An Approach Based in Petri Net for Requirement Analysis

Ermeson Andrade, Paulo Maciel, Gustavo Callou, Bruno Nogueira and Carlos Araujo
Federal University of Pernambuco (UFPE)
Brazil

1. Introduction

Embedded systems that have timing constraints are classified as real-time systems. In these systems, not only the logical results of computations are important, but also the time instant in which they are obtained. Hard real-time systems are those whose the respective timing constraints must be met at all cost, since violation might be catastrophic. Hence, time predictability is an essential issue (Barreto & Lima (2004)). In addition, the widespread expansion of mobile devices market has forced embedded systems companies to deal with several new challenges in order to provide complex systems in this market niche. In this context, energy consumption deserves special attention, since portable devices generally rely on constrained energy sources (e.g. battery) . As consequence, early estimation of the energy consumption can provide important insights to the designer about the battery lifetime as well as parts of the application that need optimization (Tavares et al. (2007)).
Nowadays, UML (UML (2005)) is the most adopted modeling language for system design in the software engineering organizations and industry. The main reasons are: (i) its friendly and intuitive notations, (ii) availability of commercial and open source tools that support the UML notations and (iii) autonomy of particular programming languages and development processes. However, UML does not provide support for quantitative notations. Quantitative notations are especially important when modeling Embedded Real-Time Systems (ERTS). Hence, we consider UML in combination with MARTE (UML Profile for Modeling and Analysis of Real-Time and Embedded systems) as specification language for the design of ERTS. MARTE foster the construction of models that may be used to make quantitative predictions regarding real-time and embedded features of systems taking into account both hardware and software characteristics (MARTE (2005)).
UML 2.0 is composed of several diagram types (e.g. activity, sequence, use case, class, timing and many others). Interaction Overview Diagram (IO) (UML (2005)) is adopted in this work due to its suitable characteristics for modeling requirements when dealing with ERTS, since UML-IO combine elements of activity diagrams with sequence diagrams to represent the embedded system behavior.
Without loss of generalization, this work aims to depict the mapping process of UML-IO into a Time Petri Net with Energy constraints (ETPN) in order to estimate the energy consumption and execution time of ERTS. These estimates are performed in the early stages of the embedded system life cycle, serving as one instrument for design decision-making process. First, the

execution time and energy consumption constraints are represented as MARTE profile annotations. After that, the ETPN model is generated by a mapping process, and finally the model is evaluated in order to find the Best Case Execution Time (BCET) and Worst Case Execution Time (WCET), the respective energy consumption, and also, adopted for qualitative analysis and verifications. Furthermore, the estimates obtained (time and energy consumption) from the model were compared with the measures obtained from the real hardware platform.
The remainder of the paper is organized as follows: Section 2 presents the adopted methodology. Section 3 introduces basic concepts regarding UML-IO, MARTE, ETPN and measuring activities. Section 4 presents the related work. Section 5 describes the mapping from UML-IO to an ETPN. Section 6 presents a case study and results. Section 7 concludes the paper and briefly discusses further work.

2. Methodology

This section introduces the set of activities related to the proposed design methodology. Figure 1 depicts the core activities of MEMBROS methodology, which organizes the activities in three groups: (i) requirement analysis; (ii) energy consumption and performance evaluation; and (iii) software synthesis. As follows, an overview of entire methodology is provided. However, this paper focuses on the activities regarding requirement analysis.

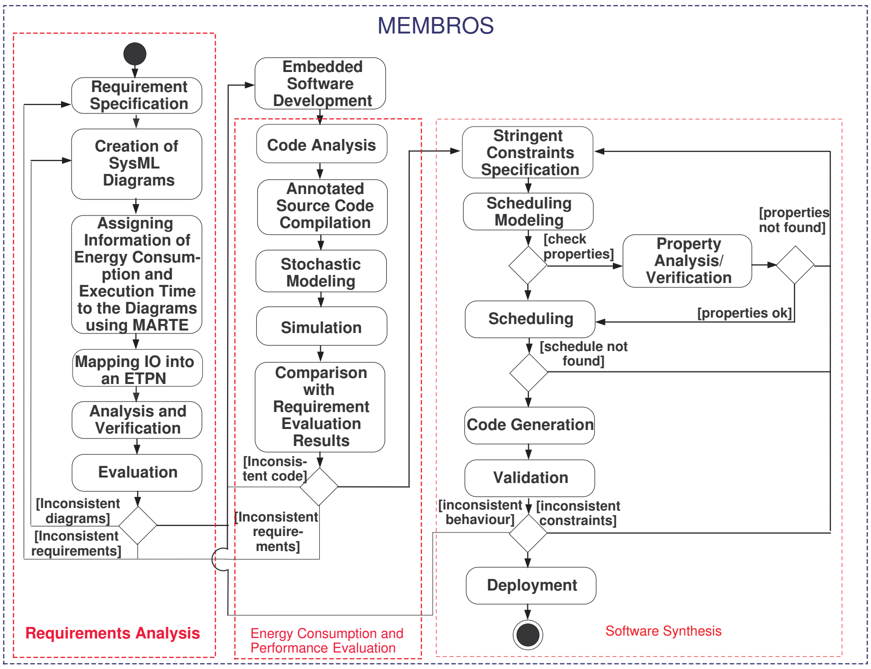
MEMBROS
Requirement Specification
Creation of SysML Diagrams
Assigning Information of Energy Consumption and Execution Time to the Diagrams using MARTE
Mapping IO into an ETPN
Analysis and Verification
Evaluation
[Inconsistent diagrams]
[Inconsistent requirements]
Requirements Analysis
Embedded Software Development
Code Analysis
Annotated Source Code Compilation
Stochastic Modeling
Simulation
Comparison with Requirement Evaluation Results
[Inconsistent code]
[Inconsistent requirements]
Energy Consumption and Performance Evaluation
Stringent Constraints Specification
Scheduling Modeling
[check properties]
Property Analysis/ Verification
[properties not found]
[properties ok]
Scheduling
[schedule not found]
Code Generation
Validation
[inconsistent behaviour]
[inconsistent constraints]
Deployment
Software Synthesis

Fig. 1. MEMBROS methodology.

Initially, the activities regarding requirements specifications are performed. After created the requirement specification, then the system requirements are modeled using a set of UML-IOs, which represent the dynamic parts of the embedded software to be developed. Since timing and energy constraints are of utmost importance in the systems of interest, the UML-IOs are annotated with timing and energy consumption information (e.g. initial estimates) using MARTE. Next, the annotated UML-IOs are automatically mapped into ETPN models in order to lay down a mathematical basis for analysis and verification of properties (e.g. absence of deadlock conditions between requirements). This activity also concerns to obtain best and worst case execution times and the respective energy consumptions, in such a way that the requirements are also evaluated in order to ensure that the timing and energy constraints can be met. As the UML-IOs are constructed by the designer, negative results in the evaluation activity may be not only related to inconsistent requirements, but also to inconsistent UML-IOs.

Afterwards, the embedded software is implemented taking into account the results obtained in previous activities. Once the source code implementation is concluded, the designer analyzes the code in order to assign probability values to conditional and iterative structures. The probability annotations allow the compiled code be evaluated in the context of time and energy consumption, in such a way that these costs may be estimated before running the code on the hardware platform. Next, the compiled code is automatically translated into a coloured Petri net (CPN) model in order to provide a basis for the stochastic simulation of the embedded software. Although not depicted in Figure 1, an architecture characterization activity is also considered to permit the construction of a library of CPN basic building blocks, which provide the foundation for the automatic generation of CPN stochastic models. From the CPN model (generated by the composition of basic blocks), a stochastic simulation of the compiled code is carried out considering the characteristics of the target platform. If the simulation results are in agreement with the requirements, the software synthesis is performed. More information about the activity of Performance Evaluation can be found in Callou et al. (2008).

Software synthesis activities are concerned with the stringent constraints (e.g. time and energy), and, in the general sense, it is composed of two subgroups of activities: (i) tasks' handling; and (ii) code generation. Tasks' handling is responsible for tasks' scheduling, resource management, and inter-task communication, whereas code generation deals with the static generation of the final source code, which includes a customized runtime support, namely, dispatcher. It is important to state that the concept of task is similar to *process*, in the sense that it is a concurrent unit activated during system runtime. For the following activities, it is assumed that the embedded software has been implemented as a set of concurrent hard real-time tasks.

Initially, a measurement activity is performed to obtain the tasks' timing information as well as the information regarding the hardware energy consumption. Next, the designer defines the specification of system stringent constraints, which consists of a set of concurrent tasks with their respective constraints, behavioral descriptions, information related to the hardware platform (e.g. voltage/frequency levels and energy consumption) as well as the energy constraint. Afterward, the specification is translated into an internal model able to represent concurrent activities, timing information, inter-task relations, such as precedence and mutual exclusion, as well as energy constraints. The adopted internal model is a time Petri net extension (TPNE), labeled with energy consumption values and code annotations. After generating the internal model (TPNE), the designer may firstly choose to perform property analysis/verification or carry out the scheduling activity. This work adopts a pre-runtime scheduling approach in order to find out a feasible schedule that satisfies timing, inter-task and energy constraints. Next, the feasible schedule is adopted as an input to the automatic code generation mechanism, such that a tailored code is obtained with the respective runtime control, namely, dispatcher. Finally, the application is validated on a DVS (Dynamic Voltage Scaling) platform in order to check system behaviour as well as the respective constraints. Once the system is validated, it can be deployed to the real environment. More information about the activity of Software Synthesis can be found in Tavares et al. (2007).

3. Background

This section presents fundamental concepts for a better understanding of the rest of the paper.

3.1 UML

The Unified Modeling Language (UML) is a standard widely adopted graphical graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.
The main goals of UML are:

- provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models;
- provide extensibility and specialization mechanisms to extend the core concepts;
- be independent of particular programming languages and development processes;
- provide a formal basis for understanding the modeling language;
- integrate best practices.

UML is composed by 13 types of diagrams, which can be divided into three parts: behavior, interaction and structural diagrams. The behavioral diagrams specify the dynamic parts used in the system being modeled. The interaction diagrams represents a subset of behavior diagrams, that emphasize the control and data flows among the things (sub-systems, components and agents) in the system being modeled. Lastly, structural diagrams define the static and structural elements used in the system being modeled.
Behavioral diagrams are composed by the following diagrams: use case (provides a high-level description of the system functionality), activity (represents the flow of data and control between activities), sequence (represents the interaction between collaborating parts of a system), state machine (describes the state transitions and actions that a system or its parts performs in response to events), communication (shows an interaction among a set of participants over the course of time), interaction overview (combine elements of activity diagrams with sequence diagrams to show the flow of program execution) and timing (shows the behaviors of elements throughout a given period of time) (UML (2005)). It is important to stress that this paper presents the mapping process of UML-IO into an ETPN.

3.1.1 Interaction Overview Diagram

An interaction overview diagram is an activity diagram in which the nodes represent interaction diagrams. Interaction diagrams can include sequence, communication, interaction overview and timing diagrams. Most of the notation for interaction overview diagrams is the same for activity diagrams. For example, initial, final, decision, merge, fork and join nodes are all the same. However, interaction overview diagrams introduce two new elements: interaction occurrences and interaction elements.

Interaction occurrences are references to existing interaction diagrams. An interaction occurrence is shown as a reference frame (Figure 2 (a)). On the other hand, interaction elements display a representation of existing interaction diagrams within a rectangular frame. They differ in that they display the contents of the references diagram inline (Figure 2 (b)).

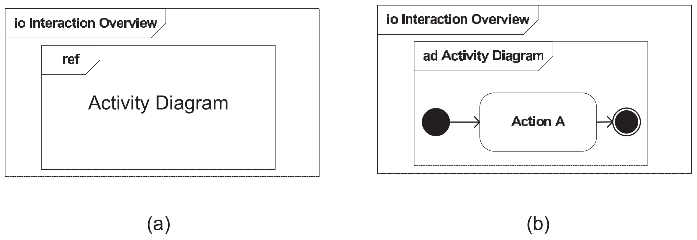

Fig. 2. New Elements: (a) interaction occurrence and (b) interaction element.

3.2 Marte

MARTE is a new UML profile standardized by the OMG. MARTE is used to define foundations for model-based description of real-time and embedded systems. These core concepts are then refined concerning: (i) modeling and (ii) analysis. The modeling part provides support required from specification to detailed design of real-time and embedded systems. On the other hand, the analysis part does not intend to define new techniques for analyzing systems, but to support them. Hence, MARTE aims providing facilities to annotate models with information required to perform specific analysis (MARTE (2005)).

Figure 3 illustrates an example of an activity diagram with time and energy constraints specified by MARTE. For this example the stereotype (*ResourceUsage*) and tagged values (*execTime* and *energy*) were used. The stereotype describes an action. Tagged values consist of a property name and an assigned value. In this example, the *ResourceUsage* stereotype describe, respectively, the delay and the energy consumption of the activity A, in this case, 10 seconds and 20 joules. The tagged values are $\{execTime = (10,'s')\}$ and $\{energy = (20,'j')\}$. More information about all stereotypes and tagged values supported by MARTE can be found in MARTE (2005).

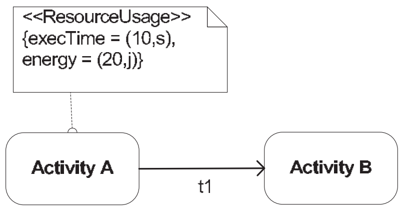

Fig. 3. Activity diagram and MARTE.

3.3 Computation Model

Time Petri Net. A Time Petri Net (TPN) (Merlin & Faber (1976)) is a bipartite directed graph represented by a tuple $\mathcal{P} = (P, T, F, W, m_0, I)$, where P (set of places) and T (set of transitions) are non-empty disjoint sets of nodes. The edges are represented by F, where $F \subseteq A = (P \times$

$T) \cup (T \times P)$. $W : A \to \mathbb{N}$ represents the weight of the edges, such that $W(f) = \{$(i) $x \in \mathbb{N}$, if $(f \in F)$, or (ii) 0, if $(f \notin F)\}$. *A TPN marking m_i is a vector ($m_i \in \mathbb{N}^{|P|}$), and m_0 is the initial marking. $I : T \to \mathbb{N} \times \mathbb{N}$ represents the timing constraints, where $I(t) = [EFT(t), LFT(t)]$ $\forall t \in T$, $EFT(t) \leq LFT(t)$. $EFT(t)$ is the Earliest Firing Time, and $LFT(t)$ is the Latest Firing Time.*
Considering the previous definition, places (P) represent local states and transitions (T) denote local actions. The set of arcs F represents the relationships between places and transitions, in such a way that arcs connect places to transitions and vice-versa. Function W assigns to each arc a natural number, which may be interpreted as the amount of parallel arcs. A marking vector m_i associates to each place a natural number, which represents the number of tokens in the respective place. Graphically, places are represented by circles, transitions are depicted as bars or rectangles, arcs are represented by directed arrows labeled with the weight, and tokens (the marking) are generally represented by filled small circles. Fig. 4 depicts a Petri net model.

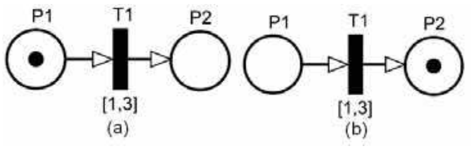

Fig. 4. Petri Net example

Time Petri Net with Energy Consumption - ETPN $\mathcal{P}_{\mathcal{E}}$. An extended TPN with energy consumption values is represented by $\mathcal{P}_{\mathcal{E}} = (\mathcal{P}, \mathcal{E})$. $\mathcal{P}$ is the underlying TPN, and $\mathcal{E}$:$T \to \mathbb{R}_+ \cup \{0\}$ is a function that assigns transitions to energy consumption values.
Enabled Transitions. A set of enabled transitions, at marking m_i, is denoted by: $ET(m_i) = \{t \in T \mid m_i(p_j) \geq W(p_j, t), \forall p_j \in P\}$
A transition $t \in T$ is *enabled*, if each input place $p \in P$ contains at least $W(p, t)$ tokens. The time elapsed, since the respective transition enabling, is denoted by a clock vector $c \in (\mathbb{N} \cup \{\#\})^{|T|}$, where # represents the undefined value for not enabled transitions. As an example, the clock vector for the net in Fig. 4(a) contains one element: $c(t_1) = 0$. At this point, the difference between static and dynamic firing intervals associated with transitions is required. The dynamic firing interval of transition t, $I_D(t) = [DLB(t), DUB(t)]$, is dynamically modified whenever the respective clock variable $c(t)$ is incremented, and t does not fire. $DLB(t)$ is the Dynamic Lower Bound, and $DUB(t)$ is the Dynamic Upper Bound. The dynamic firing interval is computed in the following way: $I_D(t) = [DLB(t), DUB(t)]$, where $DLB(t) = max(0, EFT(t) - c(t))$, $DUB(t) = LFT(t) - c(t)$. Whenever $DLB(t) = 0$, t can fire, and, when $DUB(t) = 0$, t must fire, since *strong firing mode* is adopted.
States. Let $\mathcal{P}_{\mathcal{E}}$ be a time Petri net extended with energy consumption values, $M \subseteq \mathbb{N}^{|P|}$ be the set of reachable markings (e.g. all possible markings) of $\mathcal{P}_{\mathcal{E}}$, $C \subseteq (\mathbb{N} \cup \{\#\})^{|T|}$ be the set of clock vectors, and $E \subseteq \mathbb{R}_+ \cup \{0\}$ be the set of accumulated energy consumptions. The set of states S of $\mathcal{P}_{\mathcal{E}}$ is given by $S \subseteq (M \times C \times E)$, that is, a state is defined by a marking, the respective clock vector, and the accumulated energy consumption from the initial state up to this state.
Considering the Petri net model in Fig. 4(a), the initial state is $s_0 = (m_0 = [1, 0], c_0 = [0], e_0 = 0)$.
Firing Domain. The *firing domain* for a transition t at state s, is defined by the interval: $FD_s(t) = [DLB(t), \min(DUB(t_k))], \forall t_k \in ET(m)$.

Without loss of generality, enabled transitions are only related to the marking, and firable transitions take into account the marking, and their respective clock values (the time elapsed of each enabled transition). Considering firing domain, a firable transition t at state s can only fire in the interval denoted by $FD_s(t)$. In Fig. 4(a), at the initial state $s_0 = (m_0 = [1,0], c_0 = [0], e_0 = 0)$, t_1 is firable when $c_0(t_1) = 1$ and must fire when $c_0(t_1) = 3$ ($FD_{s_0}(t_1) = [1,3]$), if it neither has been fired nor disabled.

3.4 Measurement

This section briefly describes the measurement process and also the software that automates the measuring on the target plataform. In order to obtain the energy consumption and execution time values of a microcontroller instruction set, it may be necessary to adopt some measurement techniques in case such values cannot be obtained from manuals and datasheets.
The measurement scheme is presented in Figure 5, in which a hardware platform with the LPC2106 microcontroller, an oscilloscope and a desktop computer (PC) are connected. The AMALGHMA tool (Tavares & Maciel (2006)) - Advanced Measurement Algorithms for Hardware Architectures - has been implemented for automating the measuring activities. AMALGHMA adopts a set of statistical methods, such as bootstrap and parametric methods, which are important in the measurement process due to several factors, for instance: (i) oscilloscope resolution; and (ii) resistor error. Besides, this tool has been validated considering LPC2106 datasheet as well as ARM7TDMI-S reference manual.

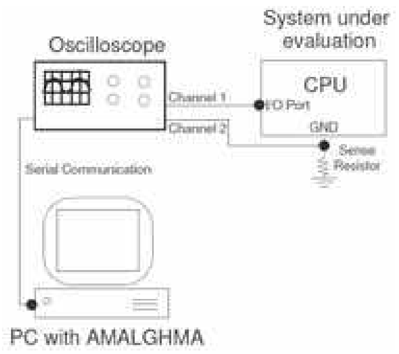

Fig. 5. Measurement Scheme.

4. Related Works

Many works are available in the literature reporting mappings of semi-formal representations to formal models. The majority of these works focus on qualitative analysis. Some of them are devoted to performance modeling, but, to the best of our present knowledge, none of them focus on both energy consumption and time evaluation of systems, besides of the papers published by our research team. It is also important to highlight that none work proposes the mapping of UML-IO into a formal model.
Merseguer et al. (2002) presented a systematic derivation of the UML-SM in fragments of a labeled Generalized Stochastic Petri Net (GSPN) that are composed into a single model that represents the behavior of the entire diagram. This work focused on the software performance evaluation in which all delays are represented by exponential distributions.

Trowitzsch & Zimmermann (2005) aimed at the transformation of UML-SM into a Stochastic Petri Nets (SPNs) for the performance evaluation of real-time systems. This approach proposed the decomposition of an UML-SM into basic elements, like states and transitions. These elements are translated into the corresponding SPN representations. The UML Profile for Schedulability Performance and Time (SPT) was used as a specification language for representing the restrictions imposed on the real-time systems. Amorim et al. (2005) proposed an approach to map Live Sequence Chart (LSC) language to an equivalent Coloured Petri Net (CPN) for analysis and verification of embedded systems' properties.
Another approach Lee et al. (2000) aimed to verify scenarios with Time Petri Net. This approach proposed a mechanism to check the acquired scenarios by indication any missing information or wrong information hidden in these scenarios. In Elkoutbi et al. (2002), the authors presented a requirement engineering process for real-time systems that aims a formal specification of the system using Timed Petri Net. In this approach, the first activity is the elaboration of the use case. Next, a corresponding scenario in form of sequence diagram is created for each use case. Afterwards, the scenario specification is derived into a Timed Petri Net. Finally, the partial TPNs are merged to obtain an integrated TPN, and then some verifications are performed.

5. Mapping UML-IO into an ETPN

This section describes how to derive the elements of UML-IO including time and energy constraints into an ETPN. The method consists of the deriving UML-IO basic elements and MARTE annotations into an ETPN. After that, all ETPN representation are composed into a single model that represents the behavior of the entire diagram. This section, firstly, presents the mapping of the elements used in this work that compose the nodes, that is, the interaction diagrams (activity and sequence). Lastly, an example illustrates the mapping of an UML-IO. Due lack of space, this paper focuses on the formalism and annotations needed to understand the case study.

5.1 Activity Diagrams

The Activity Diagrams (AD) are used to model the dynamic system aspects.

5.1.1 Mapping Activities

The activities are represented by a rectangle with rounded edges (see Figure 6 (a)). They depict the invocation of an operation that may be physical or electronic. Additionally, an action represents a single step within an activity. In the ETPN model generated by the mapping process, the activities are represented by two PN-transitions. These PN-transitions may have assigned a time interval in which the activity must be executed, that is, the maximum and minimum time bounds. One PN-transition is used to represents the execution time for the worst case, and the other PN-transition for the best case of the activity. Energy constraints can be also assigned to these PN-transitions.
Figure 6 (a) depicts the mapping of an activity into an ETPN. The *t_ex_W_A* and *t_ex_B_A* PN-transitions model the worst case and best case, respectively. The time interval for these transitions are [45,45] and [5,5] , that is, the maximum and minimum times spent by activity A. Likewise, the energy consumption for the worst and best cases assigned to these PN-transitions are [70,70] and [18,18], respectively.
Additionally, the *t_in_W_A* and *t_in_B_A* PN-transitions are used due to semantics of TPN (Strong Firing Semantics) (Merlin & Faber (1976)) in order to allow PN-transitions with longer

delay to be able to fire, hence the time interval equal to [0,0] is assigned to these PN-transitions. However, if the constraints (execution time and energy consumption) are omitted from activity *A*, then they will be mapped into a PN-transition whose maximum and minimum times are zero (Figure 6 (b)). Furthermore, the *t_ex_W_A* and *t_ex_B_A* PN-transitions (see Figure 6 (a)) are assigned with a thin interval (interval in which the upper and lower bound values are the same (David (2005))), because the state space is considerably smaller than if only one PN-transition with the interval equal to [5,45] was adopted. Hence, this model allows a faster reachability graph path search. The *in_A* place represents the activity *A* entry as well as the choice point between the worst and best related to the execution time and energy consumption of the activity *A*. The others places (*W_A*, *B_A* and *out_A*) represent, respectively, the worst case state, best case state and activity *A* exit.

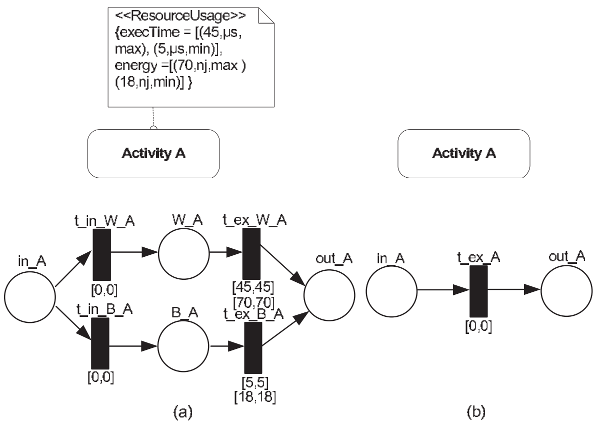

Fig. 6. Mapping activities.

5.1.2 Mapping Transitions

In UML-AD, transitions represent cause/effect activity relations. Figure 7 (a) illustrates the mapping of an AD-transition with time and energy constraints. The AD-transition (*t1*) from A to B is mapped in two PN-transitions (*t_ex_W_t1* and *t_ex_B_t1*), in which a time period of [25, 25] and [15, 15] seconds are assigned. These PN-transitions represent the execution time for the worst and best case, respectively. Figure 7 (a) also has energy constraints assigned to these PN-transitions.

AD-transitions that do not spend time and energy are mapped into PN-transitions whose maximum and minimum times are zero (Figure 7 (b)). The PN-transition generated is connected to the *out_A* and *in_B* places. These places represent, respectively, the activity *A* exit as well as the choice point between the worst and best related to the execution time and energy consumption of the *t1* AD-transition and the activity *B* entry.

5.1.3 Mapping Initial and Final States

An initial state represents the start point of a UML-AD. This initial state is mapped into a place in ETPN model, where the *staIni_A* place gets the initial marking of one token. Tokens are used in these models to simulate the dynamic behavior of systems. Furthermore, the

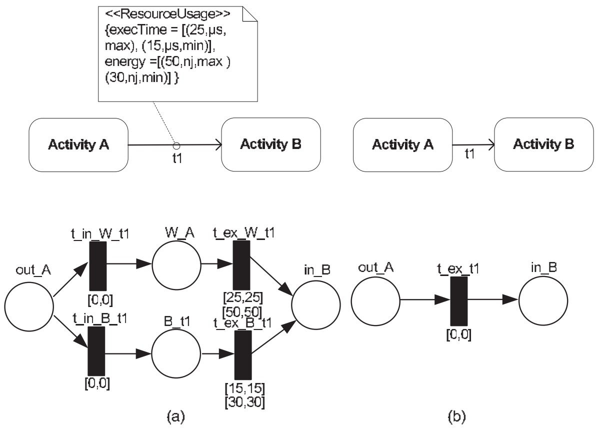

Fig. 7. Mapping transitions.

t_in_A PN-transition is used to represent the AD-transition between the initial state and the activity *A*. Figure 8 (a) illustrates a mapping of an initial state.
A final state represents the respective final state in UML-AD. This final state is mapped into a place in ETPN model (Figure 8 (b)), such that the presence of a token in the *endSta_A* place represents the end of the UML-AD. Moreover, the *t_end_A* PN-transition is used to represent the AD-transition between the activity *A* and the final state.

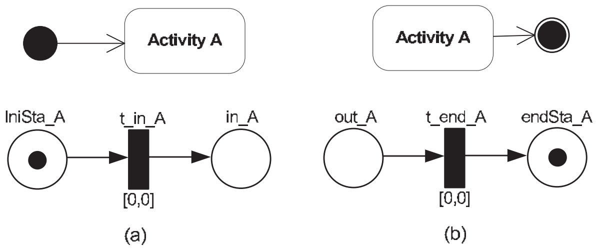

Fig. 8. Mapping initial and final states.

5.1.4 Synchronization Bar

Synchronization bars are used to split processing (fork), or to resume processing when multiple activities have been completed (join). The synchronization bars are modeled as solid rectangles, with multiple transitions going in and/or out.
Figure 9 presents an example, in which the activity A is split into 2 activities (B and C). In the mapping process, the *t_fork_A_B* PN-transition represents the split processing and the *brc_B* and *brc_C* places represent the starting points for the branches (see Figure 9). On the other hand, Figure 11 depicts the synchronization process (join), that is, the synchronization between the activities A and B. In the mapping process, the *t_join_A_B* PN-transition is used to represent the synchronization process and the *join_A_B* place represents the synchronization point. In this case, the PN-transition only can be fired if both *out_A* and *out_B* places contain a token.

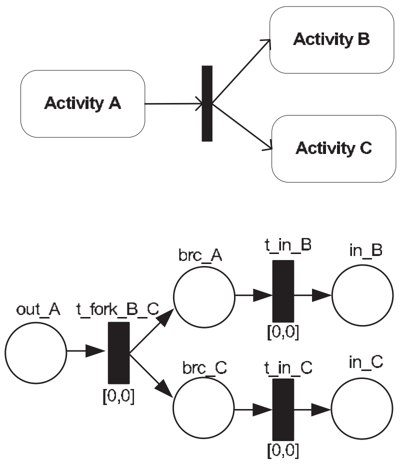

Fig. 9. Mapping fork.

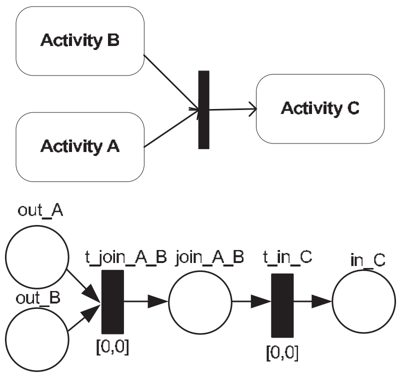

Fig. 10. Mapping join.

5.1.5 Decision

In UML-AD, decision is represented by a diamond with one flow entering and several leaving. The flows leaving include conditions although some modelers may not indicate the conditions if it is obvious. In the mapping process, the *t_in_deci* PN-transition represents the entry on the decision and the places *brc_B* represents the decision point.

5.2 Sequence Diagram

Sequence Diagrams (SD) are commonly used UML diagram for representing element's collaboration over time.

5.2.1 Mapping Lifeline and Message

A lifeline represents the involvement of a given participant in a particular interaction. The white rectangles on a lifeline (vertical dashed lines) are called activations (see Figure 12) and

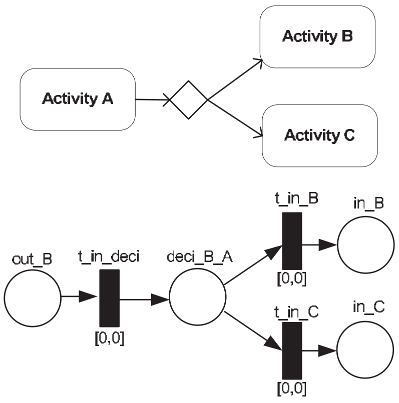

Fig. 11. Mapping decision.

indicate a participant response to a message. The communication between lifelines is performed by messages or calls, in the same order in which the events occur. Furthermore, the message specifies not only the communication type, but also the sender and the receiver. The return messages are an optional part of an UML-SD. In this paper, only asynchronous messages are considered. Figure 12 depicts an example, in which two participants (A and B) are communicating by two messages. The first message has time and energy constraints assigned. These constraints are specified by MARTE profile. The other message has neither time nor energy constraint assigned.

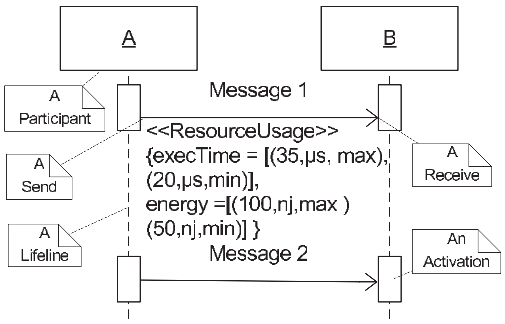

Fig. 12. UML-SD example.

Figure 13 presents the mapping of the UML-SD depicted in Figure 12 into an ETPN. The places (*start_A*, *D_1* and *end_A*) and the PN-transitions (*t_s_M1* and *t_s_M2*) represent the participant *A* lifetime. In this model, each place represents one state of that participant at a particular time instant, and the PN-transitions represent a state transition. The *start_A* place represents the start point on the lifeline *A*, and this place gets the initial marking of one token. The place *D_1* is a dummy place that interfaces the PN-transitions and the place *end_A* represents the lifeline end. Finally, *t_s_M1* and *t_s_M2* represent the message transmissions. In a similar manner, the participant *B* was mapped. The difference, in this case, is that the PN-transitions (*t_r_M1* and *t_r_M2*) represent the message receptions.

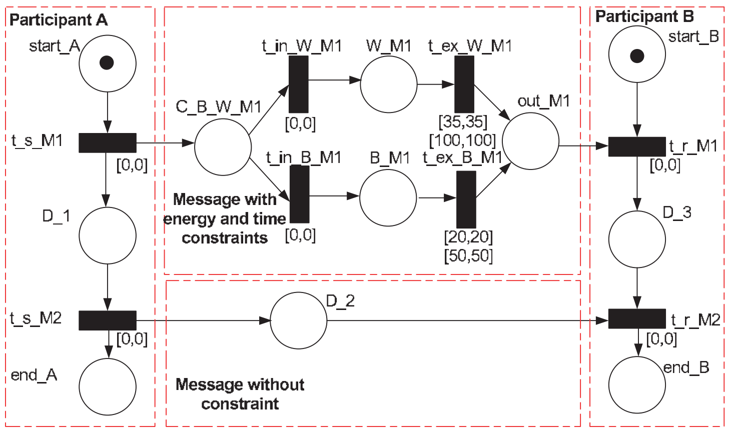

Fig. 13. Mapping UML-SD example.

Figure 13 also shows the mapping of messages. For example, *Message 1* represents the thin intervals of the execution time for the worst and best case, respectively, [35,35] and [20,20] assigned to the *t_ex_W_M1* and *t_ex_B_M1* PN-transitions. In a similar manner, the energy consumption constraints are considered. The energy consumption for the worst and best cases assigned are [100,100] and [50,50], respectively. The *C_B_W_M1* place represents the choice point between the worst and best case related to the execution time and energy consumption of the *Message 1* as well as the *Message 1* entry. The *W_M1*, *B_M1* and *out_M1* places represent, respectively, the worst case state, the best case state and the *Message 1* out.
However, if time and energy constraints are omitted from the messages as the *Message 2* shows (see Figure 12), then the mappings is performed as depicted in Figure 13, in which the place *D_2* is used as a dummy place for bridging the *t_s_M2* and *t_r_M2* PN-transitions.

5.2.2 Mapping self message

Figure 14 presents an example of self message, that is, the participant sends a message to himself.

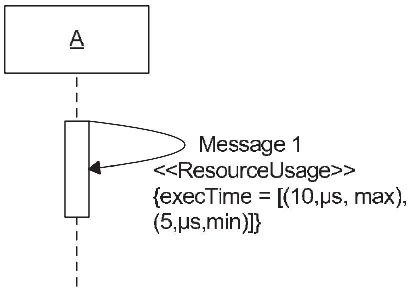

Fig. 14. Self a message example.

Figure 15 illustrates the mapping of a self message. The rules for mapping the self message are similar to the others mentioned above, the difference, in this case, is that the ETPN model backs to the same participant.

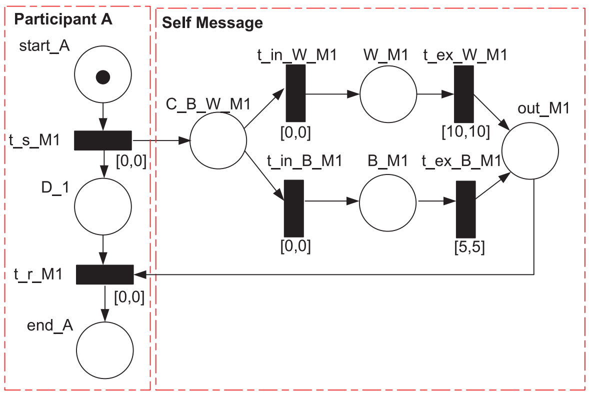

Fig. 15. Mapping self message.

5.2.3 Combined Fragments

A combined fragment is used to group sets of messages together in order to represent the SD conditional flows.

5.2.4 Mapping Alternatives

Alternatives are used to designate a mutually exclusive choice between two or more message sequences. The interaction operator *alt* (see Figure 16) shows that at least one of the operands will be executed.

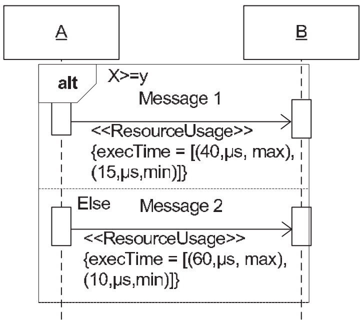

Fig. 16. Alternative combined fragment example.

Figure 17 depicts the mapping of alternatives presented in the Figure 16 into an ETPN. The rules for mapping this example are similar to the others mentioned before. However, this mapping adopts one place (alt_M1_M2) and two transitions (t_in_M1 and t_in_M2). The place represents the choice point between the operands *IF* and *ELSE* as well as the *alt* entry. One transition is used to represent the *Message 1* entry (*IF*). The other is used to represents the *Message 2* entry (*ELSE*).

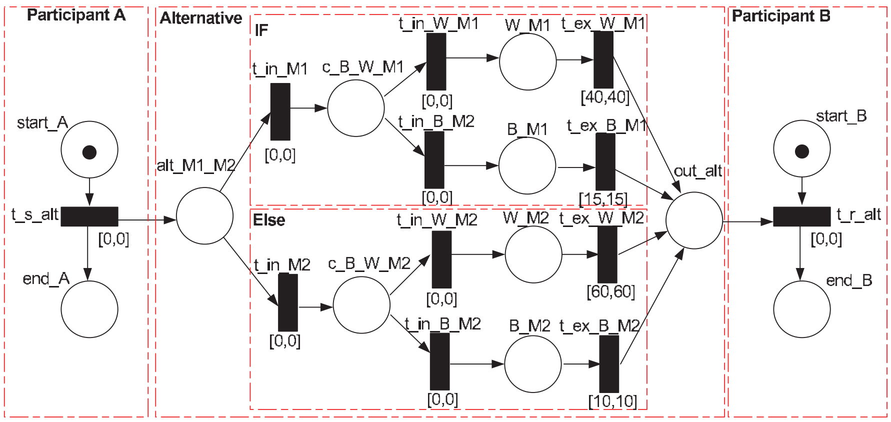

Fig. 17. Mapping alternative combined fragment example.

5.2.5 Mapping Parallel

The interaction operator *par* is used to define two or more processes that are concurrently executed. Figure 18 presents an example, in which the *Message 1* and *Message 2* are performed simultaneously.

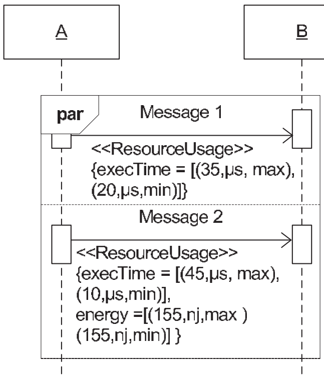

Fig. 18. Parallel combined fragment example.

Figure 19 depicts the mapping of parallel activities presented in the Figure 18 into an ETPN. The *t_in_par* and *t_syn_par* transitions represent the beginning and synchronization of concurrent activity execution, respectively. Each of these regions contains a distinct message and is mapped according to the rules mentioned before.

5.3 Mapping Interaction Overview Diagram

This section presents the mapping of an UML-IO example. Figure 20 (a) depicts this example in which two different interaction diagrams (activity and sequence) are connected by the *t1*

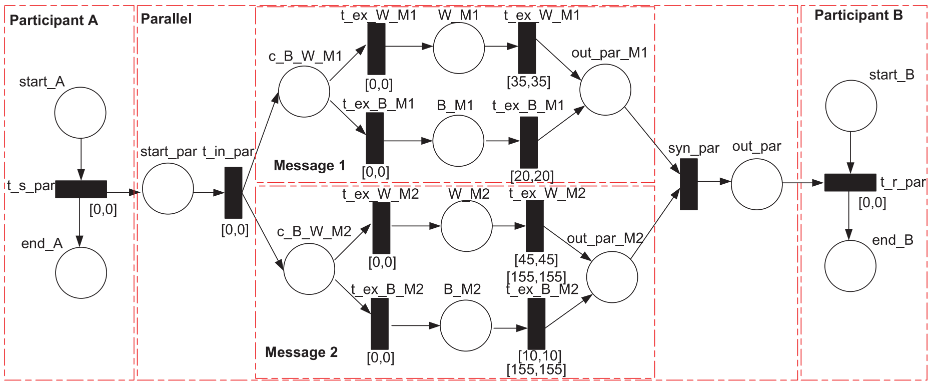

Fig. 19. Mapping parallel combined fragment.

IO-transition. The rules for mapping this example are similar to the others mentioned before. However, in the ETPN model (see Figure 20 (b)) the output arc of the *in_SD* place connects the *t_s_M1* PN-transition, and the output arc of the *t_r_M1* PN-transition connects the *out_sd* place. In this example, for sake of simplicity, the mapping of the initial and final states of UML-AD were omitted.

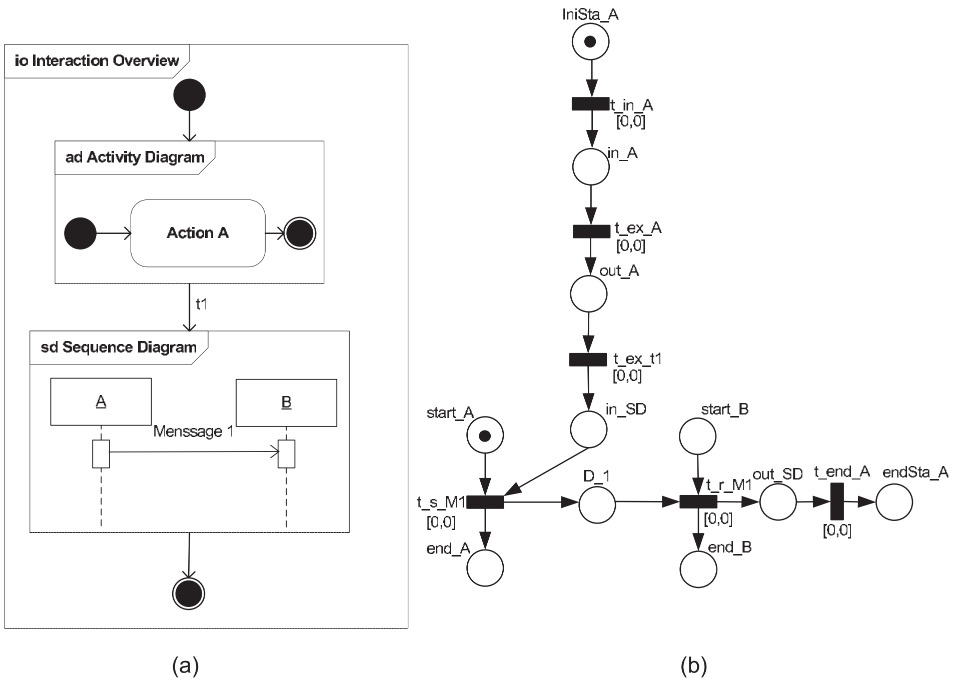

Fig. 20. Mapping UML-IO diagram.

6. A Case Study

In order to show the practical usability of the proposed mapping process, a pulse-oximeter specification (Júnior (1998)) was considered as a case study. This electronic equipment is responsible for measuring the blood oxygen saturation through a non-invasive method. A pulse-oximeter may be employed when a patient is sedated during surgical procedures. It checks the oxygen saturation to guarantee an acceptable level of blood oxygenation. This device is widely used in health care centers.

The pulse-oximeter is composed by three processes: (i) excitation, (ii) acquisition-control and (iii) management. The excitation process (see Figure 21) is responsible to dispatch pulse streams to the LEDs in order to generate radiation pulses. In this paper, only the excitation process was considered. Figure 21 depicts the excitation process according to Júnior (1998). Due to the restrictions of space, the ETPN model of the excitation process was omitted.

Figure 21 shows the UML-IO describing the excitation process. This process has requirement constraints (execution time and energy consumption). These constraints are deeply related to the hardware platform (Philips LPC2106 processor, an 32-bit microcontroler with ARM7 core). In addition, we consider the same values obtained from the pulse-oximeter specifications present in Júnior (1998) for modeling. As can be observed, the nodes represent interaction diagrams, in this case, we use the activity and sequence diagrams.

Initially the excitation process is started with the reception of an interruption in the the activity *Timer T1*. The timers are used to determine which process will be executed first. In the subsequent steps, the activities *Led Infrastructure*, *Set Led Guada* and *Set Led Red* are carry out. These activities are responsible for dispatching non-simultaneous pulse currents to the LEDs that cross the finger of a patient. In such cases, all activities have constraints of time and energy. For instance, the activity *Set Led Infra* has constraints of time and energy, and it is modeled with an <<ResourceUsage>> of: execTime = [(14,69,μs, max), (14,11,μs,min)] and energy = [(835,99, nJ, max), (795,21, nJ, min)].

After all activities responsible for dispatching pulse currents to the LEDs were performed, the activity of the next node takes place. This node is responsible for controlling the dispatch of pulse currents to the LEDs. As can be observed, an alternative combination fragment element is used to designate a mutually exclusive choice between two calls (*CalibrationNotRun()* and *CalibrationRun()*). These calls are responsible, respectively, to modify the intensity of the pulse currents or not. The calls (*CalibrationNotRun()* and *CalibrationRun()*) have constraints of time and energy. For instance, the call *CalibrationRun()* is modeled with an <<ResourceUsage>> of: execTime = [(1,87,μs, max), (1,79,μs,min)] and energy =[(180,01, nJ, max), (175,45, nJ, min)].

After the calibration of the attributes related to the LEDs, the excitation process returns to the node of the activity *Timer 1*, and waits for a new interruption to restart the process.

The following procedure is adopted to obtain the correspondent ETPN model that represents the activity diagrams present in the nodes of Figure 21. Each activity in the UML-AD is represented, basically, by two places and two PN-transitions. One place is used to represents the activity entry. The other place represents the activity exit, and the PN-transitions are used to represent the constraints assigned to activity. Additionally, two PN-transitions are adopted in the ETPN model due to semantics of TPN (Strong Firing Semantics). Besides, dummy places are used for bridging the PN-transitions. The beginning of UML-AD is depicted by initial state. The correspondent ETPN for initial state is a place with one token. Tokens are used in these ETPNs to simulate the dynamic behavior of systems. After all individual models have been built, the AD-transitions are mapped into PN-transitions in the ETPN model.

Furthermore, time and energy constraints are taken into account and included into ETPN PN-transitions.
Likewise, the following procedure is adopted to obtain the ETPN model that represents the sequence diagram presents in the node of Figure 21. Each participant is represented by at least two places and one transition in the ETPN model. One place with one token is used to represents the beginning of the involvement of a given participant. The other place represents the end of the involvement of a given participant, and the transition represents the send/receive of a message/call. After all individual ETPN models, that represent the participants, have been built, the messages/calls are mapped into transitions in the ETPN model. Messages/calls that do not have either time or energy constraints, will be mapped in one place, that is used as a dummy place for bridging the transitions. Furthermore, time and energy constraints are taken into account and included into ETPN transition.

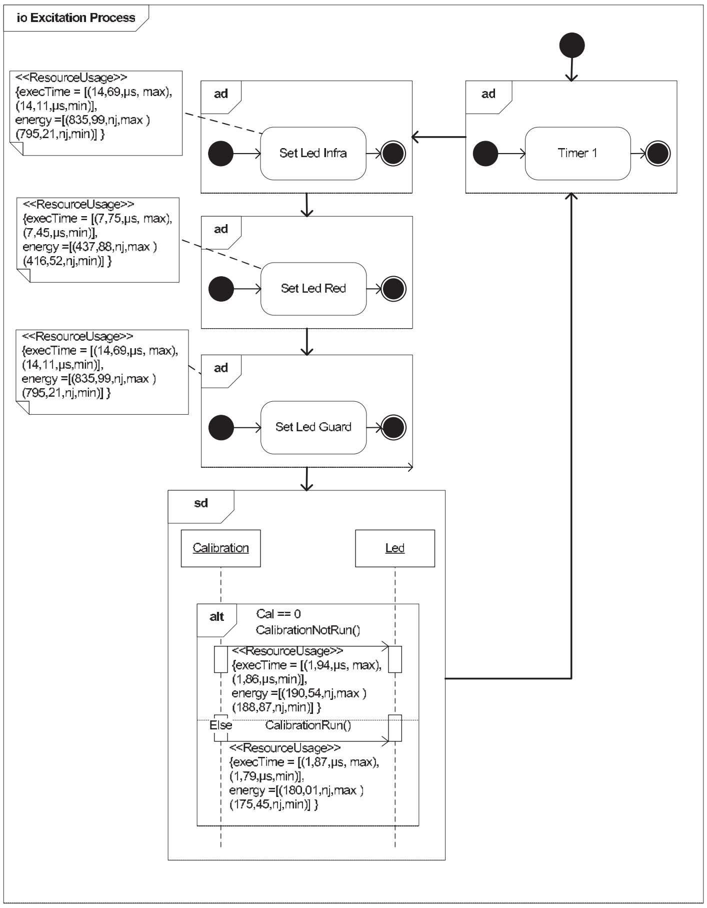
io Excitation Process
<<ResourceUsage>> {execTime = [(14,69,μs, max), (14,11,μs,min)], energy =[(835,99,nj,max) (795,21,nj,min)] }
ad
Set Led Infra
ad
Timer 1
<<ResourceUsage>> {execTime = [(7,75,μs, max), (7,45,μs,min)], energy =[(437,88,nj,max) (416,52,nj,min)] }
ad
Set Led Red
<<ResourceUsage>> {execTime = [(14,69,μs, max), (14,11,μs,min)], energy =[(835,99,nj,max) (795,21,nj,min)] }
ad
Set Led Guard
sd
Calibration
Led
alt
Cal == 0
CalibrationNotRun()
<<ResourceUsage>> {execTime = [(1,94,μs, max), (1,86,μs,min)], energy =[(190,54,nj,max) (188,87,nj,min)] }
Else
CalibrationRun()
<<ResourceUsage>> {execTime = [(1,87,μs, max), (1,79,μs,min)], energy =[(180,01,nj,max) (175,45,nj,min)] }

Fig. 21. UML-IO diagram of excitation process with time and energy constraints.

The analysis of the obtained ETPN model generated by the mapping process of the excitation process is safe, pure, homogeneous, conservative and bounded (David (2005)). The BCET

calculated of the excitation process was 37,46 ms and the WCET was 39,07 ms. Moreover, the respective energy consumption of the excitation process was also computed. The results obtained were 2182,39 μJ and 2300,4 μJ, respectively. The INA Tool (Starke & Roch (1999)) is adopted to compute the best and the worst path of the execution time. Once the worst and best path were found, then the energy consumption values are computed. In this paper, for the shortest and worst execution path time, respectively, the best and worst energy consumptions are computed.
The values measured on the hardware platform (execution time and energy consumption) were 38,88 μs and 2251,84 nJ, respectively. The analysis results show that the execution time and energy consumption, when considered the best and the worst paths, computed errors were smaller than 5% in related to the measurements conducted on hardware platform. The experimental results show that the values computed from the models are quite similar to the real measurement on the pulse-oximeter system.
UML is a user-friendly specification language that supports the specification, analysis, design, verification and validation of a broad range of complex systems. So, if the advantages of UML are allied to the power of formal models, some misinterpretations can be avoided, allowing both: reducing the risks of faults propagations from early specification to final code, and system properties analysis and verification. Hence, it can be used to reduce the risks as well as the amount of money or effort that can be spend building embedded projects.

7. Conclusions

Requirement analysis is a critical task in any embedded real-time system project. Normally, these systems have stringent timing constraints that must be satisfied for the correct functioning, since violation might be catastrophic, such as loss of human lives. In addition, there are systems where energy is another constraint that must also be satisfied. Hence, early detection of potential problems may reduce risks of faults propagations from early specification to the final code.
This work brings an approach based on ETPN for estimating embedded software execution time, energy consumption and verification of properties in early phases of the development life-cycle. The proposed method consists on a decomposition of a UML-IO into basic elements like activities, transitions and lifeline. These elements are translated into the corresponding ETPN representations. Quantitative annotations from MARTE profile, such as time and energy data, are taken into account and included into the ETPN. The obtained model is evaluated in order to check a set of qualitative properties as well as time and energy consumption requirements. That method has been applied into a practical system, namely, a pulse-oximeter, showing that this is a promising approach for modeling, analysis, and verification of real-world case scenarios.
Future research will explore the automatic generation of ETPNs from UML-IO with MARTE annotations. This work is also being extended to cover other significant UML diagrams like sequence and state machine diagrams. Another future work is related to stressing simulation and analysis capabilities to get as much significant information as possible from the Time Petri Net models.

8. References

Amorim, L., Maciel, P., Nogueira, M., Barreto, R. & Tavares, E. (2005). A Methodology for Mapping Live Sequence Chart to Coloured Petri Net, *Systems, Man and Cybernetics, 2005 IEEE International Conference on* .

Barreto, R. & Lima, R. (2004). A novel approach for off-line multiprocessor scheduling in embedded hard real-time systems, *Design Methods And Applications For Distributed Embedded Systems* .

Callou, G., Maciel, P., Andrade, E., Nogueira, B. & Tavares, E. (2008). A coloured petri net based approach for estimating execution time and energy consumption in embedded systems, ACM, New York, NY, USA, pp. 134–139.

David, R. (2005). *Discrete, Continuous, And Hybrid Petri Nets*, Springer.

Elkoutbi, M., Bennani, M., Keller, R. K. & Boulmalf, M. (2002). Real-time system specifications based on uml scenarios and timed petri nets, *2nd IEEE Internationnal Symposium on signal processing and information technology, ISSPIT'02* pp. 362–366.

Júnior, M. N. O. (1998). *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso (in portuguese)*, MSc Thesis, Departamento de Biofísica e Radiobiologia, Universidade Federal de Pernambuco.

Lee, J., Pan, J., Kuo, J., Fanjiang, Y. & Yang, S. (2000). Towards the verification of scenarios with time Petri-nets, *Computer Software and Applications Conference, 2000. COMPSAC 2000. The 24th Annual International* pp. 503–508.

MARTE, O. (2005). Profile for Modeling and Analysis of Real-Time and Embedded Systems, http://www.omgmarte.org/.

Merlin, P. & Faber, D. J. (1976). Recoverability of communication protocols: Implicatons of a theoretical study, *IEEE Transactions on Communications* **24**(9): 1036–1043.

Merseguer, J., Campos, J. & Mena, E. (2002). Performance evaluation for the design of agent-based systems: A Petri net approach, *Proceedings of the Workshop on Software Engineering and Petri Nets, within the 21st International Conference on Application and Theory of Petri Nets* pp. 1–20.

Starke, P. & Roch, S. (1999). *INA - Integrated Net Analyzer - Version 2.2*, Humbolt Universität zu Berlin - Institut für Informatik.

Tavares, E., Barreto, R., Maciel, P., Meuse Oliveira, J., Amorim, L., Rocha, F. & Lima, R. (2007). Software synthesis for hard real-time embedded systems with multiple processors, *SIGSOFT Softw. Eng. Notes* **32**(2): 1–10.

Tavares, E. & Maciel, P. (2006). Amalghma tool, http://www.cin.ufpe.br/∼eagt/tools/.

Trowitzsch, J. & Zimmermann, A. (2005). Real-Time UML State Machines: An Analysis Approach, *Object Oriented Software Design for Real Time and Embedded Computer Systems* .

UML, O. (2005). 2.0 Superstructure Specification, http://www.uml.org/.

ISBN 978-953-307-047-6
Hard cover, 752 pages
Publisher InTech
Published online 01, February, 2010
Published in print edition February, 2010