A State-Based Framework for Supervisory Control Synthesis and Verification

J. Markovski, D.A. van Beek, R.J.M. Theunissen, K.G.M. Jacobs, and J.E. Rooda

Abstract— **We extend an existing model-based framework for supervisory control synthesis with generalized control and verification state-based requirements. The former stem from the need for intuitive specification of the control requirements, whereas the latter are employed for liveness verification in order to ensure that the supervisor does not disable desired functionalities of the plant. First, we introduce generalized control requirements and show them provably equivalent to the standard state-based control requirements. In the process, we identify a class of state-based liveness requirements, which can be efficiently verified and employed in the supervisor synthesis framework to provide early feedback to the modeler.**

I. Introduction

In the last few decades, control software development has taken a more central role due to the ever increasing complexity of the machines, demands for higher quality and performance, and improved safety and ease of use. Traditionally, the requirements for the control software are formulated informally in some sort of specification documents. Thereafter, software engineers translate them into control software manually. This is a time-consuming and an error prone process, since control requirements are often ambiguous. Thus, the produced software needs to be validated against the machine. If validation fails, the code must be rewritten leading to a define-validate-redefine loop. This issue in control software design gave rise to supervisory control theory [1]–[3], where high-level supervisory controllers are synthesized automatically based upon formal models of the hardware and control requirements.

The supervisory controller observes the discrete-event behavior of the machine by receiving signals from ongoing activities, upon which it makes a decision which activities the plant is allowed to carry out and sends back control signals. This is known as a feedback loop. Under the assumption that the supervisory controller can react sufficiently fast on every input from the machine, one can model the feedback loop as a pair of synchronizing processes, where the model of the machine (referred to as a *plant*) is restricted by the model of the controller (referred to as a *supervisor*). Originally, the plant is modeled as a set of observable traces of events. The model is typically given as a set of parallel automata, whose joint recognized language corresponds to the observed traces. The events are split into controllable events, which can be disabled by the supervisor in the synchronous composition, and uncontrollable events, which must always be allowed. The *control requirements* specify allowed behavior as sequences of events, again modeled by automata, leading to event-based supervisory control [1], [2].

To alleviate spatial and computational demand, [3] proposes State Tree Structures (STSs) as underlying plant model and advocates state-based expressions as control requirements. STSs are a superset of automata and provide for hierarchical modeling of concurrent processes, while supporting efficient storage as binary decision diagrams (BDDs) [4]. The hierarchy is given in a form of a tree, i.e., one distinguishes between parent and child states. The role of automata is taken by so-called OR-superstates, in which a single simple state is active, representing a standard automaton state, or a child superstate in a lower level. The parallelism is described by AND-superstates, which denote that multiple child states are active at the same time: one state per each child. State-based control requirements [5] restrict the plant by forbidding combinations of states of the parallel components, referred to as *mutual state exclusion* requirements. The control state-based requirements also determine which events must be disabled for a given combination of states, referred to as *state-transition exclusion* requirements.

We use an existing model-based engineering framework for supervisory control synthesis [6], [7], see Fig. 2 below, to structure the process of obtaining a supervisory controller. We discuss the framework in more detail in section II. The framework and the supporting toolchain rely on the Compositional Interchange Format (CIF), see [8] and references therein. The underlying model of CIF are the interchange automata, which are supported by a vast range of conversion, simulation, verification, and visualization tools. They can model hybrid (timed) behavior, which makes them suitable for modeling the plant for simulation purposes as well. We do not give the complete syntax of interchange automata, since we focus primarily on specifying the control requirements. Thus, our visualization of the plant model should be treated as an abstraction to discrete-event behavior only. For a detailed account of supervisory control synthesis using CIF we refer the reader, e.g., to [9].

Next, we give a small example of modeling an everyday activity of starting and driving a car that illustrates the issues addressed in this paper. We consider only two aspects: the car movement, i.e., it can be standing still, moving, or transiting between these two modes, and the engine, which can be switched on or off. We assume that the driver controls the engine, acceleration, and breaking. We model the correct behavior of the driver by means of supervisory control. For instance, the engine should be off only while the car is

Research funded by European project ICT-2007.3.7.c – Control for Coordination of Distributed Systems.

J. Markovski, K.G.M. Jacobs, D.A. van Beek, R.J.M. Theunissen, and J.E. Rooda are with the Systems Engineering Group, Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands, jacobskoen@gmail.com, {J.Markovski, D.A.v.Beek, R.J.M.Theunissen, J.E.Rooda}@tue.nl.

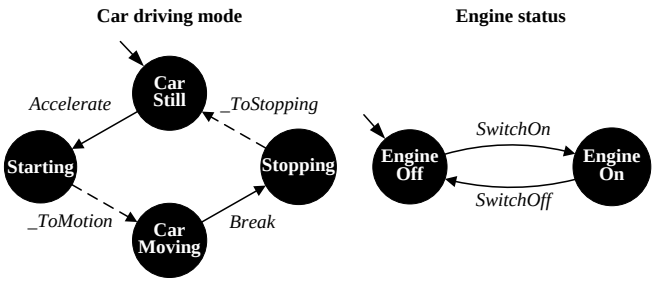

Fig. 1. Model of the car driving mode and its engine

standing still, or the car cannot start moving without the engine being switched on.

The plant model, i.e., the car driving mode and engine status, is depicted in Fig. 1. The state and transition names should speak for themselves. By full lines we denote controllable events and by dashed lines we denote uncontrollable events. Next, we attempt to model the first requirement, which states that the engine should be off only while the car is standing still. The state **CarStill** denotes that the car is standing still, whereas the state **EngineOff** denotes that the engine is switched off. Thus, to model this control requirement, we have the following intended implication:

(Plant in **EngineOff**) *implies* (Plant in **CarStill**).

The above expression states what has to be fulfilled, in contrast to mutual state exclusion, which states what is illegal. The latter form, conformable with [3], [5], is given by:

not (Plant in **EngineOff** *and* Plant in **Starting**) *and*
not (Plant in **EngineOff** *and* Plant in **CarMoving**) *and*
not (Plant in **EngineOff** *and* Plant in **Stopping**),

producing an equivalent, yet unintelligible expression.

The second control requirement states that the car cannot start moving without the engine being switched on. The event *Accelerate* denotes that the car begins to move, whereas being in the state **EngineOn** denotes that the engine is switched on. This can be straightforwardly modeled as:

(Enabling event *Accelerate*) *implies* (Plant in **EngineOn**).

Written as state-transition exclusion, we have the following expression stating when the event must be disabled:

(Plant in **EngineOff**) *implies* (Disabling event *Accelerate*).

The above 'formalization' of the control requirements follows naturally from the informal specification documents. However, as illustrated, it does not suit the form required by the tooling [3], [5]. Thus, our first contribution is identification of a set of generalized state-based control requirements typically used in specification documents, supporting complete propositional logic. Also, we build translation algorithms and a tool that we incorporate in the existing toolchain [7]. Moreover, we identified a new set of requirements that have a valid interpretation, yet not as control requirements, e.g,

(Plant in **EngineOn**) *implies* (Enabling event *Accelerate*).

The expression specifies that the car is able to accelerate when the engine is on. As a control requirement the expression is obsolete as every event that is not explicitly disabled, is allowed by the supervisor. However, we can view this requirement as a liveness property to be verified, because we want to ensure that the supervised plant behaves as intended, i.e., the car can be driven. Note that such properties are not guaranteed by the supervisory control theory. The theory only ensures fulfillment of the restricting control (safety) requirements, i.e., that nothing goes wrong, like the engine goes off while driving. The liveness properties are actually part of the validation process, where we verify that the supervision does not restrict some intended functionality. Our second contribution is, thus, a proposal of a framework based on the generalized control requirements, which aims at combining supervisory control synthesis and verification.

Further on, section II discusses the supervisory synthesis framework. Section III introduces the notion of generalized state-based control requirements. In section IV we combine supervisory synthesis and verification based on unified state-based expressions. Section V concludes the paper.

II. Model-based Engineering Framework

To structure the process of supervisory control synthesis we employ the framework depicted in Fig. 2. The modeling process begins with an informal specification of the controlled system, i.e., the desired product, written by domain engineers. A design of the controlled system follows, contrived by the domain and software engineers together. The design most importantly defines the modeling level of abstraction and the control architecture. Subsequently, it is used to separate the plant and the control requirements, a joint task of the domain and software engineers. Here, a decision is made to which extent the control is managed by the software and which part is implemented in hardware. The resulting informal documents specify the plant and control requirements, respectively. In the following, we omit the roles of the engineers as they are clear from the context.

Most plants typically contain (continuous) hybrid behavior, whereas supervisory synthesis requires a discrete-event abstraction. The hybrid model is suitable for simulation purposes, and it can be abstracted to a discrete-event model. Vice versa, one can first build a discrete-event model and, subsequently, refine it. In the design of the plant, decisions are made on the level of abstraction that is used and what is significant discrete-event and hybrid behavior. In parallel, a model of the control requirements is made following the specification documents. Section III introduces generalized control requirements that provide for easier modeling in this step, using pre-processing tools to translate formal, yet intuitive, control requirements into the form required by the supervisor synthesis tool. The discrete-event model of the plant together with the model of the control requirements are input to this tool, which automatically generates a model of the maximally-permissive supervisor. The maximal permissiveness is in the sense that the supervisor restricts the behavior of the plant as little as possible.

The succeeding steps are supposed to validate that the supervised plant behaves as initially specified. First, it is ver-

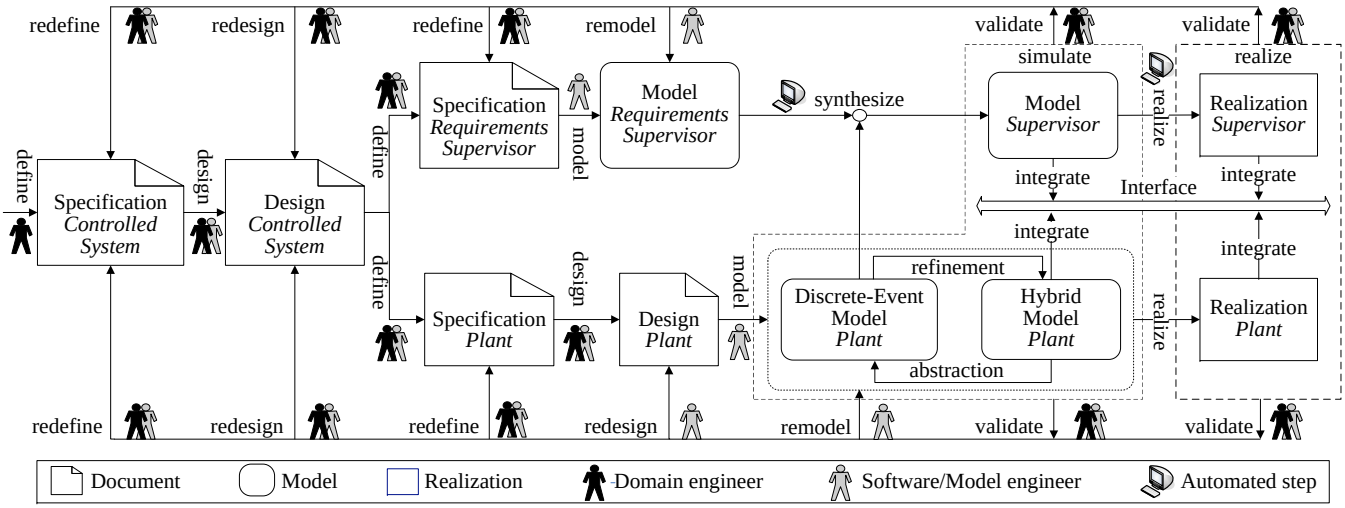

Fig. 2. Model-based engineering framework for supervisory control synthesis

ified that the discrete-event supervised plant has all desired functionalities, i.e., that the supervisor does not restrict the behavior of the plant too much. This step involves reachability analysis and/or model-checking for desired properties. In section IV we show how to combine the specification of the control requirements with the verification requirements in order to optimize the reachability analysis. Software-in-the-loop simulation is used to validate the supervisor coupled with a hybrid model of the plant, and hardware-in-the-loop simulation can be used to validate the supervisor against a prototype of the plant. If the validation is not satisfactory the control requirements and/or the plant model need to be remodeled or redefined. In certain cases, a complete revision proves to be necessary, which might even require redefining the specification of the whole controlled system. Finally, the control software is generated automatically, based on the validated models. Note that software engineers in the framework act more as 'model' engineers, shifting their focus from writing code to modeling.

In the following section we introduce generalized state-based requirements that increase modeling convenience.

III. Generalized Control Requirements

In this section we formalize the informal technique used to derive the control requirements in the introductory example, depicted in Fig. 1. We define the notion of generalized state-based control requirements and show that they are provably equivalent to the standard control requirements of [3], [5]. We assume that the plant is modeled by an indexed set of finite-state automata $\{\mathcal{A}_i \mid i \in N\}$, where the state and event set of automaton $\mathcal{A}_i$ is given by $\mathcal{S}_i$ and $\mathcal{E}_i$, respective. This can be seen as flattened STSs with an initial AND-superstate having OR-superstates as children [3]. The simplification stems from current lack of hierarchy in interchange automata that is the underlying model of the framework. (We note that ongoing research in the framework of the MULTIFORM FP7 European project aims to introduce AND/OR superstate hierarchy to interchange automata as well. An extension to support the complete hierarchy in that case is straightforward by following the steps outlined in this section.)

States have unique names, i.e., $\mathcal{S}_i \cap \mathcal{S}_j = \emptyset$ for $i \neq j$. We define $\mathcal{S} \triangleq \bigcup_{i \in N} \mathcal{S}_i$ and $\mathcal{E} \triangleq \bigcup_{i \in N} \mathcal{E}_i$ as the sets of all states and events of the plant, respectively. We use $\top$ for true, $\bot$ for false, and the following logical operators ordered by binding strength: negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\Rightarrow$, and equivalence $\Leftrightarrow$. Conjunction and disjunction over an indexed set of predicates $\{p_i \mid i \in I\}$ is given by $\bigwedge_{i \in I} p_i$ and $\bigvee_{i \in I} p_i$, respectively.

The *state predicate* $\mathbf{s}\!\downarrow$ expresses that the supervised plant is in state $\mathbf{s} \in \mathcal{S}$ and the *event predicate* $\rightarrow\!e$ expresses that the event $e \in \mathcal{E}$ is enabled by the supervisor. By $\not\rightarrow\!e \triangleq \neg\rightarrow\!e$ we denote that event e is disabled by the supervisor. That the plant occupies the states of $\mathbf{S} \subseteq \mathcal{S}$ is denoted by $\mathbf{S}\!\downarrow \triangleq \bigwedge_{\mathbf{s} \in \mathbf{S}} \mathbf{s}\!\downarrow$. That an event from the set $E \subseteq \mathcal{E}$ is enabled is given by $\rightarrow\!E \triangleq \bigvee_{e \in E} \rightarrow\!e$ and that all events in E are disabled by $\not\rightarrow\!E \triangleq \neg\rightarrow\!E$. The plant occupies exactly one state of each component, expressed by $\bigvee_{\mathbf{s} \in \mathcal{S}_i} \mathbf{s}\!\downarrow \Leftrightarrow \top$ and $\mathbf{s}\!\downarrow \wedge \mathbf{t}\!\downarrow \Leftrightarrow \bot$ if $\mathbf{s} \neq \mathbf{t}$ with $\mathbf{s}, \mathbf{t} \in \mathcal{S}_i$ for every $i \in N$.

In the given setting, the state-based control requirements of [3], [5] can be defined as follows.

Def. 1: A mutual state exclusion requirement is given by $MS ::= \top \mid \bot \mid \neg\mathbf{S}\!\downarrow$ for some $\mathbf{S} \subseteq \mathcal{S}$. A state-transition exclusion requirement is given by $ST ::= \neg MS \Rightarrow \not\rightarrow\!e$ for some $\mathbf{S} \subseteq \mathcal{S}$ and $e \in \mathcal{E}$. The standard state-based control requirements of [3] are given by $SB ::= MS \mid ST \mid SB \wedge SB$.

By $x \in X$ we denote that x is an expression of type X. The form of the mutual state and state-transition exclusion formulas are induced by the internal organization of the state tree structure, for more details see [3], [5]. The state-based control requirements impose that all requirements must hold together as stated by their conjunction in Def. 1. We note that the original definition of state-transition exclusion disables only uncontrollable events. To include disablement of controllable events is not difficult (one only has to take care that the given controllable events are disabled in the

reverse transition function Γ and the control functions f of [3]). Moreover, this type of control requirement is already supported by the synthesis tool [5].

Next, we generalize the requirements given by Def. 1.

Def. 2: The generalized mutual state exclusion requirement is given by $gMS ::= \top \mid \mathbf{s}\downarrow \mid \neg gMS \mid gMS \text{ op } gMS$, for $\mathbf{s} \in \mathcal{S}$ and $\text{op} \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$. The generalized state-transition exclusion requirement is given by $gST ::= gMS \Rightarrow \not\rightarrow E \mid \rightarrow E \Rightarrow gMS$ with $E \subseteq \mathcal{E}$. Now, generalized state-based requirements are given by $gSB ::= gMS \mid gST \mid gSB \wedge gSB$.

The gMS requirements actually support the entire propositional logic, as they support conjunction and negation. We picked the given operators as the most common ones in the informal specification documents. The gST requirements come in two forms. The former is a straightforward generalization of the ST requirements of Def. 1, whereas the latter is most commonly used in the specification documents (see the example above). The two forms are equivalent, which follows immediately from $\rightarrow E \Rightarrow g \Leftrightarrow \neg g \Rightarrow \not\rightarrow E$, where $g \in gMS$ implies that $\neg g \in gMS$ as well.

The following theorem states that for every generalized state-based requirement, there exists an equivalent standard state-based requirement. This makes the former a valid choice for supervisory control synthesis in the vein of [3].

Thm. 1: For every $g \in gSB$ (Def. 2) there exists $f \in SB$ (Def. 1) such that $g \Leftrightarrow f$.

Proof: We prove the statement in two steps, by showing that every gMS and gST requirement has an equivalent SB requirement, leading to the desired equivalence. Let $g \in gMS$. We first show that there exists $f \in SB$ such that $g \Leftrightarrow f$. By $\times_{i \in I} S_i$ we denote the Cartesian product of the sets S_i for $i \in I$. If $T \in \times_{i \in I} S_i$, then by $t \in T$ we denote that t is an element of the tuple T. Given a state $\mathbf{s} \in \mathcal{S}$, let $[\mathbf{s}] = \mathcal{S}_i$ for $i \in N$ such that $\mathbf{s} \in \mathcal{S}_i$. Then, we can replace $\neg\mathbf{s}\downarrow$ with the equivalent expression $\bigvee_{\mathbf{t} \in [\mathbf{s}] \setminus \{\mathbf{s}\}} \mathbf{t}\downarrow$ for every $\mathbf{s} \in \mathcal{S}$. Suppose that the disjunctive normal form [10] of $\neg g$ is $\bigvee_{i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigwedge_{k \in K_i} \neg \mathbf{t}_{ik}\downarrow \right)$, where $\mathbf{s}_{ij}, \mathbf{t}_{ik} \in \mathcal{S}$ for $i \in I$, $j \in J_i$, and $k \in K_i$. We have the following derivation:

$$
\begin{aligned}
&\bigvee_{i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigwedge_{k \in K_i} \neg \mathbf{t}_{ik}\downarrow \right) \Leftrightarrow \\
&\bigvee_{i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigwedge_{k \in K_i} \left(\bigvee_{\mathbf{t} \in [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}} \mathbf{t}\downarrow \right) \right) \Leftrightarrow \\
&\bigvee_{i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigvee_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}} \left(\bigwedge_{\mathbf{t} \in \mathbf{T}} \mathbf{t}\downarrow \right) \right) \Leftrightarrow \\
&\bigvee_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigwedge_{\mathbf{t} \in \mathbf{T}} \mathbf{t}\downarrow \right).
\end{aligned}
$$

Now, using that $g \Leftrightarrow \neg(\neg g)$ and De Morgan's law we have:

$$
g \Leftrightarrow \bigwedge_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \neg \mathbf{S}_{i\mathbf{T}}\downarrow = f,
$$

where $\mathbf{S}_{i\mathbf{T}} = \{\mathbf{s}_{ij} \mid j \in J_i\} \cup \{\mathbf{t} \mid \mathbf{t} \in \mathbf{T}\}$. It is straightforward that $f \in SB$ as a conjunction of MS requirements.

Now, suppose that that $g \in gST$ and suppose that $g = g' \Rightarrow \not\rightarrow E$ for some $g' \in gMS$ and $E \subseteq \mathcal{E}$. Suppose that the disjunctive normal form of g' is given by $\bigvee_{i \in I} \left(\bigwedge_{j \in J_i} \mathbf{s}_{ij}\downarrow \wedge \bigwedge_{k \in K_i} \neg \mathbf{t}_{ik}\downarrow \right)$, where $\mathbf{s}_{ij}, \mathbf{t}_{ik} \in \mathcal{S}$ for $i \in I$, $j \in J_i$, and $k \in K_i$. Following the steps from above, we can derive that $g' \Leftrightarrow \bigvee_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \mathbf{S}_{i\mathbf{T}}\downarrow$ with $\mathbf{S}_{i\mathbf{T}} = \{\mathbf{s}_{ij} \mid j \in J_i\} \cup \{\mathbf{t} \mid \mathbf{t} \in \mathbf{T}\}$. Using that $p \Rightarrow (q \wedge r) \Leftrightarrow (p \Rightarrow q) \wedge (p \Rightarrow r)$ and $(p \vee q) \Rightarrow r \Leftrightarrow (p \Rightarrow r) \wedge (q \Rightarrow r)$ for all predicates p, q, and r, we derive:

$$
\begin{aligned}
g = \; & g' \Rightarrow \not\rightarrow E \Leftrightarrow \\
& \left(\bigvee_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \mathbf{S}_{i\mathbf{T}}\downarrow \right) \Rightarrow \not\rightarrow E \Leftrightarrow \\
& \bigwedge_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \left(\mathbf{S}_{i\mathbf{T}}\downarrow \Rightarrow \not\rightarrow E \right) \Leftrightarrow \\
& \bigwedge_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I} \left(\mathbf{S}_{i\mathbf{T}}\downarrow \Rightarrow \bigwedge_{e \in E} \not\rightarrow e \right) \Leftrightarrow \\
& \bigwedge_{\mathbf{T} \in \times_{k \in K_i} [\mathbf{t}_{ik}] \setminus \{\mathbf{t}_{ik}\}, i \in I, e \in E} \left(\mathbf{S}_{i\mathbf{T}}\downarrow \Rightarrow \not\rightarrow e \right) \quad = f.
\end{aligned}
$$

Here, $f \in SB$ as a conjunction of ST requirements.

Finally, suppose that $g \in gSB$ is given by the conjunction $g = \bigwedge_{i \in I} g_i$ for some index set I where $g_i \in gMS \cup gST$. From the above we have that $g_i \Leftrightarrow f_i$ where $f_i \in SB$ for every $i \in I$. Then, the requirement $f \in SB$ given by $f = \bigwedge_{i \in I} f_i$ is equivalent to g, which completes the proof. ■

The proof of Thm. 1 is constructive and it induces a translation algorithm from generalized control requirements to the form conforming to [3], [5]. We incorporated the translation in the toolchain that supports the framework in Fig. 2 as a preprocessing step of making the model of the control requirements. We note that the algorithm optimizes the state-based expressions employing (1) $\mathbf{S}\downarrow \Leftrightarrow \bot$ if there exist $\mathbf{s}, \mathbf{t}$ and $i \in N$ such that $\mathbf{s}, \mathbf{t} \in \mathbf{S} \cap \mathcal{S}_i$ and $\mathbf{s} \neq \mathbf{t}$ and (2) $\bigvee_{\mathbf{s} \in \mathcal{S}_i} \mathbf{s}\downarrow \Leftrightarrow \top$ for all $i \in N$.

The existing implementation uses a version of Espresso to obtain the standard forms of the requirements [11]. We are also looking into the possibility of using multi-valued logic, by assigning logical values to each state predicate of the same automaton [12]. If the informal specification documents contain sufficient amount of structure, one can also use artificial intelligence methods to automatically extract the generalized control requirements by parsing and interpreting the documents [13]. Next, we extend the supervisor synthesis framework to cater for verification as well.

IV. Combining Synthesis and Verification

In this section we propose a unified framework for supervisory synthesis and verification. It is inspired by the need to verify that the supervised plant contains some desired behavior. It is based on state-based expressions stemming from the ones introduced in section III. We begin with specifying the plant and the control requirements in terms of generalized state-based requirements.

The state-based representation of the plant $P \subseteq gST$ is given by $P = \{P_e \Rightarrow \rightarrow e \mid e \in \mathcal{E}\}$, where P_e identifies *all* states in which the event $e \in \mathcal{E}$ is enabled. The state-transition exclusion requirements $C \subseteq gST$ are given by $C = \{C_e \Rightarrow \not\rightarrow e \mid e \in \mathcal{E}\}$. If $C_e \Leftrightarrow \bot$, then the event e should not be disabled. The mutual state exclusion requirements $F \in gMS$ denote forbidden states in the supervised plant.

The behavior of the supervised plant is defined as follows. By $S \subseteq gST$, where $S = \{S_e \Rightarrow \not\rightarrow e \mid e \in \mathcal{E}\}$ we define the supervisory state control feedback, i.e., S_e denotes all states in which the event e is disabled. The states of

the supervised plant are identified as all states that are *reachable* from the initial states of the plant, identified by $R \in gMS$, and *co-reachable* from the marked (final) states, denoted by $CR \in gMS$, taking into account the events disabled by the supervisor [3]. The co-reachable states that are reachable from the set of initial states are given by $RCR \in gMS$ with $RCR \triangleq CR \wedge R$. Now, the state-based representation of the supervised plant $SP \subseteq gST$ is given by $SP = \{P_e \wedge \neg S_e \wedge RCR \Rightarrow \rightarrow e \mid e \in \mathcal{E}\}$. It is known that the co-reachable states together with the supervisory feedback are sufficient to supervise the plant [3].

We have that $CR \Rightarrow \neg F$ as the co-reachable states must not be forbidden. The control requirements and the supervisory control feedback are related by $C_e \wedge CR \Rightarrow S_e \wedge CR$ for every $e \in \mathcal{E}$, i.e., the supervisor may disable more events than specified by the control requirements due to controllability issues [3]. Note that only the co-reachable set of states is affected as beyond them the feedback of the supervisor is irrelevant. One can exploit this, e.g., to optimize the storage for the supervisor. Recall that the supervisor must not disable uncontrollable events in the co-reachable set of states [3]. This is expressed as $S_u \wedge CR \Leftrightarrow \bot$, where the event u is uncontrollable. The states in which uncontrollable events are disabled are eliminated in the supervisor synthesis implying that $C_u \Rightarrow \neg CR$. We note that $C_u \wedge CR \Rightarrow S_u \wedge CR$ still holds as $C_u \wedge CR \Rightarrow \neg CR \wedge CR \Leftrightarrow \bot \Leftrightarrow S_u \wedge CR$. The predicates S and CR are computed efficiently, whereas R and, thus RCR, are not computed at all, since they are not required for the plant supervision [3], [5].

The above state-based predicates define the framework for supervisory synthesis. We noticed, however, that we are able to specify and interpret additional state-based expressions. An interesting class is $LR ::= gMS \Rightarrow \rightarrow e \mid \not\rightarrow e \Rightarrow gMS$ for $e \in \mathcal{E}$. The two possible variants of LR (refers to *Liveness Requirement*) are equivalent, as $g \Rightarrow \rightarrow e \Leftrightarrow \not\rightarrow e \Rightarrow \neg g$ for all $g \in gMS$, implying $\neg g \in gMS$ as well. As hinted in the introduction, we interpret the LR expressions as liveness requirements that specify that some desired behavior should present in the supervised plant. Another interpretation is with respect to reachability of states in the supervised plant. By $V \subseteq LR$, where $V = \{V_e \Rightarrow \rightarrow e \mid e \in \mathcal{E}\}$ we denote liveness behavior that should be present in the supervised plant. More precisely, the event e must be enabled for all reachable states of the supervised plant that are also identified by V_e. This is expressed as $V_e \wedge RCR \Rightarrow P_e \wedge \neg S_e \wedge RCR$ for all $e \in \mathcal{E}$. By $V_e \Leftrightarrow \bot$ we denote that no verification of the corresponding type is performed for the event e. Additionally, we define a set of states that identify desired (marked) behavior, given by the indexed set $D = \{D_i \in gMS \mid i \in I\}$. We require that there exists at least one state for each *desired state requirement* D_i in the supervised plant, i.e., $D_i \wedge RCR \not\Leftrightarrow \bot$ for all $i \in I$.

Now, based on the above, we propose a framework for supervisory control synthesis and verification with unified control/verification specifications given by (P, C, F, V, D). The verification of the liveness requirements follows the steps of the synthesis framework of Fig. 2:

1) On specification level, we can check if the verification requirements are conflicting with the specification of the plant and the control requirements.
2) On supervisor synthesis level, we can check if the verification requirements are conflicting with the supervisory constraints and the co-reachable states.
3) On supervised plant level, we have to verify that the requirements are satisfied for the co-reachable states that are also reachable from the initial states.

First, we consider only the specification of the control and verification requirements. At this point, we cannot predict if the states implied by liveness requirements of V are reachable. For the desired states we check that they are not all forbidden, i.e., it must hold that $D_i \wedge \neg F \not\Leftrightarrow \bot$ for every $i \in I$. The verification is efficiently performed on a 'syntactic' level looking only at the models using, e.g., Espresso [11].

In the following step, the supervisory control synthesis delivers the supervisor state feedback, given by S, and the co-reachable states, identified by CR. The liveness requirements identify events that must not be disabled by the supervisor in co-reachable states, i.e., for every $e \in \mathcal{E}$, we require that $V_e \wedge CR \Rightarrow P_e \wedge \neg S_e \wedge CR$. The states that do not satisfy this condition are counterexample states identified by $(\neg P_e \vee S_e \vee \neg CR) \wedge (V_e \wedge CR)$, which is equivalent to $(\neg P_e \vee S_e) \wedge V_e \wedge CR$ for every $e \in \mathcal{E}$. The verification fails only if the counterexample states are actually reachable from the initial states. For the desired state requirements we have to check that $D_i \wedge CR \not\Leftrightarrow \bot$ for all $i \in I$. We note that the required predicates CR and S_e for $e \in \mathcal{E}$ are easily extracted from the BDDs produced by the tool [5].

Finally, we need to verify that the verification requirements hold in the states of the supervised plant, i.e., the co-reachable states that are reachable from the initial states, identified by CR and R, respectively. As above, the liveness requirements must be satisfied for the reachable states as well, i.e., $V_e \wedge RCR \Rightarrow P_e \wedge \neg S_e \wedge RCR$ must hold for every $e \in \mathcal{E}$. Similar to above, we have that the counterexample states are identified by $(\neg P_e \vee S_e) \wedge V_e \wedge RCR$. We can rewrite this as $((\neg P_e \vee S_e) \wedge V_e \wedge CR) \wedge R$, using that $RCR = CR \wedge R$. This implies that in the final step we only need to check that none of the counterexamples of the previous steps is actually reachable from an initial state. We have a similar result for the desired state requirements, i.e., we need to verify that $D_i \wedge RCR \not\Leftrightarrow \bot$ for all $i \in I$. At present, we are developing the tools for reachability analysis in the vein of [3], [5].

In Fig. 3 we summarize the proposal for extending the supervisor synthesis framework of Fig. 2. The hierarchy of the verification is shown by the backward dependencies of the verification steps. Note that the verification that the specification is not conflicting can be performed before or in parallel with supervisor synthesis. We realize that the verification of the proposed liveness requirements is only a small part of what model-checking of the supervised plant can offer. The advantage of our approach is that the control and verification requirements are very closely connected and that they can be efficiently verified providing early feedback to the modeler. We also note that the reachability analysis is

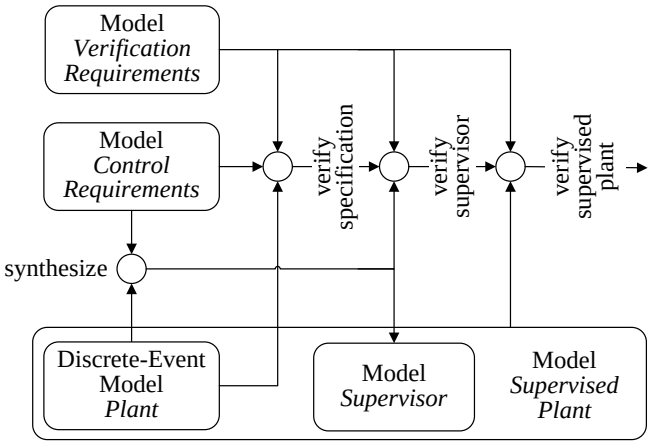

Fig. 3. Combining supervisor synthesis and verification

at most as expensive as the co-reachability analysis, which is part of the supervisor synthesis.

The combination of control and verification requirements ultimately leads to requirements of the form $CV_e \Leftrightarrow \rightarrow e$, for $CV_e \in gMS$ and $e \in \mathcal{E}$, where the left to right implication is used for supervisory control synthesis and the other for verification. The interpretation of these combined control/verification requirements is that the events from the set E must occur only in the combination of states defined by gMS in the supervised plant. Moreover, we can automatically generate the coarsest liveness requirements $W \subseteq LR$ corresponding to the above combined requirements as $W = \{P_e \wedge \neg CV_e \wedge \neg F \Rightarrow \rightarrow e \mid CV_e \Leftrightarrow \rightarrow e, e \in \mathcal{E}\}$.

V. Concluding Remarks

We proposed generalized state-based control requirements as an modeling extension of the state-based requirements of [3], providing for an intuitive and efficient modeling. We showed that the generalized requirements are provably equivalent to the standard state-based requirements. We extended the existing toolchain of supervisory synthesis framework to support the proposed generalized model of the requirements. We identified a new form of state-based expressions, which can be interpreted as liveness requirements and, thus, employed to check that the supervisor does not restrict some desired behavior of the plant. Finally, we propose a unified framework for supervisory control synthesis and verification based on the generalized state-based expressions. The proposed framework enables early feedback to the modeler as well as efficient liveness verification. Ongoing research looks into optimizing the translation of the generalized control requirements, as well as building tools for efficient reachability analysis of the supervised plant.

Plenty approaches tend to combine supervisory control synthesis with model-checking techniques. The use of model-checking techniques to perform the co-reachability analysis is advocated in [14]. A proposal to use CTL* formulas to specify the control requirements is given in [15]. An approach based on games that tends to use μ-calculus formulas instead [16]. It has been shown, however, that control requirements based on unrestricted CTL formulas is an NP-complete problem [17]. Also a translation from LTL formula to event-based control requirements is PSPACE-complete with respect to the size of the formula [18]. Thus, full-blown model checking provides thorough analysis, albeit at a high price, which often is not acceptable in the dynamic model-validate-remodel loop of our framework.

As future work we aim to deepen our understanding of the synthesis and verification algorithms, especially in the direction of automatic realization. Here, another interesting form of state-based expressions can help. It is given by $\rightarrow E \Rightarrow \not\rightarrow F$ for $E, F \subseteq \mathcal{E}$. It states that in order the events from the set E to be enabled, all events from the set F must be disabled. This, actually, amounts to prioritizing the events of the set F over the events of the set E, which is useful as a control requirement itself. This requirement has the potential to specify prioritized execution of multiple enabled controllable events.

References

[1] P. Ramadge and W. Wonham, "Supervisory control of a class of discrete event processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, 1987.

[2] C. Cassandras and S. Lafortune, *Introduction to discrete event systems*. Kluwer Academic Publishers, 2004.

[3] C. Ma and W. Wonham, *Nonblocking Supervisory Control of State Tree Structures*, ser. Lecture Notes in Control and Information Sciences. Springer, 2005, vol. 317.

[4] S. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, no. 6, pp. 509–516, 1978.

[5] C. Ma and W. Wonham, "STSLib and its application to two benchmarks," in *Proceedings of WODES 2008*. IEEE, 2008, pp. 119–124.

[6] I. Ogren, "On the principles for model-based systems engineering," *Systems Engineering*, vol. 3, pp. 38–49, 2000.

[7] R. Schiffelers, R. Theunissen, D. v. Beek, and J. Rooda, "Model-based engineering of supervisory controllers using CIF," *Electronic Communications of the EASST*, vol. 21, pp. 1–10, 2009.

[8] D. van Beek, P. Collins, D. N. Agut, J. Rooda, and R. Schiffelers, "New concepts in the abstract format of the compositional interchange format," in *Proceedings of 3rd IFAC Conference on Analysis and Design of Hybrid Systems*. Elsevier, 2009, pp. 250–255.

[9] R. Theunissen, R. Schiffelers, D. van Beek, and J. Rooda, "Supervisory control synthesis for a patient support system," in *Proceedings of 10th European Control Conference*. EUCA, 2009, pp. 1–6.

[10] H. Büning and T. Lettmann, *Propositional logic: deduction and algorithms*. Cambridge Univiversity Press, 1999.

[11] R. Brayton, A. Sangiovanni-Vincentelli, C. McMullen, and G. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984.

[12] R. Rudell, "Multiple-valued logic minimization for PLA synthesis," University of California Berkeley, Memo UCB/ERL M86/65, 1986.

[13] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River : Prentice Hall, 2004.

[14] R. Ziller and K. Schneider, "Combining supervisor synthesis and model checking," *ACM Transactions on Embedded Computing Systems*, vol. 4, no. 2, pp. 331–362, 2005.

[15] S. Jiang and R. Kumar, "Supervisory control of discrete event systems with CTL* temporal logic specifications," *SIAM Journal on Control and Optimization*, vol. 44, no. 6, pp. 2079–2103, 2006.

[16] A. Arnold, A. Vincenta, and I. Walukiewicza, "Games for synthesis of controllers with partial observation," *Theoretical Computer Science*, vol. 303, no. 1, pp. 7–34, 2003.

[17] M. Antoniotti and B. Mishra, "The supervisor synthesis problem for unrestricted CTL is NP-complete," University of California Berkeley, TR 95-062, 1995.

[18] K. Seow, "Integrating temporal logic as a state-based specification language for discrete-event control design in finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 4, no. 3, pp. 451–464, 2007.