Discrete Event Dynamic Systems manuscript No.
(will be inserted by the editor)

Conceptual design of discrete-event systems using templates

Lenko Grigorov · Brian Butler · José E. R. Cury · Karen Rudie

April 16, 2010

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, together with other relevant research, led to the proposal of a novel approach to DES problem solving called 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. It provides a high-level overview of a DES design, and facilitates the use of template models. Software supporting this methodology was implemented as an extension to existing DES software. The methodology and the implementation were 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.

1 Introduction

In the early 1980s, Ramadge and Wonham proposed a framework for the development of controllers for a class of systems, called discrete-event systems (DESs) [35]. Much of the related research since then has focused on designing feasible methods for the computation of such controllers, or supervisors. Indeed, the computations may involve enormous discrete sets which contain more than 10^6 elements [31]. The use of algorithmic tools is indispensable in this area; these computations are clearly outside the capabilities of human agents (if one is to keep the expenses reasonable). However, the sole implementation of algorithms proves to be insufficient to aid in real-world problem

L. Grigorov* and K. Rudie
Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario, Canada
E-mail: lenko.grigorov@banica.org*

B. Butler
Department of Psychology, Queen's University, Kingston, Ontario, Canada

J. E. R. Cury
Department of Automation and Systems, Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil

solving. In [13], the authors evaluate the pioneering DES software, TCT (now called CTCT, [7]). It is determined that the usability of its elementary interface is very low, despite the excellent implementation of the DES algorithms. The next generation of DES tools include graphical interfaces and the development has been driven by the need of users to visualize better the models and operations they work with. For example, the Desco software [14] and later the Supremica software [1] were developed trying to address issues surrounding the application of the DES supervision. The IDES software developed at K. Rudie's research laboratory, [39], focuses on usability and also offers a graphical user interface. However, it seems that having a graphical environment is insufficient in resolving all problems with the application of DES theory. The modelling, even when done graphically, is still much too sensitive to errors. Even a single error in one of the models may render the whole solution of a control problem incorrect. Adding to this complication, in most cases the solutions to problems are too large to be comprehended in their entirety, and thus verification becomes very hard. Lastly, even if a correct (or desired) solution is obtained, due to the specificity of models and events, it is not simple to reuse the solution in another project. This makes the application of DES control very difficult for humans, even if all underlying functionality is implemented.

In order to 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 [21]. Based on the results from the study and inspired by the work of Santos *et al.* [41], here we propose a new approach to DES problem solving within the standard supervisory control framework. There are two main ingredients in the methodology: high-level conceptual modelling and the availability of templates. We call this approach the *template design* methodology. The high-level design consists of entities and connections between them. Entities are simply finite-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 likelihood 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. Three measures—speed, experiential ease of use, and System Usability Scale scores [3]—showed improvement when the template design methodology was used. The evaluation did not uncover any negative impact to using the proposed methodology in comparison to the classical approach.

The main contributions of our work include:

- Proposal of a novel method, called template design, for the design of DES control solutions,
- Implementation of the software to support the use of the method and
- Evaluation of the proposed method in order to validate it.

The rest of this work is organized as follows. Section 2 contains an overview of relevant research. The template design methodology is described in Section 3, and the subsequent implementation in Section 4. Then, in Section 5, we discuss the results of the usability evaluation of the template design software tool. We conclude with Section 6. The appendix at the end lists the problems and questionnaires administered during the evaluation from Section 5.

Some parts of this work have already been published in [25,22,23]. Additional details of the work can be found in [24,21,20].

2 Background Review

The theoretical framework proposed by Ramadge and Wonham [35] 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 [12], 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 [12], 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 Section 3.

The work of Holloway *et al.* on condition systems, e.g., [27], 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 [42]. 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 in Section 3 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 [9], [29] and [2], 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 [19]. 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. Furthermore, improved methods for checking the nonblocking property of modular solutions have been proposed, e.g., in [15]. The template methodology described next takes advantage of modular design techniques.

3 Template Design of DESs

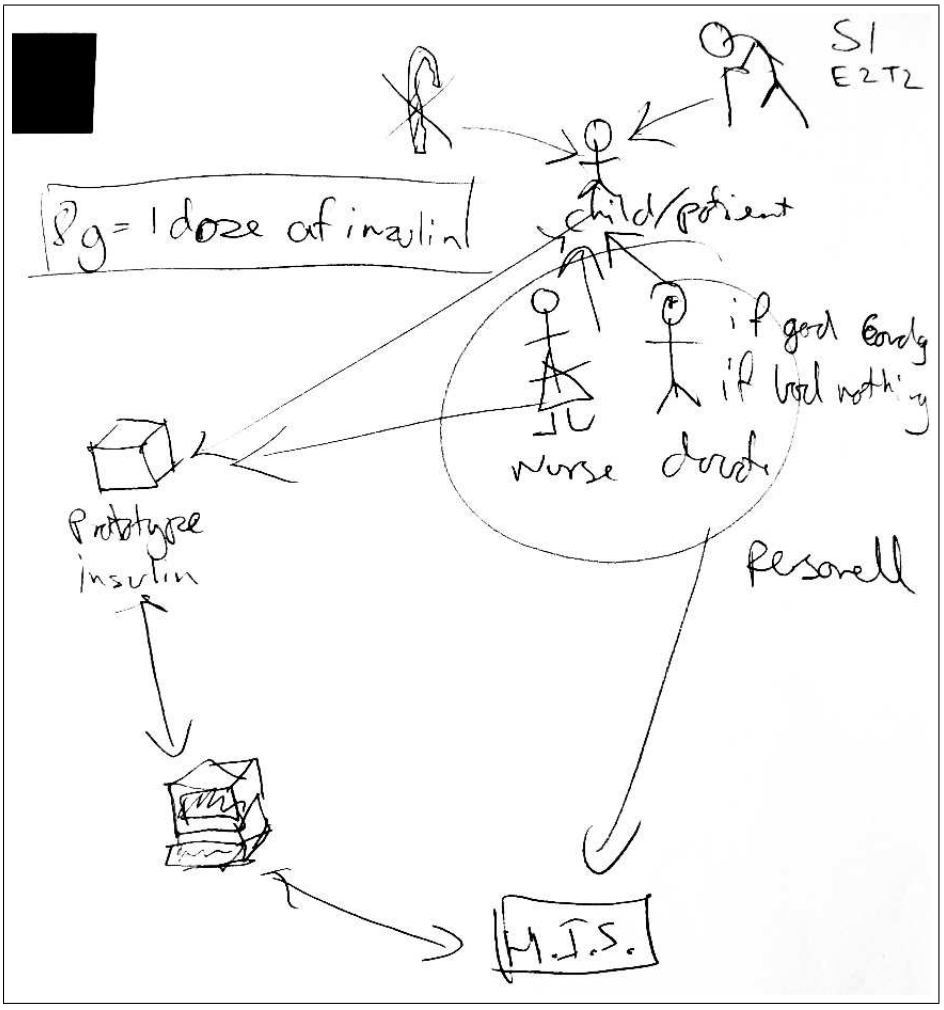
One of our observations during the study discussed in [21] is the following. When faced with a new problem, subjects frequently engaged in drawing a simple diagram of interactions between parts of the system which needed to be modeled. Examples of such diagrams can be seen in Fig. 1. It appeared that 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, the desire was to develop the approach *within* the framework of supervisory control, rather than overhaul existing theoretical results and ask experts to acquire additional background.

Here we provide the theoretical description of a new approach for the design of DES control which we termed *template design*. The details of the mathematical 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.

3.1 Framework

Before we proceed with the theoretical aspects of our work, we will describe a simple system (a part of the system from Section 4.1). 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. 2.

The framework for template design is largely based on the work of Santos *et al.* [40, 41]. 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



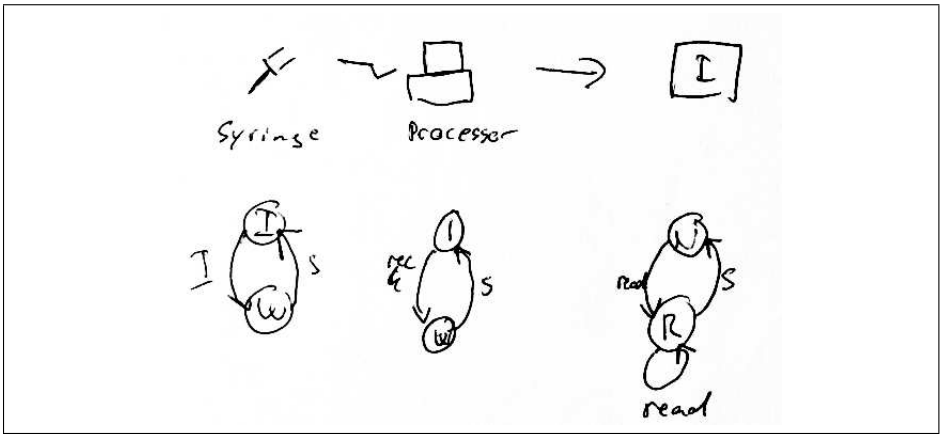
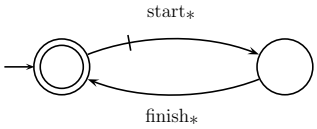
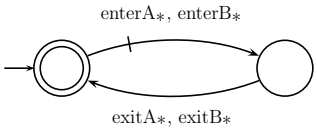


Fig. 1: Reproductions of some of the diagrams created by subjects when solving a DES problem from [21].



(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').

Fig. 2: The modules and specifications used to illustrate the template design methodology.

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 [41], 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 the modules and channels. Modules and channels as we use them here are similar to the subplants and local specifications in [9]. Finite-state automata are used for the models. Let I and J be index sets such that $|I|, |J| \in \mathbb{N}$ 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 [9]. 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. Furthermore, frequently the asynchronousness of modules is naturally satisfied as supervisory control is usually used to coordinate the joint operation of independent equipment to accomplish a complex task. 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. 2).

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 τ in the given channel should be considered equivalent to the event σ 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. Note that many channel events can be linked to the same module event; and thus the operation of a module can be governed by a number of channels (specifications). On the other hand, each channel event can be linked only to one module event, and impose restrictions on the occurrences of this event. Linking multiple module events to the same channel event would establish a direct synchronization of the module events, which would lead to ambiguity in the interpretation of the design and would violate the preconditions. Similarly, 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:

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

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, 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 [9]. The precondition for the application of this method is satisfied, i.e.,

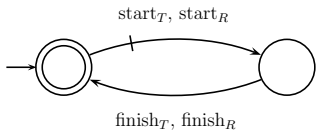


Fig. 3: E_1', the synchronized version of E_1.

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. 3). 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.

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

Finally, the algorithm from [35] for the construction of the supremal controllable sublanguage of the synchronized channel with respect to the relevant subsystem is invoked.

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

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. 3). 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. 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 defined in [9]. As long as the supervisors are nonconflicting, i.e.,

$$\|_j \overline{S_j} = \overline{\|_j S_j},$$

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.

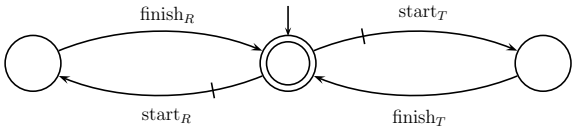


Fig. 4: The supervisor for G^1_{spec} with respect to G^1_{sys}.

3.2 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 $|K| \in \mathbb{N}, K \cap I = \emptyset = K \cap J$. Then, some of the modules, M, participating in a design can be created by instantiating the required templates. In other words, for some subset of modules G_i indexed by the subset $I' \subseteq I$, $\forall i \in I', \exists G_k \in Lib : G_i = Ins(G_k, i)$. Since the events of every template instance are named in a unique way, all instantiated modules will be asynchronous as required. Similarly, some of the channels, N, can be created by instantiating templates.

The example we used in Section 3.1 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. 2(a). The mutual exclusion specifications also share the same behavior, as shown in Fig. 2(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. 2, something very similar is described verbally.

3.3 Examples of channel templates

In this section, we provide a short list of example channel templates, i.e., generic specifications. We will not show examples of module templates as usually modules are application-specific. We will only remark that the template in Fig. 6 can be used to

create a high-level model of equipment where one is interested only in the sequence "operation is started"–"operation is finished".

Figure 5 shows a template which specifies that two events have to alternate. The template can be altered to specify an arbitrary number of alternating events (alternating event sets) or to specify a cycle of length larger than two.

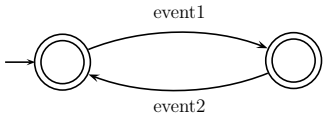


Fig. 5: Template which specifies that two events have to alternate.

Figure 6 shows a template which specifies that two events have to alternate and that the cycle has to complete once it starts. The template can be altered to specify an arbitrary number of alternating events (alternating event sets) or to specify a cycle of length larger than two. This template can be interpreted also as the specification of a single-slot buffer. It can be extended to specify a buffer with an arbitrary number of slots.

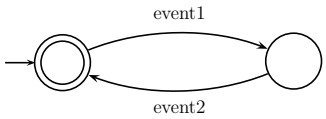


Fig. 6: Template which specifies that two events have to alternate and that the cycle has to complete once it starts.

Figure 7 shows a template which specifies that an event has to occur at least once. The template can be altered to specify an arbitrary number of required occurrences or an arbitrary set of events which need to occur.

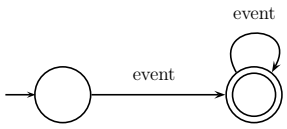


Fig. 7: Template which specifies that an event has to occur at least once.

Figure 8 shows a template which specifies two mutually exclusive sequences of events. Once one of the sequences starts, the other is prevented from occurring. This is useful, for example, when specifying that some pieces of equipment must not operate simultaneously, e.g., when accessing a shared resource. The template can be altered to specify an arbitrary number of mutually exclusive sequences or to choose if the "start" and "finish" events can occur multiple times in a single "start-finish" sequence.

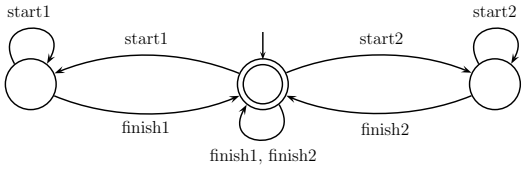


Fig. 8: Template which specifies two mutually exclusive sequences of events.

Figure 9 shows a template which couples strongly two events. If two different events are linked to "event1" and "event2" and all other events are linked to the self-loop, this template will couple strongly the two events in the following sense: if one of them occurs, the other one has to occur immediately after with no intervening events. The effect will be similar to the direct synchronization of these events. Note that the number of events in the self-loop, N, has to correspond to the number of events which must not interleave with the "event1"−"event2" sequence. The template can be altered by modifying the self-loop to accommodate links from an arbitrary number of events.

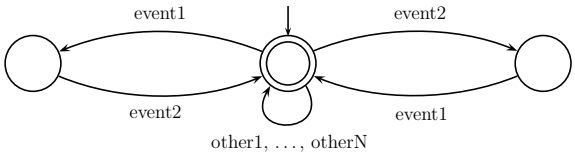


Fig. 9: Template which couples strongly two events.

3.4 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.) In Fig. 6, the model for a single-slot buffer is shown. 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 [8]. 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 parametrization approach may easily result in non-regular behaviors and specifications—and thus it cannot be readily applied in the template design framework. 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. In [52], the authors discuss supervisory control when FSA models are parameterized with Boolean variables and show that the expressive power of such models is equivalent to regular FSAs. In [43], the authors take one step further and propose an algorithm for converting parametrized models into regular FSAs.

4 Implementation of the Template Design Methodology

In this section, 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 K. Rudie's research laboratory [39].

4.1 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 Department of Automation and Systems, 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. The functionality of the testbed, shown in Fig. 10, 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 programmable logic controller (PLC).

The software prototype was implemented as an extension of IDES version 2.1. 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. 11. The models created in the software could be exported as PLC code (using the method proposed in [10]) for the direct control of the robotic testbed. More details about the prototype and the project can be found in the full report, [20].

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 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.

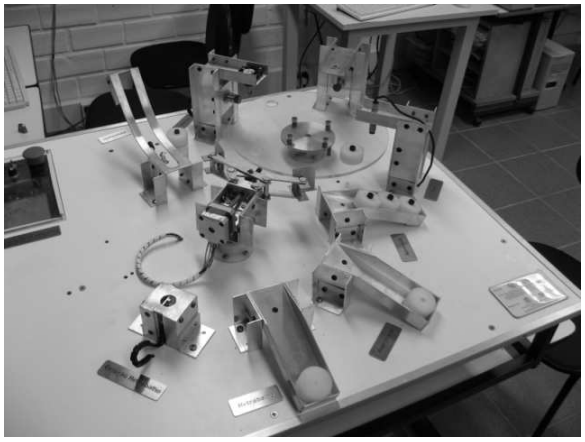


Fig. 10: The robotic testbed where template design was applied.

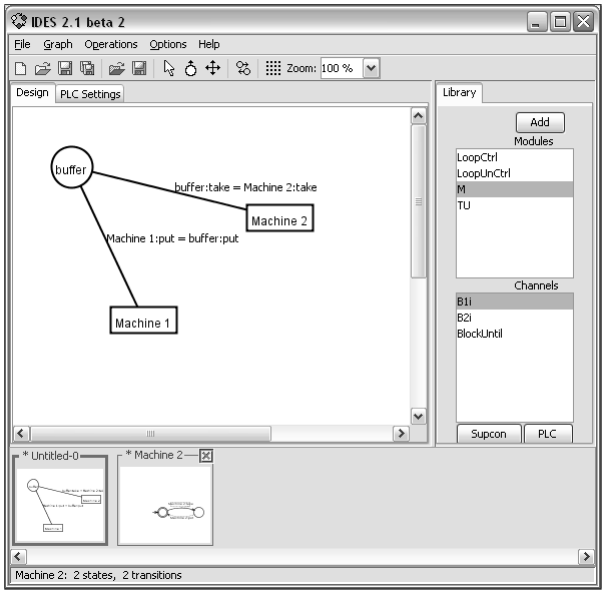


Fig. 11: The interface of the prototype template design software.

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 would then be able to delay the assignment of templates to these placeholders until more of the overall design is ready.

4.2 Template Design Plugin

The study of DES problem solving and the template design methodology served as a spring board for the development of the next major version of IDES, version 3.⋆ 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. Subsequently, the template design modelling environment was implemented as a plugin, called the TD plugin. This choice greatly reduced the development effort and provided direct access to the built-in FSA capabilities of IDES.

4.2.1 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 4.1).

The TD plugin interface is shown in Fig. 12. It consists of three main parts: the modelling area, the template library and the consistency validator.

Modelling area The modelling area, see Fig. 13, 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

⋆ At the time of writing, version 3 is still not available to the public; version 2.1 is available at http://www.ece.queensu.ca/directory/faculty/Rudie.html .

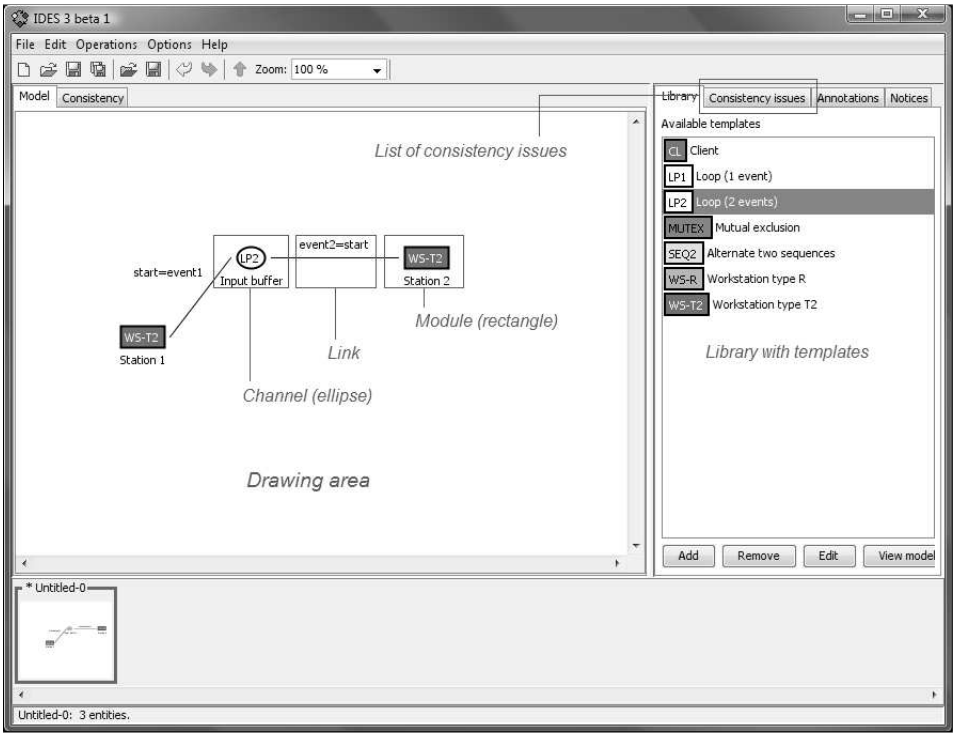


Fig. 12: The user interface of the template design plugin. The image is augmented with explanatory labels.

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 [4]. 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, [36]. 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.

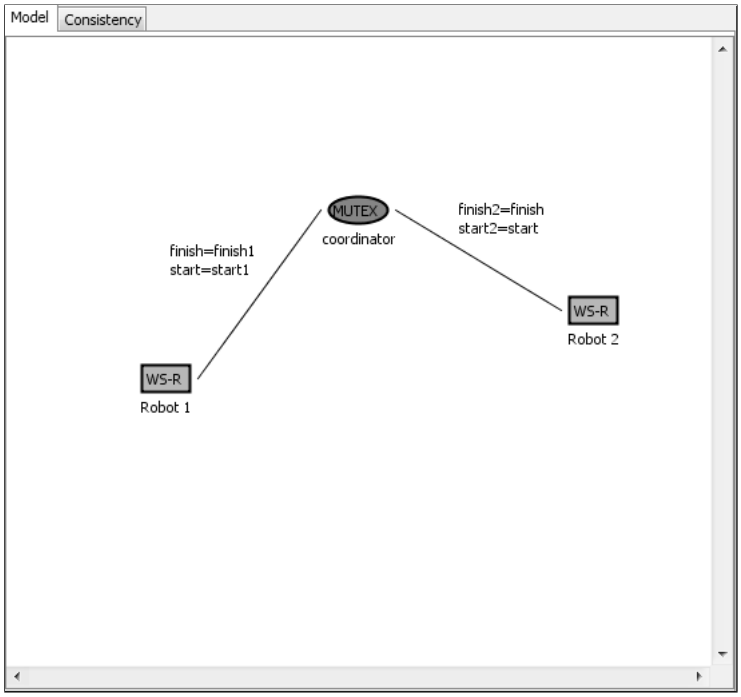


Fig. 13: The template design modelling area.

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 [33]. 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, [32]. 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 [51]). 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 through 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 [11,33]. 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 [16]. 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. 14). 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.

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

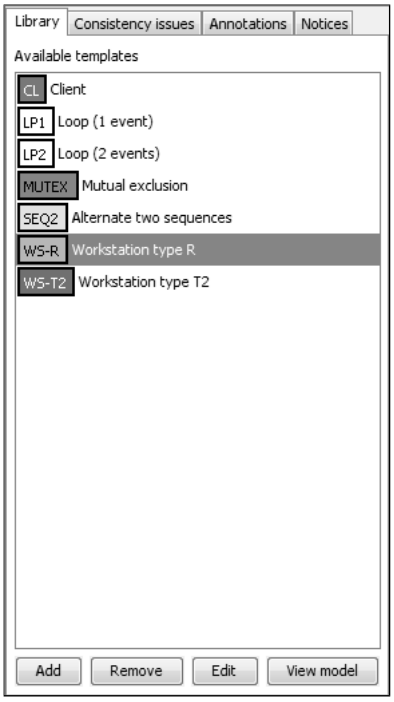


Fig. 14: The template library.

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 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. 15).

Consistency validator The consistency validator, as seen in Fig. 16, 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.

Fig. 15: 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.

Fig. 16: The list of consistency issues for template designs. In this case, a few sample issues are listed.

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 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.

4.2.2 Operations

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 nonblocking and optimal.

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

5 Evaluation of the Template Design Methodology

The last part of this work describes the evaluation of the usability of the template design implementation. It shows that, indeed, the proposed methodology of DES problem solving is advantageous in comparison to the classical approach.

5.1 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 4.1 and [20]) 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 section, 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. Furthermore, we claim that it is not necessary, at this time, to investigate the two effects separately. The template design methodology was conceived in order to make the *application* of supervisory control simpler; and this necessarily requires the employment of some implementation. Other implementations of the methodology are possible—both improvements to the software interface, and theoretical extensions to allow the use of non-FSA based template models. Until a different implementation is proposed, however, one can only speculate about the effects that would be observed.

5.1.1 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 [50]. 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 as the overall interface of the software remains the same. The TD plugin relies on the built-in functionality for manipulating

FSA models and thus the only substantial difference in using it comes from availability of the new design environment. 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".

Recruiting subjects is a very challenging task for the evaluation of DES problem solving; a similar problem was faced in [21]. The expected low recruitment rates confined our study to using a within-subjects design where all conditions are assigned to every subject (Chapter 4 in [26]). This resulted in further challenges which needed to be overcome, most notably, the prevention of task learning in within-subjects experimental designs. Subjects unavoidably learn during the performance of a task and this could affect their performance in consequent tasks. Particularly in tasks which involve problem solving, solving a problem once has obviously a huge impact on the performance when the problem is solved again.

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 [26]). 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 issue already described. Namely, the results from the tasks performed second may be meaningless. It became clear 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. After investigation of existing DES toy problems (e.g., [5,50]) and using the experience from the study of DES problem solving (in [21]) 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 4.1) 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 [18]). 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).

5.1.2 Metrics

There are many aspects of usability which can be measured. The three main categories of measures are *efficiency*, *effectiveness*, and *satisfaction* [28]. 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* [45]. 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. 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.

During the experiment, none of the subjects failed to complete a task and thus this measure is not considered in the rest of this evaluation.

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 [45]. 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 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. From previous observations (e.g., in [21]), 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. Thus, 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 [45]. It is assumed that a higher number of errors corresponds to a less effective, more difficult-to-use system. As practice has shown, [21], 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 rubric shown in Table 1 was developed.

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 [11,45], many aspects, especially the subjective experience of users, can be investigated through the administration of questionnaires. We decided to focus on a few aspects which could help compare the

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

Table 1: Rubric for the evaluation of the error rate.

template design methodology with the classical approach. The most important aspects were user confidence and subjective learnability. As discovered in the initial study of problem solving and as other researchers point out [49], 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 [11,45]. 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 [28]. To collect information about confidence and subjective learnability, in addition to using Likert-style scales from one to five [30], 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. 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. 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) [3], satisfied both conditions. Despite its simplicity, the SUS correlates well with other measures of usability [46]. Furthermore, the SUS has been used for the evaluation of many software systems, thus standardized scores have been accumulated [45, 44].

5.1.3 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 2. As

Subject	Engineering background	Knowledge of DES theory	Experience using		
			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 2: Self-reported background information about the subjects. The scale used is from one (very little) to five (very much).

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 [50] 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. This serves the goal to stimulate the subject's mind 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. Furthermore, the subject is reminded how to use IDES and the TD plugin immediately before the test and has the chance to clarify potential confusions about the software and methodology.

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.

5.1.4 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 [45,44].

5.2 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 [26,34]. Following the advice in [47] for sample size of twelve, the cutoff level of $z = 2.246$ was used (where z is the distance of the data point from the mean in terms of standard deviation units). 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 $z = 2.284 > 2.246$. 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 (incorrectly) that robots 2 and 3 must not operate concurrently and struggled to find a satisfactory solution.

There are three options for what to do with outliers in data [26,34]: data transformation, recoding or removal of outliers. We chose to remove the outliers as there are issues with the application of the other two techniques. Transformation of the data means that the value of each data point is transformed using a function, such as square root, logarithm, or inversion, to "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.

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

5.2.1 Time for task completion

As discussed in Section 5.1.2, two separate measurements were collected: time to supervisor computation and time to completion. The times for different subjects are shown in Table 3. 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. Using the Shapiro-Wilk test of normality [37], there is significant probability that the values for the time to supervisor computation and the total time (both only under the classical approach condition) are not sampled from a normal distribution. However, given the general robustness of the ANOVA test under violations of the

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 3: 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.

normality assumption [17], we decided that this test is indeed applicable in our case. Furthermore, it seems that a distribution-free (non-parametric) test is not available for mixed-design experiments (e.g., no test is suggested in [18], [26] and various other sources).

The results of the mixed factorial (one within-subjects, one between-subjects) ANOVA test are summarized in Tables 4 and 5. In the tables, of interest are the rows labelled 'Problem', 'Method' and 'Method' × 'Problem' and the columns labelled p and η^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' × 'Problem' describes the 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 η^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 [6]. A more complete description of the ANOVA table of results can be found for example in [26].

The results shown in Tables 4 and 5 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 in the data ($0.06 \leq \eta^2 < 0.14$).

Source	SS	df	MS	F	p	η^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' × 'Problem'	55513.008	1	55513.008	2.620	0.1442	0.030
'Method' × Subjects within groups	169496.042	8	21187.005			
Total	1823428.950	19				

Table 4: 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.

Source	SS	df	MS	F	p	η^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 5: 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.

- 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 ($\eta^2 > 0.14$).
- 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.

5.2.2 Error rate

The solutions of each subject were examined and the error rates were computed using the rubric from Section 5.1.2. The results are summarized in Table 6. To compare the error rates for the two methods, we used the Wilcoxon signed-rank test, a distribution-free test recommended for within-subjects ordinal data, [18]. The results of the test are $T(6) = 5.00$, $p > 0.05$ and thus a probability that the observed difference in the data is due to chance is not significantly low.

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

5.2.3 Subjective evaluation of the experience

The data from the questionnaires in Appendix B are summarized in Table 7. 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: $r > 0.5$ [6].

The open-ended questions cannot be analysed directly using similar statistical 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 8. 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

Subject	Confidence in model		Confidence in supervisor		Easy to learn approach		Easy to apply approach	
	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 7: 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).

Category	Classical Easy/Difficult	Template design Easy/Difficult
Call operations	4/0	0/2
Copy models	0/1	5/1
Create supervisors	2/0	2/0
Event synchronization	0/5	2/0
Find errors	0/1	0/1
Model new models	0/1	5/0
Model models	1/0	0/0
Modify models	1/0	2/0
Situation overview	0/2	3/0
Self-loops	5/0	2/0
User interface	1/1	1/1

Table 8: Aggregated responses to the two open-ended questions from the questionnaires in Appendix B. Each row shows how many responses indicated that the given category was “easy” and “difficult”.

situation. The single, 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 9. 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 making errors and user interface are not referred to as frequently.

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

5.2.4 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 10.

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 10: SUS scores for IDES with the TD plugin.

The SUS results were compared with the data from [44] (mean=66.41, std=12.97) using the two-tailed t-test with the Welch adjustment for unequal variances [48]. 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 [44].

5.3 Discussion

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 5.1.4, 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 [45,44]. 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.

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 [38] 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 9, 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 4.1 and [20], 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.

6 Conclusions

This paper described the research conducted in the quest for designing better DES software. The exploratory observational study of solving DES control problems, [21], served as a precursor. The think-aloud data collected from the subjects helped us to develop a list of recommendations for designing and improving DES software. These observa-

tions, 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 to design better DES software. We believe that practitioners of DES supervisory control will benefit from using the new methodology.

6.1 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 [21], 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. The largest outstanding issue, however, is the fact that our evaluation did not investigate the effect of the proposed methodology on solving complex (non-obvious) problems. It is our belief that the modularity in the design, and the visual representation of the high-level structure would have a positive impact when solving complex problems. Furthermore, the availability of standardized templates may help users to produce a successful solution by gaining insight into how to combine existing elements; recognition is easier than recall. The testing of these hypotheses will involve a series of elaborate experiments which might not be feasible, given the difficulties we experienced with subject recruitment in conducting our simpler evaluation study. Potentially, such studies could be an integral part of the aforementioned longitudinal study.

As pointed out in Section 3.4, 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 underlie 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 supervisory control algorithms which will act upon heterogeneous models.

7 Acknowledgments

We would like to thank the following people whose support and advice helped in the completion of this work: Max de Queiroz, Francisco da Silva, Guilherme Lise and Luis Marques from Federal University of Santa Catarina, Brazil, Steffi Klinge from Otto-von-Guericke University, Germany, Daniel Cownden and Roger Browse from Queen's University, Canada, and Martin Fabian from Chalmers University of Technology, Sweden. We also acknowledge the help of all volunteers who participated in the studies which form an essential part of this work. The project was supported through grants from NSERC and Queen's University, Canada, and CNPq, Brazil.

References

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. 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.
3. J. Brooke. *Usability Evaluation in Industry*, chapter SUS: a 'quick and dirty' usability scale, pages 189–194. Taylor and Francis, 1996.
4. 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.
5. C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Springer, second edition, 2007.
6. J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, second edition, 1988.
7. CTCT software. Department of Electrical and Computer Engineering, University of Toronto, Canada. Available at http://www.control.toronto.edu/DES/.
8. 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.
9. 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.
10. 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.
11. A. J. Dix, J. E. Finlay, G. D. Abowd, and R. Beale. *Human-Computer Interaction*. Prentice Hall Europe, second edition, 1998.
12. 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.
13. 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.
14. 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.
15. H. Flordal and R. Malik. Compositional verification in supervisory control. *SIAM Journal on Control and Optimization*, 48(3):1914–1938, 2009.

16. G. W. Furnas and B. B. Bederson. Space-scale diagrams: Understanding multiscale interfaces. In *Proceedings of Human Factors in Computing Systems (CHI 95)*, pages 234–241, Denver, CO, USA, 1995.
17. 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.
18. J. A. Gliner and G. A. Morgan. *Research methods in applied settings: an integrated approach to design and analysis*. Lawrence Erlbaum Associates, 2000.
19. L. Grigorov. Hierarchical control of discrete-event systems. Survey paper, School of Computing, Queen's University, Canada, 2005. Available at http://www.banica.org/research/.
20. L. Grigorov. Template design of discrete-event systems. Technical report 2007-538, School of Computing, Queen's University, Canada, 2007.
21. L. Grigorov. Observations on solving discrete-event control problems: patterns and strategies. Technical report 2009-558, School of Computing, Queen's University, Canada, 2009.
22. 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.
23. 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.
24. 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.
25. L. G. Grigorov. *Conceptual design of discrete-event systems using templates*. PhD thesis, School of Computing, Queen's University, Kingston, Ontario, Canada, 2009.
26. S. A. Haslam and C. McGarty. *Research Methods and Statistics in Psychology*. SAGE Publications, 2003.
27. 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.
28. K. Hornbæk. Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, 64:79–102, 2006.
29. R. J. Leduc. *Hierarchical Interface-based Supervisory Control*. PhD thesis, Department of Electrical and Computer Engineering, University of Toronto, 2002.
30. R. Likert. A technique for the measurement of attitudes. *Archives of Psychology*, 140:1–55, 1932.
31. 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.
32. D. A. Norman. Some observations on mental models. In *Mental Models*, pages 7–14. Lawrence Erlbaum Associates, 1983.
33. D. A. Norman. *The Design of Everyday Things*. Currency, New York, New York, USA, 1990.
34. 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.
35. 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.
36. J. Raskin. *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional, 2000.
37. D. G. Rees. *Essential statistics*. CRC Press, fourth edition, 2001.
38. E. Rogers. VIA-RAD: a blackboard-based system for diagnostic radiology. *Artificial Intelligence in Medicine*, 7:343–360, 1995.
39. 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.
40. 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.
41. 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.

42. 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.
43. M. Sköldstam, K. Åkesson, and M. Fabian. Modeling of discrete event systems using finite automata with variables. In *Proceedings of the 46th IEEE Conference on Decision and Control*, pages 3387–3392, New Orleans, LA, USA, December 2007.
44. SUS scores from 129 conditions in 50 studies. Spreadsheet available at http://measuringuserexperience.com/ (April 2009).
45. T. Tullis and B. Albert. *Measuring the User Experience*. Morgan Kaufmann Publishers, Burlington, MA, USA, 2008.
46. 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.
47. 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.
48. B. L. Welch. The significance of the difference between two means when the population variances are unequal. *Biometrika*, 29:350–362, 1938.
49. 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.
50. W. M. Wonham. Supervisory control of discrete-event systems. Available at http://www.control.toronto.edu/DES/, June 2008.
51. 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.
52. Y. Yang and P. Gohari. Embedded supervisory control of discrete-event systems. In *Proceedings of the 2005 IEEE International Conference on Automation Science and Engineering*, pages 410–415, Edmonton, AB, Canada, August 2005.

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.

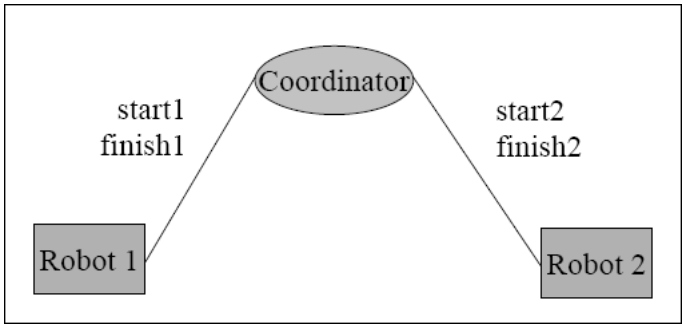
The new situation is described as follows. The factory has been modified to fit two different chips on the same board. Robot 1 has received an upgrade. During processing, it can detect defects in the circuit boards. Any time it detects a misaligned hole, it can perform an additional corrective procedure (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.

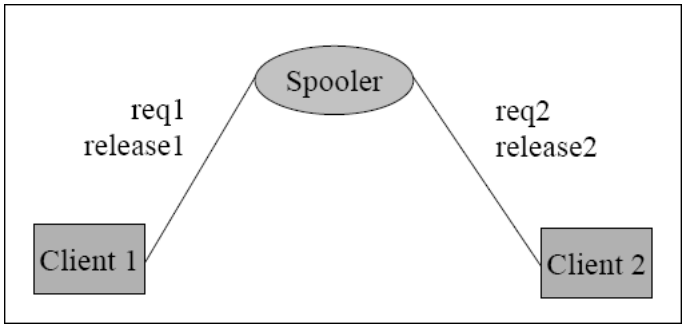
The new situation is described as follows. A third client has been added to the network, and the third client can request access to the fax machine and release the fax machine when done transmitting data—similar to the operation of the other two clients on the printer. As well, client 2 has been given 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.



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?