? Studies with the Wason selection task. *Cognition*, 31:187–276, 1989.

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

[15] C. de Oliveira, J. E. R. Cury, and C. A. A. Kaestner. Discrete event systems with guards. In *Proceedings of the 11th IFAC Symposium on Information Control Problems in Manufacturing*, volume 1, pages 90–95, Salvador, Brazil, 2004.

[16] 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.

[17] M. H. de Queiroz and J. E. R. Cury. Synthesis and implementation of local modular supervisory control for a manufacturing cell. In *Proceedings of the 6th International Workshop on Discrete Event Systems (WODES'02)*, pages 377–382, Zaragoza, Spain, October 2002.

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

[19] K. Edlund, A. G. Michelsen, and K. Rudie. Supervisory control of flowlines by modelling the legal language as inequalities. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 15–20, Ann Arbor, Michigan, USA, 2006.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multi-scale interfaces. In *Proceedings of Human Factors in Computing Systems (CHI 95)*, pages 234–241, Denver, CO, USA, 1995.

[26] G. V. Glass, P. D. Peckham, and J. R. Sanders. Consequences of failure to meet assumptions underlying the fixed effects analyses of variance and covariance. *Review of Educational Research*, 42(3):237–288, 1972.

[27] J. A. Gliner and G. A. Morgan. *Research methods in applied settings: an integrated approach to design and analysis.* Lawrence Erlbaum Associates, 2000.

[28] The Grail environment for supervisory control of discrete event systems. Department of Automation and Systems, Federal University of Santa Catarina, Brazil. Available at http://www.das.ufsc.br/~cury/grail.html.

[29] A. G. Greenwald. Within-subjects designs: To use or not to use? *Psychological Bulletin*, 83(2):314–320, 1976.

[30] L. Grigorov. Hierarchical control of discrete-event systems. Survey paper, School of Computing, Queen's University, Canada, 2005. Available at http://www.cs.queensu.ca/~grigorov/.

[31] L. Grigorov. Template design of discrete-event systems. Technical report 2007-538, School of Computing, Queen's University, Canada, 2007.

[32] L. Grigorov. Observations on solving discrete-event control problems: patterns and strategies. Technical report 2009-558, School of Computing, Queen's University, Canada, 2009.

[33] L. Grigorov, J. E. R. Cury, and K. Rudie. Design of discrete-event systems using templates. In *Proceedings of the American Control Conference 2008*, pages 499–504, Seattle, WA, USA, June 2008.

[34] L. Grigorov, J. E. R. Cury, K. Rudie, and S. Klinge. Template design and automatic generation of controllers for industrial robots. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, pages 1612–1613, Fortaleza, Ceará, Brazil, March 2008.

[35] 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.

[36] L. G. Grigorov. Control of dynamic discrete-event systems. Master's thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2004.

[37] P. L. Guernic, T. Gautier, M. L. Borgne, and C. L. Maire. Programming real-time applications with Signal. *Proceedings of the IEEE*, 79(9):1321–1336, 1991.

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

[39] S. A. Haslam and C. McGarty. *Research Methods and Statistics in Psychology*. SAGE Publications, 2003.

[40] M. Henning. The rise and fall of CORBA. *ACM Queue*, 4(5):29–34, 2006.

[41] L. E. Holloway, X. Guan, R. Sundaravadivelu, and J. Ashley, Jr. Automated synthesis and composition of taskblocks for control of manufacturing systems. *IEEE Transactions on Systems, Man, and Cybernetics: Part B*, 30(5):696–712, 2000.

[42] K. Hornbæk. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, 64:79–102, 2006.

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

[44] JavaScript programming language.
Documentation available at http://developer.mozilla.org/en/JavaScript.

[45] P. N. Johnson-Laird. *Mental Models: Towards a Cognitive Science of Language, Inference, and Consciousness.* Harvard University Press, Cambridge, Massachusetts, USA, 1983.

[46] P. N. Johnson-Laird. The shape of problems. In *The Shape of Reason: Essays in Honour of Paolo Legrenzi*, pages 3–26. Psychology Press, New York, USA, 2005.

[47] P. N. Johnson-Laird, P. Legrenzi, V. Girotto, M. S. Legrenzi, and J.-P. Caverni. Naive probability: A mental model theory of extensional reasoning. *Psychological Review*, 106:62–88, 1999.

[48] P. N. Johnson-Laird, P. Legrenzi, and M. S. Legrenzi. Reasoning and a sense of reality. *British Journal of Psychology*, 63:395–400, 1972.

[49] R. Kumar and V. K. Garg. *Modeling and Control of Logical Discrete Event Systems.* Kluwer Academic Publishers, Norwell, MA, USA, 1995.

[50] R. Kumar and V. K. Garg. Control of stochastic discrete event systems modeled by probabilistic languages. *IEEE Transactions on Automatic Control*, 46(1):593–606, 2001.

[51] R. J. Leduc. *Hierarchical Interface-based Supervisory Control.* PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 2002.

[52] A. Lefford. The influence of emotional subject matter on logical reasoning. *Journal of General Psychology*, 34:127–151, 1946.

[53] R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.

[54] F. Lin and H. Ying. Modeling and control of fuzzy discrete event systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(4):408–415, August 2002.

[55] E. Lucas. *Récréations mathématiques*, volume 3. Gauthier-Villars, Paris, France, 1883.

[56] 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.

[57] C. Ma and W. M. Wonham. Control of state tree structures. In *Proceedings of the 11th Mediterranean Conference on Control and Automation*, Rhodes, Greece, June 2003. Paper T4-005.

[58] I. S. MacKenzie, A. Seller, and W. Buxton. A comparison of input devices in elemental pointing and dragging tasks. In *Proceedings of the ACM CHI'91 Conference on Human Factors in Computing Systems*, pages 161–166, 1991.

[59] H. Marchand, P. Bournai, M. L. Borgne, and P. L. Guernic. Synthesis of discrete-event controllers based on the signal environment. *Discrete Event Dynamic Systems: Theory and Applications*, 10(4):325–346, 2000.

[60] J. Metcalfe and D. Weibe. Intuition in insight and noninsight problem solving. *Memory and Cognition*, 15:238–246, 1987.

[61] J. O. Moody and P. J. Antsaklis. *Supervisory Control of Discrete Event Systems Using Petri Nets*. Kluwer Academic Publishers, 1998.

[62] I. B. Myers, M. H. McCaulley, N. L. Quenk, and A. L. Hammer. *MBTI Manual (A guide to the development and use of the Myers Briggs type indicator)*. Consulting Psychologists Press, third edition, 1998.

[63] D. A. Norman. Some observations on mental models. In *Mental Models*, pages 7–14. Lawrence Erlbaum Associates, 1983.

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

[65] A. F. Osborn. *Applied Imagination: Principles and Procedures of Creative Problem-Solving*. Charles Scribner's Sons, New York, USA, third revised edition, 1963.

[66] J. W. Osborne and A. Overbay. The power of outliers (and why researchers should always check for them). *Practical Assessment, Research & Evaluation*, 9(6), 2004.
Retrieved May 5, 2009 from http://PAREonline.net/getvn.asp?v=9&n=6.

[67] 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.

[68] P. J. Ramadge and W. M. Wonham. The control of discrete event systems. In *Proceedings of the IEEE*, volume 77, pages 81–98, January 1989.

[69] J. Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional, 2000.

[70] D. G. Rees. *Essential statistics*. CRC Press, fourth edition, 2001.

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

[72] K. Rudie. Control of discrete-event systems. Lectures for the ELEC843 course at the Department of Electrical and Computer Engineering, Queen's University, Canada, 2002.

[73] 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.

[74] E. A. P. Santos, J. E. R. Cury, and V. J. D. Negri. Modelagem das especificações operacionais de sistemas de manipulação e montagem automatizados. In *Símposio Brasileiro de Automação Inteligente*, pages 144–149, Bauru, São Paulo, Brazil, 2003.

[75] E. A. P. Santos, V. J. D. Negri, and J. E. R. Cury. A computational model for supporting conceptual design of automatic systems. In *Proceedings of 13th International Conference on Engineering Design*, pages 517–524, Glasgow, UK, August 2001.

[76] D. Shewa, J. Ashley, and L. Holloway. Spectool 2.4 Beta: A research tool for modular modeling, analysis, and synthesis of discrete event systems. In *Proceedings of the 8th International Workshop on Discrete Event Systems*, pages 477–478, Ann Arbor, Michigan, USA, July 2006.

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

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

[79] SUS scores from 129 conditions in 50 studies. Spreadsheet available at http://measuringuserexperience.com/ (April 2009).

[80] 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.

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

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

[83] B. Tognazzini. *Tog on Interface.* Addison-Wesley Publishing Company, Inc., 1992.

[84] T. Tullis and B. Albert. *Measuring the User Experience.* Morgan Kaufmann Publishers, Burlington, MA, USA, 2008.

[85] T. S. Tullis and J. N. Stetson. A comparison of questionnaires for assessing website usability. In *Usability Professionals Association Conference*, Minneapolis, MN, USA, June 2004.

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

[87] M. Van Selst and P. Jolicoeur. A solution to the effect of sample size on outlier elimination. *The Quarterly Journal of Experimental Psychology Section A*, 47(3):631–650, 1994.

[88] G. Wallas. *The art of thought.* Harcourt, Brace and Company, New York, USA, 1926.

[89] P. C. Wason. On the failure to eliminate hypotheses in a conceptual task. *Quarterly Journal of Experimental Psychology*, 12:129–140, 1960.

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

[91] B. L. Welch. The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29:350–362, 1938.

[92] W. A. Wickelgren. *How to Solve Problems.* W. H. Freeman and Co., San Francisco, CA, USA, 1974.

[93] W. M. Wonham. Supervisory control theory: Models and methods. Informal talk at Queen's University, a version for the 24th International Conference on Application Theory of Petri Nets is available at
http://www.control.toronto.edu/DES/publish.html, 2003.

[94] W. M. Wonham. Supervisory control of discrete-event systems. Available at http://www.control.toronto.edu/DES/, June 2008.

[95] W. M. Wonham and P. J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal on Control and Optimization*, 25(3):637–659, 1987.

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

[97] 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.

# Appendix A

# Problem Definitions

## A.1 Practise Problem

This is a practise problem to remind you how to solve DES problems with IDES and to help you improve your skills before the experimental session.

The problem is described as follows. There is a lady who lives with two dogs, Toby and Ralf. Each dog has its own room where he spends the night, and he can go in or out of his room. The dogs cannot open the doors to their rooms, so the woman can stop them from going in or out. The food is served in the common area of the house, so the dogs can only eat when they are outside their rooms. Ralf is a very good dog and listens to the lady. She can tell him how much to eat. Toby is also a good dog, but when it comes to food, he is uncontrollable. He can eat and eat as long as there is food. Since Toby is so greedy, and could potentially eat Ralf's share, the lady decided that each day she must make sure that it is Ralf who eats first (he gets only one share), and it is Toby who eats second (he can eat as much as he wants as she cannot control him). Before any one of the dogs goes back into his room for the night, both dogs must have eaten (Ralf once and Toby at least once). Both dogs need to go to their rooms for the night.

Your tasks are:

- Create the models for the two dogs.
- Create the model for the control specification (Ralf eats first, Toby second; both go to sleep but not before both having eaten).
- Compute the supervisory solution for the problem.

First, solve the problem with the *classical approach*, without using a Template Design.

Then, solve the same problem using the *Template Design methodology*.

You are encouraged to consult the conductor of the experiment if you need clarification and/or assistance.

## A.2 Problem 1: Factory Problem

**Provide a discrete-event control solution to the problem of "Electronics factory"**

The initial situation is described as follows. There is a factory for electronic components with two robots. Each robot can start processing a component and finish processing it. There is control only over when the robots start processing components. Robot 1 produces circuit boards, while robot 2 produces chips that are fitted onto the boards further down the line. A circuit board is required when robot 2 outputs a chip. Thus, a coordinator is in place which makes sure that the robots alternate in producing boards and chips.

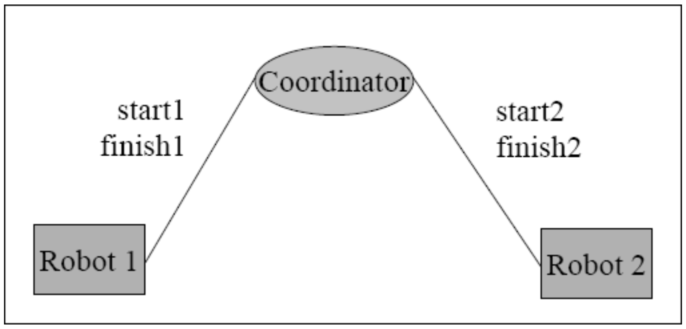
The new situation is described as follows. The factory has been modified to fit two different chips on the same board. Robot 1 has received an upgrade. During processing, it can detect defects in the circuit boards. Any time it detects a misaligned hole, it can perform an additional corrective procedure (redrilling). As well, a third robot has been installed to produce the second kind of chip. Robot 3 is of the same type as robot 2 (it can start and finish processing). Further down the line, each circuit board is fitted with one chip from robot 2 and one chip from robot 3. To guarantee correct operation of the factory, it is sufficient to add one more coordinator to guarantee that the outputs of robots 1 and 3 alternate as well.

Your tasks are the following.

- Model robot 3 and update the model of robot 1.
- Create the specification for the coordination of robots 1 and 3.
- Compute the supervisory solution for the system.
- Verify the correctness of the supervisory solution.

When you have completed all of the above tasks, please announce that you are ready. In case you decide to stop before completing all tasks, please also make an announcement.

**Note:** The models for robots 1 and 2 and for the coordination between the two robots are provided to the subjects who solve this problem. Under the template design condition, the template design for the problem is provided as well; otherwise the following, similar printed conceptual diagram is provided.



## A.3 Problem 2: Spooler Problem

**Provide a discrete-event control solution to the problem of "Device coordinator"**

The initial situation is described as follows. There is a computer network with two clients (workstations), a printer and a fax machine. Each client can request access to the printer and release the printer when done printing. There is control only over when the clients are allowed to access the printer but not over when they release the device. In order to avoid mingled printing jobs, there is a coordinator (spooler) in place which makes sure that the two clients do not acquire access to the printer at the same time.

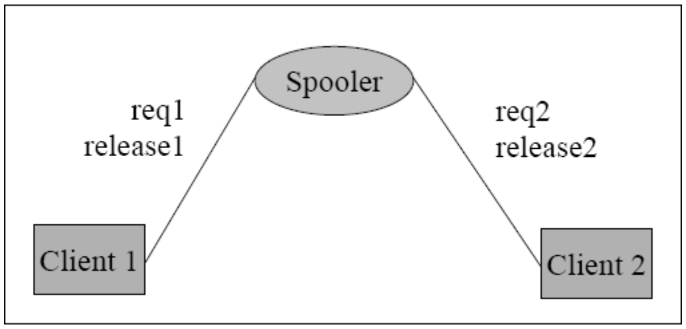
The new situation is described as follows. A third client has been added to the network, and the third client can request access to the fax machine and release the fax machine when done transmitting data—similar to the operation of the other two clients on the printer. As well, client 2 has been given permission to use the fax machine in addition to the printer. Client 2 can now request either access to the printer (and release it when done printing) or to the fax machine (and release it when done transmitting data). Again, there is control only over the requests but not over when the devices are released. Since it is necessary to guarantee the consistency of transmitted data through the fax machine, a new coordinator is required. Similar to the printer spooler, it has to prevent simultaneous access to the fax machine by clients 2 and 3.

Your tasks are the following.

- Model client 3 and update the model of client 2.
- Create the specification for the coordination of clients 2 and 3.
- Compute the supervisory solution for the system.
- Verify the correctness of the supervisory solution.

When you have completed all of the above tasks, please announce that you are ready. In case you decide to stop before completing all tasks, please also make an announcement.

**Note:** The models for clients 1 and 2 and for the printer spooler are provided to the subjects who solve this problem. Under the template design condition, the template design for the problem is provided as well; otherwise, the following, similar printed conceptual diagram is provided.



# Appendix B

# Questionnaires

## B.1 Feedback Questionnaire — Task 1

Please provide us with more information about your experience in solving the DES problem.

Did you complete the solution to your satisfaction? [Yes/No]

Answer the following questions on a scale from 1 (very little) to 5 (very much).

How confident are you in the correctness of your model?
[1 2 3 4 5]

How confident are you in the correctness of the automatically generated supervisors?
[1 2 3 4 5]

How easy was it to *learn* the problem solving methodology which you used?
[1 2 3 4 5]

How easy was it to *apply* the problem solving methodology which you used?
[1 2 3 4 5]

What difficulties did you encounter during the process of problem solving?

What aspects of the problem solving process were easy to accomplish?

## B.2 Feedback Questionnaire — Task 2

Please provide us with more information about your experience in solving the DES problem.

Did you complete the solution to your satisfaction? [Yes/No]

Answer the following questions on a scale from 1 (very little) to 5 (very much).

How confident are you in the correctness of your model?
[1 2 3 4 5]

How confident are you in the correctness of the automatically generated supervisors?
[1 2 3 4 5]

How easy was it to *learn* the problem solving methodology which you used?
[1 2 3 4 5]

How easy was it to *apply* the problem solving methodology which you used?
[1 2 3 4 5]

What difficulties did you encounter during the process of problem solving?

What aspects of the problem solving process were easy to accomplish?

Which methodology for solving DES control problems would you use in the future, given the choice?
[Classical approach/Template Design]

In your opinion, what is the biggest contribution, if any, of the Template Design methodology to DES problem solving?