Schedulability and Energy Efficiency for Multi-core Hierarchical Scheduling Systems

Abdeldjalil Boudjadar, Alexandre David, Jin Hyun Kim, Kim G. Larsen, Marius Mikučionis, Ulrik Nyman, Arne Skou
Institute of Computer Science, Aalborg University, Denmark

***Abstract*—We propose a compositional framework for modeling and analyzing the schedulability and energy efficiency of embedded hierarchical scheduling systems running on a multi-core platform. The framework is realized using Parameterized Stopwatch Automata describing the concrete task behavior. The schedulability can be verified in a compositional way using UPPAAL, and the energy profile can be generated using the statistical model checking algorithms of UPPAAL SMC. To our knowledge, our paper is the first one considering hierarchical scheduling, multi-core platforms and energy consumption simultaneously. Finally, the framework is applied to an avionics case study.**

I. INTRODUCTION

Embedded systems are an essential part of many modern products including complex safety critical real-time systems. Within the industrial domains of avionics and automotive, the safe composition of several embedded features within the one system can be achieved through the use of hierarchical scheduling. The separation between features is secured by using time partition scheduling at the system level [9]. A trend within embedded systems is to use multi-core platforms in order to increase performance and to be able to implement more functionality within one embedded system.

During the design phase of an embedded system, energy profiles for components could be used by the system designers to communicate requirements for the individual parts of the system to the OEM (Original Equipment Manufacturer) companies that produce the parts. In later development stages, energy profiles can again be used to check that the complete system satisfies its energy requirements.

In this paper we propose an approach to analyze both the schedulability and the energy consumption of hierarchical scheduling systems running on embedded multi-core platforms. This approach is realized using Parameterized Stopwatch Automata (PSA) [6], which are analyzed using UPPAAL to check the schedulability and UPPAAL SMC to generate the energy profiles (Fig. 1).

In the literature, a large amount of work has been devoted to the description and analysis of scheduling systems [12] together with energy efficiency [11]. We extend these approaches by combining, in one framework, important aspects of modern systems like (1) powerful multi-core execution platforms together with (2) a hierarchical scheduling structure, and (3) abstract as well as *concrete* task behavior. In all of the previous work, systems can be viewed as a set of abstract components competing for CPU and other resources, and having a uniform distribution of energy consumption. In contrast, our framework enables modeling concrete task behavior and differentiated energy consumption rates based on task state.

The rest of the paper is organized as follows: Related work is discussed in section II. Section III gives a general overview of our approach. Section IV gives detailed description of how the approach is implemented in terms of Parameterized Stopwatch Automata (PSA) and analyzed using UPPAAL and UPPAAL SMC. The framework is instantiated and applied to an avionics case study in section VI. Finally the conclusion is given in section VII.

II. RELATED WORK

A compositional framework for hierarchical scheduling systems was initially presented in [14] as a formal way to elaborate a compositional approach for schedulability analysis of hierarchical scheduling systems [15]. In [13], the authors dealt with a hierarchical scheduling framework for multiprocessors based on cluster-based scheduling. They use analytical methods to perform analysis, however this approach has difficulty in dealing with complicated behavior of tasks.

In [3], the authors analyzed the schedulability of hierarchical scheduling *single-core* systems using the TIMES tool [2] and implemented their framework in VxWorks [3]. They constructed an abstract task model as well as scheduling algorithms focusing on the component under analysis. However, their approach requires not only timing attributes of the component under analysis, but also timing attributes of other components that can preempt the execution of the current component.

The authors of [5] provided a compositional framework for the verification of hierarchical scheduling systems using a model-based approach. They specified the system behavior in terms of preemptive time Petri nets, and only considered a single-core execution platform.

In [1], the authors study the schedulability of real-time embedded systems under energy constraints, such as using solar panels. We extend their approach by considering hierarchy and a multi-core platform while analyzing the schedulability in a compositional way. Our framework can also generate energy profiles for the system components.

The research presented in this paper has been partially supported by EU Artemis Projects CRAFTERS and MBAT.

III. General Approach

In this paper we structure our system model as a set of hierarchical components. Each component, in turn, is the parallel composition of a set of entities with a local scheduler. A task consists of a sequence of timed actions [7]; computation steps, input, output, etc. Moreover, we consider multi-core execution environments where individual timed actions have different energy consumption rates depending on their type and the CPU on which they execute.

Figure 1 summarizes our approach. Information on the scheduling requirements of the system is combined with the hierarchical structure of the system together with a detailed description of the task behavior including the energy consumption rates of the individual timed actions. Timed actions may consume different amount of energy when using different hardware platforms. A timed action may be specified to execute on a specific piece of hardware such as the GPU or I/O unit, thus indirectly the energy consumption of different processors can be specified. All of this information is used as parameters for Stopwatch Automata templates that are part of the framework. Once a specific instance of the framework has been created, its schedulability can be checked compositionally using UPPAAL while the energy profiles of the system components can be generated using UPPAAL SMC.

An energy profile consists of a graph .... (Finish after we have made the models and queries)

1) Concrete behavior and energy consumption of tasks: A task has a concrete behavior performing a sequence of timed actions. Each timed action can either be a computation step (`Compute`), communication (`Input`, `Output`) or particular statements marking the end of the period (`Pend`) or the end of the task execution (`End`).

Definition 1 (Timed action): Given a set of action names $Acts = \{\texttt{Compute}, \texttt{Input}, \texttt{Output}, \texttt{Pend}, \texttt{End}\}$ and a multi-core platform $\mathcal{P}$, a timed action A is a one step computation given by the tuple $\langle Act, Proc, BCET, WCET\rangle$ where:

- $Act \in Acts$ is the action name,
- $Proc \subseteq \mathcal{P}$ specifies the identifiers of processors on which the timed action A can be scheduled,
- $BCET$ and $WCET$ are respectively best case and worst case execution time,

By $\mathcal{A}$ we denote the set of all timed actions.

We consider a multi-core platform and associate each timed action to a set of processors on which it can execute. Moreover, we associate to each timed action energy consumption rates specifying how much energy is consumed by this action per time unit during its execution on a given processor. To this end, we introduce the rate relation $\Gamma : \mathcal{A} \times \mathcal{P} \rightarrow \mathbb{R}^+$ which associates to the execution of each action an energy consumption rate.

Likewise, we define the behavior B of a task as a transition system $\langle L, l^0, \rightarrow\rangle$ specifying the sequence of timed actions performed by that task, where L is a set of states, $l^0 \in L$ is the initial state and $\rightarrow \subseteq L \times \mathcal{A} \times L$ is the transition relation. States can be interpreted in the semantic level as valuations of the task variables.

The behavior of a component is given by the parallel composition of the transition systems of its nested tasks.

Definition 2 (Task structure): A task T is given by $\langle Prd, BCET, WCET, Pri, B, \Gamma\rangle$ where Prd is the task period, $BCET$ and $WCET$ are respectively best case and worst case execution times of T, Pri is the priority level associated to task T, B is the task behavior defined above and Γ states the energy consumption rates of T's timed actions.
Therefore, the task specification is given by an interface $Prd, BCET, WCET$ stating the time constraints, a behavior B expressed by a sequence of timed actions and a priority Pri that will be applied for each timed action of the task in question.

2) Hierarchical scheduling: We structure our system as a set of concurrent components. Each component, in turn, can also be a parallel composition of either other components or tasks, known as the component workload. Accordingly, the leaves of our system are tasks. A rough sketch of a hierarchical scheduling system can be seen as a part of Fig. 1 (Hierarchical System Architecture). A concrete example of a hierarchical scheduling system is given in Section VI.

Roughly speaking, a component is given by an interface stating its timing requirements and a local policy for scheduling its nested entities.

Definition 3 (Component): A component C is a tuple $\langle Prd, Budget, Pri, s, \langle e_1, .., e_n\rangle\rangle$ where:

- Prd and Pri are the same as for tasks,
- $Budget$ is the amount of resource that the component guarantees to provide to its workload,
- $s \in \{EDF, FP, RM, ..\}$ is a scheduling policy,
- $\langle e_1, .., e_n\rangle$ are component entities, either tasks or components (workload).

Similarly, a system is the top level component without timing requirements ($Prd, Budget, Pri$). We emphasize the fact that our framework can be instantiated for any combination of scheduling algorithms.

IV. Compositional Framework

Our analysis for multi-core hierarchical scheduling systems aims at obtaining verified and specified task designs that satisfy given resource constraints, e.g. CPU usage and energy consumption. The analysis framework is compositional in the sense that the analysis is performed on each component individually with respect to its requirements. The schedulability of each component is verified by checking its timing specification against the interface of its sub-entities [4]. Using the same framework, each component can also be analyzed in order to generate its energy profile.

To this end, we present a behavioral model of hierarchical scheduling systems. This model consists of components and task models based on task specifications including energy profiles. We construct a hierarchical scheduling system model using Parameterized Stopwatch Automata (PSA). The schedulability is verified by model checking in UPPAAL, and the energy efficiency is analyzed by statistical model checking in UPPAAL SMC.

The system model in this paper consists of models for different CPUs (including special purpose processors), models of

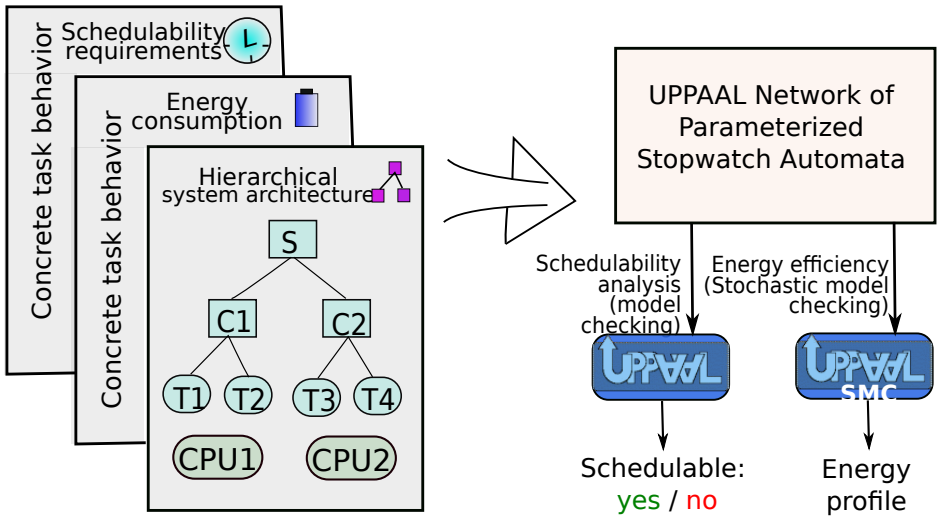

Fig. 1. Overview of the analysis framework

scheduling algorithms and models of the individual tasks. We have in our running example dedicated one of the processor cores as a specific I/O processor. In general the modeling framework can be instantiated with any number of special purpose processing units.

The computation budget of a component includes both regular execution and I/O actions of tasks. For the regular execution, the task specification can include for each timed action on which processor it should be executed. I/O timed actions are always executed on the special purpose I/O processor. Both for I/O and computation, the time consumption is part of the description of each individual timed action. Each processor has an associated energy consumption rate, so the energy consumed by a specific timed action is found by multiplying its actual execution time with the energy consumption rate of the processor on which it is executed. In the models this is achieved by the use of stopwatches.

A. Models of System in Parameterized Stopwatch Automata

The system is given in terms of a number of parameterized stopwatch automata. The important templates are described and depicted in the following sections. The system model includes templates for tasks, CPUs, scheduling algorithms, supplier models, global and local schedulers. Each template can be instantiated several times as part of the system declarations. As each template is defined with a set of parameters, it can be tailored to fit a specific application making our framework highly configurable.

1) Task Model: A task as seen in Fig. 2 serves the purpose of modeling a task in the real hierarchical scheduling system. The task interface is specified using two parameters that are the task ID and a description of the concrete behavior of the task. The concrete behavior is given as a list of timed actions. A timed action is used to represent the smallest task step that we model. It can be a chunk of computation or some I/O actions. Listing 3 shows the struct data type which is used to represent a single timed action.

Listing 1. Data structure for timed action

```
typedef     struct{
    cmdtype_t     id;
    rid_t         rid;
    time_t        bcet;
    time_t        wcet;
    pc_t          goto_n;
} cmd_t;
```

The `id` field represents the type of the timed action. The type of timed actions is given by the enumeration: `END`, `COMPUTE`, `INPUT`, `OUTPUT`, `PEND`, `GOTO`. The second field `rid` is the identifier of the processor on which this timed action can be executed. The value of `rid` can be used both to specify a particular processor or, by giving the value 0, to specify that the timed action can run on any regular processor. The framework could be made more general to match situations where the system architecture has groups of processors. The `bcet` and `wcet` are respectively the best and worst case execution times of the timed action. The `goto_n` is used together with `END`, `PEND`, `GOTO` to specify which timed action is next.

Formally, the task model is given a structure and a stopwatch automaton. The structure specifies the internal attributes used to store the task timing requirements like offset, period, execution time and deadline. The task structure is similar to that of a timed action. In our framework, the task timing requirements are given by a list where they are associated to the task by the task ID. Listing 2 shows the task structure we consider, where `pri` field states the priority level associated to the task. `initial_offset` and `offset` are respectively the initial and periodic waiting times that the task should wait for in order to being ready. The task period is given by 2 bounds `min_period` and `max_period` stating respectively the minimum and maximum values that the task period could be. For task execution, we consider both best and worst case execution times (`bcet`, `wcet`). Moreover, a task could also be preemptive or not according to the Boolean flag `preemptive`.

Listing 2. Task data structure

```
typedef struct {
  pri_t     pri;
  time_t    initial_offset;
  time_t    offset;
  time_t    min_period;
  time_t    max_period;
  time_t    deadline;
  time_t    bcet;
  time_t    wcet;
  bool      preemptive;
 } task_t;
```

The behavior of our task model is depicted in Fig. 2. Timed actions are carried out by the task model. When a task starts a new period and after waiting that the initial offset and the period offset have been elapsed, the task reaches location ChkCMD and being ready to check its timed actions. At this point, if the current timed action type is END the joins immediately location ClosingExec stating that the task execution is over. Otherwise, in the case of a COMPUTE, INPUT or OUTPUT, the task moves to location location ChkMCSchedPolicy to check which processors are able to be used to execute the current timed action.

In our framework, the following choice has been made: if the processor identifier specified by the timed action requirement is equal to 0 ($rid = 0$) means that such a timed action can be executed on any regular processor; otherwise ($rid > 0$) the timed action should be executed on the specified processor. Thus, from location ChkMCSchedPolicy of the task model, if the timed action does not specify a specific processor for its execution (cmd[pc].rid $\leq$ 0), the task is going to be scheduled on any available processor. The identifier of such a processor will be non deterministically determined from location WaitCoreID by the statements i:rid_t p_assign[tid][i]? tstat[tid].rid=i. In the case where the timed action provide the identifier of the required processor (cmd[pc].rid $>$ 0), such a processor identifier will be moved to the task level requirement (tstat[tid].rid=cmd[pc].rid). In both cases, the task moves to location ReqSched waiting to be scheduled on the designated processor. Once such a requirement is satisfied, the task moves to location RUN, where it can miss its deadline and joins location MISSDLINE; or achieves normally the execution of the current timed action and joins location ChkCMD. While a task is in the location RUN a number of stopwatches keep track of the execution time of the individual timed action and the energy consumed by the task. The energy consumption is calculated by setting the rate of the energy stopwatch to the energy consumption rate of the processor.

2) Multi-core Platform Model: The processor model we consider is depicted in Fig. 3. It consists of a multi-core platform which encompasses a set of regular processors. The multi-core model is initially waiting for a processor request. On the reception of such an event, it triggers the processor scheduler, shown in Fig. 6, together with the multi-core scheduling policy to determine which core will be used for the current request. Once a processor identifier is determined and communicated to the multi-core platform (ack_psched[policy][i]?), such a processor identifier will be communicated to the task having priority (pq.element[0]) in the waiting list pq by a synchronization over channel p_assign.

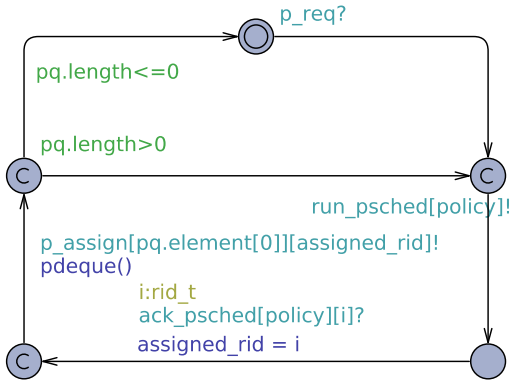

Fig. 3. Multi-core platform model

Once the first element of waiting list is served, it will removed and if the list is still non-empty the multi-core template model rerun the process of selecting other processors to serve to the waiting tasks.

B. Processor Model

The processor model depicted in Fig. 4 is used to model system resources that can only be used by one timed action at a time e.g. processors. From the state Idle the processor model is waiting for a request for its resource with the corresponding rid. From the urgent location ReqSched, the processor model checks how many elements are available in its waiting list (rq[rid].element[]). If the list is empty the processor model joins the Idle location. Otherwise, the processor model makes a call to its associated scheduling policy, waiting for the return acknowledgment from the local scheduler, indicating that it has placed the correct element as the first item in the rq[rid].element[] queue. In the Assign location the processor model again checks if the waiting list is empty, by making sure that the first element in the list is not 0. If this is not the case it executes the element and moves to the InUse location. In this location, it can be notified by a finished task execution over the finished[rid] channel or it can receive a new scheduling request. By catching a new scheduling request, the processor model moves to the location ReqSched where it triggers the scheduling policy for the new waiting list i.e. rq[rid].element[]. When a task requests a resource, it inserts itself in the waiting list of the corresponding processor.

In our compositional multi-core framework, each core is individually managed by its corresponding CPU resource model. Each task can be scheduled on different cores over its lifetime.

1) Local Scheduler: The local scheduler role consists of scheduling the different tasks designated to be executed on a same processor. In the following we will show the EDF scheduling algorithm, but our framework is highly configurable and this model can be replaced by models that implement different scheduling algorithms. In fact, the local scheduler (Fig. 5) initially waits until one CPU requests the scheduler to manage its waiting list. After joining location ChkTaskNumber, the local scheduler checks how many tasks are available in the waiting list of the given processor. If such

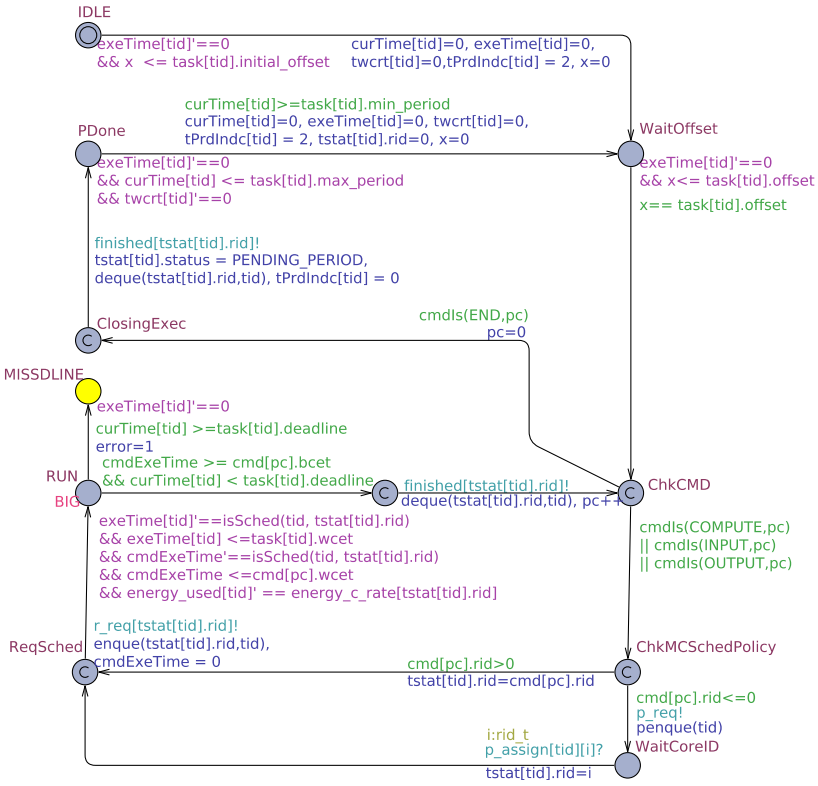

Fig. 2. Task model

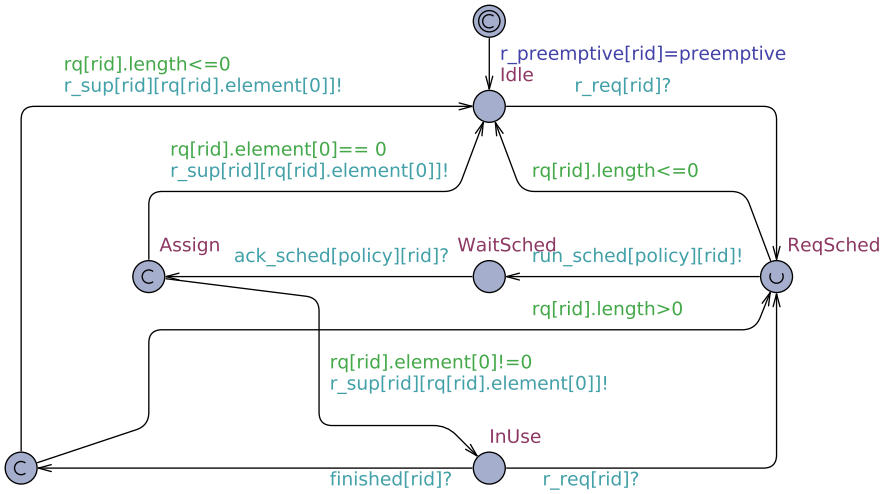

Fig. 4. Processor (resource) model

a list contains only one task, the former will immediately be scheduled and the scheduling process is over for the current request. Otherwise, the local scheduler moves to location ChkEarliestDL in order to determine the task having the earliest deadline in the waiting list of the given processor. Once a task is designated, the scheduler acknowledges the processor in question, from location AckSchedReq, and moves to the initial location.

2) Global Scheduler: The multi-core scheduling algorithm we consider, as depicted in Fig. 6, is simple and consists of designating a CPU identifier among a set of processors to be used for a request. Once a CPU identifier is determined by the scheduler, it will be communicated to the multi-core platform via a synchronization over channel ack_psched.

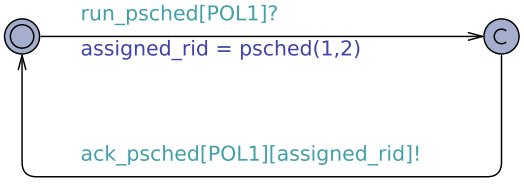

Fig. 6. Processors scheduler model

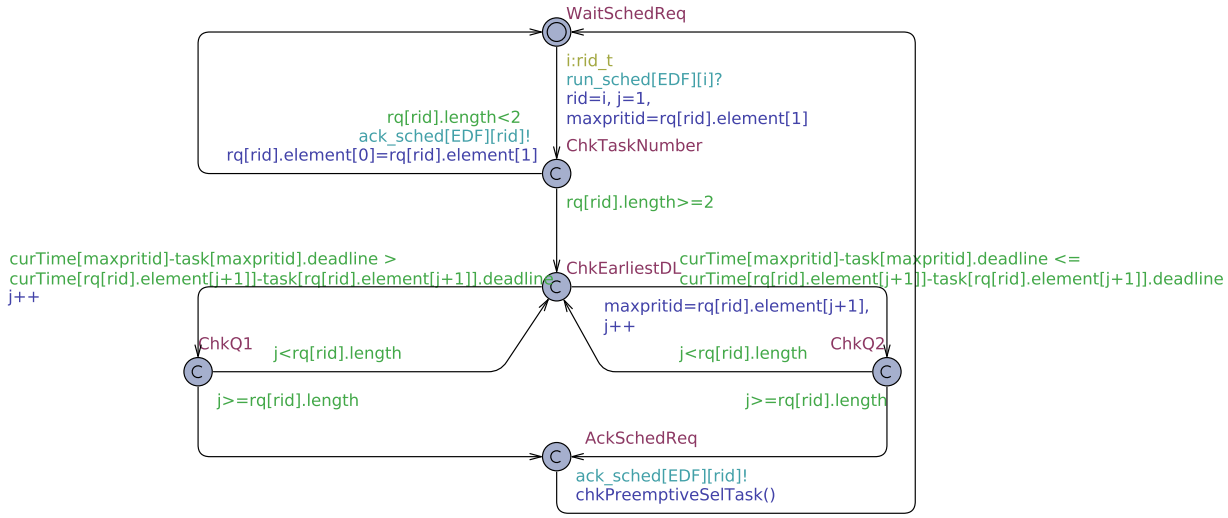

Fig. 5. Local EDF scheduler model

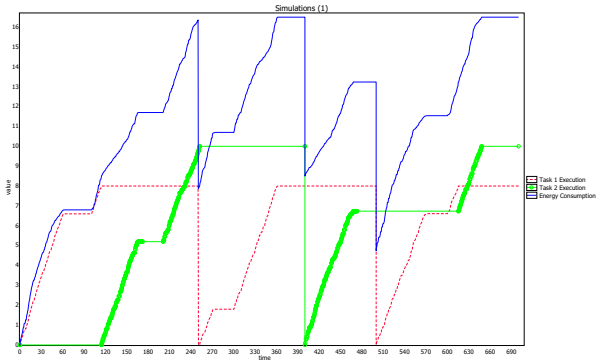

Fig. 7. Cumulated energy consumption for two individual tasks

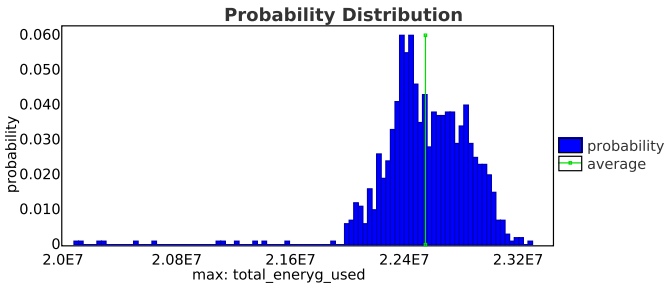

Fig. 8. Energy profile for two tasks, 1000 runs, 1000000 time units

V. Analysis: Schedulability and Energy Efficiency

In this section we explain the two forms of analysis that we apply in our framework (Fig. 1). Schedulability of a system is checked using symbolic model checking (Uppaal) and produces a YES or a NO. For the energy consumption we generate energy profiles for individual components using statistical model checking (Uppaal SMC).

A. Generation of Energy Profiles

In its lifetime a task may use different processors for its execution, so the accumulated energy consumption depends on the energy consumption rate of the processors it has used. Fig. 7 shows the individual cumulated energy consumption for two tasks. This plot is included here to show the detail with which we model and simulate the hierarchical scheduling system.

We generate an energy profile for a component by simulating a large number of runs over a reasonable large time span. The total cumulated energy consumption for each run is then plotted in a histogram as shown in Fig. 8. The energy profile is obtained using the following query:

$$E[gClock \leq simTime;1000](max:total\_energy\_used)$$

Fig. 8 can be used to characterize the energy consumption of a component. In a similar way, we could generate statistical energy profiles for complex hierarchical scheduling systems.

B. Schedulability Analysis

In the following we will first introduce some classical notation in order to set this in relation to the work presented in the paper.

From the viewpoint of the supplier and sub entities, the utilization of resources, such as CPU and energy, can be analyzed as follows: Let C be a component, $W = (T_1, T_2, ...T_n)$ the tasks, s a scheduling algorithm, and R any resource model. I denotes the collective requirements, (Prd and Bud given in *Definition 3*). For any resource model R, a scheduling unit $\Psi(W, R, s)$ is said to be schedulable if and only if:

$$\forall t > 0 \quad \mathtt{dbf}_{\mathtt{A}}(W, t) \leq \mathtt{sbf}_{\mathtt{R}}(t)$$

where $\mathtt{dbf}_{\mathtt{A}}(W, t)$ is the demand-bound function and $\mathtt{sbf}_{\mathtt{R}}(t)$ is the supply-bound function [13].

The interface I of a component $C(W, s)$ is said to be schedulable if the scheduling unit $\Psi(W, R, s)$ is schedulable with $R = I$, i.e. the resource model R satisfies the interface I of the component timing requirements.

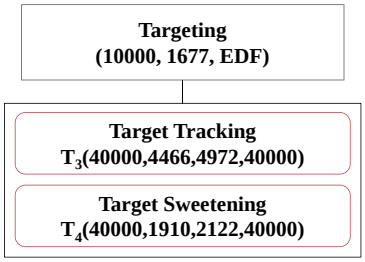

Fig. 10. Detailed view of the *Targeting* component

For an processor resource $CPU \in \mathcal{P}$, the meaning of the schedulability in [13] is adopted for this framework. To verify the schedulability, we check the following property:

$$\forall t > 0 \quad \mathtt{dbf}_{\mathtt{A_{CPU}}}(W,t) \leq \mathtt{sbf}_{\mathtt{R_{CPU}}}(t)$$

The schedulability of the system is checked using symbolic model checking the following query:

A[] (error!=1)

When this analysis is performed we only consider the worst case execution times and ignore the best case execution time. The result of the analysis is just a YES or an error trace.

VI. Case Study

In this section we utilize the presented methodology on an avionics case study.

The original case study as presented in [10] does not contain any information on the energy and consumption of tasks. Without considering energy consumption and BCET, the avionics system has also been analyzed in [8], [4]. Thus, the energy consumption rates considered in the paper are constructed for the sake of the analysis. The energy consumption rate is a parameter of the timed actions and can be updated to reflect real measurements. We also consider both WCET and BCET in our analysis, where the BCET of a task is set to 90% of the WCET as given in [10]. Again, the BCET and WCET are given at the level of timed actions and can be modified for each individual action.

Fig. 9 shows the architecture of the hierarchical scheduling system, with the interface specifications of each component and task. In this view of the system the timing attributes are specified in milliseconds. The original specification [10] actually specifies the WCET of each timed action, but the interfaces of the components have been estimated according to our previous work [4] and verified using the framework presented in the current paper.

To illustrate the generation of energy profiles we focus on two components. These components are respectively depicted in Fig. 10 and Fig. 11. When generating the energy profiles, we have converted the time units from milliseconds to microseconds. Fig. 10 and Fig. 11 also show both the WCET and BCET since both time bounds are used in the generation of the energy profile.

The timed actions of these two tasks are given the following listing:

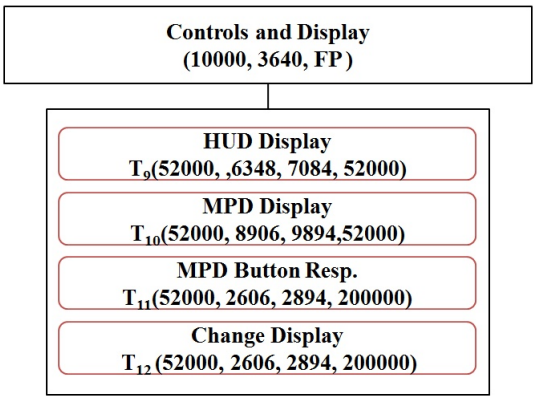

Fig. 11. Detailed view of the *Controls and Display* component

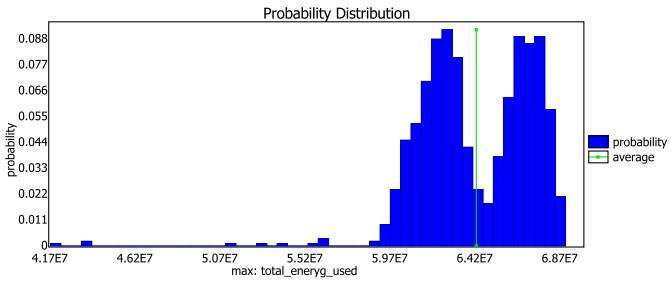

Fig. 12. Energy profile for four tasks in the component *Controls and Display*, 1000 runs, 1000000 time units

Listing 3. Data structure for timed action

```
const cmd_set_t Target_tracking = {
    { INPUT   ,  3,   110,  122,  0},
    { INPUT   ,  3,   164,  182,  0},
    { INPUT   ,  3,   100,  122,  0},
    { INPUT   ,  3,   146,  162,  0},
    { COMPUTE,  0,  3600, 4000,  0},
    { OUTPUT ,  3,   200,  222,  0},
    { OUTPUT ,  3,   146,  162,  0},
    FIN,FIN,FIN,FIN,FIN
};

const cmd_set_t Target_sweetening = {
    { INPUT   ,  3,  110, 122, 0},
    { COMPUTE,  0,  1800, 2000, 0},
    FIN,FIN,FIN,FIN,FIN,FIN,FIN,FIN,FIN,FIN
};
```

The same analysis as was performed for the *Targeting* component is also performed for the *Controls and Display* component. Fig. 11 shows the component *Controls and Display*, Fig. 12 shows the energy profile and Listing 4 shows the timed actions of that component.

Listing 4. Data structure for timed action

```
const cmd_set_t HUD_display = {
    { INPUT   ,  3,   420,  462,  0},
    { INPUT   ,  3,   146,  162,  0},
    { INPUT   ,  3,   164,  182,  0},
    { COMPUTE,  4,  5400, 6000,  0},
    { OUTPUT ,  3,   218,  242,  0},
    FIN,FIN,FIN,FIN,FIN,FIN,FIN
};

const cmd_set_t MPD_display = {
    { INPUT   ,  3,  110, 122, 0},
    { INPUT   ,  3,  452, 502, 0},
    { INPUT   ,  3,  452, 502, 0},
```

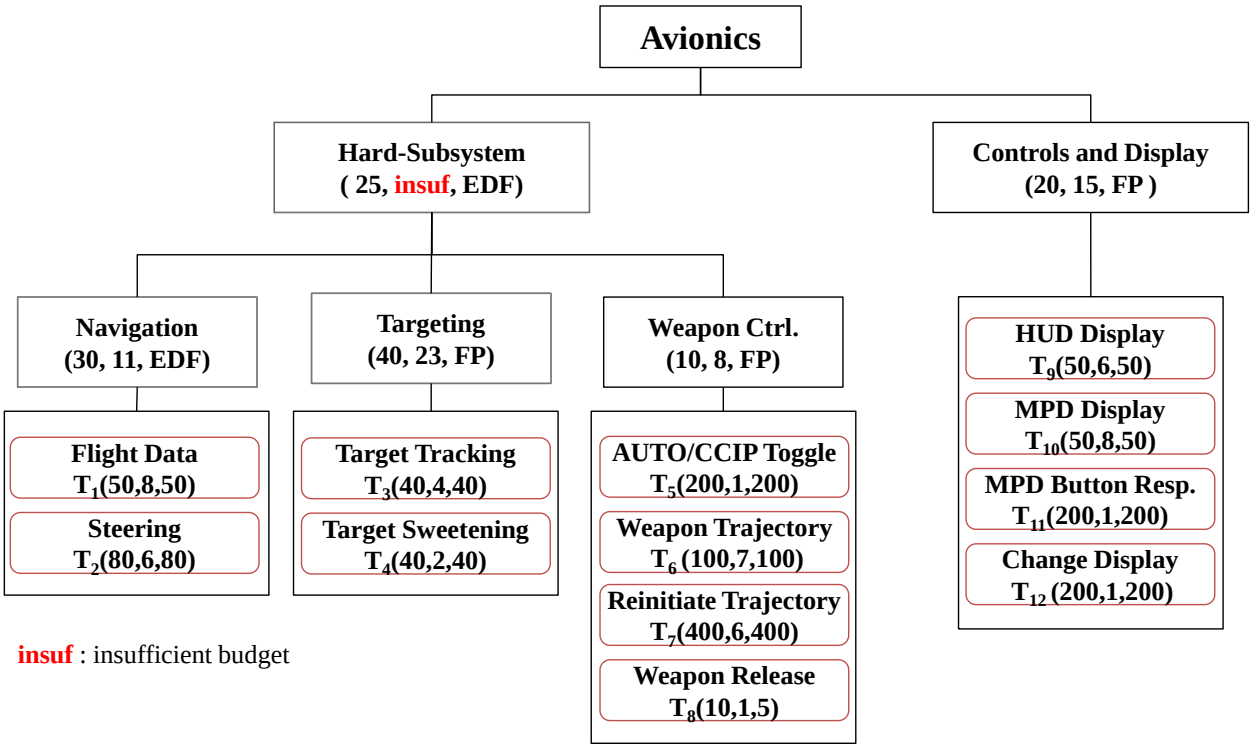

Fig. 9. Architecture of the hierarchical scheduling system

```
    { INPUT  ,  3,  218, 242, 0},
    { INPUT  ,  3,  146, 162, 0},
    { INPUT  ,  3,  146, 162, 0},
    { COMPUTE,  4,  7200, 8000, 0},
    { OUTPUT ,  3,  182, 202, 0},
    FIN,FIN,FIN,FIN
};

const cmd_set_t MPD_button_resp = {
    { INPUT  ,  3,  110, 122, 0},
    { INPUT  ,  3,  452, 502, 0},
    { INPUT  ,  3,  452, 502, 0},
    { INPUT  ,  3,  218, 242, 0},
    { INPUT  ,  3,  146, 162, 0},
    { INPUT  ,  3,  148, 162, 0},
    { COMPUTE,  0,  900, 1000, 0},
    { OUTPUT ,  3,  182, 202, 0},
    FIN,FIN,FIN,FIN
};

const cmd_set_t Change_display = {
    { INPUT  ,  3,  110, 122, 0},
    { INPUT  ,  3,  452, 502, 0},
    { INPUT  ,  3,  452, 502, 0},
    { INPUT  ,  3,  218, 242, 0},
    { INPUT  ,  3,  146, 162, 0},
    { INPUT  ,  3,  148, 162, 0},
    { COMPUTE,  0,  900, 1000, 0},
    { OUTPUT ,  3,  182, 202, 0},
    FIN,FIN,FIN,FIN
};
```

The statements FIN in the listing are used as a placeholder for no command. This is necessary as the list of timed actions has a static size. The energy profile of these two tasks is the one that was shown in Fig. 8.

In the case study, the I/O processor has processor ID 3 and the two regular processors has ID 1 and 2. A value of 0 is used to signify that a given timed action can run either processor 1 or 2. More complicated setups with different subsets of processors can be defined for other instantiations of the framework. It is in practice possible to create a task where timed actions require to be executed on different processors. The only way in which we utilize this in the current instantiation of the framework is in that all I/O action have to run on the I/O processor.

VII. Conclusions

We have presented a compositional framework for the analysis of schedulability and generation of energy profiles of hierarchical embedded multi-core real-time systems. The framework has been instantiated as reusable models given in terms of parameterized stopwatch automata (PSA) which we analyzed using Uppaal and Uppaal SMC. The reusable models ensure that when modeling a hierarchical scheduling application only the concrete task behavior and the hierarchical structure need to be specified by the system engineer. The framework also allows for instant changes of the scheduling policy at each given level in the hierarchy. The framework has been applied to a previously published avionics case-study.

As future work, we plan to study how to use the generated energy profiles for analyzing the energy efficiency of systems. To the best of our knowledge, this paper is the first paper to consider multi-core, hierarchy and energy concepts in the design of embedded real-time systems. A perspective could also be to encapsulate the framework in a tool such that it could be used in a practical development toolchain.

REFERENCES

[1] Y. Abdeddaïm and D. Masson. Real-time scheduling of energy harvesting embedded systems with timed automata. In *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications RTCSA12*, pages 31–40. IEEE Computer Society, 2012.

[2] T. Amnell, E. Fersman, L. Mokrushin, P. Pettersson, and W. Yi. Times: A tool for schedulability analysis and code generation of real-time systems. In K. G. Larsen and P. Niebert, editors, *FORMATS*, volume 2791 of *LNCS*, pages 60–72. Springer, 2003.

[3] M. Behnam, T. Nolte, I. Shin, M. Åsberg, and R. Bril. Towards hierarchical scheduling in VxWorks. In *OSPERT 2008*, pages 63–72.

[4] A. Boudjadar, A. David, J. H. Kim, K. G. Larsen, M. Mikučionis, U. Nyman, and A. Skou. Hierarchical scheduling framework based on compositional analysis using uppaal. In *Proceedings of FACS 2013*, lncs. Springer, 2013. To appear.

[5] L. Carnevali, A. Pinzuti, and E. Vicario. Compositional verification for hierarchical scheduling of real-time systems. *IEEE Transactions on Software Engineering*, 39(5):638–657, 2013.

[6] F. Cassez and K. G. Larsen. The impressive power of stopwatches. In C. Palamidessi, editor, *CONCUR*, volume 1877 of *Lecture Notes in Computer Science*, pages 138–152. Springer, 2000.

[7] A. David, K. G. Larsen, A. Legay, and M. Mikučionis. Schedulability of herschel-planck revisited using statistical model checking. In *ISoLA (2)*, volume 7610 of *LNCS*, pages 293–307. Springer, 2012.

[8] R. Dodd. Coloured petri net modelling of a generic avionics missions computer. Technical report, Department of Defence, Australia, Air Operations Division, 2006.

[9] B. Leiner, M. Schlager, R. Obermaisser, and B. Huber. A comparison of partitioning operating systems for integrated systems. In *SAFECOMP*, volume 4680 of *LNCS*, pages 342–355. Springer, 2007.

[10] C. D. Locke, D. R. Vogel, L. Lucas, and J. B. Goodenough. Generic avionics software specification. Technical report, DTIC Document, 1990.

[11] L. Niu and J. Xu. Improving schedulability and energy performance for weakly hard real-time systems. In *IPCCC, 2012 IEEE 31st*, pages 41–50, 2012.

[12] J. Rufino, J. Craveiro, and P. Veríssimo. Building a time- and space-partitioned architecture for the next generation of space vehicle avionics. In *SEUS*, volume 6399 of *LNCS*, pages 179–190. Springer, 2010.

[13] I. Shin, A. Easwaran, and I. Lee. Hierarchical scheduling framework for virtual clustering of multiprocessors. In *ECRTS*, pages 181–190. IEEE Computer Society, 2008.

[14] I. Shin and I. Lee. Periodic resource model for compositional real-time guarantees. In *RTSS*, pages 2–13. IEEE Computer Society, 2003.

[15] I. Shin and I. Lee. Compositional real-time scheduling framework with periodic model. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.