2014 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)

Formal Description of an Approach for Power Consumption Estimation of Embedded Systems

Dmitriy Shorin and Armin Zimmermann
Technische Universität Ilmenau
System & Software Engineering
P.O. Box 100 565, D-98684 Ilmenau, Germany
{Dmitriy.Shorin, Armin.Zimmermann}@tu-ilmenau.de

Abstract—**This paper presents a formal description of an approach for model-based engineering of energy-efficient automation systems. Energy consumption is an important decision criterion, which has to be included in the search for good architectural and design alternatives already on the early system design stages. In the method, we describe an embedded system with an operational model for the processor hardware and an application model for the software. UML extended with MARTE profile elements is used for this part. Both models are transformed into a stochastic Petri net using transformation rules. The procedure steps are described with formal mathematical language and graphical illustrations.**

I. Introduction

Energy consumption is an increasingly important non-functional property of embedded systems and in automation in general. Design decisions have to be based on a trade-off between energy use and other requirements. It is thus important to evaluate the effect of architectural and other design decisions in all phases of the development process based on a good prediction of the energy consumption of an embedded system. In this work, we concentrate on early design steps, in which major architectural decisions are made, which may have a significant impact on the overall system's energy consumption. Modeling methods should be developed for discrete automation systems in such a way that the energy consumption, beside other parameters, can be modeled, estimated, and finally reduced.

The Unified Modeling Language (UML) [1] is an industry standard for describing software systems. However, it is not intended to describe non-functional system properties equally well, as there are no constructs for quantitative properties. Domain profiles of the UML have been developed for this task, namely, the MARTE profile (Modeling and Analysis of Real-Time and Embedded Systems) [2] as a successor of the UML Profile for Schedulability, Performance, and Time (SPTP) [3].

UML models adopting the MARTE profile contain the necessary information for energy consumption estimation. However, they are not usable directly, as UML models are not semantically well-defined for a specification of the resulting stochastic process. For this work, we proposed UML models to be transformed into a model for which analysis algorithms exist [4], [5] and, namely, into extended deterministic and stochastic Petri nets (eDSPN) [6], such that the behavior and the properties are preserved. This was motivated by an earlier work of a former colleague, in which single extended UML state chart models describing reliability aspects of a system were transformed into uncolored stochastic Petri nets (SPN) for their analysis [7], [8].

The proposed method has been described in detail in [5], covering how the energy consumption of a microcontroller can be estimated by dividing the system into hardware and software parts. The hardware part remains the same for all applications, and is thus described in an *operational model* and specifies all possible modes of the system, possible state changes, and their associated power consumption (as well as transition times, if applicable).

On the other hand, the effect of the controlling software is captured in one or more *application models*. It describes which steps are taken and what time is spent in which mode, and may have stochastic behavior (interrupts, for instance). Thus, it contains information about the operational states used in the specified system and their duration. The model distinction follows the principle of separation of concerns in (software) engineering of complex systems. This type of distinction can be found in other fields as well, for instance in manufacturing systems, where a similar relation exists between structure (machines, transport routes) and work plans [9].

The two models together contain all the necessary information for predicting the power consumption. In this paper, we describe mathematically the proposed method for the model transformation into an SPN. The resulting model can then be used to estimate the power consumption of the system with stationary analysis.

The power consumption can be described in even greater detail than presented in this paper by using a MARTE extension termed Dynamic Power Management (DPM) profile, which was developed at Tampere University of Technology [10]. By means of this profile, power aspects of embedded systems can be described. Its main idea consists in creating an individual state machine for each hardware component, which includes necessary information for calculating power consumption. However, the formula for calculating the total power consumption is bound up with the parameters of CMOS circuits. This method goes deeper into the hardware part of the system. Though we examine automation systems on the same level, our method is alternative and attempts to be more universal and simple for the end user. In the long run, it gets its own advantages as well as disadvantages. In our method, we use the common physical formula for electric power calculation, namely $P = I \cdot U$.

A related approach similar to the one of [7], [8] is taken in [11], in which enriched UML models are translated into stochastic Petri nets. The main difference to our work is the distinction between the two aspect models and their integration during transformation. Another approach with similar goals is presented in [12], where a UML model is translated into a colored Petri net (CPN) description as supported by the CPN tools [13]. However, the resulting model tends to be rather complex and the CPN interpretation does not support a natural notion of (stochastic) time similar to the widely accepted model class of stochastic Petri nets.

In [14], the authors deal with the power consumption estimation based on the Scheduling Analysis View. For this purpose, they developed the Power Consumption Analysis View Profile that lets the user model real-time and embedded systems executing a defined set of tasks.

Transformation rules for several behavior diagrams like use case, state chart, activity and sequence diagrams are proposed in [15]. The resulting model is called Generalized Stochastic Petri Net (GSPN) [16] and is characterized by using only exponentially distributed times. This idea is then developed in [17] and, finally, in [18], where non-functional properties, like energy consumption, are taken into account. The same Petri net type (GSPN) is used in [19] to evaluate the system performance. The approach presented in [20] deals with generalized semi-Markov processes from UML state chart and activity diagrams.

This paper is structured as follows. Section II describes operational and application models and their components. Subsection II-A specifies their non-functional properties. The correspondence function given in Subsec. II-B links the states between two models. The transformation of the UML state machines into SPNs is presented in Sec. III. In Sec. IV we give an example which shows a method application. Section V concludes the paper.

II. Describing an Embedded System by Means of UML and MARTE Profile

We consider each embedded system consisting of hardware and software parts and denote its *system model* by SM. The hardware part of the system is reflected in the *operational model* OM. In this model, all possible states and transitions of the system must be given. One embedded system can be described by means of the only operational model as the hardware part of an embedded system remains the same. In case of changing the components of the embedded system under consideration, we take the new composition as a new embedded system, for which a new operational model is needed.

The software part, i.e., a set of applications running in the system, is described in the *application model* AM. This model contains information about the states used in the specified program, the algorithm sequence and durations of steps. Any modification of the software part of the system must be reflected in the application model, yet the operational model remains the same. Thus, an embedded system SM can be represented by means of one operational model OM and an application model AM referring to it, and is specified with the following tuple:

$$SM = (OM, AM)$$

The operational model is described by a MARTE-extended UML state chart containing the following elements:

$$\begin{aligned} OM \;=\; & (ST^{OM}, I^{OM}, J^{OM}, C^{OM}, TR^{OM}, \\ & \text{powerPeak}^{OM}, \text{execTime}^{OM}, \text{prob}^{OM}) \end{aligned}$$

Where ST^{OM} denote *regular states*, I^{OM} *initial states*, J^{OM} *join states*, C^{OM} *choice states* and TR^{OM} the *transitions* of a restricted class of UML state charts. The MARTE-related additional information is captured in power consumption $powerPeak^{OM}$, execution times $execTime^{OM}$, and path splitting probabilities $prob^{OM}$ (see below).

Each of the application models AM is similar to SM and defined as follows:

$$\begin{aligned} AM \;=\; & (ST^{AM}, I^{AM}, J^{AM}, C^{AM}, TR^{AM}, \\ & \text{execTime}^{AM}, \text{prob}^{AM}) \end{aligned}$$

For building both model types, we use one the behavioral UML diagram — State Machines. Only a few elements of this diagram class are used in the presented method. We will name our subclass of the UML as *UML-SC**. Operational and application models have certain common characteristics, which are described in the following.

UML elements, which may be used for building models, are listed below. Note that the statements apply to all model-specific subsets; e.g., a definition or restriction for a generic ST will cover all corresponding sets ST^{OM} and ST^{AM} in a similar manner.

- Regular states
  We denote by *ST* a (finite) set of *UML-SC** regular states[1].
- Pseudo-states
  - initial state: The (finite) set of initial pseudo-states is denoted as I.
  - join: The (finite) set of join pseudo-states is denoted as J.
  - choice: The (finite) set of choice pseudo-states is denoted as C.

  For notational convenience, all states of a model (operational or application-specific) including the set of regular states ST and all sets of pseudo-states are united in the set ST^*:

  $$ST^* = ST \cup I \cup J \cup C$$
- Transitions
  We denote by *TR* a set of *UML-SC** transitions that represent connections between all types of states. Some restrictions apply, which are detailed further.

  $$TR \subseteq ST^* \times ST^*$$

To simplify some later definitions and restrictions, the set of incoming $\text{TrIn}(st)$ and outgoing transitions $\text{TrOut}(st)$ are defined for a state $st \in ST^*$ as follows: An incoming transition represents a connection from any state to the current one.

$$\forall st \in ST^* : \text{TrIn}(st) : \{tr \in TR \,|\, tr = (\cdot, st)\}$$

[1]For sake of clarity, *UML-SC** simple states (excluding pseudo-states) are denoted as *regular states*.

An outgoing transition represents a connection from the current state to any other.

$$\forall st \in ST^* : \mathrm{TrOut}(st) : \{tr \in TR \,|\, tr = (st, \cdot)\}$$

Next, we give common properties of these elements for both operational and application models.

- Regular states
  Each regular state has at least one incoming and exactly one outgoing transition.

  $$\forall st \in ST : |\mathrm{TrIn}(st)| \geq 1 \wedge |\mathrm{TrOut}(st)| = 1$$

  The presence of the incoming transitions is not a requirement of UML; however, a regular state without any incoming transitions will never be activated, and is thus obsolete.
  For the time being, our method restricts the number of outgoing transitions to one, firstly, to exclude user's mistakes by creating models, when parallel activities begin, but can never end. Besides, permit for system multiplicity must be realized in UML models in a way, which does not seem obvious for the user. In this case, there should be given many other additional rules for creating models. For the present, we propose to examine complex systems containing two or more sub-systems (e.g. several processors) as two or more separate systems, to build operational and application models for each of them, to analyze them separately and, finally, to calculate the total power consumption as the sum of the power consumption values of all separate systems.
  For the same reason, to avoid forking, we do not introduce the fork pseudo-state of the UML state charts.

- Pseudo-states:
  - initial state: Each initial state has no incoming and exactly one outgoing transition.

    $$\forall i \in I : |\mathrm{TrIn}(i)| = 0 \wedge |\mathrm{TrOut}(i)| = 1$$

  - join: Each join pseudo-state has more than one incoming and exactly one outgoing transition.

    $$\forall j \in J : |\mathrm{TrIn}(j)| > 1 \wedge |\mathrm{TrOut}(j)| = 1$$

  - choice: Each choice pseudo-state has at least one incoming and more than one outgoing transitions.

    $$\forall c \in C : |\mathrm{TrIn}(c)| \geq 1 \wedge |\mathrm{TrOut}(c)| > 1$$

    A pseudo-state should not directly follow another pseudo-state to avoid ambiguities in the later transition probability specification. This is a restriction of UML.

    $$\forall st \in ST : |\{c \in C \,|\, (c, st) \in TR\}| \leq 1$$

  Prohibited elements are all other pseudo-states, i.e., terminate, forks, entry/exit points, shallow / deep history; they are not used in the proposed method.

A. *Non-Functional Property Specification with MARTE*

Non-functional properties are described using the MARTE profile [2]. The following table shows, which stereotypes and attributes are used in the UML-SC* models, and if they are mandatory or optional.

The states are described by means of the *ResourceUsage* stereotype of the MARTE profile. This package was specially created to consider the resource consumption in the system. Two attributes have been considered – these are $\mathrm{execTime}$ and $\mathrm{powerPeak}$. The first one reflects the duration of staying in each state (in seconds), and the second one represents the power required for the state (in Watt).

TABLE I. STEREOTYPES AND ATTRIBUTES USED IN THE MODELS

Stereotype	Attribute	Operational model	Application model
ResourceUsage	powerPeak	mandatory	not applicable
	execTime	optional	mandatory
GaStep	prob	optional	mandatory

Operational model: In the operational model, all possible states and transitions of the system under consideration are described.

All regular states must have a (real and positive) attribute $\mathrm{powerPeak}$, which specifies the power consumption of the system in the modeled state.

$$\mathrm{powerPeak}^{OM} : ST^{OM} \rightarrow \mathbb{R}^+$$

Some regular states may have an attribute $\mathrm{execTime}$, which specifies the (execution) time spent in the states. The number must be real and positive if we are referring to the non-application-specific states or be simply a default value.

$$\mathrm{execTime}^{OM} : ST^{OM} \rightarrow \mathbb{R}^+ \cup \{\epsilon\}$$

In this and following formulas, ϵ means that the attribute has no numeric value, i.e., is unset or empty.

The attribute $\mathrm{execTime}$ should be applied, if the execution time for some regular states is known and remains the same regardless of the application executed.

Application model: In this model, states and transitions of an application (a program, a thread, etc.) are described. All regular states may have an attribute $\mathrm{execTime}$. This is mandatory if the value was not specified in the corresponding regular state of the operational model to avoid missing information:

$$\mathrm{execTime}^{AM} : ST^{AM} \rightarrow \mathbb{R}^+ \cup \{\epsilon\}$$

$$\begin{aligned}\forall st \in ST^{OM} \quad : \quad & \mathrm{execTime}^{OM}(st) = \epsilon \\ & \longrightarrow \mathrm{execTime}^{AM}(st) \in \mathbb{R}^+\end{aligned}$$

To abstract in later formulas from the place where the execution time has been specified for a state, we define a generic execution time:

$$\begin{aligned}&\forall st \in ST^{AM} : \mathrm{execTime}(st) = \\ &= \begin{cases}\mathrm{execTime}^{AM}(st) & \text{if } \mathrm{execTime}^{AM}(st) \in \mathbb{R}^+ \\ \mathrm{execTime}^{OM}(st) & \text{else}\end{cases}\end{aligned}$$

In the application model, regular states immediately following a choice pseudo-state must have an attribute prob specifying the probabilities of following the path to the corresponding regular states. The value of probability must be a real and positive number.

$$\text{prob}^{AM}: \{st \in ST^{AM} | (c, st) \in TR^{AM} \wedge c \in C^{AM}\} \rightarrow \mathbb{R}^+ \cup \{\epsilon\}$$

The attribute prob has to be specified in the application model.

$$\forall c \in C^{AM}, \forall st \in ST^{AM}, (c, st) \in TR^{AM}: \text{prob}^{AM}(st) \in \mathbb{R}^+$$

The sum of the probabilities of the regular states which immediately follow the choice pseudo-state should be equal to one. This restriction helps the user understand the real probabilities of the execution of the following regular states.

$$\forall c \in C: \sum_{(c,st)\in TR} \text{prob}(st) = 1$$

However, non-observance of this recommendation will not cause any problems by transformation the models into a Petri net as the numbers will be then transformed into weights which are automatically normalized.

B. Correspondence Between the Models

This section covers some specific properties of application and operational models, as well as their relationship.

The application model must have exactly one initial state: $|I^{AM}| = 1$.

As an application runs on the system hardware, it cannot add new system states to the model, but instead only refer to them. Thus, the application states are subsets of the operational model states for a system $SM = (OM, AM) : ST^{AM} \subseteq ST^{OM}$. Each regular state of the operational model can thus be either referenced by one or more regular states in the application model or not used at all.

The same is — for the normal case — required for state transitions: $TR^{AM} \subseteq TR^{OM}$. However, from our experience there are practical cases in which purely technical intermediate steps in the operation of, e.g., a microcontroller, on which an application runs, which would be cumbersome to list in the application model and cluttering the model by repeating them each time they are necessary. As a notational convenience for modelers, we allow to skip such intermediate states of the operational model in the referencing application model. This is only possible as long as some restrictions are obeyed to avoid missing or ambiguous information. Informally, when a state transition in the application model does not exist in the operational model, there must be a path of state transitions and states in the operational model that links the source and destination states referenced by the application model. Moreover, the execution times must be defined in the operational model for all of these transitions. In the case of several paths, the one with the lowest power consumption is assumed to be meant, also avoiding circular paths.

Formally, we first define a (non-circular, finite) path in the operational model between two states st_1 and st_k as follows:

$$\begin{aligned} &\forall st_1, st_k \in ST^{OM} : \text{path}(st_1, st_k) = \\ &\{(st_1, st_2), (st_2, st_3), \ldots, (st_{k-1}, st_k) \in TR^{OM} \\ &| \, st_1 \neq st_2 \neq \ldots \neq st_k, \\ &\forall i = 2 \ldots k-1 : \text{execTime}^{OM}(st_i) \neq \epsilon\} \end{aligned}$$

It should be noted that the normal case of a direct connection between two states is a valid (minimal) path with $k = 2$ then. The execution time of the source state st_1 does not have to be set in the operational model; it can also be given in the application model.

Based on this, the *set of all paths* between the two states st_1 and st_k in the operational model is given by:

$$\forall st_1, st_k \in ST^{OM} : \text{path}^*(st_1, st_k) = \{\text{path}(st_1, st_k)\}$$

For each individual path ph between the states st_1 and st_k, the overall *path power consumption* is then defined by

$$\begin{gathered} \forall st_1, st_k \in ST^{OM}, \\ \forall ph = \{(st_1, st_2), \ldots, (st_{k-1}, st_k)\} \in \text{path}^*(st_1, st_k) : \\ \text{pathPower}(ph) = \text{execTime}(st_1) \cdot \text{powerPeak}^{OM}(st_1) + \\ + \textstyle\sum_{i=2}^{k-1} \text{execTime}^{OM}(st_i) \cdot \text{powerPeak}^{OM}(st_i) \end{gathered}$$

We are interested in the power-consumption *shortest path* path^- between st_1 and st_k defined by

$$\begin{aligned} &\forall st_1, st_k \in ST^{OM} : \\ &\text{path}^-(st_1, st_k) = \underset{ph \in \text{path}^*(st_1, st_k)}{\arg\min} \text{pathPower}(ph) \end{aligned}$$

Technically, this means a standard search for the shortest path in a directed, weighted graph.[2] It should be noted that a direct connection between st_1 and st_k will always form the shortest path if it exists, independent of whether there are other valid paths.

With these preliminaries, we are ready to restrict the relationship between state transitions in application and operational models following the informal description given at the beginning of this subsection. For each state transition in the application model, there must be (at least) a corresponding valid path in the operational model:

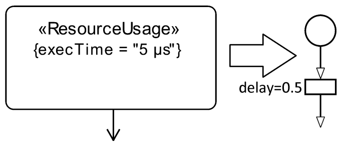
$$\forall (st_1, st_k) \in TR^{AM} : \text{path}^*(st_1, st_k) \neq \emptyset$$

III. Transforming Models into Stochastic Petri Nets

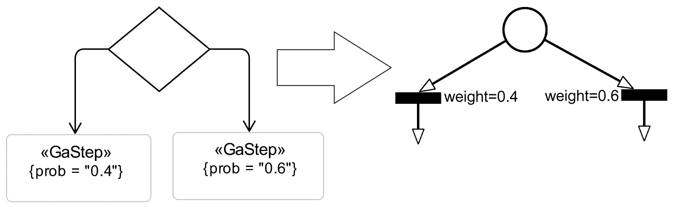
A generalized stochastic Petri net (GSPN) [16] $PN = (P, T, A, M_0, R)$ consists of the following elements:

- places: The (finite) set of places p is denoted as P, $p \in P$.
- transitions: The (finite) set of transitions t is denoted as T, $t \in T$.
- arcs: The set of directed arcs of a net connecting a place to a transition or a transition to a place is

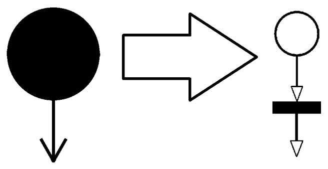
[2]We assume for simplicity that the shortest path is uniquely defined; in the case of several paths with equal power consumption, the modeler will be warned by the software tool about the possible ambiguity.



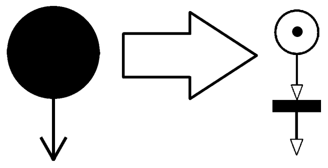
(a) regular state



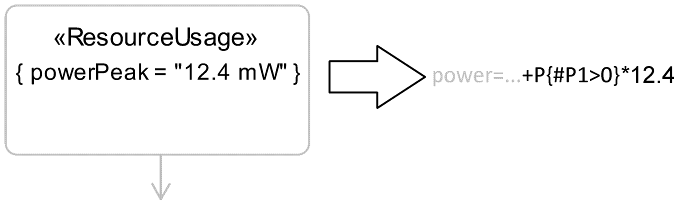
(b) choice pseudo-state



(c) join pseudo-state



(d) initial pseudo-state



(e) attribute $\mathrm{powerPeak}$

Fig. 1. Transformation rules

denoted as A, and relates the arc cardinality to the relation, i.e., the number of tokens removed or added $A : (P \times T) \cup (T \times P) \to \mathbb{N}$.

- markings: The current state of a GSPN is given by the number of tokens in each place, the marking $M : P \to \mathbb{N}$. The initial marking of a Petri net is denoted as $M_0 \in M$ and specifies the starting state.

A. Properties of Petri Net Elements

Each transition has a delay property. Its value must be either a positive real number or equal to zero. In the first case, the firing time is exponentially distributed with mean firing time given by the delay (*exponential transition*). If the delay is 0, the transition is called *immediate*.

$$\mathrm{delay} : T \to \mathbb{R}^+ \cup \{0\}$$

Immediate transitions have a weight property that is used to compute the probability of firing the transition in case of a conflict. The number must be real and positive.

$$\mathrm{weight} : T \to \mathbb{R}^+$$

In addition, we consider a *performance measure* R that specifies a reward function [21] formula over the stochastic process defined by the Petri net to calculate the power consumption later on. A formal definition of such a measure for stochastic Petri nets (see [22]) is out of the scope of this paper. Technically, it has to be defined in terms of a syntax as required by the used software tool [23].

B. Transformation Rules

To analyze the power consumption of the system, we combine application and operational models and convert them into a Petri net. For this operation, we take the application model as the basic structure. The operational model delivers missing information such as power consumption, missing states from paths, and their duration.

To denote the exact correspondence between UML models and their counterpart in the Petri net after the transformation, we introduce a (in many cases one-to-one) relationship between elements of both model types which is technically implemented by using the same names, and denote it by $st\langle p \rangle$, i.e., the state st corresponding to place p, and $p\langle st \rangle$, denoting the place p related to state st.

States and their outgoing transitions are transformed simultaneously, because UML transitions after different state types are transformed either into exponential or immediate transitions.

We use the following rules to transform UML models into a Petri net.

a) regular state (attribute $\mathrm{execTime} = "x"$*) + outgoing transition* $\Longrightarrow$ *place + outgoing exponential transition (attribute* $\mathrm{delay} = x$*):* Each UML regular state st_1 of the application model with its outgoing transition $(st_1, st_k) \in TR^{AM}$ is transformed into a Petri net place $p\langle st_1 \rangle$ and a transition $t\langle st_1 \rangle$ if such a direct transition between two states exists in the operational model. If not, the most power-efficient path must be found via other states in the operational model as defined earlier, such that all necessary information (execution time and power consumption) is given. It should be noted that in the case of such paths, the corresponding states of the operational model are leading to additional places and transitions in the Petri net each time they are referenced in the application model, just like a macro in a programming language. Therefore, the added places and transitions need to be identified based on both the source state and the sequence number in the path.

Each execution time value of the regular states ($\mathrm{execTime}$) is transformed into the attribute delay of the appropriate exponential transition of the Petri net.

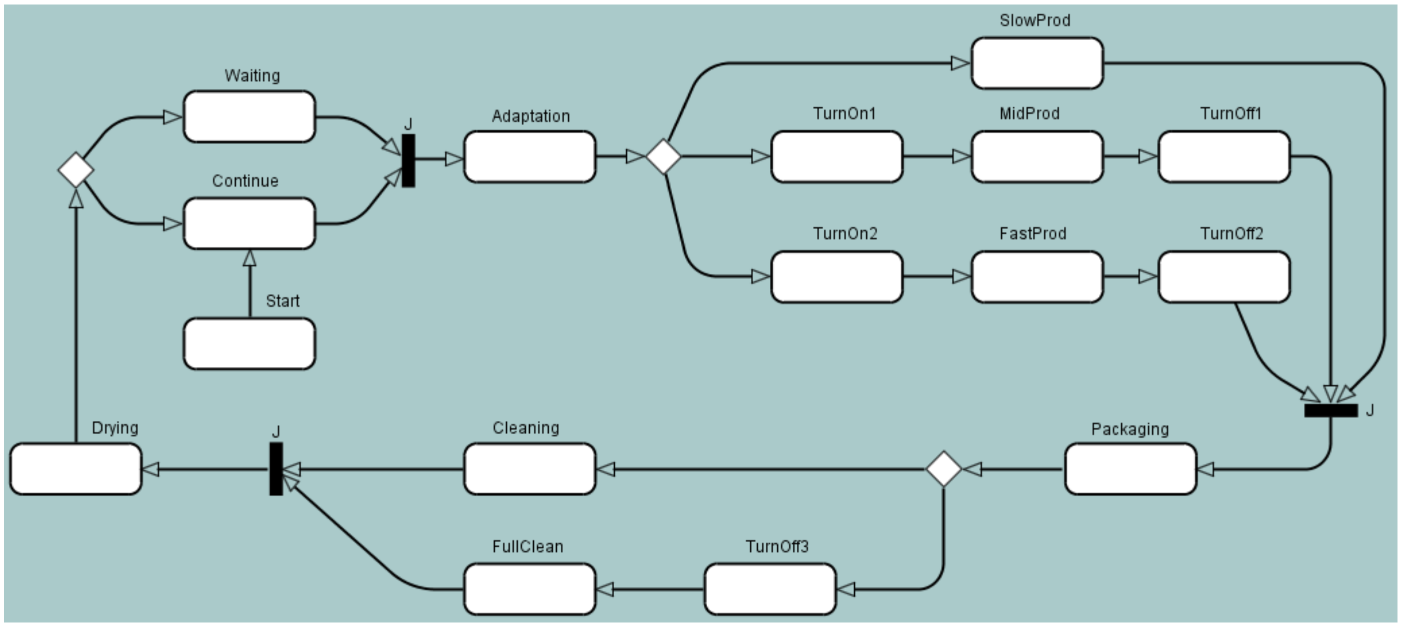


Fig. 2. Operational model of the workbench

$\forall st_1 \in ST^{AM}, (st_1, st_k) \in TR^{AM},$
$\text{path}^-(st_1, st_k) = \{(st_1, st_2), \ldots, (st_{k-1}, st_k)\}$
$\to p\langle st_1\rangle \in P$ (source place)
$\forall i = 2 \ldots k-1 : p\langle st_{1,i}\rangle \in P$ (other places)
$p\langle st_k\rangle \in P$ (destination place)
$\forall i = 1 \ldots k-1 : t\langle st_{1,i}\rangle \in T$ (transitions)
$\forall i = 1 \ldots k-1 : \text{delay}(t\langle st_{1,i}\rangle) = \text{execTime}(st_i)$ (delays)
$A(p\langle st_1\rangle, t\langle st_{1,1}\rangle) = 1$ (first arc)
$\forall i = 2 \ldots k-1 : A(t\langle st_{1,i-1}\rangle, p\langle st_{1,i}\rangle) = 1$ (arcs)
$\forall i = 2 \ldots k-1 : A(p\langle st_{1,i}\rangle, t\langle st_{1,i}\rangle) = 1$ (arcs)
$A(t\langle st_{1,k-1}\rangle, p\langle st_k\rangle) = 1$ (final arc)

The places derived during this transformation build a set P^S containing only places representing regular states: $P^S \subseteq P$.

This transformation is graphically presented in Fig. 1(a) for the simple case of a direct connection and depicts the execution time transformation.

b) choice pseudo-state + outgoing transitions (+ attributes prob $=$ *"x" of the following regular states)* $\Longrightarrow$ *place + outgoing immediate transitions (attribute* weight $=$ *x):* Each UML choice pseudo-state of the application model is transformed into a Petri net place which is followed by immediate transitions modeling the probabilistic choice.

Each probability value of the regular states, which immediately follow UML choice pseudo-states, is transformed into the attribute weight of the appropriate immediate transition of the Petri net.

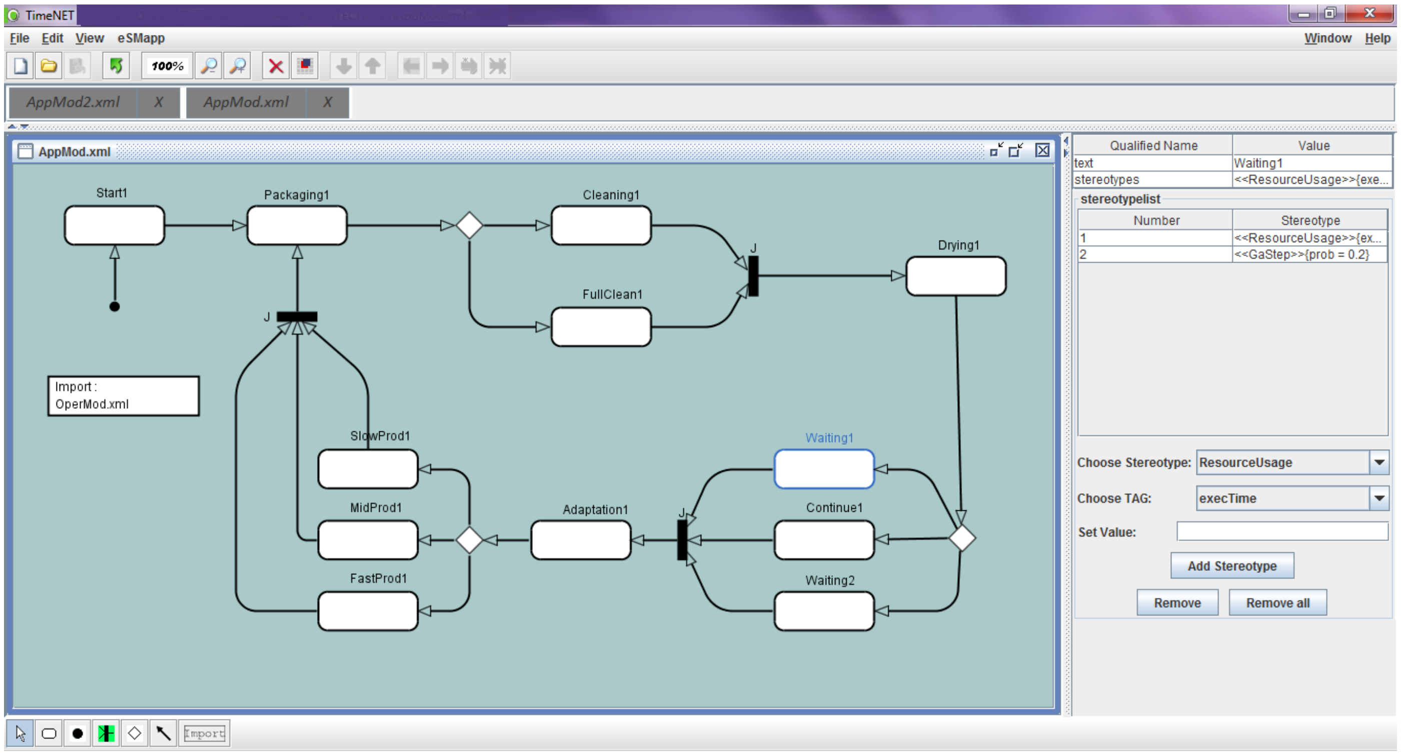
$\forall c_1 \in C^{AM}, (c_1, st_i) \in TR^{AM}$
$\to p\langle c_1\rangle \in P$ (choice place)
$\forall i = 1 \ldots k : t\langle tr(c_1, st_i)\rangle \in T$ (transitions)
$\forall i = 1 \ldots k : \text{delay}(t\langle tr(c_1, st_i)\rangle) = 0$ (zero delay)
$\forall i = 1 \ldots k : \text{weight}(t\langle tr(c_1, st_i)\rangle) = \text{prob}(st_i)$ (probabil.)
$\forall i = 1 \ldots k : A(p\langle c_1\rangle, t\langle tr(c_1, st_i)\rangle) = 1$ (arcs)
$\forall i = 1 \ldots k : A(t\langle tr(c_1, st_i)\rangle, p\langle st_i\rangle) = 1$ (arcs)

This transformation is graphically presented in Fig. 1(b) and depicts the probability transformation.

c) join pseudo-state + outgoing transition $\Longrightarrow$ *place + outgoing immediate transition:* Each UML join pseudo-state of the application model is transformed into a Petri net place which is followed by an immediate transition (see Fig. 1(c)).

$\forall j_1 \in J^{AM}, (j_1, st_1) \in TR^{AM}$
$\to p\langle j_1\rangle \in P$ (join place)
$t\langle tr(j_1, st_1)\rangle \in T$ (transition)
$\text{delay}(t\langle tr(j_1, st_1)\rangle) = 0$ (zero delay)
$\text{weight}(t\langle tr(j_1, st_1)\rangle) = 1$ (probability)
$A(p\langle j_1\rangle, t\langle tr(j_1, st_1)\rangle) = 1$ (arc)
$A(t\langle tr(j_1, st_1)\rangle, p\langle st_1\rangle) = 1$ (arc)

d) initial state: initial state + outgoing transition $\Longrightarrow$ *place (attribute* initialMarking $= 1$*) + outgoing immediate transition (Fig. 1(d)):* Each UML initial state is transformed into a Petri net place which is followed by an immediate transition. The attribute initialMarking of the Petri net place is set to one, thus setting one token as initial marking.



Qualified Name	Value
text	Waiting1
stereotypes	<<ResourceUsage>>{exe...

stereotypelist

Number	Stereotype
1	<<ResourceUsage>>{ex...
2	<<GaStep>>{prob = 0.2}

Choose Stereotype: ResourceUsage

Choose TAG: execTime

Set Value:

Fig. 3. Application model of the workbench modeled in TimeNET

$$
\begin{aligned}
&\forall i_1 \in I^{AM}, (i_1, st_1^*) \in TR^{AM} \\
&\rightarrow p\langle i_1\rangle \in P \text{ (initial state place)} \\
&\quad t\langle tr(i_1, st_1^*)\rangle \in T \text{ (transition)} \\
&\quad \text{initialMarking}(p\langle i_1\rangle\rangle) = 1 \text{ (initial marking)} \\
&\quad \text{delay}(t\langle tr(i_1, st_1^*)\rangle) = 0 \text{ (zero delay)} \\
&\quad \text{weight}(t\langle tr(j_1, st_1^*)\rangle) = 1 \text{ (probability)} \\
&\quad A(p\langle i_1\rangle, t\langle tr(j_1, st_1^*)\rangle) = 1 \text{ (arc)} \\
&\quad A(t\langle tr(i_1, st_1^*)\rangle, p\langle st_1^*\rangle) = 1 \text{ (arc)}
\end{aligned}
$$

e) power consumption: powerPeak = *"x" (regular state attribute)* $\Longrightarrow \cdots + P\{\#Name > 0\} \cdot x$ *(property* expression *of the element* measure*) (Fig. 1(e)):* Each UML regular state of the application model has its power consumption. The relevant value is taken from the corresponding regular state of the operational model. By each transformation, we extend the (initially empty) formula for power consumption estimation with a summand "$P\{\#p_i > 0\} \cdot x$", where $P\{\#p_i > 0\}$ is a probability of activation of the regular state $st_i\langle p_i\rangle$ and x is its power consumption value. The $+$ in this formula is taken as a shorthand for expression concatenation.

$$
\begin{aligned}
&\forall p_i \in P^S \\
&\rightarrow \text{expression(measure)} = \text{expression(measure)} \\
&\quad + P\{\#p_i > 0\} \cdot \text{powerPeak}(st_i\langle p_i\rangle)
\end{aligned}
$$

After finishing the transformation of the UML models into a Petri net, the power consumption of the system can be automatically calculated by software tools supporting stochastic Petri nets. In our case, TimeNET [24] is employed for modeling and analysis of stochastic Petri nets with non-exponentially distributed firing times.

IV. Example

In this section, we show how the described method is used in practice. In our earlier paper [5], we used a simple program implemented on a microcontroller to prove the accuracy of the proposed method. The worst inaccuracy of the method lay on the level of 2%. In contrast to the example used in our earlier paper [5], we use an industrial control system instead of an embedded microcontroller application to show the wide range of applicability of the method and software application. However, in contrast to the former example, in this case, the accuracy of the calculations cannot be verified on a real industrial control system.

In this example, we consider component production by a workbench with a main and two reserve motors. Its structure is depicted in Fig. 2. The workbench gets its first order and is being started (represented by the state *Start*). The process goes through the fictitious state *Continue* (see details below) and the order is adapted. Furthermore, there are three possibilities of producing components depending on performance requirements. The first one is called *Slow production* — it takes 5 minutes to create one unit. By choosing the second mode, one motor more is started and thus, the production speed reduces to 3 minutes. The third possibility is to start two reserve motors and to produce the component in only one minute. The fastest way could be the most preferable one, but the difference between these three modes is also in the power consumption. We assume that the longer it takes the workbench

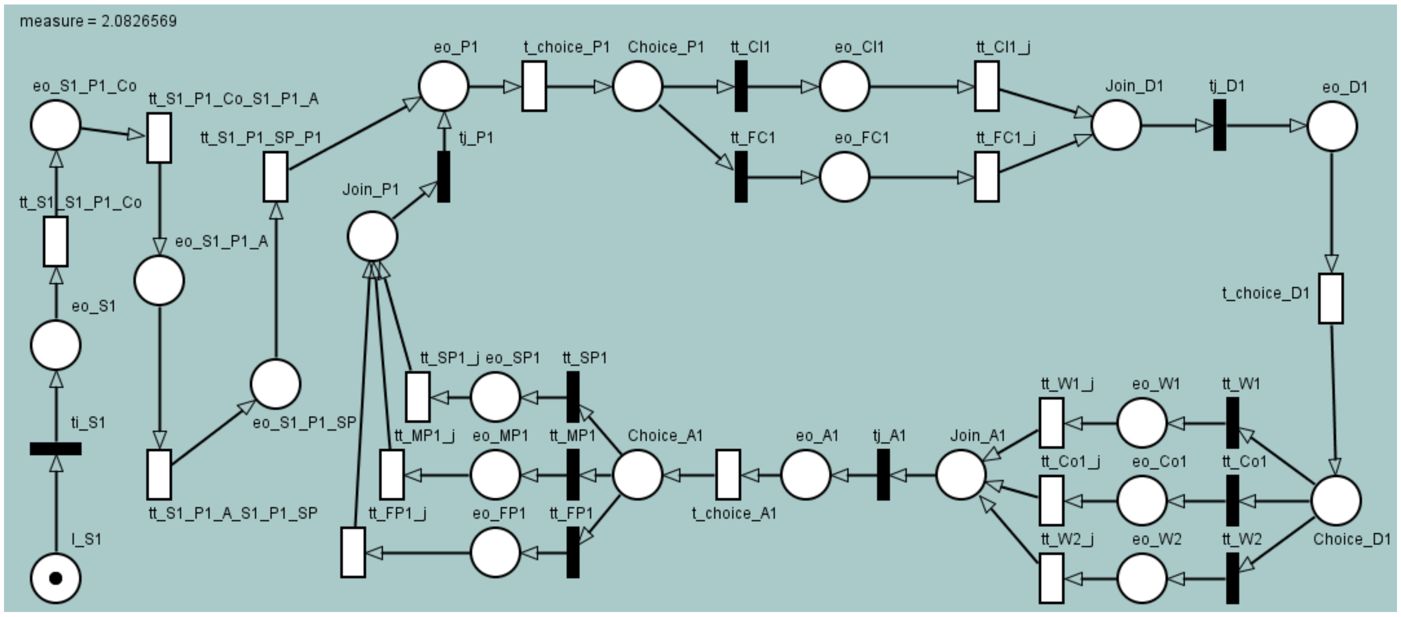


Fig. 4. Resulting Petri net of the system

to produce a unit, the less overall power it needs for this operation. The energy needed for the workbench to function in the first mode is 2.5 watt-seconds, in the second one 3 W·s and in the third – 5 W·s. Thus, the example shows a design trade-off between power consumption and other conflicting non-functional properties of a system. An overview of the attributes related to the states is given in Table II.

TABLE II. ATTRIBUTES STATED IN THE OPERATIONAL MODEL

State name	powerPeak, W	execTime, min.
Start	5	2
Waiting	0.1	
Continue	0	0.00001
Adaptation	0.2	0.1
TurnOn1	2	0.2
TurnOn2	4	0.2
SlowProd	0.5	5
MidProd	1	3
FastProd	5	1
TurnOff1	0.1	0.1
TurnOff2	0.2	0.1
TurnOff3	0.1	0.4
Packaging	0.3	0.2
Cleaning	0.3	0.1
FullClean	1	0.5
Drying	0.4	0.1

If one or two reserve motors are used in the production process, it takes extra time and power to turn them on and off (states *TurnOn1, TurnOn2, TurnOff1, TurnOff2*). When the component is produced, it will be packed (*Packaging*). After each procedure, the workbench must be cleaned. Cleaning has two modes: either a normal quick *Cleaning* or a *Full Cleaning* which takes more time and demands the main motor to be also stopped (*TurnOff3*). After that, it takes a little time to dry the workbench (*Drying*). If necessary, the main motor is being started during this process. Thus, the workbench finishes its work on the unit and goes either in the standby mode (*Waiting*) or continues its work without a pause. The value 0.00001 given in the *Continue* state is caused by the requirement for the exponential transitions of Petri net: the delay value which here represents the execution time may not be equal to zero. Otherwise, the Petri net will not be properly analyzable. However, this substitution does not influence the end result significantly. The only function of the state *Continue* itself is to be a regular (non-pseudo) state after the choice pseudostate. This restriction of UML was described in Sec. II.

Figure 2 depicts the operational model built on the basis of the system description above. After that, the user can create an application model. The one used in this example is shown in Fig. 3. When the first order arrives, the workbench starts working (*Start1*). As it is necessary to produce one unit, no matter how quickly it will be, it is enough to simply create one state *Packaging1*. The production mode will be chosen automatically while creating the SPN. The full cleaning mode should take place every ten production steps on average. Thus, the state *Cleaning1* has a probability (prob) 90% and *FullClean1* – 10%. After the *Drying*, the workbench continues its work (with the probability of 70%) or has either a short break (1 minute long, 20% probability) or a long break (5 minutes long, 10% probability). These probability values were chosen on the basis of statistical data. The further choice of the production mode depends on the demand. Although the longest mode (*SlowProd1*) is the most power-efficient one, statistically it can be used only in 20% of cases. Three out of ten units are produced in the middle-speed mode (*MidProd1*), and the half of all orders must be done while using both reserve motors (*FastProd1*). The component is then being packed (*Packaging1*) and the production cycle starts again. The element *Import* states that the XML file containing the operational model is technically linked to the current application model. An overview of the attributes given to the states is summarized in Table III.

To transform both models into a GSPN, the user clicks *eSMapp* $\rightarrow$ *eSMapp to eDSPN* in the tool menu. The information from the application model is being analyzed and the missing data is taken from the operational model. Thus, the

TABLE III. ATTRIBUTES SPECIFIED IN THE APPLICATION MODEL

State name	execTime, min.	prob
Start1		
Packaging1		
Cleaning1		0.9
FullClean1		0.1
Drying1		
Waiting1	1	0.2
Continue1		0.7
Waiting2	5	0.1
Adaptation1		
SlowProd1		0.2
MidProd1		0.3
FastProd1		0.5

power consumption is given only in the operational model. For the states, where execution time is not defined in the application model, this value is also taken from the operational model (e.g. *Start1*, *Packaging1*, *Cleaning1* and so on). Missing states between two similar states are added to make the transition stated in the application model possible (e.g. states *Continue*, *Adaptation* and *SlowProd* are missed between the states *Start1* and *Packaging1*). The parameter delay of the exponential transitions is filled up with the information from the attribute $\mathrm{execTime}$ of the respective simple states. The formula for estimating the power consumption is being constructed using the information from the attribute $\mathrm{powerPeak}$. The resulting Petri net is shown in Fig. 4.

The stationary analysis of the Petri net results in an average power consumption of the system of 2.0826569 Watt.

V. CONCLUSION

This paper presented the formal basis for a recently proposed methodology for the model-based estimation of energy consumption for embedded systems. UML extended with the MARTE profile is used for the modeling process. For the modeling part, a system is described with operational and application models, which reflect correspondingly hardware and software parts of the system. These two models are converted into a stochastic Petri net, which is then used for the performance evaluation. Finally, the design process for embedded systems can be supported by predicting the power consumption. The main contribution is the formal description of the proposed approach. Such introduction of the method let us present it within the strict frameworks of the mathematical tool, that, in its part, lets the audience and the authors as well evaluate fullness and consistency of the approach.

ACKNOWLEDGMENT

The authors would like to thank the Thuringian Ministry of Education, Science and Culture and the German Academic Exchange Service (DAAD) for the financial support of the project.

REFERENCES

[1] Object Management Group (OMG), “OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1,” May 2012. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/

[2] ——, “UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.1,” Jun. 2011. [Online]. Available: http://www.omg.org/spec/MARTE/1.1/PDF

[3] ——, “UML Profile for Schedulability, Performance, and Time Specification, Version 1.1,” Jan. 2005. [Online]. Available: http://www.omg.org/spec/SPTP/1.1/PDF

[4] D. Shorin and A. Zimmermann, “Evaluation of Embedded System Energy Usage with Extended UML Models,” in *Proceedings of the 2nd Workshop Energy Aware Software-Engineering and Development (EASED@BUIS), Softwaretechnik-Trends, 33(2)*, Oldenburg, Germany, Apr. 25, 2013.

[5] D. Shorin, A. Zimmermann, and P. Maciel, “Transforming UML State Machines into Stochastic Petri Nets for Energy Consumption Estimation of Embedded Systems,” in *2nd IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012)*, Pisa, Italy, Oct. 04–05, 2012.

[6] R. German, *Performance Analysis of Communication Systems, Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley and Sons, 2000.

[7] J. Trowitzsch and A. Zimmermann, “Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets,” in *Proceedings of the 13th International Workshop on Parallel and Distibuted Real-Time Systems*. Denver, Colorado: IEEE, Apr. 2005.

[8] J. Trowitzsch, “Quantitative Evaluation of UML State Machines Using Stochastic Petri Nets,” Ph.D. dissertation, TU Berlin, Oct. 2007.

[9] A. Zimmermann and G. Hommel, “Modelling and Evaluation of Manufacturing Systems Using Dedicated Petri Nets,” *The International Journal of Advanced Manufacturing Technology*, vol. 15, pp. 132–137, 1999.

[10] T. Arpinen, E. Salminen, T. Hämäläinen, and M. Hännikäinen, “Extension to MARTE Profile for Modeling Dynamic Power Management of Embedded Systems,” in *Proceedings of the 1st Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2010)*, Mar. 2010.

[11] G. R. d. A. Callou, P. R. M. Maciel, E. C. de Andrade, B. C. e. S. Nogueira, and E. A. G. a. Tavares, “A Coloured Petri Net Based Approach for Estimating Execution Time and Energy Consumption in Embedded Systems,” in *Proceedings of the 21st Annual Symposium on Integrated Circuits and System Design*, ser. SBCCI ’08. New York, NY, USA: ACM, 2008, pp. 134–139.

[12] E. Andrade, P. Maciel, T. Falcão, B. Nogueira, C. Araujo, and G. Callou, “Performance and energy consumption estimation for commercial off-the-shelf component system design,” *Innovations in Systems and Software Engineering*, vol. 6, no. 1-2, pp. 107–114, 2009.

[13] K. Jensen, K. L. Kristensen, and L. Wells, “Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems,” *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 9, no. 3–4, pp. 213–254, 2007.

[14] M. Hagner, A. Aniculaesei, and U. Goltz, “UML-Based Analysis of Power Consumption for Real-Time Embedded Systems,” in *Proceedings of the 8th IEEE International Conference on Embedded Software and Systems (IEEE ICESS-11)*, Changsha, China, 2011, pp. 1196–1201.

[15] J. Merseguer and J. Campos, “Software Performance Modeling Using UML and Petri Nets,” in *MASCOTS Tutorials*, ser. Lecture Notes in Computer Science, M. Calzarossa and E. Gelenbe, Eds., vol. 2965. Springer, 2003, pp. 265–289.

[16] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with Generalized Stochastic Petri Nets*, ser. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[17] J. Campos and J. Merseguer, “On the Integration of UML and Petri Nets in Software Development,” in *ICATPN*, ser. Lecture Notes in Computer Science, S. Donatelli and P. S. Thiagarajan, Eds., vol. 4024. Springer, 2006, pp. 19–36.

[18] D. Perez-Palacin, R. Mirandola, and J. Merseguer, “QoS and energy management with Petri nets: a self-adaptive framework,” *Journal of Systems and Software*, vol. 85, no. 12, pp. 2796–2811, Dec. 2012.

[19] P. King and R. Pooley, “Using UML to Derive Stochastic Petri Net Models,” in *Proceedings of the 15th UK Performance Engineering Workshop*, 1999, pp. 45–56.

[20] C. Lindemann, A. Thümmler, A. Klemm, M. Lohmann, and O. P. Waldhorst, “Performance Analysis of Time-enhanced UML Diagrams Based on Stochastic Processes,” in *Proceedings of the 3rd Workshop on Software and Performance (WOSP)*, Rome, Italy, 2002, pp. 25–34.

[21] W. H. Sanders and J. F. Meyer, "A Unified Approach for Specifying Measures of Performance, Dependability, and Performability," in *Dependable Computing for Critical Applications*, ser. Dependable Computing and Fault-Tolerant Systems, A. Avizienis and J. Laprie, Eds. Springer Verlag, 1991, vol. 4, pp. 215–237.

[22] A. Zimmermann, *Stochastic Discrete Event Systems – Modeling, Evaluation, Applications*. Springer, Berlin Heidelberg New York, 2007.

[23] ——, "Modeling and Evaluation of Stochastic Petri Nets with TimeNET 4.1," in *Proceedings of the 6th International Conference on Performance Evaluation Methodologies and Tools (ValueTools)*. IEEE, 2012, pp. 54–63.

[24] A. Zimmermann and M. Knoke, "TimeNET 4.0: A Software Tool for the Performability Evaluation with Stochastic and Colored Petri Nets," TU Berlin, User Manual, Aug. 2007. [Online]. Available: http://www2.tu-ilmenau.de/sse_file/TimeNET/Documentation/TimeNET-UserManual40.pdf