
Jasen Markovski

Abstract—We propose a model-based systems engineering framework that enables supervisor synthesis of stochastic (nondeterministic) discrete-event systems, and post-synthesis validation of functional and quantitative properties of the supervised system. Supervisory control theory deals with synthesis of models of supervisory controllers that ensure safe and nonblocking behavior, based on discrete-event models of the uncontrolled system and the control requirements. Typically, neither more elaborated functional properties nor performance metrics can be guaranteed by the synthesis procedure for large systems, due to high computational complexity. Thus, the supervised system must be validated to ensure that intended behavior is present. The framework employs a single integrated model that denotes all relevant aspects of the system. We rely on state-of-the-art tools to implement the proposed framework. For supervisor synthesis we employ Supremica, which models we extend to denote stochastic behavior. For verification, we provide a consistent translation of the supervised system to the model checker UPPAAL. To evaluate quantitative supervised behavior, first we transform the denoted system model to a labeled Interactive Markov chain and couple it with the synthesized supervisor. Then, we derive the underlying labeled Markov process and feed it to the stochastic model checker MRMC. We illustrate the framework on an industrial case study of coordinating printing maintenance procedures.

I. INTRODUCTION

Production of high-tech complex systems is faced with a new challenge of efficient development of control software. This issue becomes more prominent as machine complexity constantly increases, together with higher demands for better quality, performance, safety, and ease of use. Traditional software development techniques have proven not entirely adequate to handle the challenge as control software constantly evolves due to changes in the control requirements in the design process, leading to a costly iterative software (re)coding loop [1]. This issue gave rise to supervisory control theory, which tackles a part of this problem by investigating automated supervisory control software synthesis based on discrete-event models of the uncontrolled system and the control requirements [2], [3]. Supervisory controllers coordinate high-level system behavior by observing the discrete-event behavior of the machine. They receive sensor signals from ongoing activities, make a decision on allowed activities, and send back control signals to the hardware actuators.

A. Problem description

The promise of automatic control software generation has captured the interest of the industry with supervisory control becoming even more captivating as engineers nowadays are familiar with building models for simulation and validation purposes. Moreover, this technique enables rapid prototyping as one can couple the models with (prototype) hardware components to evaluate the control requirements, before building and testing expensive control software. However, automatic control software synthesis does not come with the guarantee that the supervised system has all intended functionalities, but it only guarantees safe and nonblocking system behavior, i.e., it prevents the system of reaching dangerous states, or deadlocking [2], [3]. Furthermore, even if the system performs functionally as intended, there are no guarantees about its performance or reliability, and additional quantitative analysis must be performed.

B. Related work

Naturally, extensions of the theory were proposed to incorporate liveness requirements during the synthesis procedures. The work of [4] extends the NuSMV model checker for synthesis employing CTL$^*$. Similarly, control requirements in CTL$^*$ are proposed and analyzed in [5]. In [6] a proposal

---

*Supported by Dutch NWO project ProThOS, no. 600.065.120.11N124.

1J. Markovski is with Department of Mechanical Engineering, Eindhoven University of Technology, The Netherlands j.markovski@tue.nl
to translate temporal logic to standard event-based control requirements is presented. Ensuring liveness for software synthesis by employing a variant of LTL is given in [7]. Unfortunately, these approaches suffer from (doubly-)exponential complexity due to enforcing of liveness during the synthesis procedure. Consequently, the proposed frameworks can handle only systems with $10^3 - 10^5$ states [4], [5], [6], [7]. Post-synthesis verification has also been proposed and implemented on several occasions. The work of [8] proposes verification of synthesized extended finite automata as a means to validate process operations for resource allocation. In [9] counterexamples are exploited to guide synthesis and verification procedures, whereas [10] investigates structural restrictions with hierarchical interfaces.

Similarly, several extensions have introduced quantitative aspects, like probabilities [11], [12], [13] and stochastic delays [14], [15], in order to ascertain that extra performance or reliability requirements are met as well. The problem of optimality has also been tackled in the field of performance evaluation, employing the wide-spread class of Markov decision processes [16]. The control problem is to schedule the control actions such that some performance measure is optimized. Stochastic game problem variants [17] that specify the control strategy using probabilistic extensions of temporal logics are also emerging in the formal methods community [18], [19], [20]. Nonetheless, these extensions suffer from similar complexity issues as ensuring liveness and/or performance requirements during synthesis is a costly undertaking.

C. Integrated framework proposal

We address the issues stated above in our proposal for an efficient integrated model-based systems engineering framework that relies on state-of-the-art tools. The framework aims to relieve some of the mentioned concerns, and set future research directions. We decouple the supervisor synthesis from validation of safety and liveness properties and performance evaluation. However, we employ only a single integrated model for all of these tasks, which denotes all necessary information. Our framework extends and subsumes prior model-based synthesis frameworks of [21], [22], [23], [24], [25], [26], which deal with separate proposals for verification and performance evaluation, respectively. We depict the core part of the framework involving the relevant models in Fig. 2A), whereas for a detailed description of the systems engineering process we refer to [23]. We work under the standard assumption that a supervisory controller can react sufficiently fast on machine input and, consequently, we model the supervisory control feedback loop as a pair of synchronizing processes [2], [3]. The model of the uncontrolled system, referred to as plant, is restricted by the model of the controller, referred to as supervisor. The coupling of the two models the behavior of the supervised system, referred to as supervised plant.

We presuppose that in Fig. 2A) the models of the control, liveness, and performance requirements, as well as the plant denoting all necessary aspects of the system, have already been made from the specification documents. For the complete process from formalization of specification documents to implementation of supervisory controllers, we refer to [22]. The plant comprises the discrete-event system behavior and it specifies its performance behavior by stochastically quantifying the duration of the events of interest. The plant and the (state-based) control requirements [22] are modeled in Supremica [27], which is also employed to synthesize a nonblocking maximally-permissive supervisor [2], [3], [27], as depicted in Fig. 2B). We employ additional syntax by means of enumerated variables attached to events to specify stochastic behavior in Supremica. For validation and verification purposes we employ the tool Supremica2UPPAAL [28] to translate the supervised plant to the model checker UPPAAL [29], as depicted by Fig. 2B).

For performance and reliability analysis, we employ labeled Interactive Markov chains [30], [25] to specify stochastic behavior in a compositional manner. We transform the annotated Supremica plant to a stochastic model by means of the transformation tool Supremica2IMC [28], see Fig 2B). Unlike other approaches, we treat stochastic delays syntactically, which is enabled by the fact that uncontrollable events and Markovian delays have the same interleaving behavior with respect to controllability [25]. Consequently, if we represent each stochastic delay by a uniquely-named uncontrollable event, we can synthesize a supervisor using a standard tool for supervisor synthesis, like Supremica. Alternatively, we can employ the supervisor that was synthesized for the annotated non-stochastic plant, as depicted in Fig 2A), as both supervisors coincide. The advantage of the latter approach is that the annotated plant comprises less states and transitions. Admittedly, it comprises additional (stochastic) variable assignments, but they do not increase the state space, as they are not used in event guards and, therefore, are completely irrelevant for the synthesis procedure and ignored by the tool [27]. Following supervisor synthesis, the (discrete-event) supervisor can be coupled with the stochastic plant, as depicted in Fig 2A). We assume that the supervised plant is a closed system, i.e., it does not rely on external inputs from the environment, which implies that its performance is defined. Then, we can reduce it to a (labeled) continuous-time Markov chain [16], [30], by employing the tool Supremica2MRMC [25], [28] as depicted in Fig. 2B). Thereafter, we feed the performance (or reliability) model to the stochastic model checker MRMC [31] and verify the given requirements, as depicted in Fig 2B) and A), respectively. We note that the developed tools and models are available for download from [28].

To illustrate our approach, we revisit an industrial case study that deals with coordination of maintenance procedures of a printing process of an Océ prototype printer [32]. Due to confidentiality issues, we can only present an (obfuscated) part of the case study. The goal of the case study is to synthesize a supervisory coordinator that ensures that quality of printing is not compromised by timely performing maintenance procedures, while interrupting ongoing print jobs as little as possible. Unlike previous attempts [32], [25],
where separate aspects of the case study were incorporated in separate models, here we define and employ a single integrated model to automatically perform both functional and quantitative analysis of the system. We synthesize a supervisor for the annotated plant and employ the supervisor to obtain both discrete-event non-stochastic and stochastic supervised plants. By using the appropriate tools as outlined above, we validate the functional aspects of the model using UPPAAL, whereas we obtain performance and reliability measures of interest using MRMC.

The rest of this paper is organized as follows. Section 2 introduces the reader to supervisory control theory and the case study that will be used as a running example throughout the paper. In section 3 we outline the translation to UPPAAL and state-based verification. Section 4 discusses the derivation of the stochastic model and some of its performance metrics, after which we finish with concluding remarks.

II. Modeling for Supervisory Control

We recall that we refer to the model of the uncontrolled system as plant, restricted by the model of the supervisory controller, which we refer to as supervisor. The model of choice is extended finite automata [33], extension of finite nondeterministic automata with state labels, data assignments, and guarded transitions. The events model observed activities of the system, and they are split into controllable events, which model interaction with the actuators of the machine, and uncontrollable events, which model observation of sensors. We recall that the synchronization of the plant and a supervisor, referred to as supervised plant, models the supervisory control loop. Therefore, the supervisor can disable controllable events by not synchronizing with them, but it must always enable available uncontrollable events by always synchronizing with them. In addition, supervised plants must satisfy the control requirements, which model allowed (safe) system behavior.

We rely on data-based control requirements, where specific states of the system will be identified by the variable assignments. We employ two prominent forms of control requirements: state exclusion, which specifies which combination of states of the concurrent components of the plant are allowed, and state-transition exclusion, which specifies which events are allowed in a given observation of states. Ideally, we should be able to refer directly to states, employing the state labels, but the current release of the synthesis tool [27] does not yet support such requirements, so we rely on variables toenumerate the states of each component. We note that the derived verification model employs the same variables to identify the corresponding states, whereas in the performance model the state labels are carried over directly.

A. Printing Process Function

We are dealing with high-tech Océ printers of [32], the control architecture of which is abstractly depicted in Fig. 3. The control architecture demands a use of a monolithic supervisor that is to be interfaced with the managers and this requirement is set by the industrial partner. The user initiates print jobs, which are assigned to the embedded software by the printer controller in order to actuate the hardware to realize them. 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. 3. Each function is differentiated to (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. Status Procedure is responsible for coordinating the other procedures given the input from the controllers. The control problem is to synthesize a supervisory coordinator that ensures that quality of printing is not compromised by timely performing maintenance procedures, while interrupting ongoing print jobs as little as possible [32]. In addition, we verify several verification and performance/reliability requirements imposed on the supervised system. The coordination rules that ensure safe behavior are given below.

We briefly describe the procedures of which the Supremica models are depicted in Fig. 3. Automata and procedure
names coincide, whereas state names hint on physical representation. Uncontrollable events are underscored, whereas variable assignments that trace the automaton state are placed below transitions labels. The variables are needed to specify the control requirements, as Supremica does not yet support direct use of state names [27]. In addition, we stochastically parameterize events that take an approximate amount of time using the parameter exp. We note that Supremica does not yet support real numbers, so we employ an enumerated variable type to stay within the domain of the modeling language, where $rXdY$ denotes the value $X.Y$. Even though with respect to modeling convenience, we had to make several compromises, as we did not have the resources to amply extend the modeling environment of the synthesis tool, we have to emphasize that the intended functionality of the tool chain is present as demonstrated with the implementation of the case study. We set the standard time unit to minutes. Then, if an event has a mean duration of $m$ minutes, the rate with which it occurs is computed as $\frac{1}{m}$. For example, the rate of $\_OpFin$ in Maintenance Operation, which denotes that the maintenance operation has finished its execution is 3.0 and, thus, we express that the average duration of the maintenance operation is one third of a minute, or 20 seconds. The other rates and their interpretation are stated in Table I.

Current Power Mode sets the power mode to run or standby depending on the enabling signals from Status Procedure. Maintenance Operation either carries out a maintenance operation or it is idle. The confirmation is sent back by the event $\_OpFin$, which synchronizes Maintenance Scheduling, Maintenance Operation, and Page Counter. Page Counter counts the printed pages since the last maintenance and sends signals when soft or hard deadlines are reached. The former signals that maintenance should be performed, but it is not yet compulsory if there are pending print jobs, and the latter is reached when maintenance must be performed to ensure print quality. The page counter is reset, triggered by the synchronization on $\_OpFin$, each time that maintenance is finished. Target Power Mode sends signals regarding incoming print jobs to Status Procedure, which should set the printing process to run mode for printing and standby mode for maintenance and power saving. Maintenance Scheduling receives a request for maintenance from Status Procedure and forwards it to the manager. The manager confirms the

<table>
<thead>
<tr>
<th>Event</th>
<th>Rate</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$_InRun$</td>
<td>20.00</td>
<td>3sec</td>
<td>Switch time from Standby to Run</td>
</tr>
<tr>
<td>$_InStb$</td>
<td>20.00</td>
<td>3sec</td>
<td>Switch time from Run to Standby</td>
</tr>
<tr>
<td>$_OpFin$</td>
<td>3.00</td>
<td>20sec</td>
<td>Duration of maintenance operation</td>
</tr>
<tr>
<td>$_SoftDln$</td>
<td>0.02</td>
<td>50min</td>
<td>Switch time from no to soft deadline</td>
</tr>
<tr>
<td>$_HardDln$</td>
<td>0.50</td>
<td>2min</td>
<td>Switch time from soft to hard deadline</td>
</tr>
<tr>
<td>$_JobFin$</td>
<td>0.20</td>
<td>50min</td>
<td>Duration of a print job</td>
</tr>
<tr>
<td>$_NewJob$</td>
<td>2.00</td>
<td>30sec</td>
<td>Time between print jobs</td>
</tr>
<tr>
<td>$_ExOper$</td>
<td>12.00</td>
<td>5sec</td>
<td>Duration of maintenance scheduling</td>
</tr>
</tbody>
</table>

TABLE I

DESCRIPTION OF THE STOCHASTIC DELAYS
scheduling with the other functions and sends a response back to Status Procedure. It also receives feedback from Maintenance Operation in order to reset the scheduling, again triggered by $OpFin$.

B. Control Requirements

As noted above, Supremica does not support direct use of the state names and, instead, we employ the corresponding variables. For example, to state that the page counter is in soft deadline, we refer to the state SoftDln from Page Counter, identified by $PC == 2$, where $==$ denotes equality.

Status Procedure is restricted by several coordination rules: 1) Maintenance operations can be scheduled only when Printing Process Function is in standby; 2) Maintenance operations can be scheduled only if soft deadline is reached and there are no print jobs in progress, or a hard deadline is passed; 3) Only scheduled maintenance operations can be started; and 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.

1) To model this property in Supremica, we identify the states Standby and from Current Power Mode and OperInProg from Maintenance Operation by $CPM == 1$ and $MO == 2$ respectively. Then, we need a state-exclusion property to model control requirements 1), i.e., we specify that no other combination of states is possible. To this end, we employ the notion of forbidden states, which force the supervisor to eliminate all states that inevitably reach them by traces of uncontrollable events [27]. To this end, we add a plant automaton that contains one uncontrollable transition with a unique label and let it target a forbidden state, as depicted in Fig. 4. The uncontrollable transition is guarded by $PM != 1 & MO != 2$, where $!=$ denotes inequality and $&$ denotes conjunction, so all states that do not conform to 1) are eliminated during supervisor synthesis.

2) States SoftDln and HardDln identify when a soft and hard deadline is reached, respectively. State NewJob of Target Power Mode states that there is a print job in progress. The event $SchOper$ is responsible for scheduling maintenance procedures. Thus, it is enabled if $(PC == 2 & TPM != 2) | PC == 3$, where $|$ denotes disjunction. To restrict the occurrence of $SchOper$, we employ a control requirement automaton with a guarded selfloop as in Fig. 4.

3) Similarly to 2), we model control requirement 3) in Fig. 4 by a selfloop guarded with $MS == 3$ as this guard identifies the state ExecNow from Maintenance Scheduling.

4) We model this requirement 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., there is a new print job identified by $TPM == 2$, and there is no need to start a maintenance operation, identified by $MS != 3$. Thus, we have a selfloop labeled by $Stb2Run$ and guarded by $TPM == 2 & MS != 3$, as depicted in Fig. 4.

On the contrary, when changing from Run to Standby power mode, the manager must be followed, unless it is overridden by a pending maintenance operation modeled by

III. Verification of Desired Functionality

We recall that the supervisor should ensure safe and non-blocking supervised behavior, but it comes with no guarantee that the intended functionality of the system is present [2], [3]. The modeler, however, specifies the control requirements with an (implicit) intent of stating the desired safe (and live) functioning of the system. Moreover, to ensure safety, the synthesis procedure may eliminate functionally important states that inevitably lead to unsafe behavior. Then, if this situation occurs, either the control requirements are too strict, or the model of the uncontrolled system is not sufficiently detailed or it is flawed, i.e., wrong assumptions have been made. Therefore, the intended behavior of the supervised system must be validated following the synthesis procedure.

A. Translation to UPPAAL

To (partially) validate the functionality of Status Procedure, which we synthesized as a supervisory coordinator, we employ the model checker UPPAAL [29]. To this end, we developed a tool Supremica2UPPAAL [28], which translates a Supremica model of the supervised plant to an UPPAAL automaton. Since UPPAAL supports the variable types of Supremica, we can employ the same variable assignments to identify states in UPPAAL. We treat the supervised plant as a closed system, i.e., it does not interact with the environment, so we translate all labeled transitions as outgoing broadcast channels [29]. We depict the translation of Maintenance Scheduling in Fig. 5 after some rearranging of the labels and the states for aesthetic purposes. We note that the variable $exp$ has been eliminated by Supremica during the synthesis process as it plays no role in the control requirements and, therefore, it is treated as redundant.

Thus, the translation preserves both the propositional signals and the transition structure of the labeled transitions.

---

<table>
<thead>
<tr>
<th>State-exclusion 1</th>
<th>State-transition exclusion 2, 3, and 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>MO == 2 &amp; CPM != 1</td>
<td>Stb2Run</td>
</tr>
<tr>
<td>Default</td>
<td>TPM == 2 &amp; MS != 3</td>
</tr>
<tr>
<td>Forbidden</td>
<td>MS == 3</td>
</tr>
<tr>
<td>block</td>
<td>Req</td>
</tr>
<tr>
<td></td>
<td>OpStart SchOper</td>
</tr>
<tr>
<td></td>
<td>Run2Stb</td>
</tr>
<tr>
<td></td>
<td>TPM == 1</td>
</tr>
<tr>
<td></td>
<td>Stb2Run</td>
</tr>
<tr>
<td></td>
<td>TPM == 2 &amp; MS != 2</td>
</tr>
<tr>
<td></td>
<td>PC == 3</td>
</tr>
<tr>
<td></td>
<td>PC == 2 &amp; TPM != 2</td>
</tr>
</tbody>
</table>

Fig. 4. Control requirements

The guard $TPM == 1 | MS == 3$ of the selfloop labeled with $Run2Stb$, again depicted in Fig. 4.

Employing the control requirements 1) – 4), depicted in Fig. 4, we synthesize a nonblocking supervisor for the plant depicted in Fig. 3, and we derive the supervised plant. The following step is to validate the functionality of the supervised plant by employing the model checker UPPAAL.
of the supervised plant. The final step is to model check the supervised plant, for which UPPAAL provides several schemes of temporal logical formulas that express both safety and liveness properties [29]. We employ the tool to validate the supervised behavior by model checking safety properties induced by the control requirements, as well as to verify that the supervised system is live and performing as intended.

We model the verification properties using the temporal logic supported by the tool. The logic is a restricted variant of CTL [29], where the combinations of $A$ and $E$, meaning for all paths and there exists a path, respectively, and $\Box$ and $\Diamond$, meaning for all states and there exists a state, respectively, are allowed, but without nesting. A useful form, referred to as leads to operator, given by $\phi \rightarrow \psi$ for $\phi, \psi \in B$ is introduced instead, which is equivalent to $A\Box(\phi \text{ imply } A\Diamond \psi)$. The standard logical operators are $\text{not}$, $\text{and}$, $\text{or}$, $\text{imply}$, $\text{leadsto}$, $\text{eventually}$, and $\text{deadlock}$ denotes presence of deadlock in a system.

### B. Verification Properties

We illustrate some of the properties that can be verified. First, we validate that Status Procedure does not have a deadlock, which should be ensured by the supervisor, by using

$$A\Box \text{not deadlock}. \quad (1)$$

Next, we validate that the state-exclusion requirement is satisfied, again ensured by the supervisor. For this task, we employ variables with the same name and value as in the Supremica model, as our translation preserves the variables with their corresponding assignments. Thus, we verify that

$$A\Box \text{MO} == 2 \text{ imply CPM} == 1. \quad (2)$$

Next, we check that if the system reaches a hard deadline and no maintenance operation is scheduled, then the maintenance operation becomes scheduled. This is specified as

$$PC == 3 \text{ and MS} == 1 \rightarrow MS == 2. \quad (3)$$

To ascertain that the maintenance procedure can be scheduled when soft deadline has been reached, we employ

$$E(\Diamond PC == 1 \text{ and MO} == 2). \quad (4)$$

Finally, we can check that Status Procedure follows the commands from the Target Power Mode manager, by verifying that if the target power mode is run, then the printing process also eventually switches to run mode as well:

$$TPM == 2 \rightarrow CPM == 3. \quad (5)$$

We note that the state-transition exclusion control requirements specify properties of states with respect to their outgoing transitions. As the system we are dealing with is deterministic, we can determine the target state of the transition and, thus, check such verification requirements. However, UPPAAL does not support directly reasoning with respect to combined verification requirements simultaneously specifying properties with respect to states and transitions. Thus, verification of nondeterministic systems, where one has to consider all target states, may not be fully supported by the tool. This suggests that UPPAAL is suitable for verifying state reachability properties, but it might be challenging to directly express properties that simultaneously consider states and their outgoing transitions. One solution involves use of variables to denote desired states as in [34].

Next, we demonstrate how to extract a Markov chain [16] from the supervised plant and we compute several performance and reliability measures using the stochastic model checker MRMC [31].

### IV. PERFORMANCE EVALUATION

To enable performance evaluation and reliability analysis, we need to extract a performance model from the supervised plant that comprises all relevant stochastic information. To this end, first we need to extract a stochastic plant model suitable for supervision. For this purpose, we employ the developed tool Supremica2IMC [28], which automatically ‘unfolds’ stochastically-parameterized transitions. Each transition that is denoted as stochastic, with an exponential rate specified by the parameter $\text{exp}$, becomes prefixed by a uniquely-named uncontrollable transition. These transitions are named $m2uX$ for $0 \leq X \leq 8$, standing for “Markovian to uncontrollable". The rate specified by the parameter $\text{exp}$ is kept in a separate “uncontrollable to Markovian” (.u2m) file, to be employed during the extraction of the performance model by the tool Supremica2IMC [28], [25]. We illustrate this process in Fig. 6A), where we depict the stochastic variant of Current Power Mode, as produced by Supremica2IMC and after some rearrangement of states and labels for esthetic purposes.

Next, we couple the stochastic plant with the supervisor and export the stochastic supervised plant from Supremica2UPPAAL.
ica in XML format, to be parsed and transformed into a
Markov chain using the Supremica2MRMC tool [28]. The
transformation extends [35] to cater for state labels. As
mentioned above, we require the two additional input files,
which specify the rates corresponding to the uniquely named
uncontrollable delays (.u2m) and the priority order of the
controllable actions (.pri). The tool Supremica2IMC caters
for the (.u2m) file automatically, whereas the modeler spec-
ifies the priority file. Ideally, an optimal priority of events
that delivers the desired performance should be deduced
automatically, which we schedule as future work. As the
supervised plant may contain action transitions, in order to
measure the performance of the plant, the action transitions
must not introduce real nondeterministic choices [30]. In
case such choices arise, the performance of the system is
undefined. To resolve this issue, one can enrich the model
with probabilistic choices that quantify the conflicting non-
deterministic choices, alter the original model to eliminate
them, or impose priorities on the action transitions. In the
latter case, we have decision supervision that optimizes the
behavior of the supervised plant [24]. As the supervisor
cannot disable uncontrollable events, they are automatically
given highest priority. For the rest, the priorities are speci-
fied by the modeler to be employed in Supremica2MRMC
for resolution of nondeterministic choice to derive a pure
Markov process.

The output of the transformation tool is a labeled
continuous-time Markov chain [31]. We carry over the state
labels from the original specification in Supremica. The state
labels are aggregated together with the states, by putting
together all state labels from states with outgoing labeled
transitions into a Markovian (stable) state. This process
is illustrated in Fig. 6B), where we depict the extraction
of the labeled continuous-time Markov chain of Current
Power Mode. Such a transformation preserves complete state
information in the abstracted states that the resulting Markov
process comprises.

A. Continuous Stochastic Logic

To specify the performance requirements, we employ Con-
tinuous Stochastic Logic [36], which is completely supported
by MRMC for continuous-time Markov chains and partially
supported for Markov decision processes. For this reason our
framework directly extracts the underlying Markov process.
Ideally, one would directly feed the stochastic supervised
plant to the stochastic model checker, only having to syntac-
tically align the output/input formats of the tools.

The logic syntax is split to state formulas and path
formulas. State formulas are employed to identify states by
their propositional labels or probability measures, whereas
path formulas identify states that satisfy time-bounded or
unbounded properties over sequences of reachable states.
Both types of formula can be coupled with probability
measures to verify performance or reliability properties. State
formulas SF have the following MRMC syntax [31]:

\[
SF ::= tt \mid ff \mid L \mid !SF \mid SF & \& SF \mid SF || SF \mid P\circ p]\{PF\} \mid S\circ p]\{SF\},
\]

where \(tt\) denotes the constant true, \(ff\) denotes the constant
false, \(L \in N\) is an atomic propositional symbol, ! denotes
negation, &\& denotes conjunction, || denotes disjunction,
\(\circ \in \{<,\leq,=,\geq,\}\), and \(p \in [0,1]\).

The probability measure operator \(P\{\circ p\}\{PF\}\) identifies
states on paths that satisfy \(PF\) and meet the probability
constraint \(\circ p\). The steady-state probability measure operator
\(S\{\circ p\}\{SF\}\) checks if the steady-state probability for being
in states that fulfill \(SF\) meets the probability constraint \(\circ p\).
These operators result in a state formulas that can be nested
to make up complex path formulas [36].

Path formulas \(PF\) combine state formulas using the next
state and the until operator, which can be bounded for a
specific time interval or unbounded:

\[
PF ::= X SF \mid SF U SF \mid X[r,r] SF \mid SF U[r,r] SF,
\]

where \(r \in \mathbb{R}\) and \(r \geq 0\). The next operator is satisfied if
the following state satisfies the state formula (in the given
time interval \([r,r]\)). The until operator identifies paths that
comprise states that satisfy the left state formula until they
reach an end state that satisfies the right state formula (again
in the given time interval for the bounded variant).

B. Verifying Performance and Reliability Requirements

To illustrate the model checking possibilities, we derived a
performance model using the rates in Table I and the priority
order of the controllable transitions: \(OpStart > Run2Stb >
Stb2Run > SchOper\). First, we check if the utilization of
the printer is higher than 80 percent. To this end, we have to
check if the steady-state probability that there is an ongoing
print job that is not interrupted by the maintenance procedure
is higher than 80 percent, i.e., 0.8, given by:

\[
S\{r > 0.8\}\{NewJob & \& !OperInProg\}.
\]

For the experimental values we took, the utilization of the
printer is 99.4 percent, so this property is verified.

Next, we check whether the probability that a print job is
interrupted in the first hour is less than 0.5, given by:

\[
P\{r < 0.5\}\{tt U[0,0.6] NewJob & !OperInProg\}.
\]

The resulting probability is 0.851, so the property is not
satisfied by the system.

Finally, we check whether the probability of postponing a
maintenance procedure that is scheduled during soft deadline
until hard deadline is reached is smaller than 0.7. To this end,
we first have to identify the states in which soft deadline is
reached and the maintenance procedure is scheduled. We can
do this by identifying the paths with probability greater than
zero that reach such a state. Then, we can employ these states
to check whether the probability to reach a hard deadline
without executing a maintenance procedure is smaller than
0.7. This property is modeled by the formula:

\[
P\{r < 0.7\}\{(P\{r > 0\}\{!OperInProg U SoftDln & \& Schld\} & \&
!OperInProg\} U HardDln\}.
\]

This probability computes at 0.04, implying that the property
is satisfied.
V. CONCLUDING REMARKS
We presented an integrated systems engineering framework in which one can model supervisory control software. The control requirements before building and testing expensive control software.

REFERENCES