Proceedings of the
46th IEEE Conference on Decision and Control
New Orleans, LA, USA, Dec. 12-14, 2007

Modeling of Discrete Event Systems using Finite Automata With Variables

Markus Sköldstam, Knut Åkesson and Martin Fabian

Abstract— **To get industrial acceptance of supervisory control theory, there is a need to bridge the gap between the signal-based industrial reality and the event-based supervisory control framework. This paper tries to do this by introducing a modeling formalism with automata extended with variables, guard expressions and action functions. The formalism is suitable for modeling plants and specifications in the supervisory control framework. An algorithm that transforms a set of extended automata into a set of ordinary automata with equivalent behavior, is presented. This allows the user to model complex behaviors with a compact representation, and at the same time use existing algorithms for analysis.**

I. INTRODUCTION

Discrete event systems (DES) are models of systems that at each time instant occupy a discrete state, and perform state-changes on the occurrence of events. Examples of such systems are manufacturing systems, communication networks and embedded systems. The behavior of a DES is described by the sequences of events that may occur, and the sequences of states that may be visited.

Supervisory control theory [1] is a general approach to synthesize control systems for DES. A supervisor may be generated using models of the plant and the specification, such that it is minimally restrictive with respect to the plant behavior, while still guaranteeing that the specification is upheld. Traditionally, regular languages and finite automata [2] have been used both for modeling and analysis of discrete event systems in the supervisory control community.

Though a large amount of promising research results have been achieved in academia, industrial acceptance of the supervisory control theory is scarce. Only a few examples have been reported [3]. A number of issues that hinder industrial use have been identified by various researchers [3], [4], [5]. Two main issues are the discrepancy between the signal-based reality and the event-based automata framework, and the lack of a compact representation of large models.

In many industrial applications, parts of a system, such as sensors, actuators and buffers, are conveniently modeled using variables. Guard expressions are used to restrict the behavior of the system and action functions are used to update variables. Physical signals that are stored in memories or sent between controllers are naturally modeled as global variables in DES models. Using variables, guards and actions help us to compactly represent large and complicated DES.

A number of frameworks have been introduced that allow compact representations of discrete event systems with complex behavior and large state-spaces. Many of these are inspired by Statecharts [6], which extends automata with hierarchy, concurrency, and communication using variables, guards and actions. While most of the concepts introduced in Statecharts are useful for modeling supervisory control problems, Statecharts is not in its original formulation suitable for supervisory control. In the supervisory control framework it is essential to model what *may* occur instead of what *should* occur, this has large consequences for how the interaction between subsystems are modeled. In Statecharts there is a causality between subsystems, this is not desired in the supervisory control framework.

Modeling frameworks based on automata extended with variables, suitable for supervisory control, are presented in [7], [8], [9], [10]. In [7] it is assumed that a variable can be updated by at the most one extended automaton and in order to do synthesis the state-space needs to be extended by additional states. In [8] automata with variables are used to implement a supervisor. The authors encode the states of a given supervisor using boolean variables. The variables are used in guards and actions attached to the events of the model. In [9] supervisory control is applied to a number of automata with variables. To ensure a least restrictive supervisor it is assumed that all variables are local i.e. not shared between automata. This is a quite strong restriction because variables cannot be used to model any interaction between subsystems. In [10] a state transition structure with a data collection used for parameterized and non-regular discrete event systems is introduced.

Though extended frameworks allow compact representations of huge state-spaces, and hence simplify the modeling of systems of industrially interesting sizes, the states do not disappear and hence potentially pose a problem when it comes to analysis. The main problem is the state-space explosion that typically occurs when the behavior of interacting sub-systems is studied. For ordinary automata, especially in the context of supervisory control, there exists a large body of work for fighting this state-space explosion. For extended frameworks, less has been done. An attractive approach is to develop algorithms that benefit from the structure given by extended modeling frameworks, see [11] and its references.

Without doubts, developing effective synthesis algorithms for automata that share variables needs to be explored further. However, it is equally important to be able to use existing algorithms that have been tested and have been proven to handle systems with large state-spaces. In this paper we present no new algorithms for analysis of extended automata models. Our approach is to transform extended models into

Department of Signals and Systems, Chalmers University of Technology, Sweden. Corresponding author: markus.skoldstam@chalmers.se

1-4244-1498-9/07/$25.00 ©2007 IEEE.

ordinary automata models with the same behavior. This way we can use extended automata for modeling and regular automata for analysis. In order to fully understand the relation between regular finite automata and extended finite automata, we feel that a new and very detailed definition of extended automata is needed. Since we only model physical systems with finite state-spaces and the current variable values are part of the system state, we only consider variables that have *finite* domains of definition.

We present a modeling formalism with automata extended with variables, guard expressions and action functions. We attach guards and actions functions to transitions since this admits local design techniques of systems consisting of many different parts. In comparison to previous work we do not put any restrictions on how variables are shared between extended automata, thus all extended automata are allowed to update all variables as long as the composition is well defined. The presented framework has been implemented in the supervisory control tool Supremica [12], [13]. In [14] it is argued that the extended finite automata (EFA) introduced in this paper may be used as a basis for efficiently representing control problems that consists of mixed logic and supervisory control problems.

We start with a motivating example of a model of a system with a complex controller specification (Section II). Section III and Section IV provide the notations used for finite automata (FA) and extended finite automata (EFA), which is our modeling framework. In Section V we present and prove a basic algorithm that transforms a single EFA to an equivalent FA. It is explained how the algorithm is extended to handle an arbitrary number of interacting EFA, and how to use the algorithm to efficiently analyze the behavior EFA models.

II. MODELING A DOSING TANK

This section illustrates some of the advantages of using EFA as a modeling tool compared with FA. We have chosen to model a unit in a chemical batch plant. The system consists of a tank and a user. The tank has an inlet valve, an outlet valve and two sensors to check the filling of the tank, S1 at the bottom and S2 at the top of the tank. Filling the tank or emptying the tank, can be requested to start or stop by the user. To meet the user requests, a supervisor/controller is designed that closes and opens the valves appropriately.

The plant and the supervisor are modeled in figures 1 and 2, respectively. State changes are modeled by the events **s1_on, s1_off, s2_on, s2_off, req_stop, req_start, close_in, open_in, close_out** and **open_out**. Sensor signals are modeled by the variables v_{s1} and v_{s2}, request signals from the user by the variable v_{req} and control signals to the valves are modeled by the variables v_{in} and v_{out}. All variables have domain $\{0, 1\}$ and zero as initial value.

The plant model consists of the extended automata User, S1 and S2. A sensor signal can go high if the inlet valve is open ($v_{in} = 1$) and it can go low if the outlet valve is open ($v_{out} = 1$). The supervisor's task is to enable or disable

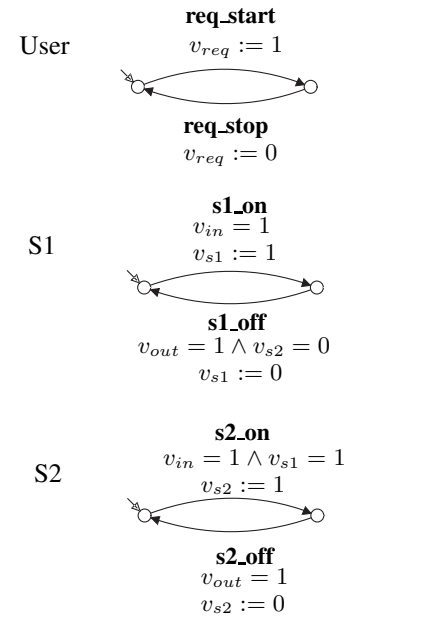

Fig. 1. A plant model of the dosing tank.

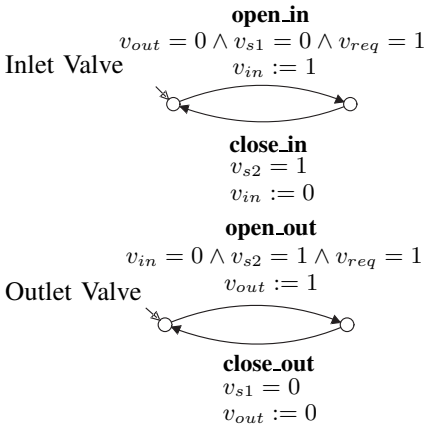

Fig. 2. A supervisor for the dosing tank.

the opening/closing of the valves such that the following conditions are satisfied:

- **a.** the inlet and outlet valves are never open at the same time;
- **b.** discharging or filling can only start when a request is present, $v_{req} = 1$;
- **c.** discharging can only start when the tank is completely filled, $v_{s2} = 1$;
- **d.** filling can only start when the tank is empty, $v_{s1} = 0$;
- **e.** for the inlet valve to close the tank must be full, $v_{s2} = 1$;
- **f.** for the outlet valve to close the tank must be empty, $v_{s1} = 0$.

In figure 2 all these requirements are expressed using guard formulas over the variables. The presented EFA model of the dosing tank may be compared to the FA model in [15], where the same process is modeled without variables. EFA facilitates the modeling of many systems and it is reasonable to expect that the benefits of using EFA for modeling increase when the complexity of the modeled system increases.

III. AUTOMATA

Typically, system behaviors are modeled using deterministic finite-state automata (FA). In what follows the word "ordinary" will be used synonymously with "deterministic finite-state".

A deterministic finite-state automaton A is a 4-tuple $A = \langle Q, \Sigma, \rightarrow, q_0 \rangle$ where Q is a finite set of states; Σ (the alphabet) is a nonempty finite set of events; $\rightarrow \subseteq Q \times \Sigma \times Q$ is the state transition function mapping elements of $Q \times \Sigma$ into singletons of Q and $q_0 \in Q$ is the initial state.

The transition function is written in infix notation $p \xrightarrow{\sigma} q$. In particular, $p \xrightarrow{\sigma}$ denotes that there exists a state q such that $p \xrightarrow{\sigma} q$. This notation is extended to strings in Σ^* in the natural way by letting

$$p \xrightarrow{\epsilon} p \text{ for all } p \in Q;$$
$$p \xrightarrow{s\sigma} q \text{ if } p \xrightarrow{s} r \text{ and } r \xrightarrow{\sigma} q \text{ for some } r \in Q.$$

For convenience, the notation $A \xrightarrow{s} q$ is introduced as a short hand for $q_0 \xrightarrow{s} q$, where q_0 is the initial state of A. The behavior of a FA is described by its language. The language of A denoted $L(A)$ is defined as $L(A) = \{s \in \Sigma^* : A \xrightarrow{s}\}$.

To deal with interacting automata we use full synchronous composition (FSC) [16]. This composition operator models that an event can occur in the synchronized system if and only if it can occur in all automata that share the event. Let $A_j = \langle Q_j, \Sigma_j, \rightarrow_j, q_0^j \rangle$, $j = 1, 2$ be two automata. The full synchronous composition (FSC) of A_1 and A_2 is

$$A_1 \| A_2 = \langle Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \rightarrow, (q_0^1, q_0^2) \rangle,$$

where $(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_1 \cap \Sigma_2$ if $p_i \xrightarrow{\sigma}_i q_i$, $i = 1, 2$; $(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_1 \setminus \Sigma_2$ if $p_1 \xrightarrow{\sigma}_1 q_1$ and $p_2 = q_2$; $(p_1, p_2) \xrightarrow{\sigma} (q_1, q_2)$, $\sigma \in \Sigma_2 \setminus \Sigma_1$ if $p_2 \xrightarrow{\sigma}_2 q_2$ and $p_1 = q_1$.

The composition operator is easily extended to simultaneous composition of multiple automata.

IV. EXTENDED AUTOMATA

An Extended Finite Automaton (EFA) is an augmentation of the ordinary automaton with guard formulas and action functions. We associate the guards and actions to the transitions in the automaton. The transitions in the EFA are enabled if and only if the guard formula is true and when a transition is taken, updating actions of a set of variables may follow. To define guard predicates we use the characteristic function χ_W of a set W. χ_W is defined by

$$\chi_W(v) = \begin{cases} 1 & \text{if } v \in W \\ 0 & \text{if } v \notin W \end{cases},$$

and is sometimes called the indicator function of W. If $\chi_W(v) = 1$ the predicate "$v \in W$" is true, and if $\chi_W(v) = 0$ the predicate is false.

Definition 1 (Extended Automaton):
An extended finite-state automaton E is a 6-tuple

$$E = \langle Q \times V, \Sigma, \mathcal{G}, \mathcal{A}, \rightarrow, (q_0, v_0) \rangle,$$

where:

- (i) $Q \times V$ is the extended finite set of states, where Q is a finite set of *locations* and V is the finite domain of definition of the variables;
- (ii) Σ is a nonempty finite set of events (the alphabet);
- (iii) $\mathcal{G} = \{\chi_W \mid W \in 2^V\}$ is the set of guard predicates over V.
- (iv) $\mathcal{A} = \{a \mid a \text{ is a function from } V \text{ to } V\}$ is a collection of action functions.
- (v) $\rightarrow \subseteq Q \times \Sigma \times \mathcal{G} \times \mathcal{A} \times Q$ is the state transition relation.
- (vi) $(q_0, v_0) \in Q \times V$ is the initial state.

We have extended the states of the ordinary automaton to $Q \times V$, where $V = V^1 \times ... \times V^n$. The finite set V is the domain of definition of an n-tuple of variables $v = (v^1, \ldots, v^n)$ with initial values $v_0 = (v_0^1, \ldots, v_0^n) \in V$. Formally speaking, the inclusion of $\mathcal{G}$ and $\mathcal{A}$ in the definition of E are superfluous since they only depend on V. In what follows, we therefore omit $\mathcal{G}$ and $\mathcal{A}$ when specifying an EFA.

The *guards* are predicates over the variables that relate each element of V, to either 1 (true) or 0 (false). We are interested in deterministic EFA, and therefore the *actions* are functions. Action functions $a \in \mathcal{A}$ maps the variable values of the present state to the variable values of the next state. Guards and actions are written as

$$\begin{aligned} k &= g(v), \text{ where } k \in \{0, 1\}; \\ w &:= a(v) = (a^1(v), \ldots, a^n(v)), \\ &\quad \text{where } w \in V. \end{aligned}$$

For convenience we use the symbol Ξ to denote implicit actions that update variables to their current value. Unlike explicit actions, Ξ can be overridden when EFA are synchronized, see [17]. If $a^i = \Xi$, we say that a^i is a *don't care updating of the variable* v^i.

The transition relation is written as $p \xrightarrow{\sigma}_{g/a} q$, where $p, q \in Q$, $\sigma \in \Sigma$, $g \in \mathcal{G}$ and $a \in \mathcal{A}$. If g is absent, it is assumed that g always evaluates to true. If a is absent, it is assumed that $a = (\Xi, \Xi, \ldots, \Xi)$ and no variable is updated during the transition [1]. Note that, the state transition relation is well defined when the guard always evaluates to false and no transition can take place. Actions and guards can be used to hide variable values in system transitions. However, it is sometimes convenient to write out the states (locations and variable values) explicitly in system transitions.

Definition 2 (Explicit State Transition Relation): Let $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0) \rangle$ be an EFA. The explicit state transition relation of E is defined as

$$\begin{aligned} \mapsto \quad := \quad & \{(p, v, \sigma, q, w) \in Q \times V \times \Sigma \times Q \times V \mid \\ & \exists p \xrightarrow{\sigma}_{g/a} q \text{ such that} \\ & g(v) = 1 \text{ and } a(v) =: w \text{ or} \\ & g(v) = 1, v = w \text{ and } a = \Xi\}. \end{aligned}$$

The explicit state transition relation is written $(p, v) \xmapsto{\sigma} (q, v')$ and it is extended to strings in Σ^* in the usual recursive way. The language $L(E)$ of an EFA E is

[1] We consider event driven transitions. Dynamic transitions that take place when the guard becomes true are not considered here.

$L(E) = \{s \in \Sigma^* \mid (q_0, v_0) \overset{s}{\mapsto}\}$. In section V, we will need the notion of deterministic EFA.

Definition 3 (Deterministic EFA):
An EFA $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ is deterministic if $(p, v) \overset{\sigma}{\mapsto} (q, v')$ and $(p, v) \overset{\sigma}{\mapsto} (q', v'')$ always implies $(q, v') = (q', v'')$.
Note that for an EFA to be deterministic all explicit transitions (not just the reachable) must have this property. This seemingly strong condition is needed to ensure that the synchronized product of two deterministic EFA remains deterministic. Observe that if the guards are distinct, we support the possibility of having multiple transitions from the same location triggered by the same event. Verifying if a given EFA is deterministic can be non-trivial for EFA with large state-spaces. However, sufficient conditions that guarantee deterministic EFA are straightforward to formulate.

A. Full Synchronous Composition, EFA

To simplify the notation when defining the full synchronous product of EFA, we assume that the EFA share all variables. This is no restriction since it is always possible to add don't care variables that are never updated. We will also assume that shared variables between EFA have the same initial values. As for ordinary automata, the composition operator models that an event can occur in the synchronized system if and only if it can occur in all EFA that share the event.

Definition 4 (FSC, EFA):
Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0)\rangle$, $k = 1, 2$, be two EFA using the shared variables $v = (v^1, \ldots, v^n)$. The Full Synchronous Composition (FSC) of E_1 and E_2 is

$$E_1 \| E_2 = \langle Q_1 \times Q_2 \times V, \Sigma_1 \cup \Sigma_2, \rightarrow, (q_0^1, q_0^2, v_0)\rangle,$$

where the state transition relation $\rightarrow$ is defined as

- $(p_1, p_2) \overset{\sigma}{\rightarrow}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \cap \Sigma_2$ if
  $\exists (p_1, \sigma, g_1, a_1, q_1) \in \rightarrow_1, \exists (p_2, \sigma, g_2, a_2, q_2) \in \rightarrow_2$ such that:
  (i) $g = g_1 \wedge g_2$,
  (ii) For $i = 1, \ldots, n$ and $\forall v \in V$:

$$a^i(v) = \begin{cases} a_1^i(v) & \text{if } a_1^i(v) = a_2^i(v) \\ a_1^i(v) & \text{if } a_2^i(v) = \Xi \\ a_2^i(v) & \text{if } a_1^i(v) = \Xi \\ v^i & \text{otherwise} \end{cases} ;$$

- $(p_1, p_2) \overset{\sigma}{\rightarrow}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_1 \setminus \Sigma_2$ if
  $(p_1, \sigma, g, a, q_1) \in \rightarrow_1$ and $p_2 = q_2$;
- $(p_1, p_2) \overset{\sigma}{\rightarrow}_{g/a} (q_1, q_2)$, $\sigma \in \Sigma_2 \setminus \Sigma_1$ if
  $(p_2, \sigma, g, a, q_2) \in \rightarrow_2$ and $p_1 = q_1$.

Note that if the action functions of E_1 and E_2 explicitly try to update a shared variable to different values, the variable is, by default, not updated. Thus, the synchronized EFA may not have the intended behavior. A sufficient condition to avoid this possibility is:

Definition 5 (Action Consistent EFA):
Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0)\rangle$, $k = 1, 2$, be two extended automata with shared variables $v = (v^1, \ldots, v^n)$. E_1 and E_2 are *action consistent* if $\forall (p_1, \sigma, g_1, a_1, q_1) \in \rightarrow_1$ and $\forall (p_2, \sigma, g_2, a_2, q_2) \in \rightarrow_2$ it is true that:

- $\forall v \in V$ such that $g_1(v) \wedge g_2(v)$ is true then $a_1^i(v) = a_2^i(v)$ or one of $a_1^i(v)$ and $a_2^i(v)$ is a don't care updating of v^i, $i = 1 \ldots n$.

Similar to definition 3, the action consistency condition is a global requirement and may be computationally expensive to check.

V. Transforming EFA to FA

Much research has been put into developing efficient algorithms and data structures for solving supervisory control problems formulated with ordinary automata. It has therefore been an important goal of our research to show how a set of EFA may be transformed into another set of equivalent FA.

In this section an algorithm for transforming EFA into equivalent FA is presented. The transformation is inspired by a translation from UML Statecharts to Finite-State Machines discussed in [18]. The algorithm relies on variables with *finite* domain of definition. It collects the information stored in the guards and actions and builds two kinds of automata, *variable automata* and *location automata*, both with relabeled event sets. The variable automata model the updating of the variables, and the location automata has the same structure as the original extended automata. From a practical point of view it is important to point out that the transformation does not destroy the modular structure, this is important because the modular structure may be exploited by efficient verification and synthesis algorithms.

Definition 6: (Isomorphic FA)
Let $E = \langle Q \times V, \Sigma_E, \rightarrow_E, (q_0, v_0)\rangle$ be an EFA and $A = \langle R, \Sigma_A, \rightarrow_A, r_0\rangle$ be a FA. Let $\mapsto_E$ be the explicit state transition relation of E. E and A are isomorphic if the following conditions hold.

(i) Σ_E=Σ_A
(ii) There exists a bijective function f from $Q \times V$ to R, such that $f(q_0, v_0) = r_0$ and
$(q, v) \overset{s}{\mapsto}_E (q', v') \Leftrightarrow f(q, v) \overset{s}{\rightarrow}_A f(q', v')$.

We place no restriction on the naming of the states in automata and therefore we call all ordinary automata that are isomorphic with the extended automaton E, *the isomorphic FA of* E. Note that, the isomorphic FA of a given EFA is obtained by replacing the state transitions relation $\rightarrow$ with the explicit state transition relation $\mapsto$.

A benefit of using EFA as a modeling tool is that the values of the variables in state transitions can be hidden. Usually, the explicit state transition relation $\mapsto$ is not known. Instead the notation $p \overset{\sigma}{\rightarrow}_{g/a} q$ is used to describe system transitions in models. Here, we present an algorithm that transforms EFA models into FA models by extracting the information in the guard and action functions. This is done by building location automata and variable automata, introducing relabeled events, composing the system using the full synchronous composition and in the last step, changing back to the original event names.

Algorithm 1 presents in detail how a single EFA is transformed to its isomorphic FA. It is assumed that the

guards have been parsed and written in disjunctive normal form $g = g^1 \vee \ldots \vee g^j$, where each and-clause

$$g^i(v) = g^{i,1}(v^1) \wedge \ldots \wedge g^{i,n}(v^n),\ i = 1 \ldots j,$$

compares the variables with a constant in $V = V^1 \times \cdots \times V^n$. In Algorithm 1 $|g|$ denotes the number of and-clauses of a guard written in disjunctive normal form and if $a^k(v^k) = \Xi$, it is understood that $v^k \xrightarrow{\sigma_i} a^k(v^k)$ means a self loop at v^k. We also assume that all actions are written as $a(v) = (a^1(v^1), \ldots, a^n(v^n))$. Any guard-action pair can be represented by multiple guard-action pairs where each action function has this form. It can be achieved by stepping through the domain of definition of all variables and creating multiple assignment functions.

Algorithm 1 (Basic Transformation Algorithm): Let $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ be an extended finite automaton where $V = (V^1, \ldots, V^n)$ is the domain of definition for the variables $v = (v^1, \ldots, v^n)$. The following steps build the isomorphic finite ordinary automaton A and define a renaming function $\Psi(\cdot)$:

1 For each transition $p \xrightarrow{\sigma}_{g/a} q$ in E, introduce $|g|$ new events, all with *unique* names. Create a one to one mapping between each renamed event and an and-clause g^i in the guard g.
2 Collect all relabeled events in the alphabet Σ'.
3 Build a location automaton $A_{loc} = \langle Q, \Sigma', \rightarrow, q_0\rangle$ representing the location changes of E. Each transition $p \xrightarrow{\sigma}_{g/a} q$ is divided into regular transitions in A_{loc} using the relabeled events in Σ'. The number of and-clauses $|g|$ of the guard determine the number of transitions obtained from $p \xrightarrow{\sigma}_{g/a} q$.
4 Build variable automata $A_v^k = \langle V^k, \Sigma', \rightarrow, v_0^k\rangle\ k = 1 \ldots n$, representing the updating of the variables. For each event σ_i in Σ', create transitions $v^k \xrightarrow{\sigma_i} a^k(v^k)$ in G_v^k if $g^{i,k}(v^k)$ is true.
5 Implement the guard by synchronizing all ordinary automata

$$A_{loc} \| A_v^1 \| \cdots \| A_v^n.$$

6 Let $\Psi(\cdot)$ be the mapping that maps each relabeled event in Σ' to its original event in Σ. The finite automaton A is obtained by applying $\Psi(\cdot)$ to all events in $A_{loc} \| A_v^1 \| \cdots \| A_v^n$.

Proposition 1: Algorithm 1 transforms an extended automaton $E = \langle Q \times V, \Sigma, \rightarrow, (q_0, v_0)\rangle$ into its isomorphic ordinary automaton A.

Proof: It follows immediately that A has the same alphabet as E. The states of A are $Q \times V^1 \times \cdots \times V^n$ so we have a trivial bijective mapping between the states of A and the states of E. It remains to prove that the transition relation of A equals the explicit transition relation $\mapsto$ of E. According to definition 2, we need to show that the transitions in A obtained from $p \xrightarrow{\sigma}_{g/a} q$ are all transitions $(p, v) \xmapsto{\sigma} (q, a(v))$, such that $g(v)$ is true. Let $V_i = (V_i^1, \ldots, V_i^n) \in V$ be the set where the and-clause g^i evaluates to true. The set of all v such that $g(v)$ is true can then be written as $\{V_1, \ldots, V_j\}$, where $j = |g|$. For each transition $p \xrightarrow{\sigma}_{g/a} q$ in E, Algorithm 1 can be described as follows.
For $i = 1, \ldots, |g|$:

(i) Create $\alpha \in \Sigma'$ and $p \xrightarrow{\alpha} q$ in A_{loc}, where α is unique.
(ii) For $k = 1, \ldots n$, create $v^k \xrightarrow{\alpha} a^k(v^k)$ for all $v^k \in V_i^k$ in A_v^k.
(iii) Synchronizing all transitions in the ordinary automata triggered by α gives: $(p, v) \xrightarrow{\alpha} (q, a(v))$, $\forall v \in V_i$ in $A_{loc} \| A_v^1 \| \cdots \| A_v^n$.
(iv) Replacing all transitions in the synchronized system triggered by α with transitions triggered by σ gives: $(p, v) \xrightarrow{\sigma} (q, a(v))\ \forall v \in V_i$ in A.

Hence, $p \xrightarrow{\sigma}_{g/a} q$ in E is mapped to $(p, v) \xmapsto{\sigma} (q, a(v))$ in A, if $g(v)$ is true. ■

Algorithm 1 transforms an EFA into a single monolithic ordinary automaton. We are interested in synthesis and verification of DES and we want to avoid the state-space explosion problem when all components are synchronized. *Therefore we only implement the first four steps of the algorithm.* Before the monolithic automaton is computed in step five, we have a model consisting of a number of ordinary automata $A_{loc}, A_v^1, \ldots, A_v^n$, whose alphabet Σ', consists of relabeled events from the original alphabet Σ. We can relate $L(E)$ to $L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)$ by extending the mapping $\Psi(\cdot)$ to strings; $\Psi(\epsilon) = \epsilon$, and $\Psi(s\sigma) = t$ if $\Psi(s) = r$ and $r\Psi(\sigma) = t$ for some $r \in \Sigma^*$. Since $\Psi(\cdot)$ projects relabeled events to their original events, it follows that $\Psi(L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)) = L(E)$.

Let $\sim$ be the equivalence relation on strings s, t in $L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)$, where $s \sim t$ if $\Psi(s) = \Psi(t)$. If the extended automaton E is deterministic then so is the isomorphic ordinary automaton A. This implies that if $s \sim t$, then s and t visit the same states in $A_{loc} \| A_v^1 \| \cdots \| A_v^n$. Hence, the language that describes the behavior (or state-changes) of the ordinary automata model is $L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)/\sim$. Most importantly, since $\Psi(\cdot)$ is a bijective mapping between $L(E)$ and $L(A_{loc} \| A_v^1 \| \cdots \| A_v^n)/\sim$ we can use the relabelled ordinary automata model obtained in step 4 of Algorithm 1 to analyze EFA models.

Typically, models are built using a number of interacting EFA. EFA that share variables and interact via the FSC can through their guard-action pairs exchange information during the synchronization process. In order to build ordinary automata that represent the synchronized behavior of multiple EFA, access to all guards and updating actions is needed for each transition in the synchronized system. *The transformation must consider all components simultaneously.* This fact implies that Algorithm 1 must be applied to all combinations of transitions in the synchronized system. Let $E_k = \langle Q_k \times V, \Sigma_k, \rightarrow_k, (q_0^k, v_0)\rangle$, $k = 1, \ldots, m$ be interacting EFA whose shared variables $v = (v^1, \ldots, v^n)$ have domain $V = (V^1, \ldots, V^n)$. The overall behavior of

the system is described by the composition of all components using the FSC, $E = E_1 \| \cdots \| E_m$.

To obtain an isomorphic ordinary automaton A of the EFA E, we apply Algorithm 1 to all transitions $p \xrightarrow{\sigma}_{g/a} q$ of E, where $p = (p_1, \ldots, p_m)$ and $q = (q_1, \ldots, q_m)$ are locations in $E_1 \| \cdots \| E_m$. The difference is that the number of location automata in step 3 increase to m, i.e. instead of only one location automaton we have $A^k_{loc} = \langle Q^k, \Sigma', \rightarrow, q^k_0 \rangle$, $k = 1, \ldots, m$. By building the location automata and variable automata transition by transition the modular ordinary automata model (in step 4 of Algorithm 1) can be implemented using an algorithm that only consumes a polynomial amount of space. A FA model of the dosing tank, without relabeled events, is given figure 3.

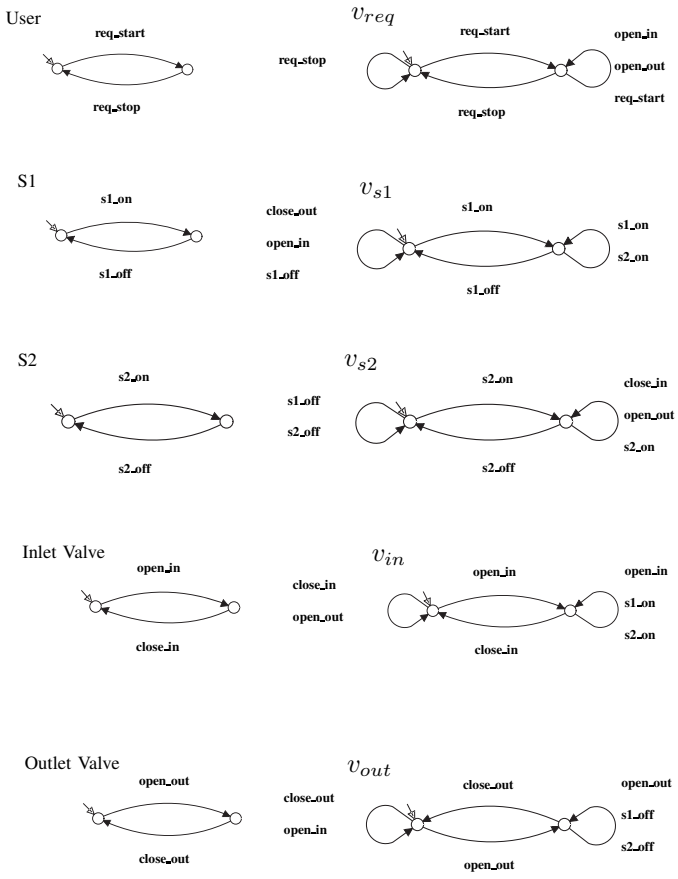

Fig. 3. A model of the dosing tank example consisting of ordinary automata. It has been generated using the first four steps of Algorithm 1.

VI. CONCLUSIONS

The proposed modeling framework of extended finite automata (EFA) can be used to design supervisors for complex systems where ordinary finite automata (FA) modeling requires complex and possibly non-intuitive solutions. Due to the use of variables, guard expressions and action functions, the EFA formalism can hide information and represent systems more compact than FA. However, developing effective synthesis algorithms for automata that share variables is not an easy task. To overcome this difficulty we have provided an algorithm that transforms systems modeled by automata with shared variables into equivalent ordinary automata models. By examining the information exchange between all components of the model, we avoid building the product of the extended model and instead, we obtain an equivalent modular FA model. The algorithm is feasible for any finite modular system whose EFA share variables with finite domain. The modeling framework and supporting algorithms have been implemented in the supervisory control tool Supremica. Since Supremica also implements state-of-the-art algorithms and data structures for dealing with large scale problems we hope that this work will facilitate the adaptation of the supervisory control ideas into industrial applications.

REFERENCES

[1] P. J. Ramadge and W. M. Wonham, "The control of discrete event systems," *IEEE*, vol. 77, no. 1, pp. 81–98, Jan. 1989.

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2nd ed., ser. Series in Computer Science. Addison-Wesley, 2003.

[3] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin, "Supervisory control of a rapid thermal multiprocessor," *IEEE*, vol. 38, no. 7, pp. 1040–1059, 1993.

[4] M. Fabian and A. Hellgren, "PLC-based implementation of supervisory control for discrete event systems," in *37th Decision and Control*, Tampa, FL, USA, 1998.

[5] X.-R. Cao, G. Cohen, A. Giua, W. M. Wonham, and J. H. van Schuppen, "Unity in diversity, diversity in unity: Retrospective and prospective views on control of discrete event systems," *Discrete Event Dynamic Systems*, vol. 12, pp. 253–264, 2002.

[6] D. Harel, "Statecharts: A visual formalism for complex systems," *Science of Computer Programming*, vol. 8, pp. 231–274, 1987.

[7] Y.-L. Chen and F. Lin, "Modeling of discrete event systems using finite state machines with parameters," in *CCA00*, Anchorage, Alaska, Sept. 2000.

[8] Y. Yang and P. Gohari, "Embedded supervisory control of discrete-event systems," in *2005*, Edmonton, Canada, August 2005, pp. 410–415.

[9] B. Gaudin and P. H. Deussen, "Supervisory control on concurrent discrete event systems with variables (extended version)," Technical University, Berlin, Tech. Rep., 2006. [Online]. Available: http://www.benoit.gaudin1.free.fr

[10] C. de Oliveira, J. Cury, and C. Kaestner, "Synthesis of supervisors for parameterized and infinity non-regular discrete event systems," in *Proceedings of the 1st IFAC Workshop on Dependable Control of Discrete Systems (DCDS'07)*, June 2007, pp. 77–82.

[11] C. Ma and W. Wonham, "Nonblocking supervisory control of state tree structures," *IEEE*, vol. 51, no. 5, pp. 782–793, May 2006.

[12] Supremica, "`www.supremica.org`. The official website for the Supremica project," 2007.

[13] K. Åkesson, M. Fabian, H. Flordal, and R. Malik, "Supremica—an integrated environment for verification, synthesis and simulation of discrete event systems," in *8th Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 384–385.

[14] K. Åkesson and M. Sköldstam, "Towards a framework for integrated supervisory and logic control," in *1st Dependable Control of Discrete Event Systems'07*, Paris, France, 2007, pp. 83–88.

[15] P. Malik and R. Malik, "Modular control-loop detection," in *8th Discrete Event Systems, WODES '06*, Ann Arbor, MI, USA, July 2006, pp. 119–124.

[16] C. A. R. Hoare, *Communicating sequential processes*, ser. Series in Computer Science. Prentice-Hall, 1985.

[17] J. M. Spivey, *The Z Notation: A Reference Manual*, 2nd ed. Prentice-Hall, 1992.

[18] R. Malik and R. Mühlfeld, "A case study in verification of uml statecharts: the profisafe protocol," *Universal Computer Science*, vol. 9, no. 2, pp. 138–151, Feb. 2003.