2008 American Control Conference
Westin Seattle Hotel, Seattle, Washington, USA
June 11-13, 2008

Design of Discrete-Event Systems Using Templates

LENKO GRIGOROV lenko.grigorov@banica.org
School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada
JOSÉ EDUARDO RIBEIRO CURY cury@das.ufsc.br
Department of Automation and Systems, Federal University of Santa Catarina, Florianópolis, Santa Catarina, Brazil
KAREN RUDIE karen.rudie@queensu.ca
Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario K7L 3N6, Canada

***Abstract*— A new methodology for the design of DES control is proposed which allows for the creation of high-level conceptual designs by using encapsulated low-level elements. The approach can be used within the standard framework of modular supervisory control. The notion of *DES templates* is introduced, where typical behaviors for both DES modules and specifications are represented in an abstract way. The control engineer creates instances of these abstractions and specifies the way the instances interact. System modeling and the design of specifications occur simultaneously. Speed and robustness of the design process are improved since there is no need to consider details or to reimplement similar parts of a system. The proposed methodology is applied to a small robotic testbed to get real-world feedback.**

I. INTRODUCTION

In control engineering, the possible behavior of some systems can be described as a set of sequences of discrete events. Ramadge and Wonham [10] propose a theoretical framework, called Supervisory Control Theory, for the modeling and control of such systems. In this framework, the discrete events are instantaneous, spontaneous, and certain control can be exercised by preemptively preventing the occurrence of some of them. Systems modeled in this framework are called discrete-event systems (DESs) and the entity exercising the control is called a *supervisor*.

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 [5], 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, lower cost of the project and more robust handling of failures. 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. Furthermore, the methodology is not cast within the supervisory control framework and it cannot take advantage of the algorithms therein.

In this work, we propose a new methodology for the design of DES control. We introduce the notion of DES templates *within* the framework of supervisory control. Typical behaviors for both DES modules and specifications are represented in an abstract way. The control engineer creates instances of these abstractions and then needs only to specify the way the instances interact. System modeling and the design of specifications occur simultaneously. Speed and robustness of the design process are improved since it is not necessary to deal with details of the system behavior, as well as to reimplement similar parts of a system. The computation of the supervisory solution can be automated. The methodology was implemented in software and support for the generation of Programmable Logic Controller (PLC) code was added. Then, we applied the approach to obtain a control solution for the hardware of a small system.

II. PRELIMINARIES

The basic theory of supervisory control of discrete-event systems [10] has been extended by researchers in attempts to resolve some of the problems in its application. The problem of state-space explosion has been addressed partially by considering *modular* or *hierarchical* supervision and by dealing with systems *incrementally*. A discussion of these topics, relevant to our paper, can be found in [14], [9] and [1], 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

978-1-4244-2079-7/08/$25.00 ©2008 AACC.

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 [7]. 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.

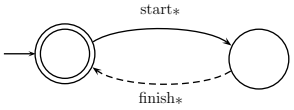
During our personal experience with the application of supervisory control, we noted further complications. First, control engineers are required to learn about finite-state automata (FSAs) and the modeling of systems and specifications using FSAs. The process of modeling is quite slow and requires a lot of attention in order to avoid errors. The presence of an error in the design is not readily observable. The design of interaction between different system modules is achieved through synchronization on common events. The designer needs to constantly maintain an overview of which events are used for that purpose. A second complication, partially a result of the method for synchronization, is that system models are usually designed on a per-use basis. It is not trivial to reuse models in other projects. Many times it is necessary to start modeling from scratch even though parts of the model have been designed on previous occasions. Third, currently DES software packages offer little, if any, support for the implementation of FSA-based supervisors so that they can be used with real equipment.

III. Template design of DESs

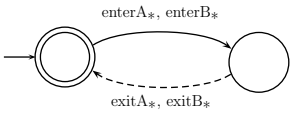
The template method for DES design is substantiated greatly by the observations made during a study of how humans solve DES control problems [8]. When faced with a new problem, subjects frequently engaged in drawing a simple diagram of interactions between parts of the system which needed to be modeled. It appeared that the subjects liked to isolate different aspects of a system before they proceeded with the low-level modeling. These observations led to the proposal for a new methodology for the design of DESs, where control engineers can focus on assembling blocks of subsystems and specifications instead of worrying about every little detail of the system.

A. Framework

Before we proceed with the theoretical aspects of our work, we will describe a part of the system from Section V. It will be used to illustrate the steps of the new methodology.



(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. 1. The modules and specifications used to illustrate the template design 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. 1.

The framework for template design is largely based on the work of Santos *et al.* [12], [13]. The authors propose a methodology for conceptual design of DESs using *entities* and *channels*. Entities are the active parts of the system (e.g., workstations). Channels are passive parts of the system which facilitate the transfer of matter and energy between entities (e.g., conveyor belts). This framework is suitable for the modeling of complex systems since it allows the simultaneous definition of both structure and functionality.

In our framework we decided to keep all the basic propositions of [13], 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 [3]. Finite-state automata (FSAs) are used for the models. Let I and J be index sets such that $|I|, |J| \in \mathbf{N}$ and $I \cap J = \emptyset$. The set of modules is

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

and the set of channels is

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

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

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

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

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

The requirement that modules be asynchronous is not a stringent restriction as discussed in [3]. The benefit of having asynchronous modules is mainly in being able to make more

uniform assumptions about the system. If some modules are not asynchronous, they can be composed until there are no dependencies between modules. The channels have to be asynchronous because they describe generic specifications. It is only with the help of links that the specifications are synchronized with the given system. In our example, $M = \{G_T, G_R, G_D\}$ and $N = \{E_1, E_2\}$ (see Fig. 1).

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

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

In other words, the function defines links between events of channels and events of modules. The interpretation of the link $C(\tau) = \sigma$ is that the event 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. For all $G_j \in N$, the restrictions of the function,

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

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

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

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

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

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

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

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

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

After a system is modeled in the proposed framework, modular control can be applied to obtain supervisors for the separate specifications. This is possible since, 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 [3]. The precondition for the application of this method is satisfied, i.e., the participating modules are asynchronous. All modules which are linked to a channel participate in the subsystem influenced by the specification determined by the channel. Let $G = (\Sigma, Q, \delta, q_0, Q_m) \in N$ be a channel. Then define $G' = (\Sigma', Q_E, \delta', q_0, Q_m)$ as the synchronized channel G where all channel events have been replaced with their corresponding module events, i.e.,

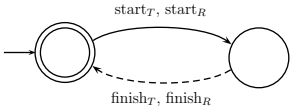


Fig. 2. The synchronized version of E_1.

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

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 (see Fig. 2). Channel E_2 is synchronized in a similar way. Furthermore, $C(E_1) = \{G_T, G_R\}$ and $C(E_2) = \{G_T, G_D\}$.

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

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

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

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

Finally, the algorithm from [10] 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 (see Fig. 2). Thus, no self-loops are introduced into the channels, i.e., $G^1_{spec} = E'_1$ and $G^2_{spec} = E'_2$. The supervisor S_1 obtained for G^1_{spec} with respect to G^1_{sys} is shown on Fig. 3. It is easy to see that the simultaneous operation of the table and the arm is avoided. The supervisor for G^2_{spec} is analogous.

The last step involves checking whether the supervised system is nonblocking, as defined in [3]. As long as the supervisors are nonconflicting, i.e.,

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

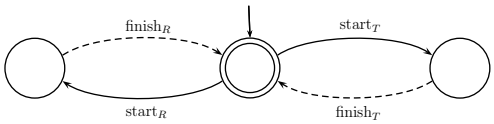


Fig. 3. The supervisor for G^1_{spec} with respect to G^1_{sys}.

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.

B. Templates

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

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

$$\Sigma_p = \{\sigma_p \mid \sigma \in \Sigma\},$$
$$\delta_p(q, \sigma_p) = \delta(q, \sigma).$$

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

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

The example we used in Section III-A 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. 1(a). The mutual exclusion specifications also share the same behavior (see Fig. 1(b)). Thus, if templates are used, the system modules can be instantiations of a generic "workstation" template, while the channels can be instantiations of a generic "mutual exclusion" template. If one looks again at the caption of Fig. 1, something very similar is described verbally.

C. Parametrization

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

One possible approach to the parametrization of FSAs is described in [2]. There, a regular FSA is augmented with a *data collection*. The data collection is a vector of scalars which can range over some set. A vector of unary functions is associated with each transition in the FSA. For example, a buffer can be modeled as a single state with two self-looped transitions, "insert" and "remove", and a single integer in the data collection to keep track of the number of items in the buffer. Then, the functions "$+1$" and "-1" will be applied to the integer when "insert" and "remove", respectively, occur. In such a system, control can be based on predicates about the current state of the system and on the current value of the data collection. The authors propose a method to compute the supremal controllable sublanguage of a system by incrementally backtracking with the predicates until the control decisions do not attempt control of uncontrollable events. Unfortunately, the use of this model may easily result in non-regular behaviors and specifications. This is the reason why the model cannot be readily applied in the template framework proposed in this work. A potential solution would be to restrict the type of data collections that can be used. For example, each scalar in a data collection could be restricted to belong to a closed integer interval. However, even in this case it is necessary to find an efficient transformation from the parametrized model into a "simple" FSA.

IV. Software package for template design

The theoretical foundation of template design does not require any graphical representations. However, as it has been discovered in practice [6], [8], the lack of graphical representation may significantly influence the usability of a software package. Thus, one of the principles we decided to follow in designing our software was to use a graphical interface. A screenshot of the interface is shown in Fig. 4. We decided to use boxes to represent instances of modules and circles to represent instances of channels. The links between modules and channels are represented as lines connecting the boxes and circles. The user of the software can create and manipulate the graphical elements using the mouse cursor.

Template design is just a high-level interpretation of modular supervisory control. Finite-state automata underlie all elements of the design and regular DES operations are applied at the low level to produce the supervisors. Thus, any software which supports template design has to be able to

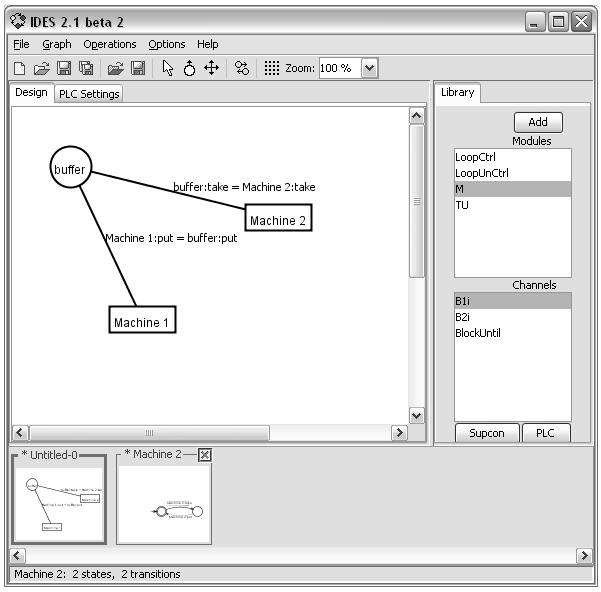


Fig. 4. The template design software interface.

perform all functions of a regular DES tool as well. Instead of writing a completely new software package, we decided to extend the IDES software developed at Rudie's research laboratory, [11]. Its architecture supports the addition of extensions, it offers an advanced graphical interface infrastructure, and it can be used on all major computer platforms since it is developed in Java.

Since the purpose of template design is to make the application of DES theory easier, we decided to try to streamline the complete process of application: from modeling to control of the real hardware. In many cases the real system is controlled by a PLC unit; this is the case in our example system as well. Thus, we focused on the generation of PLC code from the template design. There are many ways how to convert FSAs into code, however, the method proposed in [4] seems to be most suitable for two reasons: it converts FSAs directly into PLC code, and it is designed with modular control in mind. Since this approach is generic, the users still need to make manual modifications to insert hardware-specific instructions. In our software, for each event in the template design the user can specify a snippet of PLC code. Then, during PLC code generation, this code will be incorporated into the automatically produced code.

V. Example application

In order to test the applicability of template design of DESs, the methodology was used to design a controller for a robotic testbed at the Department of Automation and Systems, Federal University of Santa Catarina, Brazil. The functionality of the system, shown in Fig. 5, is to retrieve parts from an input buffer, perform operations on the parts and test if the operations were successful. Depending on the outcome of the test, the given part is output into one of a number of buffers (such as "accepted", "reprocess", etc.) The system is controlled via a Siemens S7-200 series PLC unit.

The part of the system we used included four modules: the input buffer for new parts, the arm with a grabber, the rotating table which moves a part to the different workstations and one of the workstations, the drill. The arm with the grabber was simplified to perform only one (high-level) activity: retrieve a part from the input buffer and place it on the table. The specifications applied to the system were as follows. First, there has to be mutual exclusion between the table and the arm and between the table and the drill. In other words, the table should not turn while one of the other units is in the middle of completing an operation. Second, there has to be underflow control for the input buffer, i.e., the arm should not retrieve a part if there are no parts in the buffer. Third, there has to be control over the sequence of operations: after a part is placed on the table, the table has to turn before the drill operates on the part.

At the start of the modeling, it was assumed that the template library contains all relevant templates (such as the ones in Fig. 1). Then, the modules and channels were created by instantiating the templates and linking the relevant events. The supervisory control algorithms were performed to obtain modular supervisors, to check the local modularity property and to generate the corresponding PLC code. The code was downloaded onto the PLC unit and the testbed was started.

VI. Discussion and conclusions

The template design of DESs is based on theoretical work, however, the main motivation for its conception was making the application of DES control simpler. The following improvements were envisioned.

- Faster design of systems. The use of pre-built templates not only reduces the time to mechanically input new FSAs but also the time to mentally consider low-level details of FSA implementations.
- More robust designs. Fewer errors can be made during the design since it is not necessary to manually copy FSAs and to keep track of the names of events in different modules and specifications.

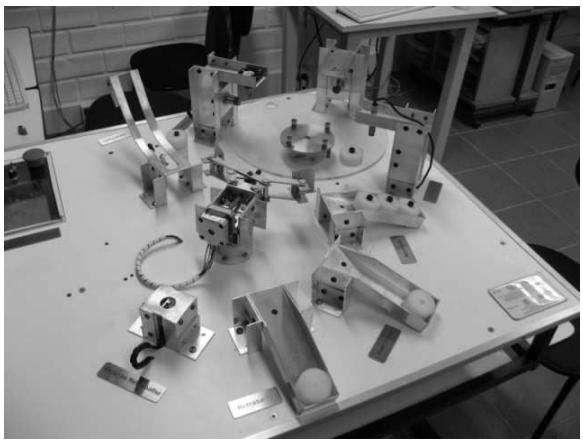


Fig. 5. The robotic testbed where template design was applied.

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

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. The users no longer have to keep track of event name consistency between modules and between specifications. Synchronization is not achieved by naming events consistently but rather by visually linking them. Then, it is easy to try different synchronization strategies and it is possible to use a single template instance in a number of ways without having to always rename events. This property seemed to be especially liberating since renaming events is laborious and error-prone. In our project it was necessary to go through a large number of iterations where the system was simplified with different approaches. This rapid prototyping would not have been feasible if all operations had to be called manually and if event names had to be changed for every new approach.

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

Further work should also focus on providing support for real-time interaction between model and running system. The current implementation of the software supports only one-way interaction with the real system—PLC code is generated from the abstract description of the DES supervisors and it is downloaded to the PLC controller. However, it is not possible to receive any feedback from the real system when it runs. Feedback which is not real-time, such as a log of the executed events and how much time they took, may be used for analysis of the performance of the controlled system. It would be much more interesting, however, to be able to connect the software with the system controller (such as a PLC) in order to obtain real-time feedback. This could be used in many ways: from animating on the screen the execution of the system to providing high-level control from within the software, if the PLC code is equipped to delegate the control of some events to the software. By incorporating real-time feedback, a designer could swap control specifications during the system runtime and thus immediately observe the effect of such changes.

VII. Acknowledgments

We would like to thank the following people whose help and support were crucial for the completion of this work: Max de Queiroz, Francisco da Silva, Guilherme Lise and Luis Marques from Federal University of Santa Catarina, Brazil and Steffi Klinge from Otto-von-Guericke University, Germany. The project was supported through grants from NSERC and Queen's University, Canada, and CNPq, Brazil.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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