Coordination of Resources using Generalized State-Based Requirements

J. Markovski * **K.G.M. Jacobs** * **D.A. van Beek** *
L.J.A.M. Somers ** **J.E. Rooda** *

* *Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands (e-mail: J.Markovski@tue.nl, jacobskoen@gmail.com, D.A.v.Beek@tue.nl, J.E.Rooda@tue.nl).*
** *Océ Research & Development and Department of Mathematics and Computer Science, Eindhoven University of Technology, The Netherlands (e-mail: Lou.Somers@oce.com)*

Abstract: Control and coordination is an important aspect of the development of complex machines due to an ever increasing demand for better functionality, quality, and performance. We develop a coordinator for maintenance procedures for a high-tech Océ printer that eliminates undesired behavior which stems from unrestricted interaction of its distributed components. To this end, we extend and employ a model-based engineering framework for supervisory controller synthesis. We generalize standard state-based control requirements to increase modeling convenience. We model the use case with 23 generalized state-based requirements, which translate to 1000+ requirements in the original form.

Keywords: supervisory control, model-based engineering, coordination, printers

1. INTRODUCTION

Over the last few decades, the complexity of high-tech machines has increased due to market demands for higher quality, better performance, new functionalities, improved safety, and ease of use. To satisfy the ever-changing market demands and to keep products competitive, both development costs and time-to-market have to be minimized. This puts high demands on the development of control and coordination for complex machines. In this paper, we deal with a high-end printer, which consist of three major parts as depicted in Fig. 1. We focus on the printing process (**2b**), which produces and fuses the toner images onto a paper sheet that runs along the paper path.

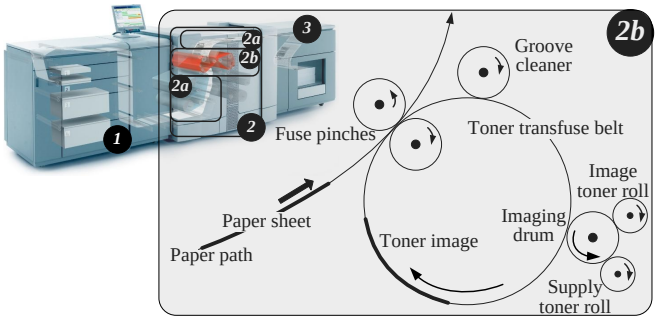

Fig. 1. High-end printer: 1. Paper input; 2. Print engine: a. Paper path, b. Printing process; 3. Finisher.

The printing process consists of several distributed components, each one controlled by embedded control software as depicted in Fig. 1. The process applies the toner image onto the toner transfuse belt and fuses it onto the paper sheet. To maintain high printing quality, several maintenance operations have to be carried out, some of which are: (1) toner transfuse belt jittering, which displaces the transfuse belt to prolong its lifespan due to its wearing by contact with the paper edges; (2) black image operation, which prints completely black pages to remove paper dust from the toner transfuse belt; (3) coarse toner particles removal operation, which removes leftover particles of toner.

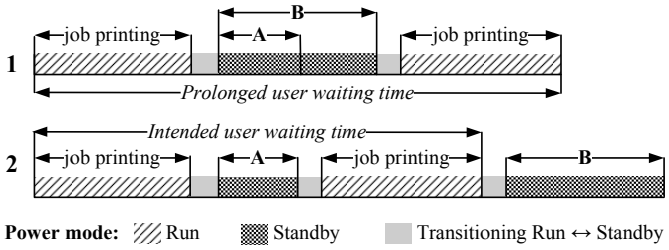

Fig. 2. User waiting time due to component interaction.

Most of the maintenance operations are scheduled after a given number of prints, but must be carried out after a given hard threshold. For example, after 500 prints there is need to perform a black image operation, unless there is an active print job, but after 700 prints it must be executed to maintain the printing quality, even if an ongoing print job needs to be interrupted. To perform a maintenance operation, the printing process typically has to change its power mode, e.g., from Run mode, used for printing, to Standby mode, required for maintenance. However, the change of power mode can actually trigger pending maintenance operations, which may unnecessary prolong the user waiting time.

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

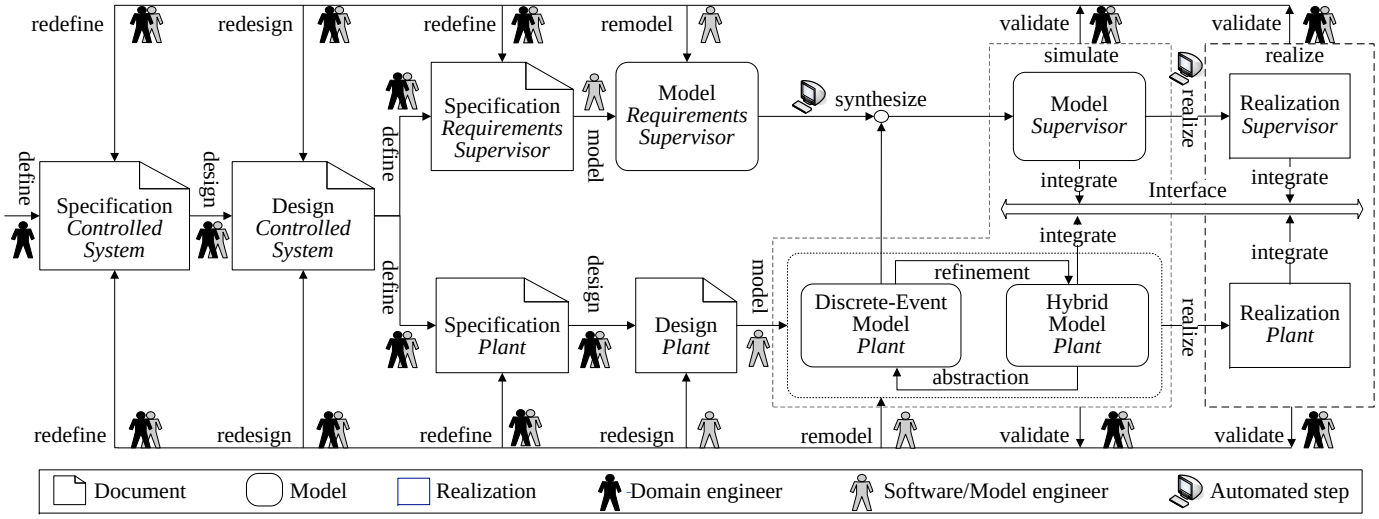

Fig. 3. Supervisory control model-based engineering framework.

As an illustration, in the upper part of Fig. 2 we depict situation **1**, where due to inevitable execution of maintenance operation **A**, the ongoing print job is suspended and the power mode of the printer is changed to Standby. Situation **2** depicts the desired behavior. However, an unwanted situation occurs, i.e., the power mode change triggers a longer, yet postponable maintenance operation **B**. For instance, a black image operation (**A**) must be performed, which takes the time needed to print one page and is activated often, but the switching of the power mode triggers a longer maintenance operation, e.g., toner transfuse belt jittering (**B**). Thus, in such situation the user has to wait unnecessarily for all maintenance operations to finish, after which the power mode can be changed to Run, which is required for printing.

The goal of the research performed for this use case was to eliminate undesired emergent behavior due to interactions of otherwise correctly-functioning distributed components, with primary focus at coordinating maintenance operations. To solve this problem we employed supervisory control theory for discrete-event systems, see Ramadge and Wonham (1987) and Cassandras and Lafortune (2004), specifying control requirements in a state-based manner as Ma and Wonham (2005). We utilize the model-based engineering framework of Ogren (2000) and a corresponding toolchain of Schiffelers et al. (2009) supported by the underlying Compositional Interchange Format (CIF), see van Beek et al. (2009). Our contributions in this paper are: (1) further extension of the model-based engineering framework triggered by identification of control requirements typically used in specification documents; (2) formalizing and extending the expressiveness of state-based control requirements to support complete propositional logic instead of the given restricted format accepted by the existing toolset; (3) solving, simulating, and implementing the use case with the new technique; and (4) identifying a new set of control requirements that can be interpreted in the setting of supervisory control synthesis, but are not implemented in the existing tooling. Section 2 discusses the supervisory control synthesis framework. In Section 3 we formalize the notion of generalized state-based control requirements and discuss the toolchain. An illustrative example of the modeling process is given in Section 4. We conclude and discuss possible extensions and application of the generalized state-based requirements in Section 5.

2. SUPERVISORY CONTROL SYNTHESIS

We synthesize a coordinator for the maintenance operations with respect to the change of the power mode (see Fig. 6 below). The coordinator has the form of a supervisory controller that supervises the unrestricted interaction of the distributed components of the printing process, referred to as *plant*. This is performed by imposing a set of *control requirements* that describe which interactions are allowed under which circumstances.

We follow the methodology of model-based engineering using the framework depicted in Fig. 3. The modeling starts with the informal specification of the controlled system, i.e., the desired product, written by domain engineers. This specification is used to contrive a design of the controlled system in which software engineers take part as well. The design, among other things, is important as it defines the level of abstraction and the control architecture. Subsequently, the design is used to define separately the plant and the control requirements, which is a joint task of the domain and software engineers. We do not explicitly state the role of the engineering further on, as it is clear from the context. This results in informal documents specifying the plant and control requirements, respectively. In the use case, the plant is given by the distributed components of the printing process, which already exists implemented as a prototype. The control requirements are given in documents written in spoken language. So, we focus on formal modeling of the control requirements and generating the control software that implements the coordinator.

In addition to modeling the control requirements, we also make a discrete-event model of the plant that is required for the supervisory control synthesis. A hybrid model of the plant already exists in a native software-in-the-loop simulation environment, see van der Hoest (2006). Next, a model of a maximally-permissive supervisor is synthesized automatically using the discrete-event model of the plant and the model of the control requirements.

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 the models of the plant and the control requirements. 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. In the use case, we validate the coordinator by software-in-the-loop simulation. In this way the existing hardware is simulated to enable visualization of maintenance operation execution, which visualization not available for hardware-in-the-loop simulation. During the development process, the control requirements change, but all that is needed is to remodel them provided that the plant model is validated. On certain occasions, a complete revision is necessitated, 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 act more as 'model' engineers, shifting their focus from writing code to modeling.

To support the intuition of the reader about the modeling process, we give a sample of the document Specification *Requirements Supervisor* (see Fig. 3): *"A maintenance operation can be performed only if the printing process is in Standby."* In the following section we formalize the control requirements and we discuss the proposed generalization.

3. GENERALIZED CONTROL REQUIREMENTS

In the dawn of supervisory synthesis, the plant was modeled as a set of possibly synchronizing finite automata, occupying one state per component, see Ramadge and Wonham (1987) and Cassandras and Lafortune (2004). The control requirements specified allowed behavior as sequences of events, again modeled by automata, leading to event-based supervisory control. To alleviate spatial and computational demand, Ma and Wonham (2005) propose the use of state tree structures, as the underlying model of the plant, and advocates state-based expressions as control requirements. State tree structures are a superset of automata and enable hierarchical modeling and support efficient storage. State-based control requirements restrict the plant by forbidding combinations of states of the parallel components, referred to as *mutual state exclusion* requirements. They can also state which events must be disabled for a given combination of states, referred to as *state-transition exclusion* requirements.

Here, we employ interchange automata, see van Beek et al. (2007), the underlying model for CIF, to describe the plant. They are supported by a vast range of conversion, simulation, verification, and visualization tools, see van Beek et al. (2009). In the examples, however, we do not give the complete syntax of CIF automata, since we focus primarily on specifying the control requirements. Thus, our visualization of the plant model should be treated as abstracted CIF automata that specify discrete-event behavior only. For a detailed account of supervisory control synthesis using CIF automata we refer the reader to Theunissen et al. (2009) and Schiffelers et al. (2009).

Next, we motivate the need for generalized state-based control requirements by means of a simple example. We take as an example the informal requirement stated in the previous section: *"A maintenance operation can be performed only if the printing process is in Standby."* The plant model that describes the situation is depicted in Fig. 4. The names of the states speak for themselves. Events are depicted by outgoing arrows, whereas incoming arrows with no source denote initial states. Controllable events, which can be restricted by the supervisor are drawn with full lines, whereas uncontrollable events, which must be always allowed, are depicted with dashed lines and their names begin with an underscore. Note that we abstract from marked (or final) states, see Ramadge and Wonham (1987), as they play no role in the specification of the control requirements.

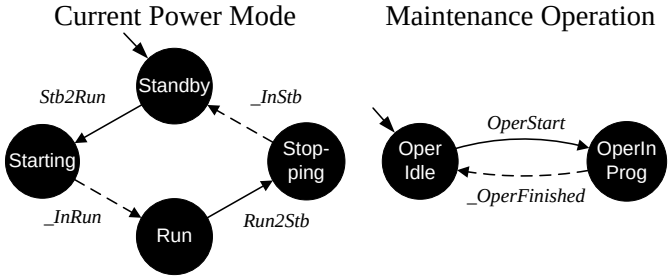

Fig. 4. Power mode and maintenance operation automata.

To formalize this control requirement we focus on the meaning of the states. The automaton on the left gives the power modes of the printing process. The state Standby denotes that the power mode is set to Standby. The status of the maintenance operation is given by the automaton on the right, with the state OperInProg denoting that the operation is in progress. Thus, we state the following intended implication:

Plant in OperInProg *implies* Plant in Standby.

The above expression states what has to be fulfilled, in con-trast to mutual state exclusion, which states what is illegal. The latter form, conformable with Ma and Wonham (2008), is given by:

not (Plant in OperInProg *and* Plant in Starting) *and*
not (Plant in OperInProg *and* Plant in Run) *and*
not (Plant in OperInProg *and* Plant in Stopping),

producing an equivalent, yet unintelligible expression.

Next, we formalize the above informal technique and show that there is always an translation from the 'generalized' to the standard control requirement. 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 $\mathcal{A}_i$ is given by $\mathcal{S}_i$ and $\mathcal{E}_i$, respective. This can be seen as a simplified state tree structure with an initial AND-superstate having OR-superstates as children, stemming from current lack of hierarchy in interchange automata. We note that ongoing research in the framework of the MULTIFORM FP7 European project aims to introduce AND/OR superstate hierarchy to interchange automata.

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 $\mathcal{E} \triangleq \bigcup_{i \in N} \mathcal{E}_i$. We use 1 and 0 for truth values of propositional logic and the standard logical operators: negation $\neg$, conjunction $\wedge$, disjunction

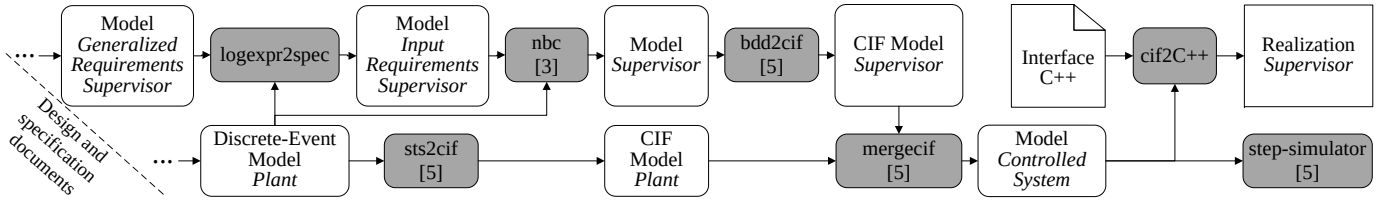

Fig. 5. Toolchain: The generalized state-based requirements are translated by the conversion tool logextp2spec.

$\vee$, implication $\Rightarrow$, and equivalence $\Leftrightarrow$. Conjunction and disjunction over an indexed set I of predicates p_i is given by $\bigwedge_{i\in I} p_i$ and $\bigvee_{i\in I} p_i$, respectively.

The *state predicate* $\mathsf{s}\downarrow$ expresses that the supervised plant is in state $\mathsf{s} \in \mathcal{S}$ 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 $\mathsf{S} \subseteq \mathcal{S}$ is denoted by $\mathsf{S}\downarrow \triangleq \bigwedge_{\mathsf{s}\in S} \mathsf{s}\downarrow$. 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 $\bigvee_{\mathsf{s}\in\mathcal{S}_i} \mathsf{s}\downarrow \Leftrightarrow 1$ and $\mathsf{s}\downarrow \wedge \mathsf{t}\downarrow \Leftrightarrow 0$ if $\mathsf{s} \neq \mathsf{t}$ with $\mathsf{s}, \mathsf{t} \in \mathcal{S}_i$ for every $i \in N$.

We formalize the state-based control requirements of Ma and Wonham (2005).

Definition 1. A mutual state exclusion requirement is given by $MS ::= 0 \mid 1 \mid \neg\, \mathsf{S}\downarrow$ for some $\mathsf{S} \subseteq \mathcal{S}$. A state-transition exclusion requirement is given by $ST ::= \neg\, MS \Rightarrow \not\rightarrow e$ with $\mathsf{S} \subseteq \mathcal{S}$ and $e \in \mathcal{E}$. The state-based control requirements are given by $SB ::= MS \mid ST \mid SB \wedge SB$.

The form of the mutual state and state-transition exclusion formulas are induced by the internal organization of the state tree structure, see Ma and Wonham (2005). The state-based control requirements impose that all formulas must hold together as stated by their conjunction in Definition 1. We note that originally the ST requirements of disable only uncontrollable events. To include disablement of controllable events is not difficult (the controllable events must be disabled in the reverse transition function Γ and the control functions f of Ma and Wonham (2005)). Moreover, the ST requirement of Definition 1 is already supported by the synthesis tool. Next, we generalize the state-based control requirements.

Definition 2. The generalized mutual state exclusion requirement is given by $gMS ::= 1 \mid \mathsf{s}\downarrow \mid \neg\, gMS \mid gMS \; op \; gMS$, for $\mathsf{s} \in \mathcal{S}$ and $op \in \{\wedge, \vee, \Rightarrow, \Leftrightarrow\}$. 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 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 direct generalization of the ST requirements of Definition 1, whereas the latter is most commonly used in the specification documents (see the example above). Both forms are equivalent, which follows from $\rightarrow E \Rightarrow g \Leftrightarrow \neg\, g \Rightarrow \not\rightarrow E$, where g is a gMS requirements, making $\neg\, g$ a gMS requirement as well.

As follows, we state that for every gMS and gST requirement, there exists an equivalent MS and ST requirement, respectively, making the generalized state-based control requirements a valid choice for supervisory control synthesis.

Theorem 3. For every gSB requirement g there exists SB requirement f such that $g \Leftrightarrow f$.

Proof. We sketch the proof that for every gMS requirement g there exists a SB requirement f such that $g \Leftrightarrow f$. The proof for gST requirements is similar. We refer to Jacobs et al. (2009) for the complete proof. We start with the conjunctive normal form of g. To obtain the form of a SB requirement, we eliminate the state predicates prefixed by negation using that $\neg\, \mathsf{s}\downarrow \Leftrightarrow \bigvee_{\mathsf{t}\neq\mathsf{s},\mathsf{t}\in\mathcal{S}_i} \mathsf{t}\downarrow$ where $\mathsf{s} \in \mathcal{S}_i$ for some $i \in N$. Finally, we use the De Morgan's and distributivity laws to obtain the form of a SB requirement as a conjunction of MS requirements.

The proof of Theorem 3 is constructive and it induces a translation algorithm, which we incorporated in an existing toolchain for supervisory control synthesis based on CIF and depicted in Fig. 5. The input to the toolchain are the models of the plant and the control requirements derived from the framework of Fig. 3. Our main contribution is the translation given by the tool "logexpr2spec", which translates the generalized state-based control requirements to an input suitable for the tool "nbc" of Ma and Wonham (2008). The model of the supervisor is given in BDD format which can be incorporated in CIF, together with the model of the plant in STS format. More information about these formats can be found in Ma and Wonham (2005). The CIF formats of the plant and supervisor models are then merged and analyzed by means of simulation. We also implemented a dedicated tool "cif2C++" that generates C++ code, given an interface to the printer control architecture. The following section discusses the control architecture and illustrates the modeling process.

4. COORDINATING MAINTENANCE OPERATIONS

An abstract view of the control architecture of a high-end printer is depicted in Fig. 6. Print jobs are sent to the printer by means of the user interface. The printer controller communicates with the user and assigns print jobs to the embedded software, which actuates the hardware to realize print jobs. The embedded software is organized in a distributed way, per functional aspect, such as, paper path, printing process, etc. Several managers communicate with the printer controller and each other to assign tasks to functions, which take care of the functional aspects.

We depict a printing process function comprising one maintenance operation in Fig. 6. We abstract from all timing behavior, which can be present in some control

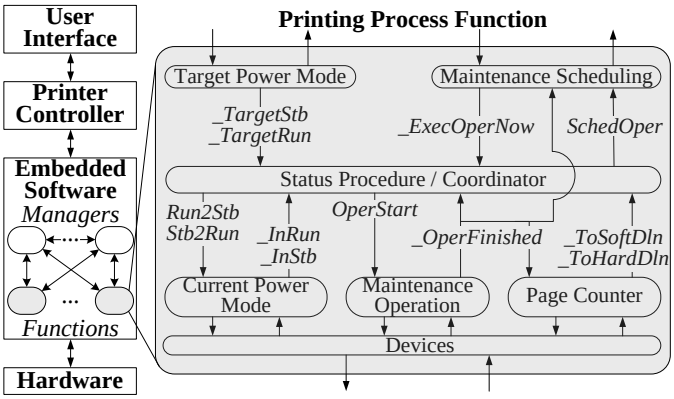

Fig. 6. Printing process function.

signals, e.g., execute a maintenance procedure after a given delay. Each function is hierarchically organized as follows: (1) controllers: Target Power Mode and Maintenance Scheduling, which receive control and scheduling tasks from the managers; (2) procedures: Status Procedure, Current Power Mode, Maintenance Operation, and Page Counter, which handle specific tasks and actuate devices, and (3) devices as hardware interface.

We describe the procedures. The Status Procedure is responsible for coordinating the other procedures given the input from the controllers. It will be implemented as a supervisory coordinator. We define the coordination rules below. The Current Power Mode procedure sets the power mode to Run or Standby depending on the enabling signals from the Status Procedure $Stb2Run$ and $Run2Stb$, respectively. The confirmation is sent back via the signals $\_InRun$ and $\_InStb$, respectively. Maintenance Operation either carries out maintenance operation or it is idle. The triggering signal is $OperStart$ and the confirmation is sent back by $\_OperFinished$. The Page Counter procedure counts the printed pages since the last maintenance and sends signals when soft and hard deadlines are reached using $\_ToSoftDln$ and $\_ToHardDln$, respectively. The counter is reset each time the maintenance is finished, by receiving the confirmation signal $\_OperFinished$ from Maintenance Operation. The controller Target Power Mode defines which mode is requested by the manager by sending the control signals $\_TargetStb$ and $\_TargetRun$ to the Status Procedure. Maintenance Scheduling receives a request for maintenance from Status Procedure via the signal $SchedOper$, which it forwards to a manager. The manager confirms the scheduling with the other functions and sends a response back to the Status Procedure via the control signal $\_ExecOperNow$. It also receives feedback from Maintenance Operation that the maintenance is finished in order to reset the scheduling.

The plant model of the Printing Process Function is depicted in Fig. 7. The names of the automata coincide with the names of the procedures. Since we are modeling an open system with a lot of freedom, the automata that model the managers have uncontrollable events that synchronize with external processes. The goal is to synthesize a coordinator that implements Status Procedure, which coordinates the maintenance procedures with the rest of the printing process. The control signals of Fig. 6 are modeled as events, and their names correspond. The signal $\_OperFinished$ is shared by multiple components and, thus, it is modeled as a synchronizing event. State names hint on their physical representation. The following coordination requests describe the behavior of the Status Procedure: (1) Maintenance operations can be performed only when the Printing Process Function is in Standby; (2) Maintenance operations can be scheduled only if soft deadline has been reached and there are no print jobs in progress or a hard deadline is passed; (3) Maintenance operations can be started only after being scheduled; (4) The power mode of the Printing Process Function must follow the power mode dictated by the managers, unless overridden by a pending maintenance operation.

Now, we formalize the above coordination requirements following the methodology outlined in previous sections. For the sake of clarity, we omit brackets when clear from the context. Requirement 1) is directly formalized as: $G_1 = \mathsf{OperInProg}\downarrow \Rightarrow \mathsf{Standby}\downarrow$, discussed above in the example of Fig. 4 in Section 3.

We schedule a maintenance operation using the signal $SchedOper$. States SoftDeadline and HardDeadline identify when a soft and hard deadline is reached, respectively. The power mode requested by the managers shows if there are any print jobs in progress. If the target power mode is Run, then there are print jobs in progress and, otherwise, not. So, requirement 2) is modeled as $G_2 = \rightarrow SchedOper \Rightarrow (\mathsf{SoftDeadline}\downarrow \wedge \neg\, \mathsf{TargetRun}\downarrow) \vee \mathsf{HardDeadline}\downarrow$.

For requirement 3), state ExecuteNow denotes that the maintenance operation has been scheduled by the managers and it can be started. We start the operation by sending the signal $OperStart$ from Maintenance Operation. This is modeled by $G_3 = \rightarrow OperStart \Rightarrow \mathsf{ExecuteNow}\downarrow$.

We model requirement 4) separately for switching from Run to Standby mode and vice versa. We can change from Run to Standby mode if this is required by the manager, i.e., the target power mode is Run, and there is no need to start a maintenance operation, i.e., Maintenance Scheduling is not in ExecuteNow. This is modeled as $G_{41} = \rightarrow Stdb2Run \Rightarrow (\mathsf{TargetRun}\downarrow \wedge \neg\, \mathsf{ExecuteNow}\downarrow)$. Contrariwise, when changing from Run to Standby power mode the manager must be followed, unless it is overridden by a pending maintenance operation, modeled as $G_{42} = \rightarrow Run2Stdb \Rightarrow (\mathsf{TargetStandby}\downarrow \vee \mathsf{ExecuteNow}\downarrow)$.

With the model of the coordination control requirements given by $\{G_1, G_2, G_3, G_{41}, G_{42}\}$, we synthesize a supervisory coordinator by using the toolchain of Fig. 5. The generalized requirements translated to 12 standard requirements. The complete plant model consists of 25 automata with state spaces ranging from 2 to 24 states with 23 generalized coordination requirements. These translated to more than 1000 state-based requirements using the preprocessing tool "logexpr2spec" (see Fig. 5).

5. CONCLUDING REMARKS AND FUTURE WORK

We proposed generalized state-based control requirements as an extension of the state-based requirements of Ma and Wonham (2005) to support complete propositional logic, providing for an intuitive and efficient modeling. We showed that they are provably equivalent to the standard state-based requirements. Our experiences show a con-

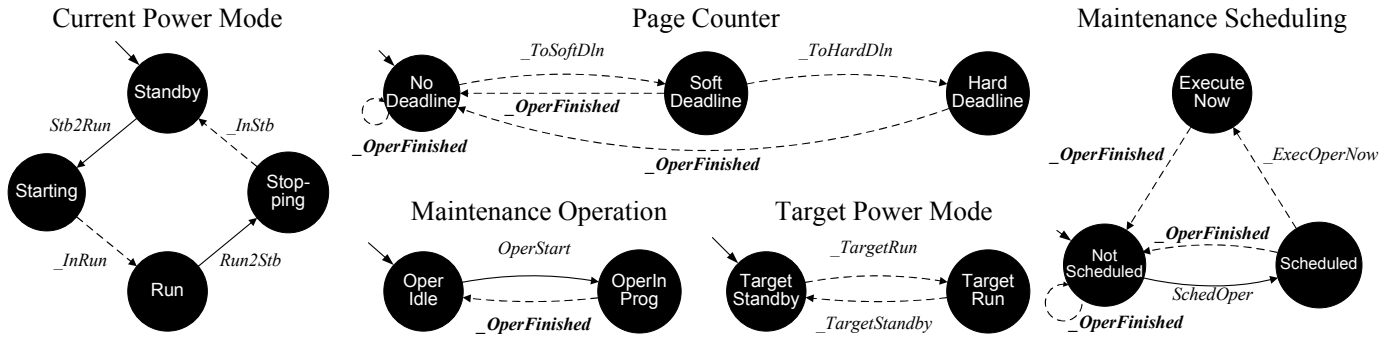

Fig. 7. Plant model of the Printing Process Function.

siderable gain in the number and clarity of the specified requirements, e.g., for the use case 23 generalized requirements produced more than 1000 standard requirements. The increase is mainly due to the elimination of the negation before state predicates by complementing it as residing in the remaining states of the same component, see Theorem 3. Note that if one would directly specify the standard control requirements, then this might optimize the number of requirements. However, in that case, the emphasis would not be in the clear and intuitive modeling, but in fitting the control requirements in the specific form.

We employed the generalized state-based control requirements to conveniently specify coordination rules. This come as no surprise as supervisory control sometimes requires coordination when, e.g., using modular supervision, see Ramadge and Wonham (1986). The coordination issues of Fig. 4 were resolved by correctly switching the power modes and scheduling execution of the maintenance operations. Finally, we extended the toolchain of supervisory synthesis framework of Fig. 3 to support the proposed generalized model of the requirements.

Additionally, we can specify forms of state-based expressions, which do not fit the above formats. One such class is $gMS \Rightarrow \rightarrow E$, where $E \subseteq \mathcal{E}$. One interpretation of these expressions is that the events from the set E must be enabled if the supervised plant resides in the combination of states identified by gMS. As a control requirement this is obsolete, as all events that are not explicitly disabled are allowed. Alternatively, one can see the expression as a liveness property for verification, requiring that the supervisor does not restrict the plant too much, i.e., some desired functionality is present in the supervised plant. We aim to look further in combining supervisory control synthesis with verification using the above expressions as 'verification' requirements. This would amount to unified control/verification requirements of the form $gMS \Leftrightarrow \rightarrow E$ where the left to right implication is used for supervisory control synthesis and the other for verification. The interpretation of these combined requirements is that the events from the set E must occur only in the combination of states defined by gMS.

Another interesting form is $\rightarrow E \Rightarrow \not\rightarrow F$ 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 requirements itself. We detected these requirements as $\rightarrow E \Rightarrow g$ and $g \Rightarrow \not\rightarrow F$, for g a gMS expression, which implies $\rightarrow E \Rightarrow \not\rightarrow F$. The requirements can also be employed for implementation purposes, e.g., to prioritize execution of multiple enabled controllable events.

REFERENCES

Cassandras, C. and Lafortune, S. (2004). *Introduction to discrete event systems*. Kluwer Academic Publishers.

Jacobs, K.G.M., Markovski, J., van Beek, D.A., Rooda, J.E., and Somers, L.J.A.M. (2009). Specifying state-based supervisory control requirements. SE Report 09-06, Eindhoven University of Technology.

Ma, C. and Wonham, W.M. (2008). STSLib and its application to two benchmarks. In *Proceedings of WODES 2008*, 119–124. IEEE.

Ma, C. and Wonham, W. (2005). *Nonblocking Supervisory Control of State Tree Structures*, volume 317 of *LNCIS*. Springer.

Ogren, I. (2000). On the principles for model-based systems engineering. *Systems Engineering*, 3, 38–49.

Ramadge, P.J. and Wonham, W.M. (1986). Modular supervisory control of discrete event systems. In *Analysis and Optimization of Systems*, volume 83 of *LNCIS*, 202–214. Springer.

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

Schiffelers, R.R.H., Theunissen, R.J.M., van Beek, D.A., and Rooda, J.E. (2009). Model-based engineering of supervisory controllers using CIF. *Electronic Communications of the EASST*, 21, 1–10.

Theunissen, R., Schiffelers, R., van Beek, D., and Rooda, J. (2009). Supervisory control synthesis for a patient support system. In *Proceedings of 10th European Control Conference*, 1–6. EUCA.

van Beek, D.A., Collins, P., Nadales Agut, D.E., Rooda, J.E., and Schiffelers, R.R.H. (2009). New concepts in the abstract format of the compositional interchange format. In *Proceedings of 3rd IFAC Conference on Analysis and Design of Hybrid Systems*, 250–255. Elsevier.

van Beek, D., Reniers, M., Schiffelers, R., and Rooda, J. (2007). Foundations of a compositional interchange format for hybrid systems. In *Hybrid Systems: Computation and Control*, volume 4416 of *LNCS*, 587–600. Springer.

van der Hoest, S. (2006). The development of a software-in-the-loop simulation framework for testing real-time control software. SAI report, Stan Ackerman Institute, Eindhoven University of Technology.