12th IFAC/IEEE Workshop on Discrete Event Systems
Cachan, France. May 14-16, 2014

Unified Model for Synthesis and Optimization of Discrete Event and Hybrid Systems ⋆

Bengt Lennartson * **Oskar Wigström** * **Martin Fabian** *
Francesco Basile **

* *Department of Signals and Systems, Chalmers University of Technology, SE-412 96 Göteborg, Sweden (e-mail: bengt.lennartson@chalmers.se).*
** *Dip. Ingegneria dell'Informazione, Ingegneria elettrica e Matematica applicata (DIEM), Università di Salerno, Italy, (e-mail: fbasile@unisa.it).*

Abstract: A recently proposed generic discrete event model is further developed and exemplified in this paper. Since every transition is expressed as a predicate on the current and next values of a set of variables, the model is called Predicate Transition Model (PTM). It is briefly illustrated how a number of well known discrete-event models, including automata and Petri nets extended with shared variables, can be formulated and synthesized in the PTM framework. More specifically modular Petri nets with shared variables (PNSVs) are shown to be significantly more readable compared to ordinary Petri nets. PTMs are also naturally extended to hybrid systems, and finally it is shown how easy and efficiently PNSVs can be optimized concerning performance based on Constraint Programming. To summarize, the proposed modeling framework unifies and simplifies both synthesis, optimization and implementation of discrete event systems.

Keywords: discrete-event systems, automata, Petri net, supervisory control, synthesis, hybrid systems, optimization

1. INTRODUCTION

The lack of a unified model representation for discrete signal and discrete event systems (DESs) is well known, see Cassandras and Lafortune (2008). For continuous systems differential equations and transfer functions serve this purpose. In a recent paper a unified model for DESs is proposed, called State-Vector Transition model, Lennartson et al. (2014). Inspired by classical continuous-time state space models, the state of a system is represented by a number of state variables x_j, where the domain of definition of the individual variables can be symbolic states as in automata, or integer values as in Petri nets. All variables together constitute a state-vector x, and a transition from one value of x to an updated next value $\acute{x}$ is enabled when a related predicate $\mathcal{C}(x, \acute{x})$ is satisfied. To emphasize the predicate formulation of the transition model, it is in this paper called Predicate Transition Model (PTM).

This type of model was introduced by Manna and Pnueli (1991) as a generic model for transition systems, and the predicate $\mathcal{C}(x, \acute{x})$ is a natural formulation in model checking , see Clarke et al. (2000). In the supervisory control community a slightly different formulation has been used based on predicate transformers, cf. Kumar et al. (1993). The transition predicate $\mathcal{C}(x, \acute{x})$ is shown to be very useful both from a modeling and a computational point of view.

Communication between different discrete event models is often obtained by shared events and full synchronous composition Hoare (1978), as in automata and Petri nets. For automata extended with variables, one version called Extended Finite Automata (EFAs), Sköldstam et al. (2007), communication and synchronization can also be determined by *shared variables* that are updated in more than one EFA. The same feature is used for PTMs, where it is assumed that any variable in the state-vector x can be assigned to new values in more than one model.

In Lennartson et al. (2014) it is shown that the suggested PTM includes automata, EFAs, Petri nets (PNs) and colored Petri nets (CPNs) and their synchronous composition as special cases. Since EFAs are automata extended with variables, PNs are also extended with additional shared variables, including guards and actions. This model, called *Petri net with shared variables* (PNSV), is an interesting complement to the formal PTM, that does not include any specific graphical model structure. In many situations a set of local PNSVs can give a clear and understandable graphical model. For the suggested PTM a synthesis procedure is also developed where supervisor guards are generated. Based on a set of local plant and specification models, synchronized in the PTM framework, it is shown how a controllable, nonblocking and maximally permissive supervisor can be computed.

The contribution of this paper is to briefly summarize the PTM in Lennartson et al. (2014), and clarify some important aspects related to controllability. The benefit of modular PNSV is also further exemplified and motivated in this paper. Already for a system with three straight sequences of token flows, it is illustrated how an ordinary PN becomes hard to read, while the PNSV model gives a clear view of the modeled behavior. It is also shown how PTMs are naturally extended to include continuous behavior, which results in modular Hybrid Predicate Transition Models (HPTMs). Finally, it is illustrated

⋆ This work was carried out within the Wingquist Laboratory VINN Excellence Centre within the Area of Advance - Production at Chalmers, and supported by VINNOVA and the Swedish Science Foundation. The support is gratefully acknowledged.

how especially the PNSV model has a structure that makes it extremely simple to convert to a Constraint Programming (CP) model, which is used for minimization of the make span for the PNSV model discussed above. Note that CP, see Baptiste et al. (2001) is most often significantly more efficient for this type of performance optimization compared to more classical Mixed Integer Linear Programming (MILP) solvers, Manne (1960). To summarize: the proposed PTM unifies the classical DES model structures, also including discrete signals in a natural way, it gives a modular framework both for discrete event and hybrid systems, and the model is shown to be useful both for supervisor synthesis and performance optimization.

2. PREDICATE TRANSITION MODEL

A formal definition of PTMs is given in this section, see further details in Lennartson et al. (2014) where the model is called state-vector transition model. Since PTMs include a number of predicates, note that a subset W of a set X also can be defined by the *predicate* mapping $\mathcal{W} : X \to \mathbb{B}$ as $\mathcal{W}(x) = 1$ iff $x \in W$ and $\mathcal{W}(x) = 0$ iff $x \notin W$.

2.1 Definition of the Predicate Transition Model

Consider a universal ordered set (tuple) of discrete variables $(x_1, \ldots, x_n)$, where the domain of definition for each variable x_j is X_j. A subset of these variables are included in a tuple x, for which a transition model is now defined.

Definition 1. (Predicate Transition Model). A predicate transition model G is a 6-tuple

$$G = \langle \Omega_x, X, \Sigma, T, \mathcal{X}_i, \mathcal{X}_m \rangle \tag{1}$$

where:

(i) $\Omega_x = \{j_1, \ldots, j_n\}$ is the index set for the tuple $x = (x_{j_1}, \ldots, x_{j_n})$.
(ii) $X = X_{j_1} \times \cdots \times X_{j_n}$ is the domain of definition for x.
(iii) Σ is a finite set of events.
(iv) T is a finite set of transitions. Each transition is a tuple $(\sigma, \mathcal{C})$, where $\sigma \in \Sigma$ and $\mathcal{C} : X \times X \to \mathbb{B}$ is a predicate on the current value x and the next value $\acute{x}$.
(v) $\mathcal{X}_i : X \to \mathbb{B}$ is a predicate, defining possible initial values of x.
(vi) $\mathcal{X}_m : X \to \mathbb{B}$ is a predicate, defining desired marked values of x.

□

The reason to introduce the index set Ω_x is that variables will later be arbitrarily shared between different local models. Generally, the domain of the variables may be infinite, as for unbounded PNs, but for computational reasons a finite domain is normally assumed. The predicates are generated by boolean expressions, including conjunction $\wedge$, disjunction $\vee$, and negation $\neg$, while relations between variable values involve the operators $=, \neq, <, >, \leq$, and $\geq$.

A transition $(\sigma, \mathcal{C})$ is *enabled* when the predicate $\mathcal{C}(x, \acute{x})$ is true. When the enabled transition is executed the event σ occurs. For a model with tuple $x = (x_1, x_2)$, a transition predicate $\mathcal{C} \equiv x_1 < 2 \wedge x_2 = 0 \wedge \acute{x}_1 = 2$ means that the transition is enabled when $x_1 < 2 \wedge x_2 = 0$, and the next value of x_1 is $x_1 = 2$. Since there is no condition on the next value for x_2, a natural assumption is that it keeps its current value, i.e. $\acute{x}_2 = x_2 = 0$, cf. Sköldstam et al. (2007); Lennartson et al. (2014).

Also note the condition on the next value $\acute{x} \in X$. When for instance the domain $X = \{0, 1, 2\}$, the conditions $\acute{x} = x + 1$ and $\acute{x} = x - 1$ implicitly include the additional guards $x < 2$ and $x > 0$, respectively. These conditions do not need to be explicitly introduced, since they are achieved by the domain of definition for $\acute{x}$.

2.2 State Transition Model

For analysis and synthesis purposes, the predicate transition model in (1) is now formulated as an explicit state transition model. This model defines the semantics of the PTM. To formally define the fact that no condition on $\acute{x}_j$ in $\mathcal{C}(x, \acute{x})$ implies that x_j keeps its current value, consider the index set

$$\Omega_{\mathcal{C}} = \{j \mid \text{condition on } \acute{x}_j \text{ in } \mathcal{C}(x, \acute{x})\}$$

This means that any expression involving $\acute{x}_j$, such as $\acute{x}_j = x_j + 2$ or $\acute{x}_j < 3$, implies that $j \in \Omega_{\mathcal{C}}$. No condition on $\acute{x}_j$ in $\mathcal{C}(x, \acute{x})$ yields $j \in \Omega_x \setminus \Omega_{\mathcal{C}}$, and the variable x_j will be assumed to keep its current value, i.e. $\acute{x}_j = x_j$. Introducing the keep-current-value predicate

$$\mathcal{C}_{cv}(x, \acute{x}) \equiv \bigwedge_{j \in \Omega_x \setminus \Omega_{\mathcal{C}}} \acute{x}_j = x_j, \tag{2}$$

the *complete transition predicate* for transition $(\sigma, \mathcal{C})$ becomes

$$\Phi(x, \acute{x}) \equiv \mathcal{C}(x, \acute{x}) \wedge \mathcal{C}_{cv}(x, \acute{x}) \tag{3}$$

Consider e.g. the predicate $\mathcal{C} \equiv x_1 = 1 \wedge \acute{x}_2 = 3$ and the index set $\Omega_x = \{1, 2\}$. Then $\Omega_{\mathcal{C}} = \{2\}$, and the complete transition predicate $\Phi \equiv x_1 = 1 \wedge \acute{x}_2 = 3 \wedge \acute{x}_1 = x_1$.

The defined complete transition predicate is the basis for the definition of the explicit state transition model. Indeed, it can be considered as an *evaluated* PTM, where $\Phi(x, \acute{x})$ is determined for all possible combinations of values of the variables.

Definition 2. (State Transition Model). A state transition model (STM) of a PTM $G = \langle \Omega_x, X, \Sigma, T, \mathcal{X}_i, \mathcal{X}_m \rangle$ is a 5-tuple

$$\widehat{G} = \langle X, \Sigma, \rightarrow, X_i, X_m \rangle \tag{4}$$

where X and Σ are defined in Definition 1, and using the complete transition predicate $\Phi(x, \acute{x})$ in (3)

- $\rightarrow = \{(x, \sigma, \acute{x}) \in X \times \Sigma \times X \mid \exists (\sigma, \mathcal{C}) \in T : \Phi(x, \acute{x})\}$;
- $X_i = \{x \in X \mid \mathcal{X}_i(x)\}$;
- $X_m = \{x \in X \mid \mathcal{X}_m(x)\}$.

□

The STM $\widehat{G}$ is an automaton, where the state transition relation is written $x \xrightarrow{\sigma} \acute{x}$, which is recursively extended to strings in Σ^* by letting $x \xrightarrow{\varepsilon} x$ for all $x \in X$, and $x \xrightarrow{s\sigma} z$ if $x \xrightarrow{s} y$ and $y \xrightarrow{\sigma} z$ for some $y \in X$. A path from an initial state in $\widehat{G}$ to a state x is written $\widehat{G} \xrightarrow{s} x$, while a path from a state x to a marked state in X_m is denoted $x \xrightarrow{s} X_m$. Furthermore, $x \xrightarrow{s}$ means that $x \xrightarrow{s} y$ for some $y \in X$, and $x \xrightarrow{s}\!\!\!\!\!\!\rightarrow\ y$ implies that no string $s \in \Sigma^*$ exists such that $x \xrightarrow{s} y$, while $x \xrightarrow{s}\!\!\!\!\!\!\rightarrow$ means that $x \xrightarrow{s}\!\!\!\!\!\!\rightarrow\ y$ for all $y \in X$.

The PTM G in (1), and its corresponding STM $\widehat{G}$ in (4), naturally include nondeterministic behavior, but a *deterministic* PTM has a single initial state, and the transitions $x \xrightarrow{\sigma} \acute{x}$ and $x \xrightarrow{\sigma} \grave{x}$ always imply $\acute{x} = \grave{x}$. Many properties of a PTM are naturally expressed by its STM representation (4). One

important exception is however the synchronous composition, presented in the next subsection.

2.3 Synchronous Composition

The synchronous composition of PTMs is defined based on Hoare's full synchronous composition Hoare (1978), but extended to include shared variables.

Definition 3. (Synchronous Composition of PTMs). Let $G^k = \langle \Omega_x^k, X^k, \Sigma^k, T^k, \mathcal{X}_i^k, \mathcal{X}_m^k \rangle$, $k = 1, 2$, be two PTMs, including their individual tuple of variables x^k. The synchronous composition of G^1 and G^2 is then defined as

$$G^1 \| G^2 = \langle \Omega_x^1 \cup \Omega_x^2, X, \Sigma^1 \cup \Sigma^2, T, \mathcal{X}_i^1 \wedge \mathcal{X}_i^2, \mathcal{X}_m^1 \wedge \mathcal{X}_m^2 \rangle$$

where the domain of definition X and the corresponding tuple of variables x are defined based on the index set $\Omega_x^1 \cup \Omega_x^2$, according to Definition 1. The transition $(\sigma, \mathcal{C}) \in T$ is defined for each combination of $(\sigma, \mathcal{C}^k) \in T^k$, $k = 1, 2$, such that

$$\mathcal{C}(x, \acute{x}) \equiv \begin{cases} \mathcal{C}^1(x^1, \acute{x}^1) \wedge \mathcal{C}^2(x^2, \acute{x}^2), & \sigma \in \Sigma^1 \cap \Sigma^2 \\ \mathcal{C}^1(x^1, \acute{x}^1), & \sigma \in \Sigma^1 \backslash \Sigma^2 \\ \mathcal{C}^2(x^2, \acute{x}^2), & \sigma \in \Sigma^2 \backslash \Sigma^1 \end{cases} . \quad (5)$$

□

When the next value conditions in G^1 and G^2 are in conflict (due to update of one or more shared variables to different values), no transition will occur. The index set $\Omega_\mathcal{C}$ that depends on $\mathcal{C}(x, \acute{x})$ can also be determined by the local index sets, where $\Omega_\mathcal{C} = \Omega_\mathcal{C}^1 \cup \Omega_\mathcal{C}^2$ for $\sigma \in \Sigma^1 \cap \Sigma^2$, $\Omega_\mathcal{C} = \Omega_\mathcal{C}^1$ for $\sigma \in \Sigma^1 \backslash \Sigma^2$, and $\Omega_\mathcal{C} = \Omega_\mathcal{C}^2$ for $\sigma \in \Sigma^2 \backslash \Sigma^1$.

The reason why the keep-current-value predicate $\mathcal{C}_{\Omega_x \backslash \Omega_\mathcal{C}}$, is not involved in the synchronization in (5), only the predicate condition $\mathcal{C}$, is that shared variables can be updated in different PTMs. The keep-current-value predicate is therefore a global property that can be determined first after all local models have been synchronized. Then the set $\Omega_\mathcal{C}$ for the global model and the complete transition predicate $\Phi(x, \acute{x})$ in (3) are determined.

3. RELATED DISCRETE EVENT MODELS

The relation between the proposed PTM and other well known discrete event models is now discussed. Especially, the benefit of PNs with shared variables (PNSVs) is illustrated.

3.1 Models Based on Automata

Finite automata An ordinary finite automaton (FA) can be described by only one variable x, taking symbolic values from a discrete set $X = \{q_1, \ldots, q_n\}$. A transition from a state q_i to a state q_j is modeled by the predicate $\mathcal{C}_{FA}(x, \acute{x}) \equiv x{=}q_i \wedge \acute{x}{=}q_j$.

Extended finite automata (Sköldstam et al. (2007)) An EFA is an automaton extended with a tuple of variables v with domain V, and a location variable ℓ, with domain $L = \{\ell_1, \ldots, \ell_n\}$. Hence, the total tuple of variables is $x = (\ell, v)$, and $X = L \times V$. The condition predicate $\mathcal{C}$ is specialized into a guard condition g on the current value of v, and an action function that updates v to a new value $\acute{v} := a(v)$. A transition from location ℓ_i to location ℓ_j is represented by the following PTM predicate $\mathcal{C}_{EFA}(x, \acute{x}) \equiv \ell{=}\ell_i \wedge \acute{\ell}{=}\ell_j \wedge g_{ij}(v) \wedge \acute{v} = a_{ij}(v)$, including corresponding guard and action function.

3.2 Models Based on Petri Nets

Petri nets A PN is a bipartite directed graph where a set P of *places* is connected to a set T of *transitions*. $Pre(p_i, t_j) = w$ ($Post(p_i, t_j) = w$) means that there is an arc from place p_i (transition t_j) to transition t_j (place p_i) with weight w. A *marking* is a vector m that assigns to each place p_i a non-negative integer number of tokens denoted as m_i. Furthermore, a transition t_j is *enabled* iff $m_i \geq Pre(p_i, t_j)$ for each place $p_i \in P$. If t_j is enabled it may fire, yielding an updated marking $\acute{m}$ for each place p_i as $\acute{m}_i = m_i + Post(p_i, t_j) - Pre(p_i, t_j)$.

In a PTM, the marking vector m for a PN is the tuple of variables, and the transition predicate for transition t_j can be formulated as

$$\mathcal{C}_{PN}(m, \acute{m}) \equiv \bigwedge_{i=1}^{n} \big(m_i \geq Pre(p_i, t_j) \wedge \acute{m}_i = m_i + Post(p_i, t_j) - Pre(p_i, t_j) \big). \quad (6)$$

Petri nets with shared variables In the same way as variables can be added to ordinary automata, resulting in EFAs, shared variables and related guards and actions can be added also to a Petri net, resulting in a PNSV. Extra variables in a tuple v then results in the total tuple of variables $x = (m, v)$. Additional guards and actions on the transitions in an ordinary PN are then included in a predicate $\mathcal{C}_V$, which results in the following transition predicate for a PNSV

$$\mathcal{C}_{PNSV}(x, \acute{x}) \equiv \mathcal{C}_{PN}(m, \acute{m}) \wedge \mathcal{C}_V(x, \acute{x}). \quad (7)$$

The following example illustrates how a PN can be simplified by introducing shared variables in modular PNSVs.

Example 1. Consider the PN in Fig. 1, where two common resources R_1 and R_2 are shared between three straight sequences. The places and arcs between these sequences model the mutual exclusion conditions for the two resources, as well as the precedence condition c_1 before c_3 for each individual token. In Fig. 2 three PNSVs are presented, where the two shared resource places are replaced by the shared variables R_1 and R_2, and the precedence condition is modeled by the shared variable P. The domain of the variables R_1 and R_2 models their capacity, and with the short cut notation

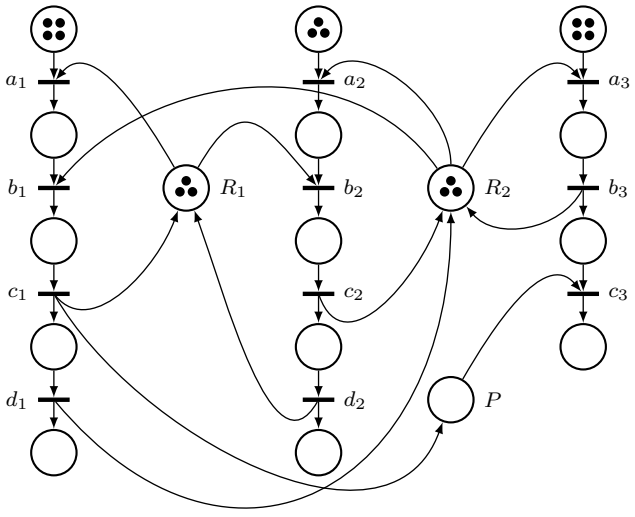

Fig. 1. Petri net including two shared resources R_1 and R_2 and a precedence token place P.

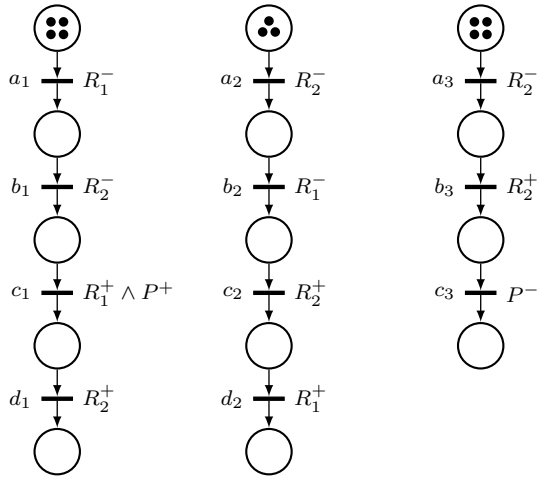

Fig. 2. Three PNSVs with shared variables R_1, R_2 and P, increased and decreased by the short-cut condition (8).

$$R^{\pm} \equiv \acute{R} = R \pm 1 \tag{8}$$

for a variable R, the resources are booked (-) and unbooked (+) in an equivalent way in the ordinary PN in Fig. 1 and the PNSVs in Fig. 2. The precedence condition based on P has also en equivalent behavior.

The PNSVs are easily expressed as three synchronized PTMs, illustrated for a smaller example in Lennartson et al. (2014). Here, we only observe that the transition conditions on the shared variables are exactly those that are given on R_1, R_2 and P in Fig. 2. □

In this example the predicates on the shared variables belong to $\mathcal{C}_V$. Generally, $\mathcal{C}_V$ may include guards and actions on both the number of tokens in m and the variables in v. An example of an action on m is the reset of tokens.

A clear benefit of PNSVs is that graphical modeling can be used where it has its strength, typically showing flows of items between places in a PN. Complicated logical conditions related to mutual exclusion, synchronization, precedence relations, and supervisory control conditions are preferably expressed by logical conditions. This is clear already in the simple PN models in Fig. 1 and Fig. 2, but is much more obvious for larger systems. Logical conditions between places far away from each other generate complex graphical models. These models may be much harder to understand then modularized PNSVs with added explicit transition predicates, where variable names can be assigned to clearly express the meaning of for instance resource booking, as in Fig. 2.

Unified model for synthesis In the next section it will be shown how synthesis of supervisors can be performed for PTMs. Since all models described in this section are special cases of the PTM, it also means that the same synthesis approach can be applied to all these model classes.

4. SUPERVISORY CONTROL

Supervisory control theory (SCT), see Ramadge and Wonham (1987), is a formal framework for synthesizing a supervisor S for a given plant G, such that a specification K is satisfied for the closed loop system $G\|S$. The plant G is normally represented as a number of synchronized subplants, i.e. $G = G^1\| \cdots \|G^{N_G}$. Local specifications K^j, $j = 1, \ldots, N_K$ are also synchronized to a common specification $K = K^1\| \cdots \|K^{N_K}$. The total specification $G\|K$ is then also a *candidate of a supervisor* to control the plant such that the specification K is fulfilled. From now on, a finite number of states is assumed, as well as all models being *deterministic*.

4.1 Controllable and Nonblocking System

In SCT, the events are divided into two disjoint subsets: *controllable events*, denoted by Σ_c, that can be prevented to be executed by the supervisor (i.e., disabled); and *uncontrollable events*, denoted by Σ_u, which cannot be influenced by the supervisor. Two important properties are considered in SCT namely controllability and nonblocking.

Controllability Traditionally, controllability has been defined as a language condition, while a similar state based definition was denoted *completeness*, see Ramadge and Wonham (1987). Later, different definitions of controllability in state space form have also appeared, cf. Fabian (1995); Ouedraogo et al. (2011), following the basic principle that the closed loop system must be able to follow any uncontrollable event enabled in the plant states that the closed loop system can reach. Since our synthesis formulation is state based, the same type of controllability definition is also chosen in this paper.

Before controllability is defined for PTMs, a simple state projection function is introduced. A state x^k in a local model G^k is related to the corresponding state x in a synchronized system G including G^k by the projection function $x^k = P^k(x)$. Hence, $P^k(x)$ only preserves those variables from G that are involved in G^k.

Definition 4. (Controllability). Let $G^k = \langle \Omega_x^k, X^k, \Sigma^k, T^k, \mathcal{X}_i^k, \mathcal{X}_m^k \rangle$, $k = 1, 2$, be two PTMs, where $\Sigma^2 \subseteq \Sigma^1$. Then, G^2 is *controllable* with respect to G^1 and a set of uncontrollable events $\Sigma_u \subseteq \Sigma^1$ if, for every string $s \in \Sigma^{1*}$ and every uncontrollable event $\sigma_u \in \Sigma_u \cap \Sigma^2$ such that $\widehat{G^1\|G^2} \xrightarrow{s} x$ and $P^1(x) \xrightarrow{\sigma_u} P^1(\acute{x})$ in $\widehat{G}^1$, it also holds that $x \xrightarrow{\sigma_u} \acute{x}$ in $\widehat{G^1\|G^2}$. If $x \not\xrightarrow{\sigma_u}$, then x is an *uncontrollable state*. □

This definition is general in the sense that a system with shared variables can model many different scenarios. Assume for instance that G_1 is a plant model where some variables represent internal states in the plant, and other variables are input signals determined by a supervisor G_2. The update of the internal plant state variables are then naturally modeled as uncontrollable transitions, while the input signals are updated at controllable transitions. This means that input variables keep their current values when any internal plant variable is updated at an uncontrollable transition. On the contrary, the plant variables keep their current values when any input signal signal is updated at a controllable transition. Hence, the internal plant state variables can be considered as *uncontrollable variables*, while the input signals to the plant are *controllable variables*. This interpretation gives a signal based formulation of a plant G_1 controlled by a supervisor G_2.

Nonblocking Nonblocking is now defined for an arbitrary PTM G, although the application is the closed loop system $G\|S$.

Definition 5. (Nonblocking). Let $G = \langle \Omega_x, X, \Sigma, T, \mathcal{X}_i, \mathcal{X}_m \rangle$. A state $x \in X$ is *reachable* if $\widehat{G} \xrightarrow{s} x$, and it is *coreachable*

if $x \xrightarrow{s} X_m$ where $X_m = \{x \in X \,|\, \mathcal{X}_m(x)\}$. A PTM G is *nonblocking* if every reachable state is also coreachable. □

4.2 Supervisor State Set Generation

In the generation of a controllable and nonblocking supervisor S, given a supervisor candidate $G\|K$, the first step is to identify all *uncontrollable states* in $G\|K$ with respect to G. The identification of these forbidden states is determined in the following proposition, where two restrictions are included. For the uncontrollable transitions in the specification K it is assumed that guards, but no actions, are included i.e. $\Omega_{\mathcal{C}^K} = \varnothing$, and the variables included in the specification K are assumed to be a subset of the plant variables in G, i.e. $\Omega_x^K \subseteq \Omega_x^G$.

Proposition 1. (Uncontrollable states). Let $G = \langle \Omega_x^G, X^G, \Sigma^G, T^G, \mathcal{X}_i^G, \mathcal{X}_m^G \rangle$ and $K = \langle \Omega_x^K, X^K, \Sigma^K, T^K, \mathcal{X}_i^K, \mathcal{X}_m^K \rangle$. Assume that $\Sigma^K \subseteq \Sigma^G$ and consider all transitions $(\sigma, \mathcal{C}^T) \in T^K$ such that $\sigma \in \Sigma_u$. Furthemore, assume that then $\Omega_{\mathcal{C}^K} = \varnothing$, and $\Omega_x^K \subseteq \Omega_x^G$. The set of uncontrollable states in $G\|K$ with respect to G is then given by the set

$$X_u^{G\|K} = \{x \in X^{G\|K} \,|\, \exists (\sigma, \mathcal{C}^G \wedge \mathcal{C}^K) \in T^{G\|K}, \\ \exists \acute{x} \in X^{G\|K} [\Phi^G(x, \acute{x}) \wedge \neg \mathcal{C}^K(P^K(x), P^K(\acute{x}))]\}$$

Proof: According to Def. 4 for $G^1 = G$ and $G_2 = K$, the statement is proven by evaluating the transitions where the event $\sigma \in \Sigma_u \cap \Sigma^K$ and $\neg(\Phi^G \rightarrow \Phi^{G\|K}) \equiv \Phi^G \wedge \neg \Phi^{G\|K} \equiv \mathcal{C}^G \wedge \mathcal{C}_{cv}^G \wedge \neg(\mathcal{C}^G \wedge \mathcal{C}^K \wedge \mathcal{C}_{cv}^{G\|K})$. Since $\Omega_{\mathcal{C}^K} = \varnothing$, and $\Omega_x^K \subseteq \Omega_x^G$, we find that $\Omega_x^G \cup \Omega_x^K = \Omega_x^G$ and $\Omega_{\mathcal{C}^G} \cup \Omega_{\mathcal{C}^K} = \Omega_{\mathcal{C}^G}$. Thus, $\mathcal{C}_{cv}^{G\|K} = \mathcal{C}_{cv}^G$ and the result follows. □

Most often there is no additional specification predicate on transitions with uncontrollable events, which results in $X_u^{G\|K} = \varnothing$. In Ouedraogo et al. (2011) that is even assumed. However, to avoid unnecessary restrictions, $X_u^{G\|K}$ takes care of cases where guards are also added on transitions with uncontrollable events.

Given the uncontrollable states $X_u^{G\|K}$ and any explicitly forbidden states $X_f^{G\|K}$, a supervisor S can be synthesized by removing these states and additional blocking and extended uncontrollable states. This is done by removing states from the EST model $\widehat{G\|K}$ in a fixed point iteration, starting with $X_f^0 = X_u^{G\|K} \cup X_f^{G\|K}$ and iterating until $X_f^{k+1} = X_f^k = X_f$. To express this, the following forbidden state model is defined.

Definition 6. (Forbidden state model). Let $\widehat{G} = \langle X, \Sigma, \rightarrow, X_i, X_m \rangle$. A corresponding model excluding a set of *forbidden states* $X_f \subseteq X$ is then defined as $\widehat{G}_{\setminus X_f} = \langle X \setminus X_f, \Sigma, \rightarrow_{\setminus X_f}, X_i \setminus X_f, X_m \setminus X_f \rangle$ where $\rightarrow_{\setminus X_f} = \{(x, \sigma, \acute{x}) \,|\, x, \acute{x} \notin X_f\}$ □

The supervisor states, the *safe states*, are $X^S = X^{G\|K} \setminus X_f$, where all reachable blocking and uncontrollable states are included in X_f. Moreover, assume that only those states that are necessary to remove are included X_f. A maximally permissive, controllable and nonblocking supervisor

$$\widehat{S}_{X_f} = \widehat{G\|K}_{\setminus X_f} \tag{9}$$

is then achieved.

4.3 Supervisor Guards

Supervisor guards will now be added to the original PTM $G\|K$, such that the closed loop system will stay within the safe state set X^S. First, two additional state sets are introduced, cf. Miremadi et al. (2011) , the set of states X_σ^S where the event σ must be allowed by $\widehat{S}_{X_f}$, and the set of states $X_{\neg\sigma}^S$ where the event σ must be forbidden. For all other states it does not matter if the supervisor allows or forbids σ. Together with the fact that a supervisor can only restrict controllable events, this implies that for every event $\sigma \in \Sigma_c \subseteq \Sigma_G$ a supervisor guard $\mathcal{C}_\sigma^S(x)$ is generated such that:

$$\mathcal{C}_\sigma^S(x) \equiv \begin{cases} \top, & x \in X_\sigma^S \\ \bot, & x \in X_{\neg\sigma}^S \\ \text{don't care}, & x \in X \setminus (X_\sigma^S \cup X_{\neg\sigma}^S) \end{cases}. \tag{10}$$

These supervisor guards are preferably implemented by adding them to the plant PTMs G^k, $k = 1 \ldots, N_G$. The transition relation sets T^k are then replaced by

$$T_{\mathcal{C}}^k = \{(\sigma, \mathcal{C}^k \wedge \mathcal{C}_\sigma^S) \,|\, (\sigma, \mathcal{C}^k) \in T^k\}$$

resulting in modified plant models $G_{\mathcal{C}}^k$. This is possible since $\sigma \in \Sigma_c \subseteq \Sigma_G$. For a shared event σ, due to the synchronous composition, it is only necessary to add the guard to one of the plant models that include this event. This results in a distributed PTM supervisor

$$S = G_{\mathcal{C}}^1 \| \cdots \| G_{\mathcal{C}}^{N_G} \| K^1 \| \cdots \| K^{N_K}$$

where the synchronous composition may be implemented online. With a reasonable size of the supervisor guards, it is easy to implement this supervisor in an industrial control system, and with smaller guards it is even possible for a user to identify how the supervisor restricts the behavior of the plant.

Example 2. Consider the PNSVs in Fig. 2. Since the resources R_1 and R_2 are booked in opposite order, a deadlock may occur. A maximally permissive and nonblocking supervisor can in this case be formulated, independently of the capacity of R_1 and R_2, by the supervisor guards

$$\mathcal{C}_{a_1}^S \equiv R_1 > 1 \vee (R_2 + m_2^3) > 0, \\ \mathcal{C}_{a_2}^S \equiv R_1 > 0 \vee (R_2 + m_2^3) > 1,$$

where m_2^3 is the number of tokens in the second place after event a_3 in the right PNSV in Fig. 2. This unsymmetric contribution comes from the fact that a token in this place without any restriction follows by an action R_2^+ at the event b_3. These supervisor guards guarantee that the deadlock state $R_1 + R_2 + m_2^3 = 0$ never occurs. □

General techniques to reduce the size of supervisor guards $\mathcal{C}_\sigma^S(x)$ (10) are proposed in Miremadi et al. (2011). The same technique can also be used to generate compact supervisor guards for PTMs. Furthermore, the computation of the safe states in the supervisor, which is based on reachability search, can be made symbolically to save memory and execution time. More specifically, the set of all transition relations $\rightarrow$ can be formulated logically as a transition predicate $\mathcal{T}$. To restrict x and $\acute{x}$ to their domain of definition X, the state predicate $\mathcal{X}(x) \equiv x \in X$ is introduced. Then the following event transition predicate defines the set of transitions with label σ,

$$\mathcal{T}_\sigma(x, \acute{x}) \equiv \bigvee_{(\sigma, \mathcal{C}) \in T} \mathcal{C}(x, \acute{x}) \wedge \mathcal{C}_{cv}(x, \acute{x}) \wedge \mathcal{X}(x) \wedge \mathcal{X}(\acute{x}).$$

The total transition predicate, including all transitions in $\rightarrow$, is finally expressed as $\mathcal{T}(x, \acute{x}) \equiv \bigvee_{\sigma \in \Sigma} \mathcal{T}_\sigma(x, \acute{x})$ This predicate formulation of the transition set $\rightarrow$ can be directly translated

to BDDs, which means that efficient reachability search can be performed.

One critical problem in the BDD formulation is, however, that the number of BDD nodes becomes very large during the construction, while the size of the final BDD is more manageable. A well known recommendation is then to partition the total transition, in our case by formulating individual BDDs for each $\mathcal{T}_\sigma$, which gives a disjunctive partitioning of $\mathcal{T}$. This partitioning approach is a key factor to reduce the memory size (number of nodes in the BDDs), but also the computation time. A typical experience for larger systems is memory overflow when no partitioning is involved. Further details on this partitioning can be found in Fei et al. (2014).

5. HYBRID SYSTEMS

Hybrid systems including a combination of discrete and continuous dynamics are naturally modeled by extending the PTM with continuous variables, and a predicate on the differential equations for these variables. Such a model is called a *hybrid predicate transition model* (HPTM) model

Definition 7. (Hybrid Predicate Transition Model). A hybrid predicate transition model G is an 8-tuple

$$G = \langle \Omega_x, X, \Sigma, T, \mathcal{X}_c, \mathcal{X}_i, \mathcal{X}_m, \mathcal{X}_{inv} \rangle \tag{11}$$

where Ω_x, X, Σ, T, $\mathcal{X}_i$, and $\mathcal{X}_m$ are defined in Definition 1 and

(i) $\mathcal{X}_c : X \times X \to \mathbb{B}$ is a predicate on the current value x and the time derivative $\dot{x}$.
(ii) $\mathcal{X}_{inv} : X \to \mathbb{B}$ is a predicate, defining desired invariants of x that must always be satisfied.

□

Of course, the time derivative of a state has a meaning only if the state has a continuous domain of definition. Assume for simplicity that those states are collected in the vector x_c. Then, even a differential inclusion such as $\dot{x}_c \geq f_1(x_c) \wedge \dot{x}_c \leq f_2(x_c)$ may determine the continuous part of the trajectory. In the same way as keep current value, is the normal assumption for those variables that are not updated in a discrete transition, the default time derivate of a variables x_j is $\dot{x}_j = 0$ for those variables that are not assigned any specific time derivative in $\mathcal{X}_c$.

Example 3. Consider a tank process with an area A, an input flow q_{in}, an output flow q_{out}, and a height level $h(t)$. The input flow is controlled by a discrete valve u, where $u \in \{0, 1\}$. The opening and closing actions of this valve have a time delay T_d that is modeled by a clock $c(t)$. The continuous-time model is then given by the predicate

$$\mathcal{X}_c(x, \dot{x}) \equiv A\dot{h}(t) = q_{in}u - q_{out} \wedge \dot{c} = v,$$

and the following predicates

$$\begin{aligned}
\mathcal{C}_1 &\equiv h \geq h_{max} \wedge u = 1 \wedge \acute{v} = 1 \\
\mathcal{C}_2 &\equiv c = T_d \wedge u = 1 \wedge \acute{c} = 0 \wedge \acute{v} = 0 \wedge \acute{u} = 0 \\
\mathcal{C}_3 &\equiv h \leq h_{min} \wedge u = 0 \wedge \acute{v} = 1 \\
\mathcal{C}_4 &\equiv c = T_d \wedge u = 0 \wedge \acute{c} = 0 \wedge \acute{v} = 0 \wedge \acute{u} = 1
\end{aligned}$$

define the discrete updates of both the continuous variable, the clock c, and the discrete variables u and v. The desired maximum and minimum tank levels are h_{max} and h_{min}, respectively. The invariant predicate $\mathcal{X}_{inv}$, traditionally included in hybrid automata Alur et al. (1993), can be used to determine necessary transitions from a state, while transition conditions in principle can express the same conditions but then at one or more transitions. To conclude, the invariant predicate add some more flexibility in the modeling of transitions from one state to another, either in terms of a continuous or a discrete state jump.

Compared to the well established hybrid automaton, see Alur et al. (1993), this HPTM is more general and flexible. The logical behavior can be expressed not only by transitions between specific locations, but also by combining locations with discrete variables in the same way as discrete variables can be added to ordinary automata and Petri nets.

A synchronization of local HPTMs is also naturally achieved, applying the same rules for $G^1 \| G^2$ as in Definition 3, adding the rules

$$\begin{aligned}
\mathcal{X}_c(x, \dot{x}) &= \mathcal{X}_c^1(x^1, \dot{x}^1) \wedge \mathcal{X}_c^2(x^2, \dot{x}^2) \\
\mathcal{X}_{inv}(x) &= \mathcal{X}_{inv}^1(x^1) \wedge \mathcal{X}_{inv}^2(x^2)
\end{aligned}$$

This formulation makes it easy to formulate modular representations of hybrid systems.

6. OPTIMIZATION

Optimization of discrete event systems is a challenging area. In this session we will focus on performance optimization. This is of special interest for PNs with a token flow as illustrated by the modular PNSVs. Assuming that every token has a minimal execution time in each place, a typical goal is to optimize how fast all tokens can pass from their first to their last place. This time, called the make span will now be minimized for the PNSV model in Fig. 2. It is illustrated how well this model is suited for optimization based on Constraint Programming (CP) Baptiste et al. (2001).

The PNSV model in Fig. 2 is directly translated to a CP model. The solution, although just an example, demonstrates the principles how this can be done for an arbitrary number of local PNSVs, communicating by discrete shared variables.

Introduce the following bounded set of integers $I^n = \{1, \ldots, n\}$ and $Z^n = \{0\} \cup I^n$. For each token $j \in I^{m_0^k}$ in task $k \in I^3$ and place $i \in I^{N^k}$ the execution time is $T^k(i)$. Execution is assumed to occur in all places except the first and the last one. Hence, the first execution place ($i = 1$) is the second place in each straight sequence in Fig. 2.

Given the continuous transition times $t^k(i, j)$ for each token $j \in I^{m_0^k}$, the precedence constraints are

$$t^k(i, j) + T^k(i) \leq t^k(i + 1, j), \quad \forall i \in I^{N^k - 1}$$

where we note that the total duration time in each place may be larger than the execution time. The continuous criterion

$$T_f = \max_{k \in I^3, j \in I^{m_0^k}} t^k(N^k, j),$$

the make span, is minimized. The shared variables $R_1 \in Z^3$, $R_2 \in Z^3$ and $P \in Z^4$ generate restrictions by the following cumulative constraints

$$\begin{aligned}
\text{cum}(R_1) = R_1^0 &- \sum_{j=1}^{m_0^1} \text{step}(t^1(1, j), 1) + \sum_{j=1}^{m_0^1} \text{step}(t^1(3, j), 1) \\
&- \sum_{j=1}^{m_0^2} \text{step}(t^2(2, j), 1) + \sum_{j=1}^{m_0^2} \text{step}(t^2(4, j), 1)
\end{aligned}$$

$$\text{cum}(R_2) = R_2^0 - \sum_{j=1}^{m_0^1} \text{step}(t^1(2,j),1) + \sum_{j=1}^{m_0^1} \text{step}(t^1(4,j),1) - \sum_{j=1}^{m_0^2} \text{step}(t^2(1,j),1) + \sum_{j=1}^{m_0^2} \text{step}(t^2(3,j),1) - \sum_{j=1}^{m_0^3} \text{step}(t^3(1,j),1) + \sum_{j=1}^{m_0^3} \text{step}(t^3(2,j),1)$$

$$\text{cum}(P) = P^0 + \sum_{j=1}^{m_0^1} \text{step}(t^1(3,j),1) - \sum_{j=1}^{m_0^3} \text{step}(t^3(3,j),1)$$

with initial values $R_1^0 = 3$, $R_2^0 = 3$ and $P^0 = 0$. The function $\text{step}(t, \ell)$ denotes a step function of magnitude ℓ at time t and $\text{cum}(R)$ specifies that the expression must be within the domain of defininition for variable R at all time instances. The step function can also be regarded as a resource booking interval of infinite length, see Aggoun and Beldiceanu (1993) for details on the cumulative constraint.

It is clear from this example that it is very easy and natural to express a PNSV as a CP model, where the precedence relations between the tokens in each local PNSV are determined by their transition and execution times. What is more interesting is the easy way the integer variables R_1, R_2 and P are updated, representing the logical constraints of the token flow. All incremental updates of a variable say R, by expressions R^+ and R^-, are in the CP model determined by the corresponding step function. This function specifies at which time the corresponding discrete variable will be updated. Then the optimization algorithm distributes the transition times such that the desired criterion is minimized satisfying the necessary constraints.

This example shows how discrete shared variables easily can be introduced in the optimization model. Shared events are also simply handled by introducing the same transition time for those transitions that have the same shared event. If there are more than one shared event with the same label in two local models, all combinations must be included, introducing alternative choices to involve all the alternative scenarios. This formulation is easy to apply, and our experience also shows that the optimization is very efficient. In most cases the CP solver gives an answer significantly faster than for instance a more traditional Mixed Integer Linear Programming (MILP) solver, see Wigström and Lennartson (2012). For the example in this session, the CP solver gave an optimal solution more than ten times faster than a corresponding MILP approach.

7. CONCLUSIONS

A generic predicate transition model has been presented, from which it is shown how a nonblocking, controllable and maximally permissive supervisor can be implemented in terms of control guards added to the original model. Furthermore, since PTMs include automata, EFAs and Petri nets as special cases, the presented framework is a unified, flexible and attractive approach for supervisor synthesis. The suggested PTM has also a graphical correspondence, including shared variables to Petri nets. This flexible model structure is shown to be both more readable than ordinary Petri nets, and a natural formulation for efficient performance optimization based on constraint programming.

REFERENCES

Aggoun, A. and Beldiceanu, N. (1993). Extending chip in order to solve complex scheduling and placement problems. *Mathematical and Computer Modelling*, 17(7), 57–73.

Alur, R., Courcoubetis, C., Henzinger, T., and Ho, P. (1993). Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Lecture Notes in Computer Science*, volume 736, 209–229. Springer Berlin / Heidelberg.

Baptiste, P., Pape, C.L., and Nuijten, W. (2001). *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer.

Cassandras, C.G. and Lafortune, S. (2008). *Introduction to Discrete Event Systems*. Chapter 3, pp. 133-221, Springer, 2nd edition.

Clarke, E.M., Grumberg, O., and Peled, D.A. (2000). *Model Checking*. MIT Press.

Fabian, M. (1995). *On Object Oriented Nondeterministic Supervisory Control*. Ph.D. thesis, Control Engineering Laboratory,Chalmers University of Technology, Göteborg, Sweden.

Fei, Z., Miremadi, S., Åkesson, K., and Lennartson, B. (2014). Efficient Symbolic Supervisor Synthesis for Extended Finite Automata. *IEEE Transactions on Control Systems Technology*, 11.

Hoare, C.A.R. (1978). *Communicating sequential processes*, volume 21 of *Series in Computer Science*. ACM. doi: 10.1145/359576.359585.

Kumar, R., Garg, V., and Marcus, S. (1993). Predicates and predicate transformers for supervisory control of discrete event dynamical systems. *IEEE Transactions on Automatic Control*, 38(2), 232–247.

Lennartson, B., Basile, F., Miremadi, S., Fei, Z., Hosseini, M.N., Fabian, M., and Åkesson, K. (2014). Supervisory Control for State-Vector Transition Models - A Unified Approach. *IEEE Transaction on Automation Science and Engineering*, 11(1), 33–47.

Manna, Z. and Pnueli, A. (1991). *The temporal logic of reactive and concurrent systems*. Springer-Verlag New York, Inc., New York, NY, USA.

Manne, A.S. (1960). On the job-shop scheduling problem. *Operations Research*, 8(2), 219–223.

Miremadi, S., Åkesson, K., and Lennartson, B. (2011). Symbolic Computation of Reduced Guards in Supervisory Control. *IEEE Transactions on Automation Science and Engineering*, 8(4), 754–765. doi:10.1109/TASE.2011.2146249.

Ouedraogo, L., Kumar, R., Malik, R., and Åkesson, K. (2011). Nonblocking and Safe Control of Discrete-Event Systems Modeled as Extended Finite Automata. *IEEE Transactions on Automation Science and Engineering*, 8(3), 560–569.

Ramadge, P. and Wonham, W. (1987). Supervisory control of a class of discrete event processes. *SIAM Journal of Control and Optimization*, 25(1), 635–650.

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

Wigström, O. and Lennartson, B. (2012). Scheduling model for systems with complex alternative behaviour. In *Proc. 8th IEEE Conference on Automation Science and Engineering (CASE 2012)*, 587–593. Seoul.