# Transforming UML State Machines into Stochastic Petri Nets for Energy Consumption Estimation of Embedded Systems

Dmitriy Shorin and Armin Zimmermann
*Ilmenau University of Technology*
*System & Software Engineering*
*P.O. Box 100 565, D-98684 Ilmenau, Germany*
*Dmitriy.Shorin@tu-ilmenau.de; Armin.Zimmermann@tu-ilmenau.de*

Paulo Maciel
*Federal University of Pernambuco*
*Center for Informatics*
*50740-540 Recife-PE, Brazil*
*prmm@cin.ufpe.br*

***Abstract*—This paper presents an ongoing effort towards a methodology for the model-based engineering of energy-efficient automation systems. Energy consumption is an increasingly important decision criterion, which has to be included in the search for good architectural and design alternatives. In the paper, a modelling and performance evaluation technique is proposed, which describes an embedded system with an operational model of the processor hardware and an application model of the software. UML extended with MARTE profile elements is used for this part. Both models are transformed into a stochastic Petri net (SPN), for which we give transformation rules. It is then possible to predict the energy consumption of the hardware / software system by a standard evaluation of the Petri net. An example is provided.**

***Keywords*-UML; MARTE; Petri nets; energy consumption; energy efficiency;**

## I. Introduction

Energy consumption is an increasingly important non-functional property of embedded systems. Design decisions have to be based on a trade-off between energy and other requirements. Mobile systems which rely on batteries or energy-harvesting techniques are an example for which this is especially true. Specific processors have been developed which allow to be run in several energy-saving modes.

It is important to evaluate 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 system properties equally well, as there are no constructs for non-functional 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 SPT profile [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 propose UML models to be transformed into a model for which analysis algorithms exist and, namely, into SPNs [4], so that the behavior and the properties are preserved. This is an extension of an earlier work, in which extended UML statechart models were transformed into uncolored SPNs and analyzed [5], [6]. A similar approach is taken in [7], where the work mentioned is applied to the energy consumption evaluation. This paper extends our previous work in [8], where it has been shown how the energy consumption of the microcontroller can be estimated. For the modeling of the microcontroller work we also used the MARTE profile of the UML.

Here, we go a step further by explicitly addressing the hardware part of the system, which will be the same for all applications. It is described in an *operational model* and specifies all run modes of a processor (microcontroller), the possible state changes, and their associated energy consumption (as well as transition times, if applicable). This information can be taken from data sheets, and the model has to be constructed only once for a specific CPU.

On the other hand, the effect of the controlling software is captured in an *application model.* 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 program and their duration.

The two models together contain all the necessary information for a prediction of the energy consumption. We propose a method for their combined transformation into a SPN in this paper. The resulting model can then

978-3-901882-46-3 ©2012 IFIP

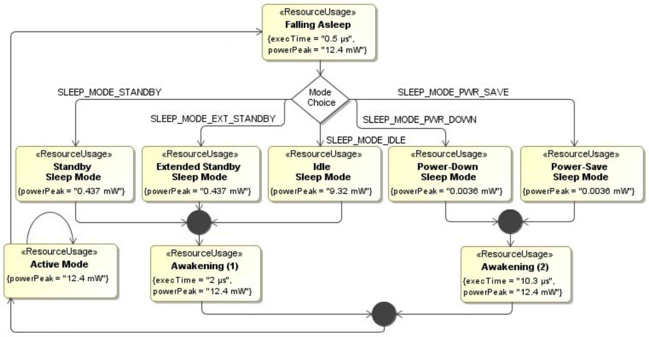


Figure 1: Operational model of the microcontroller

be used to estimate the energy consumption of the system with standard Petri net tools such as TimeNET [9].

The next sections describe each step of the method in detail. Sections II and III describe the operational model and the application model, respectively. The method that transforms UML state machines into SPNs is presented in Sec. IV. The example given adopts the energy-controllable ATMEL microcontroller ATxmega128A1 [10]. Section V concludes the paper.

## II. Operational model

This model contains all the necessary information for analysing the system under consideration. All possible states and transitions of the system should be described. The conditions of possible choices in the model should be specified, and the power required for each state must be provided. State duration is not an obligatory parameter in this model, as it can be given later in the application model. However, if for some states it is a constant value, regardless of a program executed on the system, it should also be introduced. All the state names have to be unique. An operational model representing all the possible states of the microcontroller is given in Fig. 1. The following conditions are implemented: internal oscillator operating on the frequency of $f = 2\ MHz$, supply voltage of $V_{CC} = 3.3\ V$, and ambient air temperature of $T = 24°C$.

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

All possible falling asleep, sleep and awakening modes of the microcontroller are described here. The active mode could also have been divided into several and even into a bulk of states with different values of energy consumption, depending on the estimation accuracy required. However, our research revealed that the power of this state differs in the range of ±5%. We take it as negligible for our aim. Thus, only one active state with a mean power value is being used in the presented operational model.

Without fail, all the other states also contain information about the power required for a specified operating frequency of the microcontroller. Besides, the execution time is already given for two states (falling asleep and both awakening modes) in the operational model (Fig. 1). This is due to the fact that the abovementioned modes always last a certain given time [10].

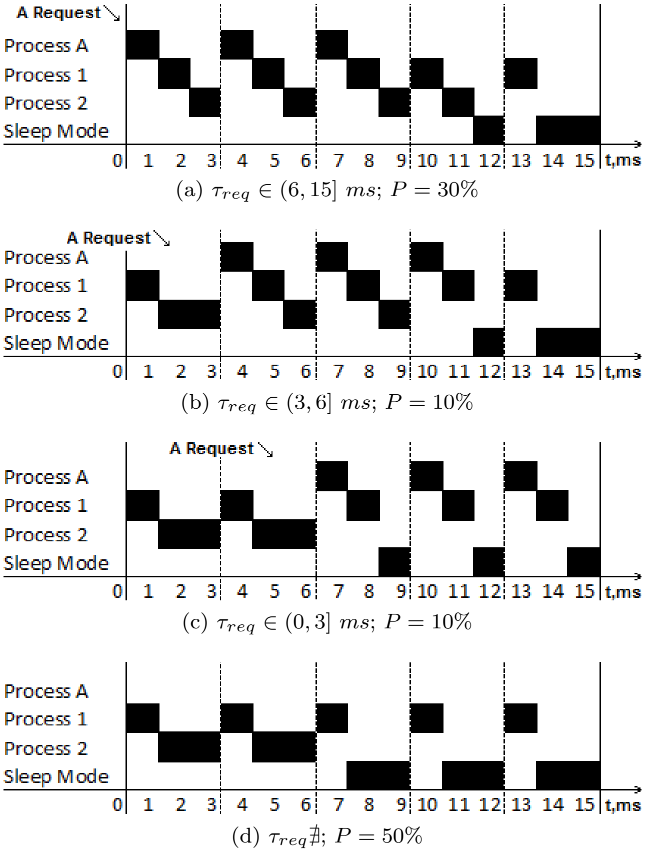
This example also contains a sleep mode choice. According to the command used in the program of the microcontroller, one or another sleep mode can be chosen and therefore, different power is required. Thus, the sleep mode choice is deterministic.

There are some possibilities which might be useful in difficult cases, when a developer does not have enough information available and cannot gain it. Firstly, describing only the states that are indeed in use. The unused modes may be skipped. In this case, the operational model will not represent the real state of the system, but the method will still show its workability. Secondly, power consumption of some states cannot be measured, but still must be given. In this case, this parameter can be roughly estimated. Depending on the aims, one can take the maximum power possible for the respective state, its minimum or average value.

We did not make a thorough investigation of the power consumption for the falling asleep and both awakening modes, which would have been hard to detect. An estimate is used instead. This is an example of a simplification mentioned above.

## III. Application model

At the second stage of the method proposed, an application model has to be created. It contains information about the states used in the specified program and their duration. The order of transitions between the modes must comply with the order specified in the operational model. However, the states by which both the power consumption and the execution time given in the operational model may be skipped in the application model. For the correspondence detection between these two models, the modes implemented in the application model must be named the same as the states of the operational model. They may also include subsidiary signs or numbers. It is supposed that in the application model, the non-specified duration of some states defined in the operational model will be given. If an application contains a choice, probabilities of each alternative should be defined. If not, all the options are considered as



(a) $\tau_{req} \in (6, 15]$ $ms$; $P = 30\%$

(b) $\tau_{req} \in (3, 6]$ $ms$; $P = 10\%$

(c) $\tau_{req} \in (0, 3]$ $ms$; $P = 10\%$

(d) $\tau_{req} \nexists$; $P = 50\%$

Figure 2: Tasks schedule

equiprobable. It is imperative to mark the initial state of the system.

Some works related to the analysis of power consumption of the embedded systems, e.g. [11], concentrate on different types of tasks. In the following example application, the microcontroller has 3 processes to execute. Two of them are periodic and one aperiodic. Process 1 has the duration of 1 $ms$ and is executed every 3 $ms$. Process 2 has the duration of 4 $ms$ and is executed every 15 $ms$. Process A (aperiodic) should be executed for 3 $ms$ within the global period of 15 $ms$. It appears in 50% of cases. The microcontroller falls asleep if there are no more tasks left at the moment.

First, the tasks have to be scheduled to be executed by the microcontroller. There are different methods for this. As we have an aperiodic process, it forces us to use one of the advanced algorithms which can consider aperiodic jobs as well as periodic. As an example for this application, the polling method was chosen. A poller is a periodic task with a polling period and its execution time. The poller is ready for execution periodically and is scheduled together with the periodic tasks in the system according to the given priority-driven algorithm. When it executes, it examines the aperiodic job queue. If the queue is non-empty, the poller executes the job at the head of the queue. The poller suspends its execution or is suspended by the scheduler either when it has executed for the allowed unites of time in the period or when the aperiodic job queue becomes empty. If at the beginning of a polling period the poller finds the aperiodic job queue empty, it suspends immediately and will be able to examine the queue again until the next polling period. [12].

We use a schedule of real-time processes following a rate-monotonic approach as an arbitrary application example for our method. The proposed method incorporating description, transformation, and analysis of energy-consuming embedded systems is, however, not restricted to such simple scheduling setups.

A request for the aperiodic task can appear equiprobable during the global period time. However, according to the method, it could be executed only during the polling time. We specify the poller period of 3 $ms$ and its execution time of 1 $ms$. Thus, the probability of the event that a request for executing the aperiodic process appears is 10% for each of 5 poller periods of 3 $ms$. However, requests appeared after the sixth microsecond of the period can be executed only in the course of the next global period, because if not, after the sixth microsecond of the period, there is no enough polling time for finishing the aperiodic task.

Thus, using the polling method, the following schedule alternatives were revealed (Fig. 2). As mentioned above, the aperiodic process does not appear in 50% of cases (Fig. 2d). Cases when the aperiodic request appears either in the first or in the second polling period occur each with the probability of 10% (Fig. 2b- 2c). The case when the aperiodic task is executed immediately after the beginning of the global period (Fig. 2a) can occur with the possibility of 30%, as the execution of all requests appeared after the 6th microsecond are moved to the next global period.

The availability of the schedule lets us create the application model in UML (Fig. 3). It is made by means of the *ResourceUsage* (attribute *execTime*) and the *GaStep* (attribute *prob* represents the probability of the step to be executed (for a conditional execution)) stereotypes [2].

The order of the transitions between the modes strictly corresponds to the operational model (Fig. 1). Falling asleep and awakening modes are not reflected in this model, because the necessary information concerning power consumption and execution time is already given in the operational model. The given values of the execution time relate only to the modes by which this parameter was not specified earlier, namely, by the active and sleep modes. It is caused by the fact that these time parameters are actually specified by the programmer

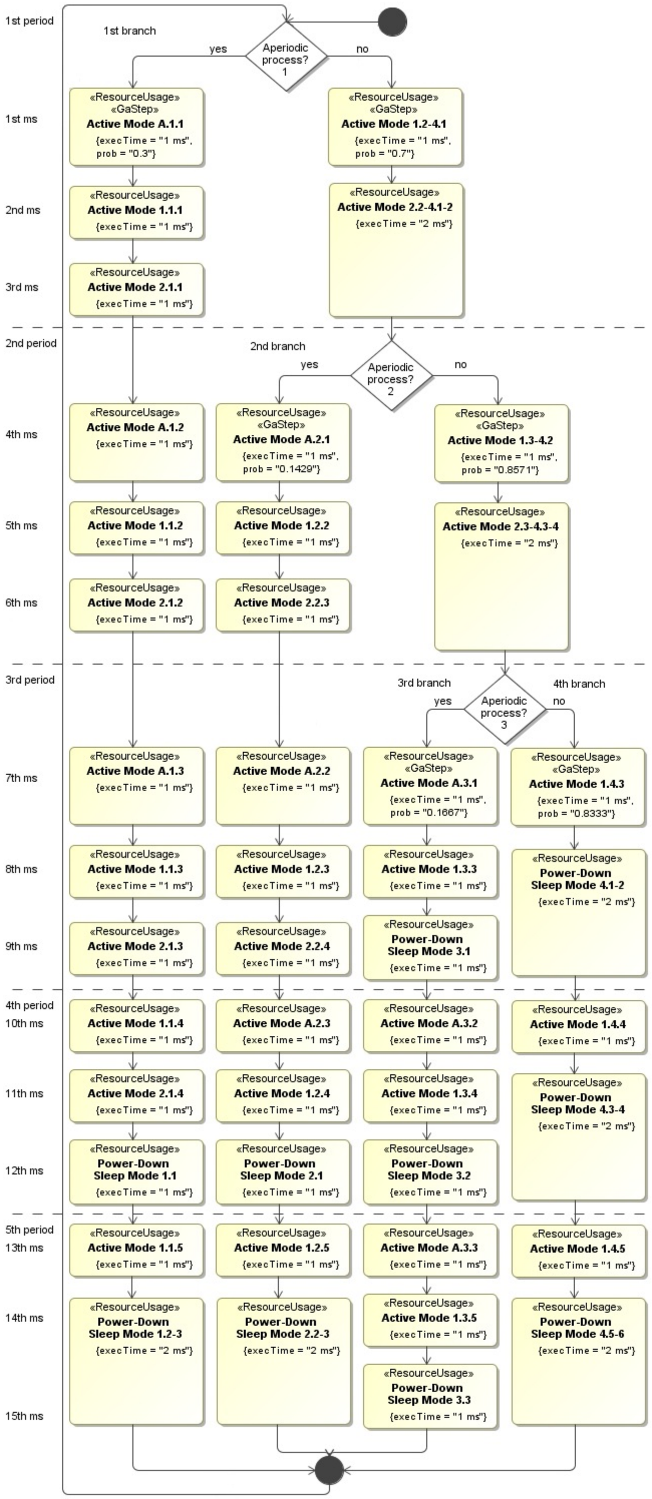

1st period
1st branch
Aperiodic process? 1
yes
no
1st ms
«ResourceUsage» «GaStep» Active Mode A.1.1 {execTime = "1 ms", prob = "0.3"}
«ResourceUsage» «GaStep» Active Mode 1.2-4.1 {execTime = "1 ms", prob = "0.7"}
2nd ms
«ResourceUsage» Active Mode 1.1.1 {execTime = "1 ms"}
«ResourceUsage» Active Mode 2.2-4.1-2 {execTime = "2 ms"}
3rd ms
«ResourceUsage» Active Mode 2.1.1 {execTime = "1 ms"}
2nd period
2nd branch
Aperiodic process? 2
4th ms
«ResourceUsage» Active Mode A.1.2 {execTime = "1 ms"}
«ResourceUsage» «GaStep» Active Mode A.2.1 {execTime = "1 ms", prob = "0.1429"}
«ResourceUsage» «GaStep» Active Mode 1.3-4.2 {execTime = "1 ms", prob = "0.8571"}
5th ms
«ResourceUsage» Active Mode 1.1.2 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.2.2 {execTime = "1 ms"}
«ResourceUsage» Active Mode 2.3-4.3-4 {execTime = "2 ms"}
6th ms
«ResourceUsage» Active Mode 2.1.2 {execTime = "1 ms"}
«ResourceUsage» Active Mode 2.2.3 {execTime = "1 ms"}
3rd period
3rd branch
4th branch
Aperiodic process? 3
7th ms
«ResourceUsage» Active Mode A.1.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode A.2.2 {execTime = "1 ms"}
«ResourceUsage» «GaStep» Active Mode A.3.1 {execTime = "1 ms", prob = "0.1667"}
«ResourceUsage» «GaStep» Active Mode 1.4.3 {execTime = "1 ms", prob = "0.8333"}
8th ms
«ResourceUsage» Active Mode 1.1.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.2.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.3.3 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.1-2 {execTime = "2 ms"}
9th ms
«ResourceUsage» Active Mode 2.1.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode 2.2.4 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 3.1 {execTime = "1 ms"}
4th period
10th ms
«ResourceUsage» Active Mode 1.1.4 {execTime = "1 ms"}
«ResourceUsage» Active Mode A.2.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode A.3.2 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.4.4 {execTime = "1 ms"}
11th ms
«ResourceUsage» Active Mode 2.1.4 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.2.4 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.3.4 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.3-4 {execTime = "2 ms"}
12th ms
«ResourceUsage» Power-Down Sleep Mode 1.1 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 2.1 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 3.2 {execTime = "1 ms"}
5th period
13th ms
«ResourceUsage» Active Mode 1.1.5 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.2.5 {execTime = "1 ms"}
«ResourceUsage» Active Mode A.3.3 {execTime = "1 ms"}
«ResourceUsage» Active Mode 1.4.5 {execTime = "1 ms"}
14th ms
«ResourceUsage» Power-Down Sleep Mode 1.2-3 {execTime = "2 ms"}
«ResourceUsage» Power-Down Sleep Mode 2.2-3 {execTime = "2 ms"}
«ResourceUsage» Active Mode 1.3.5 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.5-6 {execTime = "2 ms"}
15th ms
«ResourceUsage» Power-Down Sleep Mode 3.3 {execTime = "1 ms"}

Figure 3: Application model

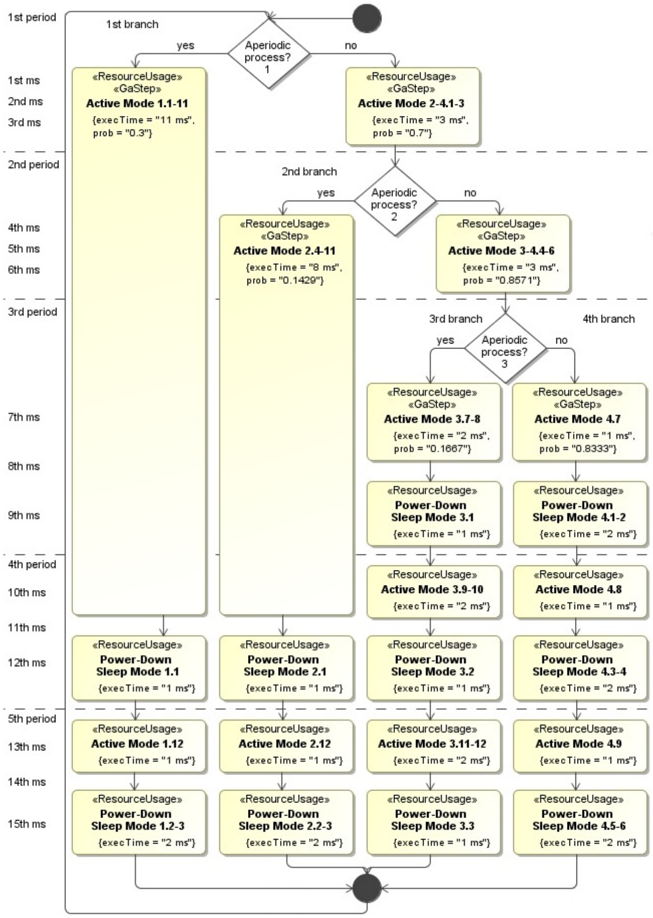

1st period
1st branch
Aperiodic process? 1
yes
no
1st ms
2nd ms
3rd ms
«ResourceUsage» «GaStep» Active Mode 1.1-11 {execTime = "11 ms", prob = "0.3"}
«ResourceUsage» «GaStep» Active Mode 2-4.1-3 {execTime = "3 ms", prob = "0.7"}
2nd period
2nd branch
Aperiodic process? 2
4th ms
5th ms
6th ms
«ResourceUsage» «GaStep» Active Mode 2.4-11 {execTime = "8 ms", prob = "0.1429"}
«ResourceUsage» «GaStep» Active Mode 3-4.4-6 {execTime = "3 ms", prob = "0.8571"}
3rd period
3rd branch
4th branch
Aperiodic process? 3
7th ms
«ResourceUsage» «GaStep» Active Mode 3.7-8 {execTime = "2 ms", prob = "0.1667"}
«ResourceUsage» «GaStep» Active Mode 4.7 {execTime = "1 ms", prob = "0.8333"}
8th ms
9th ms
«ResourceUsage» Power-Down Sleep Mode 3.1 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.1-2 {execTime = "2 ms"}
4th period
10th ms
«ResourceUsage» Active Mode 3.9-10 {execTime = "2 ms"}
«ResourceUsage» Active Mode 4.8 {execTime = "1 ms"}
11th ms
12th ms
«ResourceUsage» Power-Down Sleep Mode 1.1 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 2.1 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 3.2 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.3-4 {execTime = "2 ms"}
5th period
13th ms
«ResourceUsage» Active Mode 1.12 {execTime = "1 ms"}
«ResourceUsage» Active Mode 2.12 {execTime = "1 ms"}
«ResourceUsage» Active Mode 3.11-12 {execTime = "2 ms"}
«ResourceUsage» Active Mode 4.9 {execTime = "1 ms"}
14th ms
15th ms
«ResourceUsage» Power-Down Sleep Mode 1.2-3 {execTime = "2 ms"}
«ResourceUsage» Power-Down Sleep Mode 2.2-3 {execTime = "2 ms"}
«ResourceUsage» Power-Down Sleep Mode 3.3 {execTime = "1 ms"}
«ResourceUsage» Power-Down Sleep Mode 4.5-6 {execTime = "2 ms"}

Figure 4: Simplified application model

and, hence, can be changed in any way. The beginnings of the state names in this state chart agree completely to the existing in the operational model. However, each state name has an extension in form of numbers. It is caused by the fact that the state names in any UML model must be unique if the states are not duplicated.

The following conventional signs were implemented. For the active mode, X.Y.Z means that the process number X (1, 2 or A for aperiodic) will be executed in the branch number Y (see cases of Fig. 2) and this is the microsecond number Z of the respective task in the present global period. For the sleep mode, the parameter X falls away, because the sleep modes already vary by their names.

This model also contains choice elements (*Aperiodic process?*), they are numbed in succession. The states following by these elements also contain information about the probability of executing each of them. In each case, the sum of the choice states equals 1. It should be noted that the indicated probabilities of the states *Active Mode A.2.1* and *Active Mode A.3.1* are not equal, though the probability of the second and the third

Table I: Transformation rules

| # | UML model | Petri net |
|---|---|---|
| 1 | state | place |
| 2 | transition | transition |
| 3 | *execTime="x"*<br>(state attribute) | *delay=x*<br>(exponential transition property) |
| 4 | *powerPeak="x"*<br>(state attribute) | $\ldots + P\{\#Name > 0\} * x + \ldots$<br>(property *expression*,<br>element *measure*) |
| 5 | choice | place & immediate transitions |
| 6 | *prob="x"*<br>(state attribute) | *weight=x*<br>(immediate transition property) |
| 7 | junction | place & immediate transition |
| 8 | initial state | *initialMarking=1*<br>(place property) |

branches are equal (Fig. 2b- 2c). The reason is that in the first case, the probability of 10% is a part of the rest for three branches 70% (10%/70% ≈ 0.1429), while in the second case, the probability of 10% is a part of the rest for two branches 60% (10%/60% ≈ 0.1667).

The application model in Fig. 3 looks complex and can be simplified owing to the following. As it was mentioned in Sec. II, the power consumption of the active mode stays at almost the same level independently of the process executed by the microcontroller. Thus, for the application model, it is significant whether the microcontroller is in the active or sleep mode. All the active modes can be united and presented as one with a longer execution time. In so doing, we cause no inaccuracy in the process of energy consumption estimation. The simplified application model is presented in Fig. 4.

Now, two created models include all the necessary information for estimating the power required for the system. At the same time, the data is not redundant; no data is duplicated. It is also important to notice another advantage of the models division. The application creator may not care about the necessary power. He can even have no information at all about this parameter, but it is still important to know the general appearance of the operational model to be able to create a correct application model.

## IV. Translating UML state machines into SPNs

Two created UML state machines (Fig. 4 and 1) are combined and converted into a Petri net. The application model is taken as the basic one for this operation. The missing information (missing states, power, duration) is taken from the general operational model.

The transformation rules used are listed in Table I.

It is important to notice that while an active mode is really transformed into one place, a sleep mode of the application model in real requires taking into account 3 states: falling asleep, sleep mode and awakening. In the process, the duration time given in the application model

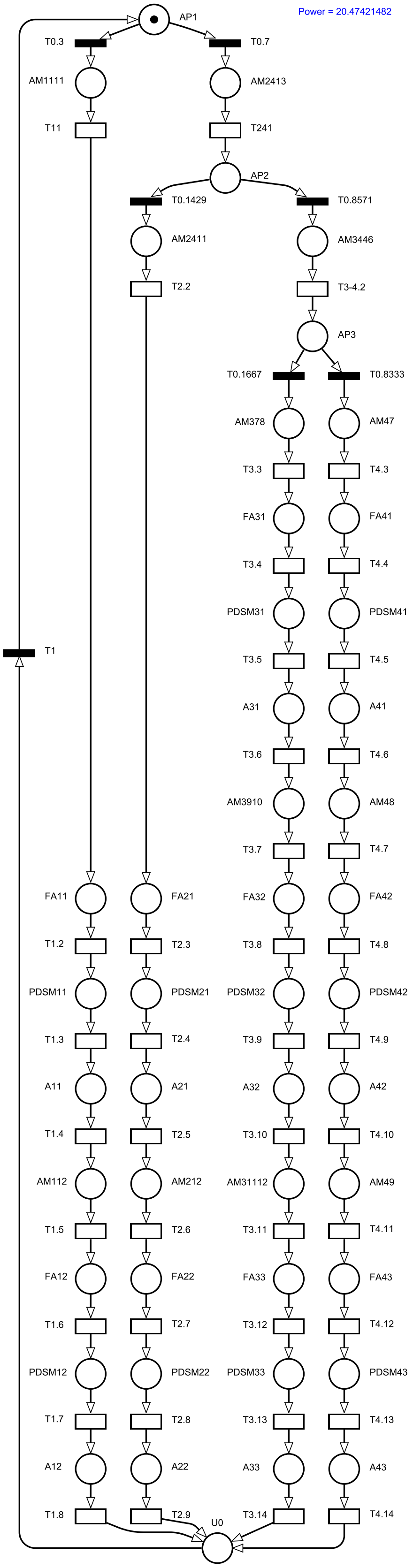


Figure 5: Petri Net reflecting the application execution

must be divided into 3 parts for each state. 0.5 $\mu s$ are allocated for falling asleep mode, 10.3 $\mu s$ for awakening mode. Thus, the sleep mode oneself will last 10.8 $\mu s$ less than given in the application model.

In the process of transformation, the state duration given for user's convenience in seconds in the UML models will be transformed into the delay in clock cycles in the Petri net. By the operating internal oscillator frequency of 2 $MHz$, one clock cycle is equal to 0.5 $\mu s$.

All the other elements used in the UML state machines (text boxes, separators, notes) are not taken into account in the course of transformation.

Figure 5 represents a Petri net created according to the application and operational models given (Fig. 4 and 1) using the above-mentioned transformation rules (Table I).

This example is built in TimeNET [9], a special software tool for the modelling and analysis of stochastic Petri nets with non-exponentially distributed firing times. The calculation of the power occurs automatically in the course of the static analysis. The result (in $mW$) is given in Fig. 5 under the name *Power*. In this case, 20.474 $mW$ is the power needed for executing the application within the global period of 15 $ms$. The energy consumption of the microcontroller after a certain time could be determined as the power needed for a global period multiplied by the number of global periods.

The instance given in this paper should be considered as a simplified example for describing the possibilities of the method proposed. In actual fact, more complex systems must be analysed. Increase of the states number will lead to the growth of the scheduling options quantity and thus, to the expansion of the models. The influence of this demerit can be considerably reduced by implementing the scheduling policies into Petri Nets. Thus, it will not be necessary to make a preliminary scheduling like in Fig. 2. This effort will be taken next.

## V. Conclusion

This paper presented a methodology for the model-based estimation of energy consumption for embedded systems. The UML language extended with the MARTE profile is used for the modeling process. The main contribution is to describe the overall processor behavior with an independent general operational model, while the software applications are specified in the second (application) model referencing the first one. The two models are converted into a stochastic Petri net, which is then used for a performance evaluation. Transformation rules are given, and an example is presented. Finally, the design process for embedded systems can be supported by predicting the energy consumption.

## Acknowledgment

The authors would like to thank the German Academic Exchange Service (DAAD) for the financial support of the project.

## References

[1] Object Management Group (OMG). (2011, Aug.) OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.3. [Online]. Available: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/

[2] ——. (2011, Jun.) UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems, Version 1.0. [Online]. Available: http://www.omg.org/spec/MARTE/1.1/PDF

[3] ——. (2005, Jan.) UML Profile for Schedulability, Performance, and Time Specification, Version 1.1.

[4] A. Zimmermann, *Stochastic Discrete Event Systems - Modeling, Evaluation, Applications.* Springer, Oct. 2007.

[5] J. Trowitzsch and A. Zimmermann, "Towards quantitative analysis of real-time UML using stochastic Petri nets," in *Proc. 13th Int. Workshop on Parallel and Distibuted Real-Time Systems.* Denver, Colorado: IEEE, Apr. 2005.

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

[7] 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.

[8] D. Shorin and A. Zimmermann, "Model-based development of energy-efficient automation systems," in *RTAS 2011 - The 17th IEEE Real-Time and Embedded Technology and Applications Symposium. Work-In-Progress Proceedings*, Chicago, IL, Apr. 11–14, 2011, pp. 37–40.

[9] 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.

[10] *ATxmega64A1/128A1/192A1/256A1/384A1 Preliminary*, Atmel Corporation, Sep. 2010. [Online]. Available: http://www.atmel.com/Images/doc8067.pdf

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

[12] J. W. S. Liu, *Real-time systems.* Prentice Hall, 2000.