Techniques for the Parametrization of Discrete-Event System Templates ⋆

Lenko Grigorov * **Karen Rudie**

Department of Electrical and Computer Engineering, Queen's University, Kingston, Ontario

* *lenko.grigorov@banica.org*

Abstract: Two techniques for the parametrization of templates in the template design methodology are introduced. First, the use of finite-state automata extended with variables is demonstrated. This kind of parametrization allows one to vary the occurrences of events by varying a parameter. Second, a new way of parametrizing templates is proposed, named compositional parametrization. With this technique, a template consists of the composition of a number of structurally identical components whose events are indexed differently, depending on a parameter. This kind of parametrization allows one to vary the structure and event set of the template by varying a parameter. An enhancement of compositional parametrization with the use of special selector transitions is also discussed. This extension allows one to model interactions between components which depend on the total number of components in a composition and thus cannot be modelled by simple parametrization of event indices. With the choice of a specific parameter, regular finite-state automaton models are obtainable from the templates parametrized with any of the aforementioned techniques. Thus, such parametrized templates can be easily introduced in the template design modelling environment and they integrate seamlessly with the supervisory control framework. Motivating examples to illustrate the application of template parametrization are provided.

Keywords: discrete-event systems, parametrization, template design, finite automata, supervisory control.

1. INTRODUCTION

The application of discrete-event supervisory control, as proposed in Ramadge and Wonham (1987), has not been very widespread since the introduction of the methodology. A number of reasons have contributed to this lack of uptake. One of them is the large amount of computation required to synthesize controllers for non-trivial systems. Researchers have recognized this drawback and much effort has focused on mitigating the computational size of the supervisory control problem, e.g., by considering local modular control, de Queiroz and Cury (2000), or by using hierarchical structures and symbolic representations, Ma and Wonham (2003). Our research started with an attempt to understand the process of applying supervisory control theory from the human factors point of view, Grigorov (2009). Based on this investigation, we proposed a new methodology for the design of discrete-event system (DES) controllers by using templates, Grigorov et al. (2008, 2010).

The usability of the template design methodology was evaluated and it was found that the novel approach offers advantages over the classical in DES problem solving. However, it was noted that a further advantage may be gained if a foundation for the parametrization of templates is developed. In this paper, we propose and demonstrate two methods of template parametrization. The first one is based on using Extended Finite Automata (EFA), Sköldstam et al. (2007), as the underlying template models. The second method introduces compositional parametrization, where templates are instantiated by composing structurally equivalent components whose events have parametrized indices. Additionally, we describe an extension of the compositional parametrization where special selector transitions are used. This enhances the expressiveness of the model. For each method, we give examples of basic templates and then demonstrate how they can be parametrized. As with our original template design approach, the use of parametrized templates does not require modifications to the established theoretical framework of supervisory control. Parametrization occurs at the level of the user interaction with the modelling environment.

2. PRELIMINARIES

The template design methodology is based on the construction of a conceptual design by using pre-built components, called templates. Templates have already been proposed in the context of DESs, e.g., in Ekberg and Krogh (2006) and Holloway et al. (2000). However, the main inspiration for our work came from the work of Santos et al. (2001) as it allowed us to position the template design approach completely *within* the supervisory control framework.

⋆ This work was supported by NSERC, Canada.

2.1 Template Design Framework

The template design approach is described in more detail and illustrated with a small example in Grigorov et al. (2008). Here, we will only review briefly the mathematical background.

A template design 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 de Queiroz and Cury (2000). Finite-state automata are used for the models, but the framework does not preclude the use of other formalisms if the required operations are available for them. 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 have to be asynchronous. This requirement permits the construction of a consistent graphical representation as in the software from Grigorov et al. (2010). All channels have to be asynchronous as well because they describe generic specifications.

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, imposing restrictions on the occurrences of this event. 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.

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. The first step is to use the links to synchronize the channels in the template design. 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.,

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

After channels are synchronized, either modular or monolithic supervisor synthesis techniques can be used to obtain the control solution.

In the proposed framework, a template is simply a model of some common discrete-event behavior, e.g., the model of a robot type which is used in many places on the factory floor. In the supervisory control setting, the model would be a finite-state automaton (FSA). Using a template, 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., $\Sigma_p = \{\sigma_p \mid \sigma \in \Sigma\}$ and δ_p is the corresponding update of δ. Thus, for example, creating the DES modules for ten workstations would be reduced to instantiating the corresponding template with ten different indices.

Since templates can describe both system behavior (i.e., modules) and restrictions on behavior (i.e., channels), the use of templates 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.

3. TEMPLATE PARAMETRIZATION

The parametrization of templates is a natural extension of the template design framework. 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., a buffer with two slots, a 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 could be parametrized in terms of capacity—and then at instantiation one would be able to choose the specific capacity to be used.

One of the first papers discussing the use of parametrized finite-state automata to model DESs is by Chen and Lin (2000). The authors propose a model where an automaton is augmented with parameters from some vector space, $v \in V$, which usually ranges over the set of natural numbers. Then, transitions in the FSA can be conditioned on the value of the parameters and, upon occurrence, may update the value of the parameters. The conditions are called *guards* and represent predicates on v. The occurrences of transitions may update the parameters using some functions $f(v)$. For example, a transition on the event "insert" in a buffer may be conditioned on a counter variable not having reached a limit, and may

increase the value of the counter by one. The control of such DESs may take advantage of the parameters in addition to the information in the states of the FSA, as proposed by de Oliveira et al. (2004). Control decisions are based on predicates about the current state of the system and on the current value of the parameters. 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 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. In Sköldstam et al. (2007), the authors propose an algorithm for converting parametrized models, called *Extended Finite Automata* (EFA), where each scalar in the data collection is restricted to belong to a closed integer interval, into regular FSAs.

A different approach to parametrization is discussed by Bherer et al. (2009). The authors discuss control of parametrized DESs under three similarity assumptions, namely, the process, mask, and specification similarity assumptions. The first two assumptions constrain the type of DESs considered to be from a class where processes can be described as the composition of a replicated structure. The specification similarity assumption constrains the control specifications to be symmetric, i.e., to pose the same requirements on any subset (of a given size) of the process components. It is proved then that, under these similarity assumptions, it is possible to construct a supervisor for a given specification for a system of arbitrary size (i.e., consisting of the composition of an arbitrary number of replicas), n, based on the system of fixed size (i.e., consisting of the composition of a fixed number of replicas), n_0, where $n_0 \leq n$ and it is understood that usually n_0 is much smaller than n. Two of the limitations of this work are that only state-based safety specifications are considered, and that processes can consist of the composition of only one replicated structure. Nevertheless, under the right conditions, the proposed approach enables the efficient synthesis of compact and scalable supervisors.

The parametrization of templates can naturally take advantage of the research described above. Templates are essentially FSAs of which copies are made during instantiation. (Here let us point out again that the basic template design framework does not preclude, theoretically, the use of formalisms other than FSAs.) Thus, it is simple to incorporate parametrization through the use of EFAs. Indeed, in the following, we will give examples where such parametrization is suitable to express the desired flexibility in a template model. However, we will demonstrate that not all desired parametrization can be accomplished in this way. We then propose another method of template parametrization which bears similarity to the parametrization with replicated structures from Bherer et al. (2009), and provide illustrating examples.

3.1 Parametrization using variables

Parametrization of template models using variables is a simple way to introduce greater flexibility in the templates

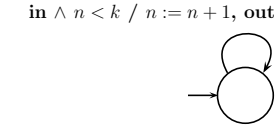
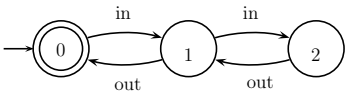
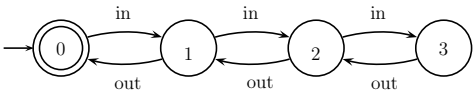


Fig. 1. The EFA representation of a buffer of size k. Here, n is an integer variable from the interval $[0, k]$, with the initial value $n = 0$ and an accepting value $n = 0$.



(a) $k = 2$



(b) $k = 3$

Fig. 2. The regular FSA models for a buffer, derived from the EFA representation in Fig. 1.

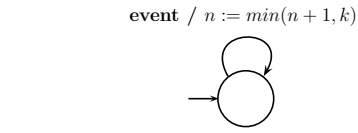


Fig. 3. The EFA representation of the specification that an event has to occur at least k times. Here, n is an integer variable from the interval $[0, k]$, with the initial value $n = 0$ and an accepting value $n = k$.

that can be used in a template design. The specific parametrization approach we use in the following two examples is borrowed from the work of Sköldstam et al. (2007). However, as discussed earlier, there are other similar approaches which can be substituted if so desired.

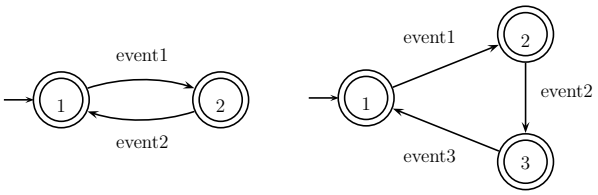
Let us consider the template for a buffer. Without parametrization, a separate template must be available for each or all buffer capacities that need to be considered. However, with the use of a single variable, the EFA in Fig. 1 can serve as a generic buffer template where the parameter k specifies the desired capacity of the buffer. Indeed, when $k = 2$ or $k = 3$, for example, one can convert the EFA into the ordinary FSAs shown in Fig. 2. The conversion can be performed by using the algorithm from Sköldstam et al. (2007).

Another template which can be easily parametrized is the template which specifies that an event has to occur at least k times (see Fig. 3).

3.2 Compositional parametrization

Parametrization of templates with variables is suitable for situations when different sequences of events could be desired, but where the sequences are constructed using the same set of events. However, this is not always the case. Here we will discuss two templates where new behavior is achieved by extending a basic pattern with new events.

First, let us consider the template which will specify that k different events have to happen in sequence. The templates for $k = 2$ and $k = 3$ are shown in Fig. 4. As it can be observed, the two templates are not related simply



(a) $k = 2$ (b) $k = 3$

Fig. 4. Specifications that k events have to happen in sequence.

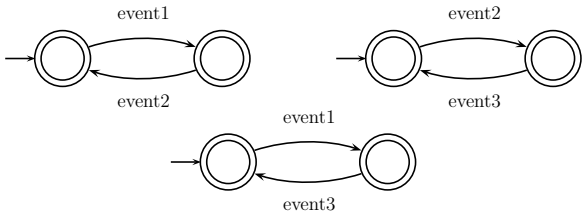


Fig. 5. A set of components which can be composed using synchronous product to obtain the model from Fig. 4(b).

by some functional variation of the generated event sequences. Rather, the template for $k = 3$ incorporates one extra event in comparison to the template for $k = 2$. Thus, the parametrization should be in terms of events rather than variables. The parametrization of the template can be expressed much more conveniently as the parallel composition of a number of similarly structured components, as shown in Fig. 5. Let us formalize this different kind of parametrization, which we will call *compositional parametrization*. The basic proposition here is that we will create a generic component automaton with parametrized events, and then obtain a number of specific derived automata which we will compose. A similar idea is used in Bherer et al. (2009), however, in the context of scalable supervisor synthesis. Here we are only interested in the parametrization of templates and thus we need not impose all the restrictions necessary for ensuring feasibility of such synthesis. Most importantly, the patterns of synchronization between components of the composition can be much more complex; it is possible to use a number of generic automata in the same composition, different subsets of events can be shared by different subsets of components, and it is possible to have parameter-dependent synchronization.

Before we define compositionally parametrized automata, first we will establish a few preliminaries. Since we are aiming to define FSAs with parametrized events, and since the parameter k is a natural number, the set of potential events in the model is unbounded. Let us construct this event set by enumerating all potential events, $\Sigma_\infty = \{\sigma_n | n \in \mathbb{N}\}$. Then, each event from the components will be represented by some σ_i.

Compositionally parametrized FSAs consist of the composition of structurally similar component automata which differ only in the indices of their events (as shown in Fig. 5). In order to parametrize event indices, with the parameter $i \in \mathbb{N}$, let us consider a finite set of functions, F, such that $\forall f \in F,\ f : \mathbb{N} \to \mathbb{N}$. For example, σ_{i+1} would equal the event σ_3 when $i = 2$ and σ_4 when $i = 3$. Such parametrization allows us to define automata where

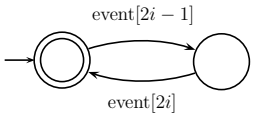
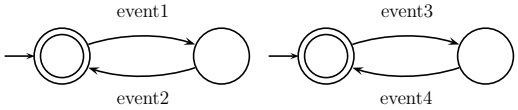


Fig. 6. A prototype automaton with parametrized events.



(a) $i = 1$ (b) $i = 2$

Fig. 7. Components derived from the prototype in Fig. 6 for some values of the parameter i.

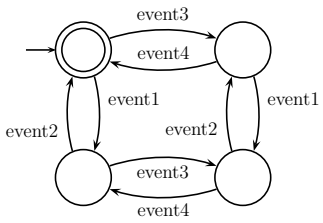


Fig. 8. The compositional unit based on the prototype in Fig. 6 for $1 \leq i \leq 2$.

the specific indices of the events depend on the parameter i, and thus it is possible to construct an arbitrarily large series of similar automata just by varying i. An FSA with parametrized events can be seen in Fig. 6. [1] Let us call the generic structure of the components in a composition a *prototype* and, given F, let us denote it $G^p = (\Sigma_F, Q, \delta, q_0, Q_m)$. Here, the alphabet is $\Sigma_F = \{\sigma_f | f \in F\}$ and all other elements of the tuple are defined as usual.

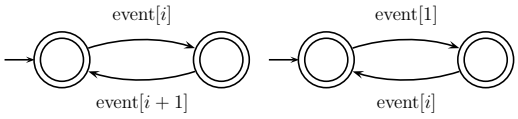
In order to compute compositions of components, it is necessary to derive from the prototype a component for each relevant i. We denote the derived component $G^p(i) = (\Sigma(i), Q, \delta(i), q_0, Q_m)$. Here, $\Sigma(i) = \{\sigma_{f(i)} | \sigma_f \in \Sigma_F\}$ is the set of events constructed by computing the specific event indices for the given i. Similarly, the transition function $\delta(i)$ is derived from the prototype function δ by using the specific event indices for i. In other words, $\delta(i)(q, \sigma_{f(i)}) = q'$ for some $q, q' \in Q$ and $f \in F$ if and only if $\delta(q, \sigma_f) = q'$. The components derived from the prototype in Fig. 6 for $i = 1$ and $i = 2$ are shown in Fig. 7.

We need to define one last construct before we give the formal definition of compositional parametrization. Let us choose the lower and upper bounds, lo and hi, respectively, for the parameter i (where $lo, hi \in \mathbb{N}, lo \leq hi$). Then, we call the composition $\|_{i=lo}^{hi} G^p(i)$ a *compositional unit* and we denote it $\langle G^p, lo, hi \rangle$. The compositional unit of the prototype in Fig. 6 for $lo = 1, hi = 2$ is shown in Fig. 8.

We define the *compositional parametrization* of automata as follows: an FSA $G[k]$ is said to be compositionally parametrized if it can be expressed as

$$G[k] = A \| \langle G_1^p, lo_1(k), hi_1(k) \rangle \| \ldots \| \langle G_l^p, lo_l(k), hi_l(k) \rangle,$$

[1] In the figures, square brackets are used instead of subscripts for readability.



(a) Prototype G_1^p. (b) Prototype G_2^p.

Fig. 9. The prototype automata which can be used for the compositional parametrization of the template which specifies that a number of events have to occur in a sequence.

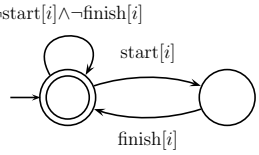


Fig. 10. The prototype for the components of a compositional parametrization of mutual exclusion.

where A is some FSA, $l \in \mathbb{N}$, $G_1^p, \ldots, G_l^p$ are some prototype FSAs, and for $j \in [1, l] : lo_j : \mathbb{N} \to \mathbb{N}, hi_j : \mathbb{N} \to \mathbb{N}$ are parametrized boundaries such that $\forall k : lo_j(k) \leq hi_j(k)$. In other words, a compositionally parametrized FSA can be expressed as the composition of a conventional automaton and a number of compositional units.

Now, let us return to templates from Fig. 4. A template, T, with compositional parametrization can be constructed as follows: $T[k] = \langle G_1^p, 1, k-1\rangle \| \langle G_2^p, k, k\rangle$, where the prototypes G_1^p and G_2^p are shown in Figs. 9(a) and 9(b), respectively. Then, $T[2]$ and $T[3]$ will be the same as the corresponding automata shown in Fig. 4. Furthermore, using this parametrized template, it will be possible to obtain the desired regular FSAs for an arbitrary value $k > 1$ (the model is undefined when $k = 1$).

3.3 Compositional parametrization with selector transitions

In the previous section, we introduced compositional parametrization and we demonstrated how it can be used to parametrize a template. However, compositional parametrization as it stands is not powerful enough to express models where the composition of individual components depends on the synchronization of non-local events (by local events, we mean events expressible as σ_f). Sometimes, synchronization has to occur across a subset of all events where the size of the subset depends on the parameter k. For example, in a parametrized template for mutual exclusion, the number of events that need to be excluded in each component will depend on the overall set of components in the composition. A sketch of the desired solution is shown in Fig. 10.

Here, we propose an extension of compositional parametrization which allows one to define parametrized transitions which depend on the final size of the composition, k. First, let us consider a set, H, of indicator functions selecting a subset of $\mathbb{N}$ depending on a parameter $i \in \mathbb{N}$, i.e., $\forall h \in H,\ h : \mathbb{N} \times \mathbb{N} \to \{0, 1\}$. For a given $h \in H$ and some parameter $i \in \mathbb{N}$, let us also denote the subset selected by the function $h_N(i) = \{n \in \mathbb{N} | h(i, n) = 1\}$. We call a transition labelled with σ_h for some $h \in H$ a *selector transition*. The label σ_h does not directly introduce new events into a model, rather it serves to select a subset of already existing events. For example, the transition on $\sigma_{i\neq 2}$ will select the events $\Sigma_\infty \setminus \{\sigma_2\}$, intersected with the set of events actually used in the composition. In this sense, the set of selected events may vary with the number of components in a composition. The definition of a prototype is updated to be $G^p = (\Sigma_{FH}, Q, \delta, q_0, Q_m)$, where $\Sigma_{FH} = \{\sigma_f | f \in F\} \cup \{\sigma_h | h \in H\}$.

When selector transitions are used, it is no longer possible to derive a component from the prototype based only on the parameter i. The exact semantics of selector transitions (i.e., which events are selected) depend also on which specific components participate in the composition. Thus, when deriving components from a prototype, it is necessary to choose the range of i *a priori*. Let $lo, hi \in \mathbb{N}, lo \leq hi$ are some chosen bounds for i, $lo \leq i \leq hi$. Let $R(F, lo, hi) = \{d | \exists f \in F, \exists i, lo \leq i \leq hi, f(i) = d\}$, i.e., it signifies the union of the ranges of the functions in F over the interval $[lo, hi]$. In other words, $R(F, lo, hi)$ contains the indices of all events σ_f that will appear in the composition for $lo \leq i \leq hi$. Then, we denote the derived component ${}^{hi}_{lo}G^p(i) = ({}^{hi}_{lo}\Sigma(i), Q, {}^{hi}_{lo}\delta(i), q_0, Q_m)$. Here, ${}^{hi}_{lo}\Sigma(i)$ is the set of events constructed as follows.

- All regular parametrized events belong to the event set, $\Sigma(i) \subseteq {}^{hi}_{lo}\Sigma(i)$.
- All events in the composition which are selected by a selector transition belong to the event set, $\{\sigma_d | h \in H, d \in h_N(i) \cap R(F, lo, hi)\} \subseteq {}^{hi}_{lo}\Sigma(i)$.
- No other events belong to ${}^{hi}_{lo}\Sigma(i)$.

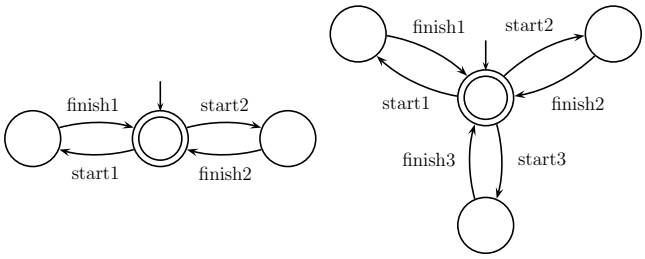
The transition function ${}^{hi}_{lo}\delta(i)$ is defined as follows.

- The transition function is identical to $\delta(i)$ for all regular parametrized events. If $\delta(q, \sigma_f) = q'$ for some $q, q' \in Q$ and $f \in F$, then ${}^{hi}_{lo}\delta(i)(q, \sigma_{f(i)}) = q'$.
- Selector transitions are replaced with individual transitions for all the selected events. If $\delta(q, \sigma_h) = q'$ for some $q, q' \in Q$ and $h \in H$, then $\forall d \in h_N(i) \cap R(F, lo, hi)$, ${}^{hi}_{lo}\delta(i)(q, \sigma_d) = q'$.
- No other transitions exist in the derived component, i.e., in all other cases, ${}^{hi}_{lo}\delta(i)$ is undefined.

Finally, we need to update the definition of compositional units. The compositional unit $\langle G^p, lo, hi\rangle \stackrel{def}{=} \|_{i=lo}^{hi}\, {}^{hi}_{lo}G^p(i)$. The definition of compositional parametrization takes advantage of this update automatically.

As indicated previously, the prototype component for a mutual exclusion template will have to use selector transitions in order to exclude the occurrences of other events, *regardless of how many they are*. If we call the model from Fig. 10 G^p, then we can use compositional parametrization to define the parametrized mutual exclusion template as follows: $MUTEX[k] = \langle G^p, 1, k\rangle$. Note that the event labels in the automaton on Fig. 10 do not follow the expected format only for the purpose of convenience. As already explained, the variation of event symbols can be mapped to events in Σ_∞. Furthermore, once such a mapping is defined, the selector transition in Fig. 10 can be rewritten to conform to the format used in the definition of selector transitions. The regular FSAs which can be obtained from $MUTEX[k]$ for $k = 2$ and $k = 3$ are shown in Fig. 11.

Finally, let us point out that, under certain conditions, selector transitions can be expressed as the event masks



(a) $k = 2$ (b) $k = 3$

Fig. 11. The FSAs obtained from the parametrized mutual exclusion template $MUTEX[k]$ for some values of the parameter k.

introduced in Kumar and Heymann (2000). However, it might be necessary to define separate event masks for every pair of interacting components (compositional units), of which there can be an arbitrary number—in essence necessitating the use of parametrization for masks.

4. CONCLUSIONS

In this paper we introduced two techniques for the parametrization of templates to be used with the template design methodology. The first technique makes use of FSAs extended with variables, e.g., as per Sköldstam et al. (2007). The second technique consists of varying the number of components in the composition which constitutes the template model, somewhat similar to the approach of Bherer et al. (2009). We also discussed an extension of the compositional parametrization by using special selector transitions which allow the modelling of interactions between components depending on the total number of components in a composition. All parametrization techniques and extensions result, with the choice of a specific parameter, in the synthesis of regular FSA models. Thus, such parametrized templates can be easily introduced in the template design modelling environment, Grigorov et al. (2010), and they integrate seamlessly with the supervisory control framework.

In our opinion, the most important next step to be taken is to implement the proposed parametrization techniques in the template design software from Grigorov et al. (2010). This will make parametrized templates available to users, and allow us to evaluate the usefulness and usability of the techniques. We also envision the development of in-use parametrization approaches, where instantiated templates will be parametrized automatically depending on how they are used. For example, an instance of a "mutual exclusion" template could vary its behavior dynamically depending on how many entities are connected to it. Finally, we believe that further investigation of the relation between compositional parametrization of templates and controller synthesis for parametrized DESs, Bherer et al. (2009), may lead to interesting results. Namely, it may be possible to use the scalable controller synthesis method within a template design when all prerequisites are met.

ACKNOWLEDGMENT

We would like to thank José Cury from the Department of Automation and Systems, Federal University of Santa Catarina, Brazil for the lively discussions which stimulated our research on template parametrization.

REFERENCES

Bherer, H., Desharnais, J., and St-Denis, R. (2009). Control of parameterized discrete event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 19(2), 213–265.

Chen, Y.L. and Lin, F. (2000). Modeling of discrete event systems using finite state machines with parameters. In *Proceedings of the 2000 IEEE International Conference on Control Applications*, 941–946. Anchorage, Alaska, USA.

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

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

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

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

Grigorov, L., Butler, B.E., Cury, J.E.R., and Rudie, K. (2010). Conceptual design of discrete-event systems using templates. To appear in *Journal of Discrete-Event Dynamic Systems*.

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

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

Kumar, R. and Heymann, M. (2000). Masked prioritized synchronization for interaction and control of discrete event systems. *IEEE Transactions on Automatic Control*, 45(11), 1970–1982.

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

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

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

Sköldstam, M., Åkesson, K., and Fabian, M. (2007). Modeling of discrete event systems using finite automata with variables. In *Proceedings of the 46th IEEE Conference on Decision and Control*, 3387–3392. New Orleans, LA, USA.