2^{nd} International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011)

July 5th, 2011, Porto, Portugal

In conjunction with the 23^{rd} Euromicro Conference on Real-Time Systems (ECRTS 2011)

Table of Contents

Message from the Program Chairs

Research in real-time systems has gone very far from the initial seminal papers back in the 70s. Many algorithms, design methodologies, techniques and tools have been proposed, spanning several application areas, from RTOS to distributed systems, from safety critical to soft real-time systems. However, unlike other research areas (e.g., networking) there are no widely recognized reference tools or methodologies for comparing different research works in the area.

In fact, the comparison among results achieved by different research groups becomes non-trivial or impossible due to the lack of common tools or methodologies by means of which the comparison is done. For example, different authors use different algorithms for generating random task sets, different application traces when simulating dynamic real-time systems, different simulation engines when simulating scheduling algorithms. Therefore, research in the field of real-time and embedded systems would greatly benefit from the availability of well-engineered, possibly open tools, simulation frameworks and data sets which may constitute a common metrics for evaluating simulation or experimental results in the area. Also, it would be nice to have a possibly wide set of reusable data sets or behavioural models coming from realistic industrial use-cases over which to evaluate the performance of novel algorithms. Availability of such items would increase the possibility to compare novel techniques in dealing with problems already tackled by others from the multifaceted viewpoints of effectiveness, overhead, performance, applicability, etc.

The ambitious goal of the International Workshop on Anaysis Tools and Methodologies for Embedded and Real-time Systems is to start creating a common ground and a community to collect methodologies, software tools, best practices, data sets, application models, benchmarks and any other way to improve comparability of results in the current practice of research in real-time and embedded systems. People from industry are welcome to contribute as well with realistic data or methods coming from their own experience.

The workshop seeks original contributions on methods and tools for real-time and embedded systems analysis, simulation, modelling and benchmarking. We look for papers describing well-engineered, highly reusable, possibly open, tools that can be used by other researchers.

Areas of interest include, but are not limited to:

- Simulation of real-time, distributed and embedded systems
- Simulation of multi-core, many-core and massively parallel and distributed systems
- Modeling, analysis and simulation of Operating Systems components

- Tools and methodologies for real-time analysis
- Instrumentation of Operating Systems
- Tracing methods and overhead analysis
- Power consumption models and experimental data for real-time power-aware systems
- Middleware components and mechanisms for distributed infrastructures supporting real-time and QoS-aware Cloud Computing applications
- Realistic case studies and reusable data sets
- Comparative evaluation of existing algorithms

We would like to thank the Euromicro organization for having allowed us to organize this event, and particularly Gerhard Fohler for his prompt and ready support. We would like to thank all the authors for having submitted their work to the workshop for selection, the Program Committee members for their effort in reviewing the papers, the presenters for ensuring interesting sessions, and the attendees for participating into this event. We hope that interesting ideas and discussions will come out of the presentations, demos and the questions that will alternate along the day. We hope you will find this day interesting and enjoyable.

The WATERS 2011 Chairs
Giuseppe Lipari and Tommaso Cucinotta

Real-Time Systems Laboratory
Scuola Superiore Sant'Anna, Pisa (Italy)
{g.lipari, t.cucinotta} @ sssup.it

Program Committee

- Andrea Acquaviva (Politecnico di Torino, Italy)
- Mark Bartlett (University of York, UK)
- Ian Broster (Rapita Systems Ltd, York, UK)
- Roberto Bucher (SUPSI, Manno, Switzerland)
- Gerhard Fohler (Technische Universitaet of Kaiserslautern, Germany)
- Christopher D. Gill (Washington University, St. Louis, Missouri)
- Michael Gonzalez (Universidad de Cantabria, Spain)
- Lucia Lo Bello (University of Catania, Italy)
- Damir Isovic (Mälardalen University, Sweden)
- Julio Medina (Universidad de Cantabria, Spain)
- Thomas Nolte (Mälardalen University, Sweden)
- Luigi Palopoli (University of Trento, Italy)
- Rodrigo Santos (Universidad Nacional del Sur, Bahia Blanca, Argentina)
- Simon Schliecker (Symtavision GmbH, Braunschweig, Germany)
- Douglas C. Schmidt (Vanderbilt University, Nashville, TN)
- Marisol Garcìa Valls (Universidad Carlos III de Madrid, Spain)
- Zlatko Zlatev (IT-Innovation Center, Southampton, UK)

SMFF: System Models for Free

Moritz Neukirchner, Steffen Stein and Rolf Ernst
Technische Universität Braunschweig
Braunschweig, Germany
neukirchner|stein|ernst@ida.ing.tu-bs.de

***Abstract*—Evaluation of scheduling, allocation or performance verification algorithms requires either analytical performance estimations or a large number of testcases. In many cases, e.g. if heuristics are employed, extensive sets of testcase systems are imperative. Oftentimes realistic models of such systems are not available to the developer in large numbers.**

We present SMFF (System Models for Free) - a framework for pseudorandom generation of models of real-time systems. The generated system models can be used for evaluation of scheduling, allocation or performance verification algorithms. As requirements for the generated systems are domain-specific the framework is implemented in a modular way, such that the model is extendible and each step of the model generation can be exchanged by a custom implementation.

I. Introduction

During the development of e.g. scheduling or allocation algorithms or algorithms for performance verification, testcase systems are required to evaluate the applicability and performance. If formal proofs of correctness or analytically derived performance estimations can be given a small set of such systems is sufficient. However, in many cases this is not possible e.g. if heuristics are employed. In this case the algorithm has to be tested with an extensive set of testcases. For many algorithm developers, especially in academia, system models are not available in large numbers. Manually creating such system models is very time-consuming and might not respect requirements on randomness.

In this paper we address this issue and present SMFF - a framework for parameter-driven generation of models of distributed real-time systems. These models incorporate a description of the platform, of the software applications mapped onto the platform and the associated scheduling and timing parameters, thus covering the entire model specification.

As system models, that are used for algorithm evaluation, have to resemble real-world systems, requirements on testcase systems may be highly domain- and problem-specific. The presented framework provides a high degree of modularity, allowing the user to extend the system-model and to replace algorithms for system model generation, thus making the framework a universal tool for testcase generation. It is available as open source software from [1]. The algorithms presented in this paper and provided along with SMFF are example implementations and were developed for the evaluation of an algorithm to find execution priorities in static-priority-preemptively scheduled systems under consideration of timing constraints [2].

The key features of the SMFF framework are:

- Parameter-driven generation of complete system models for use as testcases
- A modular framework architecture to allow exchange of generation algorithms
- Extendible data structures to allow customization of the system model

The SMFF framework is no simulation or benchmarking environment. Thus, we do not address the issues of simulation or performance monitoring. We rather provide models as input for such tools.

This paper is structured as follows. First we will discuss how the process of system model generation can be structured into discrete steps. We will then give an overview of related work and how previous approaches relate to this structure. In section IV we will define the main terms and the system model used throughout the paper. The following sections will address the single steps of the model generation and the implemented algorithms. Section IX covers aspects of the implementation of the framework and its modularity. In section X we will present an example of system model generation with the SMFF framework. Then we will conclude the paper.

II. General Approach

In this section we will outline the general approach to generate system models of distributed real-time systems.

We propose to divide the process of model generation into six steps as depicted in fig. 1. The six steps can be grouped into the categories *structural definition*, *real-time property definition* and *constraint definition*. While some steps can be executed independent of each other, some require other steps to be performed beforehand (indicated by arrows).

The structural definition is composed of the *platform model generation*, the *application model generation* and the *application mapping*. During platform model generation the architecture graph is constructed. Thus this step defines the topology of the platform. In the application model generation step, application graphs are created defining the logical structure of software applications. The final step of the structural definition maps the application models onto the platform model, defining the distribution of the application on the hardware platform. Naturally this step can only be performed after architecture and application models have been created.

Real-time property definition is composed of the steps of *assignment of timing properties* and of *assignment of scheduling parameters*. In the first step tasks and communication between tasks are assigned an execution model such as best-case execution and worst-case execution times. Furthermore tasks can be assigned an activation model (e.g. activation period and activation jitter). Thus this step defines the timing of each entity of an application in isolation. This step can only be

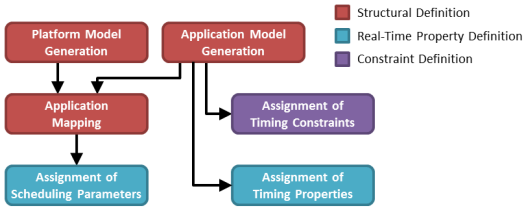


Fig. 1: General testcase generation flow

performed after the application models have been generated. In the second step scheduling parameters, such as execution priorities for static priority preemptively (SPP) scheduled resources, are assigned to the tasks and communication. This step can only be executed after the application mapping, as scheduling parameters depend on the scheduling strategy of the resources tasks and communication have been mapped to. After these two steps the timing behavior of the system is completely specified.

The last category is composed of only one step - the *assignment of timing constraints*. In assignment of timing constraints limitations on e.g. end-to-end path latencies or output jitters can be defined. Also this step can only be performed after the application model has been created, as constraints are specified for entities of an application.

Except for the outlined dependencies the steps of the system model generation can be performed in arbitrary order and independent of each other. This gives great flexibility to the developer as it is possible to e.g. load manually defined platform models and only perform the remaining steps of the testcase generation with the provided platform model. Also the developer may introduce further dependencies to tailor the testcase generation process for his specific needs. E.g. one may make the assignment of timing properties depend on the application mapping to include processor-specific execution times.

III. Related Work

We now review previous approaches to generation of testcase system models and highlight how these approaches relate to the structure of testcase generation as described above.

For evaluation of real-time algorithms (e.g. scheduling, allocation or performance verification) many developers rely on handcrafted example systems, to highlight strengths and shortcomings of different approaches (e.g. [3], [4], [5]). Others rely on benchmark models of real applications such as MPEG2 decoders (e.g. [6]) or on more comprehensive benchmark suites as e.g. [7]. A third approach many developers take for evaluation and comparison of their algorithms is parameter-driven testcase generation (e.g. [8], [9]).

If any of the first two approaches is taken, the results of an algorithm evaluation tend to be fairly well reproducible as the set of system models is well specified and oftentimes publicly available. However the number of testcases typically is low. As a result statements about average performance of an algorithm may be inaccurate, as the examples might not be representative of the targeted domain of application or might not cover common corner-cases properly. This may bias the evaluation of an algorithm.

The approach of pseudorandom parameter-driven testcase generation can provide more accurate results on average performance, as an algorithm can easily be evaluated against a large set of system models. Many developers use custom algorithms to generate testcases. Common approaches are to select a platform and application model manually and to assign timing properties in specified bounds (e.g. [9]) or to use a fixed platform and generate task sets (i.e. application models) automatically (e.g. [8], [10]). In both cases common corner-cases, that might only occur for e.g. certain platforms, might be neglected. To the best of our knowledge there exists no single approach that addresses all steps of parameter-driven pseudorandom system model generation. Instead parts of the generation process have been addressed.

A fairly comprehensive tool for automatic testcase generation is task-graphs for free (TGFF) [11]. TGFF generates task graphs based on a parameter set that allows detailed influence on the topology. Furthermore it allows to generate timing properties for all tasks (periods and execution times) and latency constraints on paths. Thus it addresses the three steps of application model generation, assignment of timing properties and assignment of timing constraints. TGFF has been widely used for generation of task sets (e.g. [12], [13], [14]). However, if the algorithm under evaluation targets distributed systems with communicating tasks, platform topology and application mapping may be relevant. TGFF is not able to perform these steps of model generation, though.

The steps of application model generation and assignment of timing properties have also been studied thoroughly in the scope of single and multi-processor schedulability analysis. Here benchmarking includes generation of task sets and assignment of timing parameters such as execution times and activation periods. In many cases timing parameters of fixed-size task sets are assigned so that a specific processor load is accomplished by randomly assigning activation periods and setting the execution times to match the required utilization (see e.g. [15], [9]). [16] gives an analysis of properties of commonly used algorithms to accomplish this task and discusses their respective properties with respect to the task set parameters generated. It shows that the chosen algorithm can bias the benchmark favoring one schedulability test or the other and proposes new algorithms for timing parameter generation that have been widely adopted in the community [17], [18], [19]. These findings should be considered when designing an algorithm for timing parameter generation in the described flow of pseudorandom system generation.

We are not aware of any tool capable of generating complete system models covering the entire testcase generation flow. With the SMFF framework we aim to incorporate all steps of the testcase generation into a single framework, allowing developers to generate complete system models to evaluate their algorithms. In the following we will first provide a detailed description of the system model and then address the single steps of the testcase generation flow.

IV. System Model

In this section, we will briefly introduce the main terms used throughout the paper and elaborate on the system model.

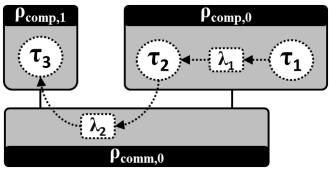


Fig. 2: System Model

The term *platform* refers to the hardware software will run on. We assume that a platform consists of set of processors (*computational resources*) interconnected by a set of communication media (*communication resources*). A platform is modeled as a bipartite graph with the two types of resources being the two types of vertices. The undirected edges between the vertices denote connectivity between two resources.

Applications are running on the platform. An application consists of a set of *tasks* and a set of communication channels (*task links*). By a task link we understand a communication entity that may be established/scheduled on a processor or communication medium. An application is modeled as a directed bipartite graph with the tasks and task links being the two types of vertices. Edges denote communication of a task via a task link while information flows in the direction of the edge. All task links have an in-degree and out-degree of 1. This model slightly deviates from the common model of task graphs, where tasks are modeled as vertices and communication between tasks as edges. We chose to modify this model to provide better expressiveness for mappings of task links. Regular task graphs can be transformed to the modified model by replacing edges by a task link vertex and two edges. We call two tasks *adjacent* to each other if a task link exists that is adjacent to both tasks. This notion of adjacency is identical to adjacency of tasks in regular task graphs.

Tasks are mapped on resources, signifying that a task is executed on that resource. Task links can be mapped on a computational or a communication resource indicating that communication is established via/on this resource.

An example system is depicted in fig. 2. It is composed of a platform of two computational and one communication resource ($\rho_{comp,0}$, $\rho_{comp,1}$ and $\rho_{comm,0}$, respectively) and one application of three tasks (τ_1-τ_3) that communicate over two task links (λ_1, λ_2). Task link λ_1 is mapped on a computational resource, while λ_2 is mapped on the communication resource.

As we are dealing with real-time systems timing properties can be defined. A task can be assigned an activation model and an execution model. An activation model may be a specification of e.g. an activation period and an activation jitter. An execution model can be specified by e.g. best-case and worst-case execution time (BCET and WCET). An execution model is also assigned to task links that are mapped on a communication medium. In addition to the timing properties a set of constraints on timing properties can be provided, e.g. constraints on end-to-end path latency.

In the following sections we will explain the single steps of system model generation. We will present possible characterizations for the system models and present exemplary algorithms for each step.

Algorithm 1 Platform Generation

INPUT: `numRes`, `CRes%`

1: `numCRes` = value from Binomial Distribution(n=`numRes`, p=`CRes%`)
2: **for** $i = 1$ **to** $i =$`numCRes`-1 **do**
3: connect comm. resources $i - 1$ and i to random comp. resource
4: **for** all unconnected comp. resources **do**
5: connect to random comm. resource
6: **for** all comm. resources connected to only one comp. resource **do**
7: connect to random comp. resource
8: Random pruning of superfluous connections

V. Platform Generation

Automatically generated platform models should resemble the architecture of real-world systems to provide a sensible basis for evaluation. Thus, in order to be applicable to various application domains an algorithm for generation of platform models should be parametetrizable to produce a wide range of different architectures. We have implemented an exemplary algorithm that allows to influence the size (i.e. number of computational resources) and degree of connectivity (i.e. communication structure between computational resources) of generated platform models.

The algorithm produces platform graphs that are one bipartite connected component. Furthermore it ensures, that each communication resource is connect to at least 2 computational resources. Algorithm 1 shows the pseudo-code for the platform generation. The algorithm requires the two parameters `numRes` and `CRes%`. `numRes` directly defines the number of computational resources in the system. It allows the user to scale the overall system size. The second parameter (`CRes%`) controls the degree of connectivity of the platform. It allows to set the average number of communication resources as percentage of the number of computational resources. E.g. if set to 5%, on average the generated platforms will have 0.05 times as many communication resources as computational resources. As a result low values of `CRes%` will tend to generate bus-like architectures, while higher values will tend to create architectures with gateways.

First, the algorithm determines the number of communication resources based on a binomial distribution (line 1). Then the algorithm ensures that all communication resources are contained in one connected component (lines 2-3), that all comp. resources are connected to at least one comm. resource (lines 4-5) and that all comm. resources are connected to at least two comp. resources (lines 6-7). In a last step (line 8) connections are pruned, such that no two computational resources are connected to the same two communication resources, if this does not violate any of the required platform criteria.

VI. Application Generation

As platform models, also application models are highly diverse for different domains (e.g. highly parallel applications vs. sequential task graphs). TGFF [11] already provides sophisticated algorithms for parameter-driven task graph generation allowing to reflect this diversity. It has been used extensively in various projects (e.g. [12], [13], [14]). The exemplary algorithm of SMFF is based on TGFF but extends the functionality by support for cyclic task graphs. The remaining parameters are identical to the old algorithm of TGFF (for a complete description of TGFF's functionality refer to [20]).

Algorithm 2 Application Generation

```
INPUT: numTasks, diffNumTasks, taskMaxDegrIn, taskMaxDegrOut,
    cyclicGraph
 1: get task graph from TGFF(numTasks, diffNumTasks, taskMaxDegrIn,
    taskMaxDegrOut)
 2: if cyclicGraph then
 3:     undirect edges
 4:     find cycles
 5:     redirect edges
 6: convert task graph to application model
```

Algorithm 2 shows the application model generation in pseudo-code. The first four parameters (`numTasks`, `diffNumTasks`,`taskMaxDegrIn`, `taskMaxDegrOut`) are passed directly to TGFF (line 1). TGFF can only create non-cyclic task graphs. If cycles are allowed in the created applications (parameter `cyclicGraph`, line 2), the direction of all edges of the task graph is deleted (line 3). A cycle search is performed on the undirected graph (line 4) and edges are re-directed (line 5). In a final step (line 6) the task graph is transformed to the framework's application model.

VII. Application Mapping

The application mapping completes the structural definition of a system model and determines the degree of distribution of an application on the platform. Applications may be either clustered, i.e. all tasks are located on only a few resources, or widely spread, i.e. tasks are distributed across many resources. However some domains may have very specific requirements on application mapping. E.g. if the generated system model should resemble a client-server setup, a path of the application graph should start and end on the same resource while some intermediate task has to be mapped on a different resource.

The example mapping algorithm is tailored to generate mappings for "sensor-actuator-like" applications. More specifically, it only maps two tasks of an application on the same resource, if they are adjacent in the application model. At the same time it tries to distribute the tasks of the application across several resources. This mapping is performed by a probabilistic algorithm that enforces the first criterion and lets the user control the degree of distribution with a parameter. The algorithm only supports chains of tasks without forks and joins. Its pseudocode is shown in Algorithm 3.

As long as the application is not completely mapped (line 2) the algorithm alternatingly selects the first/last task of the task chain, that is not yet mapped (lines 3-6). For the selected task the set of resources, that this task can be mapped on, is calculated based on distances in the platform graph and the distances to already mapped tasks of the application graph (line 7). In the following step a probability with which the task is mapped to a resource is calculated for every resource in this set (line 8). This step is based on a weighted random number generator, which selects a value from a given set with a probability corresponding to the value's weight (non-normalized probability mass). The probabilities for all other resources are initialized with 1 and are then modified by the three parameters that are passed to the mapping algorithm. `kPredecessor` and `kSuccessor` are factors applied to the weight of a resource, if the predecessor or successor of the task to be mapped, is mapped on a resource (line 8). The third parameter `kResDist` is applied only to the end

Algorithm 3 Application Mapping

```
INPUT: kPredecessor, kSuccessor, kResDist
 1: temp=0
 2: while not all tasks mapped do
 3:     if (temp++)%2==0 then
 4:         get first unmapped task of task chain
 5:     else
 6:         get last unmapped task of task chain
 7:     calculate set of possible resources
 8:     calculate probability of all possible resources
 9:     if temp==2 then
10:         apply kResDist to probabilities
11:     map task on resource based on probability distribution
```

of the task chain. Each resource's weight is multiplied by $\texttt{kResDist}^{\text{distance}}$, where distance is the distance to the resource that the first task was mapped on (line 10). Thus, `kResDist` allows to control the degree of distribution of an application. In a final step a resource is chosen from the set of possible resources based on probability mass and the task is mapped to that resource.

VIII. Definition of Real-time Properties and Constraints

The previous three sections have elaborated on the algorithms for the structural definition of a system model. Now, we focus on the remaining steps of timing property and scheduling parameter assignment and generation of constraints. Depending on the developer's needs these three steps may be closely related; e.g. if testcases are required where some constraints are violated, the assignment of timing properties and scheduling parameters have to be performed beforehand.

A. Assignment of Scheduling Parameters

Along with the framework we provide an exemplary algorithm for assignment of scheduling parameters for SPP scheduled resources. Execution priorities for tasks and task links are assigned randomly. Execution priorities that were set beforehand are left unchanged.

B. Assignment of Timing Properties

In this step of testcase generation the timing properties of all tasks and task links are defined. In the example algorithm of SMFF this comprises assignment of an activation period, an activation jitter, BCET and WCET . SMFF uses the UUniFast algorithm presented in [16] for this step. UUniFast assigns task execution times, such that the resource utilization assumes a specified value, while the distribution of the task execution times is uniform.

In the SMFF implementation of UUniFast activation periods are assigned uniformly in a specified interval [`minActPeriod`, `maxActPeriod`]. The user furthermore specifies an interval for resource utilization [`minResU`, `maxResU`]. From this interval a random value is chosen for each resource and the task execution times are assigned according to UUniFast to achieve that resource utilization. BCETs are assigned as user-specifiable percentage of WCETs [`bcetPercentage`].

C. Assignment of Timing Constraints

The assignment of timing constraints for a testcase system model is of particular importance when evaluating algorithms for optimization or design space exploration, as it may significantly influence the number of feasible system configurations

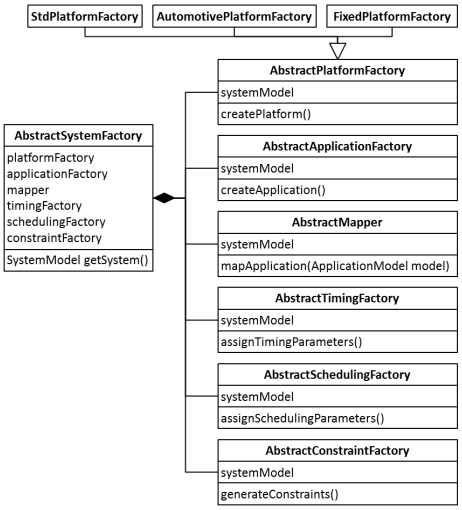


Fig. 3: Organization of System Generation Stages

(i.e. configurations that do not violate any constraint).

The implemented algorithm is again targeted at sensor-actuator-like applications. It defines paths from the first task to the last task of a task chain. The value of the constraint is defined as multiple of the sum of all WCETs along the path. This factor - the constraint *laxity* - is randomly selected from a user-defined interval [`minLaxity`, `maxLaxity`]. Smaller laxity values result in more tightly constrained systems.

IX. Implementation and Modularity

Until now we have focused on the testcase generation algorithms of the SMFF framework. However, as previously indicated, requirements on generated system models may be diverse, depending on application domain, scheduling algorithm and algorithm under evaluation. Thus the presented implementations of the generation steps may not be suitable for every user and the system model might not be sufficient. To account for the diverse requirements the testcase generation framework is implemented in a modular way, allowing extension of the data structures as well as replacement of all generation steps. The framework was implemented in Java for ease of development and platform independence. The next paragraphs give a short overview of the software architecture of SMFF, highlighting the aspects of modularity and extensibility.

A. Model Generation Infrastructure

In order to enable a flexible combination of the system model generation steps described in this paper, SMFFs generation logic is based on an aggregation of factories, each responsible for one step of the generation process. For an overview refer to fig. 3. All factories are grouped in a central system generation factory orchestrating the order of the single generation steps. The SMFF model generation core library as depicted in fig. 3 does not provide any generation logic, but merely defines the relationship between the single factories and their APIs to ensure seamless integration of different implementations of the model generation stages. An actual implementation, as the one discussed in this paper needs to provide an implementation for each of the factories.

The algorithms described in this paper are shipped with

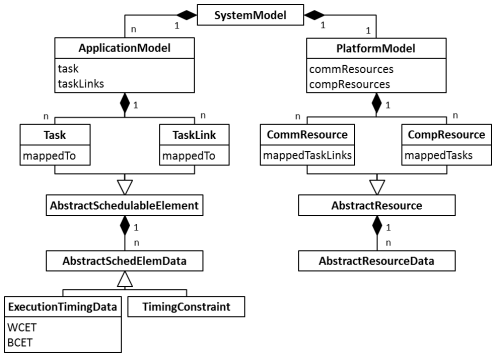


Fig. 4: System Model and Extension Points

SMFF as standard implementations, e.g. StdPlatformFactory. As an example, fig. 3 shows two additional possible implementations of a platform factory (AutomotivePlatformFactory, FixedPlatformFactory), which could be integrated into the flow. This shows that a user of the tool may replace e.g. the standard platform generation logic with a platform factory specific to his own requirements, while using the standard implementations of the other factories for the remaining steps.

Similarly, e.g. the default order imposed on the stages of system model generation can be altered by replacing the standard implementation of the system factory.

B. Model Representation

Next, we present the system model data structure as shown in fig. 4. It consists of a platform model and multiple application models. Platform models are comprised of computational and communication resources, represented by the CommResource and CompResource class respectively. Similarly, application models consist of tasks and task links, each represented by a distinct class. Additionally, application and platform models manage adjacency information about their parts defining the bipartite graphs described in section IV. A mapping is specified as a relation between task or task links and communication and computational resources. In the diagram this is indicated by the relevant local variables.

In order to reflect the flexibility of the model generation framework also in the system model data structure, all model elements allow the attachment of additional classes for extension of data and functionality without modification of the basic data structure. Fig. 4 shows that application as well as platform model components may be associated with a set of data elements. Although not shown in the figure, data extension points also exist for the system, application and platform models, thus enabling a very flexible extension of the model depending on the specific use-case of SMFF.

C. Framework Extensions

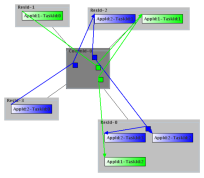
This flexible data structure allows usage of the generation framework for multiple purposes. For example, we implemented an interface to the performance verification tool SymTA/S [21] to allow verification of timing constraints. This also allows to e.g. only generate systems that are schedulable. All relevant data and functionality needed to transform the system model to a SymTA/S compliant representation as well as retrieving data from the SymTA/S tool is encapsulated in data extensions of model elements. Another existing extension

Algorithm 4 Sample System Model Generation

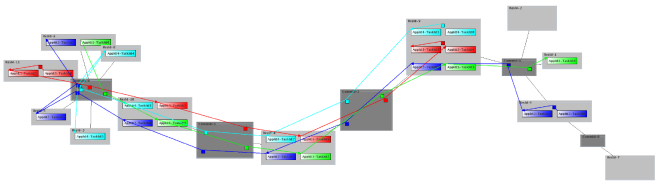
```
INPUT: systemFactoryData
1: // create system factory
2: StdSystemFactory systemFactory = new StdSystemFactory(systemFactoryData);
3: // create new system model
4: SystemModel systemModel = systemFactory.generateSystem();
5: // create XML file
6: new ModelSaver("SystemModel.xml").saveModel(systemModel);
7: // create PDF file
8: PdfPrinter.convertToPdf(systemModel, "systemGraph.pdf");
```

Param. Set	1	2
numRes	4	12
Cres%	35%	35%
minTasks	2	3
maxTasks	4	7
kPredecessor	3.0	3.0
kSuccessor	3.0	3.0
kResDist	1.5	1.5

(a) Factory Parameters



(b) Small System



(c) Large System

Fig. 5: Example Systems generated with SMFF

is a *Visualization Plugin* to the SMFF framework. It enables the display of a graph representation of the system model as well as the export to a pdf file. This allows the user to quickly grasp the platform and application graphs and the application mappings of the generated system models. The *XML Load/Store Plugin* completes the SMFF model generation suite, allowing easy integration with other tools.

X. Testcase Generation Example

In this section, we give a brief example on how to use SMFF in practice. We assume that all necessary factories for the system generation stages are present. We used the algorithms specified earlier on in this paper for generation of the example systems shown below.

Algorithm 4 shows the necessary code to generate a system from a set of parameters as needed for the different model generation stages, which we assume to be given. In order to generate a system model, one merely needs to instantiate a system factory, in this case the StdSystemFactory supplied with SMFF (line 2). A system model is generated each time the `generateSystem()` function of the factory is called (line 4). The following lines show the code necessary to save the model to an XML file (line 6) and create a pdf file (line 8) containing a graphical representation of the system.

Fig. 5 shows two example systems that have been generated by SMFF and exported using the visualization plugin. The relevant parameters to the system model generation algorithms as described in the previous sections are summarized in figure 5a. The parameters in column 1 resulted in the system model shown in fig. 5b, the model shown in fig. 5c corresponds to the parameter set in the second column. Note that multiple application models were generated for both systems. Each application model is depicted in a separate color.

XI. Conclusion

In this paper we have presented *System Models for Free* (SMFF) - a framework for parameter-driven generation of models of distributed real-time systems. SMFF can generate completely specified system models, including specification of platform architecture, of application models, mapping of applications and definition of timing and scheduling parameters and timing constraints.

As illustrated with examples, the user can easily and quickly generate pseudorandom system models for use in his field of application, thanks to supplied standard implementations of parameter-driven factories. More advanced users can take advantage of the flexible infrastructure of SMFF, by replacing implementations of system generation steps or extending the system model to tailor SMFF to fit their specific needs.

If you would like to use SMFF in your project, feel free to contact any of the authors or visit http://smff.sourceforge.net.

References

[1] M. Neukirchner, "System models for free (smff)," Internet. [Online]. Available: http://smff.sourceforge.net

[2] M. Neukirchner, S. Stein, and R. Ernst, "A lazy algorithm for distributed priority assignment in real-time systems," in *Proc. of 2nd IEEE Workshop on Self-Organizing Real-Time Systems (SORT)*, 2011.

[3] S. Perathoner, E. Wandeler, L. Thiele, A. Hamann, S. Schliecker, R. Henia, R. Racu, R. Ernst, and M. G. Harbour, "Influence of different abstractions on the performance analysis of distributed hard real-time," *Design Automation for Embedded Systems*, vol. 13, pp. 27–49, 2009.

[4] J. Real and A. Crespo, "Mode Change Protocols for Real-Time Systems: A Survey and a New Proposal," *Real-Time Systems*, vol. 26, pp. 161–197, 2004.

[5] J. G. García and M. G. Harbour, "Optimized priority assignment for tasks and messages in distributed hard real-time systems," in *Proc. of the IEEE Workshop on Parallel and Distributed Real-Time Systems*, 1995.

[6] T. Cucinotta and L. Palopoli, "QoS Control for Pipelines of Tasks using Multiple Resources," *IEEE Trans. on Computers*, vol. 59, pp. 416–430, 2010.

[7] A. R. Weiss, "The standardization of embedded benchmarking: pitfalls and opportunities," in *Int'l. Conf. on Computer Design (ICCD)*, 1999.

[8] T. D. ter Braak, P. K. F. Hölzenspies, J. Kuper, J. L. Hurink, and G. J. M. Smit, "Run-time spatial resource management for real-time applications in heterogeneous mpsocs," in *Proc. of DATE'10*, 2010.

[9] B. Andersson, S. Baruah, and J. Jonsson, "Static-priority scheduling on multiprocessors," in *Proc.. 22nd IEEE Real-Time Systems Symp. (RTSS)*, 2001.

[10] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "LITMUS^{RT} : A Testbed for Empirically Comparing Real-Time Multiprocessor Schedulers," in *RTSS*, 2006.

[11] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proc. of the 6th Int'l. workshop on Hardware/software codesign (CODES/CASHE)*, 1998.

[12] V. Kianzad, S. Bhattacharyya, and G. Qu, "Casper: an integrated energy-driven approach for task graph scheduling on distributed embedded systems," *Application-Specific Systems, Architecture Processors, 2005. ASAP 2005. 16th IEEE Int'l. Conf. on*, pp. 191–197, 2005.

[13] M. Schmitz, B. Al-Hashimi, and P. Eles, "Energy-efficient mapping and scheduling for dvs enabled distributed embedded systems," in *Proc. of the conf. on Design, automation and test in Europe (DATE)*, 2002.

[14] L. Shang and N. K. Jha, "Hardware-software co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable fpgas," in *VLSI Design*, 2002.

[15] E. Bini, G. Buttazzo, and G. Buttazzo, "A hyperbolic bound for the rate monotonic algorithm," in *Proceedings of the 13th Euromicro Conference on Real-Time Systems*, 2001.

[16] E. Bini and G. Buttazzo, "Measuring the performance of schedulability tests," *Real-Time Syst.*, vol. 30, pp. 129–154, 2005.

[17] E. Bini, T. H. C. Nguyen, P. Richard, and S. Baruah, "A response-time bound in fixed-priority scheduling with arbitrary deadlines," *Computers, IEEE Transactions on*, vol. 58, pp. 279–286, 2009.

[18] V. Pollex, S. Kollmann, K. Albers, and F. Slomka, "Improved worst-case response-time calculations by upper-bound conditions," in *DATE*, 2009.

[19] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *Computers, IEEE Transactions on*, vol. 58, pp. 1250 –1258, 2009.

[20] K. Vallerio, *Task Graphs for Free (TGFF v3.0)*, April 2008.

[21] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst, "System level performance analysis - the symta/s approach," *Computers and Digital Techniques, IEE Proc. -*, vol. 152, pp. 148–166, 2005.

On the Gap between Schedulability Tests and Automotive Task Model

Saoussen Anssi[1], Stefan Kuntz[1], Sébastien Gérard[2], François Terrier[2]
[1]*Continental Automotive France SAS, PowerTrain E IPP*
1 Avenue Paul Ourliac - BP 83649, 31036 France
{saoussen.ansi, stefan.kuntz}@continental-corporation.com
[2]*CEA LIST, Laboratory of model driven engineering for embedded systems,*
Point Courrier 94, Gif-sur-Yvette, F-91191 France
{ sebastien.gerard, francois.terrier}@cea.fr

***Abstract*- In this paper, we study the adequacy of available schedulability tests for monoprocessor fixed-priority systems to enable performing scheduling analysis for automotive applications. We show that, in spite of the work carried out during the last decade to enhance these tests in order to support more realistic task model, a gap still exists between the task model considered in these tests and automotive task model. However, we claim that an extension of these tests is possible to support some of the uncovered automotive features. The aim of this study is to raise discussion and make researchers involved in the development of such schedulability tests be aware of the gap still existing between current schedulability tests and automotive task model. The study is illustrated by showing the concrete challenges faced when applying scheduling analysis to a real automotive case study.**

I. INTRODUCTION

Today, embedded automotive systems often involve hard real-time constraints intended to ensure full system correctness [1]. Power train and chassis applications, for example, include complex (multi-variable) control laws, with different sampling periods, for use in conveying real-time information to distributed devices. One hard real-time constraint controlled in power train applications is ignition timing, which varies with engine position. The latter is defined by a sporadic event characterizing the flywheel zero position. End-to-end response times must also be bounded, since a too long control loop response time may not only degrade performance, but also cause vehicle instability. These constraints have to be met in every possible situation.

Automotive software development costs are sharply impacted by wrong design choices made in the early stages of development but often detected after implementation. Most timing-related failures are detected very late in the development process, during implementation or in the system integration phase. Timing verification is usually addressed by means of measuring and testing rather than through formal and systematic analysis. For this reason, innovative and complex functionalities are not implemented in a cost-efficient way[1]. The benefits of defining an approach that permits early timing verification for automotive systems are thus obvious. Such an approach would enable early prediction of system timing behavior and allow potential weak points in design to be corrected as early as possible

In this context, we aim at developing an approach that enables automotive system designers predicting early the timing behavior of a system before the costly implementation phase of a project begins.

Quantitative analysis techniques (such as scheduling or performance analysis) are good candidates for analyzing non-functional properties for automotive applications. Using these techniques, designers could detect infeasible real-time architectures, and therefore prevent costly design mistakes, while providing an analytical basis for assessing the design tradeoffs associated with resource optimization. For these reasons, in our approach, we suggest to use the scheduling analysis as an early verification technique for automotive systems.

Task scheduling and schedulability analysis have traditionally been the most studied topics within the field of real-time systems [2]. The problem of scheduling fixed-priority task set with hard deadlines on a uniprocessor was first studied by Liu and Layland. They gave a worst case performance analysis of the rate monotonic scheduling algorithm [3]. Liu and Layland work has come through the work of a large group of researchers who extended it in a number of ways. For instance, many necessary and sufficient feasibility tests have been developed to predict the schedulability of a

[1] These statements are based on the study of current automotive software development practices and particularly in the case of Continental

task set. These tests are based on the calculation of the worst-case response time of a task, which is the longest time between the arrival of a task and its subsequent completion. The feasibility of the task can be checked by comparing its worst-case response time to its deadline.

Adapting these schedulability tests to enable scheduling analysis for automotive applications has been tackled by some recent academic studies [4] but it is still in its very beginning. Results from these works will be discussed later in this paper. Some commercially available scheduling analysis tools, such as SymTA/S [5] are said to be based on techniques extending such feasibility tests to take into account automotive task model specificities. However, information about the algorithms used in these techniques is not publicly available.

In this paper, we aim at determining the extent to which some of these tests can be used to perform scheduling analysis for automotive applications. This means answering the following questions: to what extent the task model considered in these tests do map a realistic automotive task model. To answer this question, we firstly propose a characterization of an automotive task model; we list the most important features that should be considered when developing a schedulability test for such task model. Then, we show to what extent the available feasibility tests satisfy these features. Although, we show that some features are already covered by some schedulability tests, we conclude that a gap still exists between automotive task model and the assumptions considered in such feasibility tests. However, we claim that an extension of some of these tests is possible to support some of the uncovered automotive features.

The paper is organized as follows. In section II, we give a characterization of automotive task model. Then, in section III we present a brief state-of-the art of the available schedulability tests and show to what extent they cover the automotive task model features. Section IV is dedicated to highlight the challenges met to apply schedulability analysis to automotive systems. This is done through performing scheduling analysis to an automotive use case using an open source tool that implements some of the studied schedulability tests.

II. AUTOMOTIVE TASK MODEL

In this section, we present a characterization of automotive task model. We consider task model of monoprocessor applications based on OSEK/VDX, i.e. with a fixed priority scheduling. The features listed in this section have been collected based on the author's experience in the automotive domain and particularly in Powertrain context at Continental.

Automotive task model is characterized by the following features:

- **Arbitrary deadlines**: Automotive tasks may have deadlines less, equal or greater than their periods. Deadlines greater than periods are used for functions that are distributed over several processors. In this case, the deadlines of these functions are a sort of end-to-end deadlines that can be greater than function periods.
- **Offsets/variable offsets**: In an automotive application, a task start may be delayed by a value called offset or phasing. The particularity of some automotive tasks is that their phasing depends on the variable engine speed.
- **Preemptive and cooperative tasks:** In automotive applications, there are two kinds of tasks: Preemptive tasks and cooperative tasks. Preemptive tasks can be interrupted by higher priority tasks at any time and can interrupt any lower priority task at any time of its execution. Cooperative tasks can be interrupted by higher priority cooperative tasks only in predefined points called schedule points. Figure 1, shows an example of a system with preemptive and cooperative tasks. Task T1 is a preemptive task having the highest priority, task T2 and T3 are both cooperative tasks, T2 has got higher priority than T3. As the figure shows, T2 waits until the schedule point of T3 to start executing while T1, being preemptive, interrupts T2 before its schedule point.

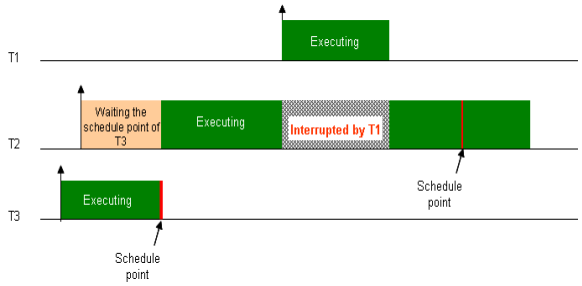


Figure 1: Preemptive and Cooperative Tasks

Cooperation is needed in automotive task model to ensure data consistency while avoiding a long blocking time of higher priority tasks that may result from fully non-preemptive tasks.

- **Same priority level:** In automotive task model, different tasks may have the same criticality and hence the same priority level is assigned to them. To schedule these tasks the FIFO (First In First Out) protocol is used. This means that if two tasks having the same priority are waiting to start ex-

ecuting, the task that was activated first will start its execution first.

- **Dependant tasks:** In automotive task model, task dependency may result from the use of shared resources. It may also result from precedence relationship between tasks called "task chaining" where the activation of a task is triggered by the termination of its predecessor. In Powertrain applications, task chaining is needed to enable e.g. the activation of timing tasks by the OS clock where this clock is modeled as a periodic task that triggers other tasks at the end of each instance of it.
- **Heterogeneous recurrences:** In automotive task model, tasks can be time-triggered or event triggered. Event triggered tasks are activated by the arrival of events that can be sporadic or singular (arrives only once). Time-triggered tasks are periodic tasks that are activated at predetermined points in time. In automotive, there are two kinds of periodic tasks, timing tasks and engine-synchronous tasks. Timing tasks have timing recurrences (e.g. 1ms, 10ms, etc). Engine-synchronous task recurrences are expressed in engine angle degree rather than time (e.g. 2°crk). In fact these recurrences depend on the Camshaft and Crankshaft positions that vary with the engine speed (*The camshaft is the element of the engine that allows the opening and the closure of intake and exhaust valves. The crankshaft is the part of the engine that translates reciprocating linear piston motion into rotation*). Hence, expressing the period of such tasks in time depends also on the engine speed. For instance, for a 6 cylinder system, a task that should be activated each 120°crk has got a recurrence of 3.3ms at 6000rpm and 13.33ms at 1500rpm. This variable aspect of recurrence should be taken into account when designing a schedulability test for such task model.
- **Changing execution profile:** The execution time of some automotive tasks depend of the engine speed. For instance, some tasks are deactivated beyond a certain speed (i.e. their computation time becomes zero). Hence, to analyze the real time behavior of such systems, the schedulability test used should take into account task variable execution times.
- **Self-suspending tasks:** During its execution, a task may suspend itself to wait for one or more event.
- **System overheads:** In automotive task model, overheads may results from: the computation time required to activate a task, the computation time required to schedule the tasks, the computation time to terminate a task and reschedule and finally the computation time to lock and unlock a resource.

III. ADEQUACY BETWEEN SCHEDULABILITY TESTS AND AUTOMOTIVE TASK MODEL

This section studies to what extent, the above-mentioned task model features are supported by the available schedulability tests. First, we present a brief historical review of the development of these tests (we focus on tests developed for fixed priority systems). In the second part we discuss to what extent the automotive task model features are supported by these tests.

A. Schedulability Tests: Brief historical review

In this section, we present a historical review of the most known results achieved within schedulability test development for fixed-priority monoprocessor systems

In 1973, Liu and Layland published a paper on the scheduling of periodic tasks that is generally regarded as the foundational and most influential work in fixed priority real time scheduling theory [3]. They considered the following assumptions: 1) all tasks are periodic, 2) all tasks are released at the beginning of period and have a deadline equal to their period, 3) all tasks are independent, i.e., have no resource or precedence relationships, 4) all tasks have fixed computation time or, at least, an upper bound on their computation time which is less than or equal to their periods, 5) no task may voluntary suspend itself, 6) all tasks are fully preemptible, 7) all overheads are assumed to be 0, 8) there is just one processor. Based on this model, Liu and Layland gave a sufficient utilization-based condition for the feasibility of a fixed priority task set scheduled with the rate monotonic algorithm [3]. They proved that a set of n periodic tasks, each having a computation time C_i and a period T_i is feasible with this algorithm if

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n\left(2^{\frac{1}{n}} - 1\right)$$

Due to the limitations of Liu and Layland test (pessimistic condition, unrealistic task model with deadlines equal to periods, task priorities have to be assigned according to the rate monotonic policy) more complex feasibility tests were developed to address the above limitations. In 1987, Lehoczky et al. [6] gave the first exact schedulability test for the Liu and Layland task model. Concurrently, another group of researchers looked at the problem of determining the worst case response time of a task. Joseph and Pandya [7] and

Audsley et al. [8] developed independently an algorithm to compute the worst-case response time R_i of a task τ_i as the least-fixed-point of the following recursive equation:

$$R_i = C_i + \sum_{j=1}^{i-1} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

In 1982, Leung [9] considered fixed priority scheduling of a set of tasks with deadlines less than their periods. Lehoczky [10] considered another relaxation of the Liu and Layland model to permit a task to have a deadline greater than its period. The Lehoczky approach uses the notion of "busy-period". A "level i busy period" is defined as the maximum time for which a processor executes tasks of priority greater than or equal to the priority of task i. Lehoczky shows how the worst-case response time of a task i can be found by examining a number of windows, each defined to be the length of the busy period starting at the window, and each window starting at an arrival of task i. In the early 1990, Tindell [11] extended the Lehoczky response time analysis providing an exact test for tasks with arbitrary deadlines.

A further relaxation of Liu and Layland task model is to permit tasks to have specified offsets (phasing). Tindell proposed in [12] a test for fixed priority tasks in which task offsets can be taken into account. This test has been later extended to by Palencia and Gonzalez [13].

Wang and Saksena [14] introduced a feasibility test where they take into account non-preemptible tasks in addition to preemptible ones.

B. Schedulability Tests Evaluation

As shown in the previous section, many feasibility tests have been developed to extend the Liu and Layland original test and be closer to a realistic task model. In this section, we show to what extent the automotive task features presented previously are supported by these tests. We give also some suggestion of possible extensions of some of these tests to support the uncovered features.

- **Arbitrary deadlines**: For automotive task model, the schedulability test should support tasks with deadlines less, equal or greater than periods. The Leung [9] test remains incomplete as it assumes that deadlines are either less or equal to periods. It is the same case for the Lehoczky [10] test as it considers only tasks with deadlines greater than periods. In addition, Lehoczky test restrict all tasks to have $D_i = kT_i$ where k is constant across all tasks. However, the Tindell test [11] for tasks with arbitrary deadlines can be used as it allows deadlines to be less, equal or greater than periods. In addition, unlike the test proposed by Lehoczky, the analysis given by Tindell places no restrictions upon the relative task periods. Hence, the Tindell test described in [11] supports the arbitrary deadlines feature of automotive task model. However, to use this test, the remaining features should also be covered
- **Offsets/variable offsets**: The Tindell test [11] described above assumes that all tasks are released at beginning of period. Hence this test is incomplete and does not allow accurate analysis for automotive tasks with offsets. However, Tindell developed a second test [15] that allows tasks to have offsets. In this test the concept of transaction is introduced. A transaction is a collection of related tasks and it has got a period. A member task of a transaction no longer has a specific period associated with it; it has rather an *every* attribute e_i whereby it is permitted to run at most every e_i transaction invocation. Task offsets are defined with reference to the start of the transaction and are assumed to be static and less than the period of the transaction. This test can be used for automotive tasks with static offsets. However, as mentioned previously, the offsets of some automotive tasks vary from one activation to the next, making, hence, the Tindell test inefficient to analyze such tasks. For such tasks, the test developed by Gonzalez and Palencia [13] can be used. This test extends the Tindell test for static offsets to allow analysis for tasks with static and dynamic offsets. It also enhances the Tindell test by allowing offsets to be greater than task periods. To be able to use this test to analyze automotive task model, the test should also take into account the feature described above, such as the mixed preemptive and cooperative scheduling.
- **Preemptive and cooperative tasks:** Unfortunately, the Palencia and Gonzalez test described above [13] assumes that all tasks are fully preemptible. It is the same also for the Tindell tests. Non preemptible tasks are considered in the Wang and Saksena test [14]. They defined the concept of preemption threshold that, as for task priority, is assigned offline and remains constant at run-time. When a task is released, it is inserted to the ready queue at its priority, when the task starts executing, its priority is raised to its preemption threshold and it keeps this priority until it finishes execution. Based on this model, they gave an algorithm to calculate the worst-case response time of tasks using the busy-period technique. The task model considered in

this test is slightly different from our model in term of preemptibility. In fact, in this test, a non-preemptible task cannot be interrupted by a higher priority task until it terminates its execution. In our model, interruption of cooperative tasks by other cooperative tasks is forbidden only during the execution of task sections (code section between two schedule points). We think that such behavior can be captured with the notion of preemption threshold in a quite generic way by bringing this preemption threshold at the task section level rather than task level. This same approach is used in Hladik et al. test [4] to calculate the blocking time of a task due to the locking of a shared resource by a lower priority task in a simplified OSEK task model (more details about the Hladik test will be given in next section). Hence in our case the blocking time of a cooperative task T1 is the execution time of the longest section of a lower priority task T2 if this section has got a preemption threshold equal or greater than the priority of T1. In conclusion, the mixed preemption-cooperation aspect of automotive task model can be covered by extending the Wang and Saksena test. In addition, this test takes into account tasks with arbitrary deadlines as it is based on the extension of previous work such as the Tindell test for arbitrary deadlines. However, this test does not support tasks with static or dynamic offsets.

- **Same priority level:** All the tests described above have been developed assuming different priority levels for different tasks except the Hladik et al. test [4]. In this test, Hladik gives a schedulability test for a simplified OSEK/VDX task model. In this model, tasks can be preemptive or not with arbitrary deadlines and shared resources. Moreover different tasks are allowed to have the same priority level and the FIFO scheduling is used for these tasks. The test uses the busy-period technique to calculate the worst-case response time for tasks. To calculate this response time, the test calculates first the worst case start time of an instance of a task. In this start time, the delay due to the execution of the tasks having the same priority level and activated before the considered task instance is added. The Hladik test covers hence more features then the previously described tests. However, this test does not take into account tasks with offsets. In addition, the test does not deal explicitly with cooperative tasks. Nevertheless, as mentioned before, this test adapts the preemption threshold notion to calculate blocking time due to shared resource locking. We think that this same approach can be used to calculate the blocking time due to the non-preemptible sections of cooperative tasks.
- **Dependant tasks:** Except the Hladik test, all the above mentioned tests assume that tasks are independent. The Hladik test described in [4] deals only with task dependency resulting from shared resource use. In this test, the OSEK IPCP protocol is considered for shared resource. Based on the work of Sha et al. in [16], The test considers that the worst-case blocking time of a task T is reduced to at most the duration of at most one critical section of a lower priority task that is using a resource whose ceiling priority is higher than or equal to the priority of task T. Hladik used the preemption threshold notion to calculate this blocking time showing that a task T can be blocked by at most one critical section of a lower priority task with a preemption threshold higher than or equal to the priority of T. In [17], Hladik et al. extend this test to take into account task chaining. However the test proposed is not an exact one since it presents only an upper bound of the worst-case response time.
- **Heterogeneous recurrences:** As mentioned previously, automotive task model contains both sporadic and periodic tasks. Periodic tasks have either timing recurrence or angular recurrence. Most of the schedulability tests presented above assume that tasks are purely periodic or sporadic with a known minimum inter-arrival time. Hence automotive periodic and sporadic tasks are taken into account. However, the problem remains unsolved concerning engine-synchronous tasks with angular recurrences. During engine running, the period of these tasks vary significantly making it impossible to consider a mean inter-arrival time for them. Moreover, these tasks present usually hard deadlines and have quite high priorities. In scheduling theory, tasks with such features are called aperiodic tasks. Among the work performed in system scheduling, most works focused on the problem of how to design feasible systems with such tasks [18], but none of the schedulability tests calculating the worst-case response times has considered this kind of tasks. As for a fixed engine speed the timing recurrence of engine-synchronous tasks is constant (hence they become purely periodic), one can think to use the above-mentioned schedulability tests to perform schedulability analysis at a fixed engine speed. However, the worst-case response times found for a particular speed are not valid for other engine speeds. Extending the available schedulability tests to cover all the engine

speeds seems to be intractable and inefficient. The main reason is that the computation of the response time is not an exact computation when the model becomes too complex.

- **Changing execution profile:** As mentioned previously, during engine running, the execution time of some tasks varies significantly as some treatments are performed or not depending on the current engine speed. Tasks with varying execution times have never been considered by the schedulability tests. All of them assume that tasks have a known maximum computation time. Extending these tests to take into account this variability in execution times seems also to be intractable.
- **Self suspending tasks:** self suspending tasks are not taken into account by schedulability tests. All of them assume that no task may voluntary suspend itself. However extending some of these tests to take into account this feature seems to be feasible. For example, one can use the transaction notion used in Tindell and Gonzalez tests [12, 13]. This way a self-suspending task can be modeled as a transaction composed of two tasks, the first task corresponds to the code before the suspension and the second task to the code after the suspension. As the activation of the second task depend on the computation time of the first one, the second task will have a variable offset. The work of Gonzalez on variable offsets can thus be used.
- **System overheads:** System overheads are assumed to be null in all above-mentioned tests. In [19], Bimard and Goerge give an algorithm to calculate the worst case response time of tasks where interference of kernel overheads are included into worst-case response time computation equation. In this test, overheads resulting from resource sharing are not considered as it assumes that tasks are independents. In addition this test does not take into account tasks with offsets as well as self suspending tasks.

C. Conclusion

As shown in the previous section, some features of the automotive task model are already handled by some of the available schedulabilitiy tests such as arbitrary deadlines and offsets. For other features such as engine-synchronous tasks and varying execution times which are quite important, they are unfortunately not taken into account. Extending the presented schedulability tests to end up with a final test covering all the necessary features seems to be quite challenging. On one hand, each of these tests focus only on some features and neglects the other (For instance tests dealing with offsets do not consider cooperative tasks while tests dealing with non-preemptive tasks do not consider task offsets, etc). Hence one should find a way to combine the different techniques suggested in these tests. This means that ending up with an exact test is not guaranteed. Besides, engine-synchronous recurrences and variable execution times have never been tackled in these tests. This makes us think that extending these tests to cover these features is intractable. Although these evaluation results show that some automotive task model features can be covered by the extension/combination of some of these tests, they show also that a gap still exists concerning other features which are considered to be quite important in automotive applications.

IV. ILLUSTRATION

In this section, we show the concrete challenges faced when applying scheduling analysis to automotive applications. The work consists in performing scheduling analysis to an automotive use case using an open source tool, MAST [20], that is based on some of the above mentioned feasibility tests. The response times obtained are compared with values obtained from measurements done on the application using a test bench.

A. MAST overview

MAST is an open source tool that offers, for fixed priority systems, a suite of schedulability tests such as the RMA test developed by Liu and Layland [3] as well as the Gonzalez et al. test mentioned previously [13]. The tool also implements tests dedicated for distributed systems. The MAST model is based on the notion of *transaction* which represents a succession of interrelated activities. Each *activity* represents the execution of an *operation* which represent, itself a small segment of code execution. A transaction is characterized by an external event and a succession of activities. The output event of each activity in the transaction is the input event of the subsequent activity. For the first activity, the input event is the external event of the transaction. Tasks are represented by *scheduling servers* in MAST. A MAST scheduling server represents a schedulable entity in a processing resource. The allocation of operation to scheduling servers is ensured through the description of the corresponding activity.

B. Use case presentation

The system to be analyzed is a simplified configuration of a Powertrain software platform. The Table below summarizes the tasks with their properties:

Table 2. Powertrain task model

Timing tasks					
Task	**Priority**	**Period (ms)**	**Deadline (ms)**	**Type**	**Sections WCETs (ms) ****
T1_T	9	1	0.2	Preemptive	0.14
T2_T	7	5	0.5	Cooperative	{0.01}
T3_T	4	10	10	Cooperative	{0.12, 0.09, 0.11, 0.05}
T4_T	3	40	40	Cooperative	{0.38}
T5_T	2	100	100	Cooperative	{0.27, 0.22, 0.29, 0.2, 0.25, 0.12}
T6_T	3	10	20	Cooperative	{0.29, 0.14, 0.2, 0.31}
Engine-synchronous tasks					
Task	**Priority**	**Period (°CRK)**	**Deadline (° CRK)**	**Type**	**Sections (ms) ****
T1_ENG	5	120	= period	Cooperative	{0.7}
T2_ENG	5	360	= period	Cooperative	{0.02}

** The WCETs used in this example were measured using internal methods and tools that for confidentiality reasons cannot be presented here

The "section WCET" column shows either the worst case execution time of the non-preemptible sections for a cooperative task or simply the worst case execution time of a preemptive task.

The periods of T1_ENG and T2_ENG depend on the Crankshaft position (measured in °crk) that, itself, depends on the engine speed. Some treatments performed by T1_ENG are no more activated beyond 6500rpm, meaning that the computation time of this task changes beyond this speed. The engine-synchronous tasks should be executed in this order: T1_ENG then T2_ENG, we ensure this by introducing an offset of 45°crk on the activation of T2_ENG. Hence, like periods, this offset varies also with the engine speed.

The Tasks T3_T, T4_T, T5_T and T6_T are activated by the termination of T2_T. In addition, T3_T and T6_T which have the same recurrence are activated alternatively. This is ensured by introducing an offset of 5ms on the activation of T6_T.

C. Analysis challenges

To analyze this system, we choose the Gonzalez et al. test [13] that is implemented in this tool as an "offset_based" test. Although this test does not cover all the features of our model, it is the closest one among the tests available in this tool.

Analyzing this system as it is using the above mentioned tool is not possible because of the angular recurrences of engine-synchronous tasks and the varying execution time of T1_ENG. In fact, this tool uses the notion of physical time as measured by a unique time base meaning that an angular base can not be described. In addition, the chosen schedulability test does not support varying execution times. To convert the recurrence of engine-synchronous tasks in milliseconds, we are obliged to choose a fix speed for which we perform the analysis. For a 6 cylinder engine running at 6000rpm, the T1_ENG has got a recurrence of 3.33ms, T2_ENG has got a recurrence of 10ms *(the calculation of these recurrences is based on formula describing the configuration parameters of the engine that for confidentiality reasons can not be presented here).* At this speed, T1_ENG has got the execution time described in the table 1 and T2_ENG offset is equal to 1.25ms. We will hence perform the analysis with these values. However we should keep in mind that the analysis results are valid only for an engine running at this speed, which is very limiting as, during its functioning, an engine usually runs at different speeds.

To perform analysis with the "offset_based" test, we should first develop the Powertrain model analyzable with this technique. As shown in the use case description, the cooperative timing tasks are dependant as they have precedence relationships. It is the same for the engine-synchronous tasks. Hence, the first idea is to model a transaction for each kind of tasks, i.e. a first transaction for cooperative timing tasks and a second one for engine-synchronous tasks (the preemptive task will be modeled by a third transaction). However, for cooperative timing tasks, we will obtain a non-linear transaction as all cooperative timing tasks are activated by the same task T2_T (we obtain a transaction with a fork relationship). This is unfortunately not supported by the MAST offset_based technique that considers only linear transactions. We are obliged hence to relax some dependencies of the timing tasks. We will consider for example only the chaining of T3_T by T2_T and the remaining tasks will be modeled by independent transactions.

D. MAST model for Powertrain dynamic architecture

In the Powertrain MAST model, each section described in table 1 is modeled as an operation for which we specified the WCET. The tasks are modeled as scheduling servers. A first transaction is modeled to represent the interrelation between T1_ENG and T2_ENG. For this transaction, we specified an external event with 3.33 ms as period. This means that the activities of this transaction execute each 3.33ms. However we must capture also that the activity of T2_ENG is not executed in each occurrence of the transaction but rather each 10ms. To model this situation in MAST, we use a *rate divisor event handler*. Rate divisor is a kind of activity that only generates one output when a number of input events equal to the rate factor have arrived. In our case, a rate divisor with rate factor equal to 3 is then placed between the activity of T1_ENG and the activity of T2_ENG. To specify the offset on the activation of T2_ENG, we use the *offset event handler*. This is a kind of activity that generates its output event

after a time interval has elapsed from the arrival of a referenced input event. A second transaction is created in a similar way to model the chaining of the task T3_T by T2_T and all remaining tasks are modeled by independent transactions for which we specify an external event having as period the period of the task. For each transaction, timing requirements can be attached to output events. In particular each deadline described in table 1 is represented in MAST either as a *hard local deadline* (for intermediate tasks in a transaction) or *hard global deadline* (for transactions containing only one task).

E. Analysis results evaluation

To evaluate the response times obtained, we compare them with response time values that we measured using a test bench. This done by measuring the time elapsed from the activation of a task until its termination. These measurements are performed at 6000rpm and, of course, take into account all the features of the Powertrain tasks (cooperation, chaining, FIFO for similar priorities, offsets, deadlines, etc). Table 2 shows the response times from the analysis and the measurements

Table 2. Task response times from measurements and analysis

Task	Measured response time (ms)	Response time from analysis (ms)	Deadline
T1_T	0.14	0.14	0.2
T2_T	**0.9**	**0.15**	**0.5**
T3_T	2.86	1.39	10
T4_T	2.8	2.6	40
T5_T	10.2	4.66	100
T6_T	5.74	2.19	20
T1_ENG	1.73	0.85	3.33
T2_ENG	1.6	1.03	10

As the table shows, there is an obvious divergence between the measured values and the values from analysis. Especially, although the values determined from analysis are supposed to give the worst case response times, these values are lower than the measured values for all the tasks. This is due to the fact that the analysis technique used does not take into account the blocking of cooperative tasks by non-preemptible sections of lower priority cooperative tasks as well as the delay due to the execution of tasks having the same priority that have been activated earlier. Due to these reasons, the response time calculated for the task T2_T tells that the task is schedulable as the value determined is less than the deadline. However, this is not compatible with the measured value that shows that with this configuration (and taking into account the cooperative aspect) the task T2_T misses its deadlines (as the measured response time is greater than its deadline). Furthermore, we should keep in mind that the system configuration that have been analyzed is different from the original configuration that we were obliged to adapt in order to cope with the limitations of the technique and the tool.

V. CONCLUSION

In this paper, we discussed the adequacy between available schedulability tests and a realistic automotive task model. We showed that although some automotive features are already covered by some of these tests, a gap still exists between them which make the application of scheduling analysis for automotive systems a challenging task. For some uncovered features, we made some suggestions about possible extensions of available tests to take into account these features. In this paper, we illustrated the study by attempting to perform scheduling analysis to an automotive real case. This allowed us highlighting the challenges met during this task. The limitations of the used technique are also shown through the comparison of the analysis results with measurements performed on the considered use case. In this paper, the focus is made on monoprocessor systems. An important feature that should be also considered when developing schedulabiltity tests for automotive applications is the distributed aspect. Especially particular communication protocols such as CAN and Flexray should be taken into account.

REFERENCES

[1] N. Nave, F. Simonot-Lion, editors: The Automotive Embedded Systems Handbook. Industrial Information Technology series, CRC Press / Taylor and Francis, ISBN 978-0849380266, December 2008.

[2] L. Sha, T. Abdelzaher, K. Arzen, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A. Mok, Real Time Scheduling Theory: A Historical Perspective, Real-Time Systems Journal, November-December 2004.

[3] Liu, C. L. and J. W. Layland: 1973, Scheduling alghorithms for multiprogramming in a hard real-time environment.

[4] P. Hladik, A. Deplanche, S. Faucou, and Y. Trinquet, Schedulability analysis of OSEKNVDX applications, in 15th International Conference on Real-Time and Network Systems, 2007.

[5] SymTA/S website (www.symtavision.com/symtas.html)

[6] Lehoczky, J. P., L. Sha, and D. Y. Ding: 1989, The rate monotonic scheduling algorithm: exact characterization and average case behaviour'. In: Proc. 10th IEEE Real-Time Systems Symposium.

[7] Joseph, M. and P. Pandya: 1986, Finding response times in a real-time system. BCS Computer Journal 29(5), 390-395.

[8] Audsley, N. C., A. Burns, M. Richardson, and A. J. Wellings: 1991, Hard real-time scheduling: the deadline monotonic approach. In: Proc. 8th IEEE Workshop on Real-Time Operating Systems and Software. Atlanta, GA, USA,

[9] Leung, J. Y. T. and J. Whitehead: 1982, On the complexity of fixed-priority scheduling of periodic, real-time tasks. *Performance Evaluation (Netherlands)*

[10] Lehoczky, J. P.: 1990, Fixed priority scheduling of periodic task sets with arbitrary deadlines. In: Proc. 11th IEEE Real-Time Systems Symposium.

[11] Tindell, K., A. Burns, and A. J. Wellings: 1994a, An extendible approach for analysing fixed priority hard real-time tasks. *Real-Time Systems*

[12] K. Tindell, Adding Time-Offsets to Schedulability Analysis, Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994.

[13] J.C. Palencia Gutiérrez and M. González Harbour, Schedulability Analysis for Tasks with Static and Dynamic Offsets. Proceedings of the 18th. IEEE Real-Time Systems Symposium, Madrid, Spain, December 1998.

[14] Y. Wang and M. Saksena. Scheduling fixed-priority tasks with preemption threshold. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications (RTCSA'99)*, 1999

[15] K. Tindell, Adding Time-Offsets to Schedulability Analysis, Technical Report YCS 221, Dept. of Computer Science, University of York, England, January 1994

[16] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization *IEEE Transactions on Computers*, September 1990

[17] P. Hladik, A. Deplanche, S. Faucou, and Y. Trinquet, Computation of worst-case response time of OSEK/VDX applications with precedence relations. In 14th International Conference on Real-Time and Network Systems, 2006

[18] Sha, L., J. P. Lehoczky, and R. Rajkumar: 1986, Solutions For Some Practical Problems in Prioritizing Preemptive Scheduling'. In: Proc. 7th IEEE Real-Time Sytems Symposium

[19] F. Bimbard and L. George. FP/FIFO feasibility conditions with kernel overheads for periodic tasks on an event driven osek system. In *Proceeding of the Ninth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2006*

[20] M. Gonzalez Harbour, J. J. Gutiérrez Garcia, J.C. Palencia Gutiérrez, J.M. Drake Moyano. MAST: Modelling and Analysis Suite for Real Time Applications. 13rd Euromicro conference on real Time Systems (ECRTS'01). June 2001.

FORTAS : Framework fOr Real-Time Analysis and Simulation

Pierre Courbin
ECE Paris, LACSC
37 quai de Grenelle, 75015 Paris, France
Email: courbin@ece.fr

Laurent George
University of Paris-Est, LISSI
120 rue Paul Armangot, 94400 Vitry sur Seine, France
Email: lgeorge@ieee.org

***Abstract*—Research in real-time scheduling has produced a large number of algorithms with their associated feasibility conditions to respond to the increasing complexity of multiprocessors architectures. However, it is difficult to find tools able to evaluate and compare these algorithms based on simulations or on analytical tests. Our tool named FORTAS offers to facilitate the comparison between different algorithms for uniprocessor and multiprocessors real-time scheduling. Developed in Java with a programming paradigm oriented to modules and abstraction, it gives the user the opportunity to develop their own extensions. Moreover, it proposes to automate the process of comparing different algorithms: generation of sets of tasks, computation of results for each algorithm and generation of graphs for comparison. FORTAS has already been used effectively for various published papers ([1], [2], [3], [4]).**

I. Introduction

There is no need to recall the growing importance of multiprocessors architectures including multicore systems addressed by the field of real-time scheduling. These architectures have brought a lot of questions to this area, and its own set of answers: algorithms, techniques, optimizations etc.

Many solutions have been proposed by the community to meet this challenge: either for preemptive or for non-preemptive scheduling, with fixed or dynamic priority, based on a global scheduling approach, on a partitioning approach or on the recent semi-partitioning scheduling approach.

Everyone develops his idea, discovers many advantages and would like to share with the community so that it can use, understand the tricks and possibly be inspired in order to improve the original idea. But when comes the time to test the solution and present it, we are often faced to a problem: on what basis can we compare? How can we compare ourselves as fairly as possible, without bias the results by implementing the best possible pre-existing solutions? Too few common ways of generating sets of tasks used for simulations or common way to implement other solutions are existing.

The tool presented in this paper does not respond to each questions, but it proposes a perfectible, extensible, open and scalable solution for these concerns. A minimalist *Graphical User Interface* (GUI) is available for those who want fast performance and basic usage. The code is open and offered to those who want complete control and specific results. This paper presents some functionalities of this tool.

Section II presents the related work and our motivation. Then, we present the four main options identified as the most common: test a scheduling algorithm (section III), observe a scheduling (section IV), generate tasks and sets of tasks (section V) and edit and run an evaluation of performance comparison (section VI). A final section is devoted to conclusions and future work related to this tool (section VII).

II. Related work and motivations

A. Brief overview of existing tools

Several tools, commercial or free, are already available to study real-time systems. On the commercial side, the goal is usually an analysis and a complete design of a particular system. Examples are TimeWiz (TimeSys Corp.) or RapidRM (Tri-Pacific Software Inc.) based on the Rate Monotonic Analysis methodology (RMA). On the other hand, free projects proposed by the academic community generally respond to specific needs and are not always flexible or even maintained. For projects still in development, we can cite MAST [5] which proposes a set of tools to analyze and represent the temporal and logical elements of real-time systems. Cheddar [6] mainly focuses on theoretical methods of real-time scheduling and proposes a simulator and the majority of existing feasibility tests. STORM [7] defines the hardware platform and software as an XML file and then conduct simulations on scheduling. Others tools like RESCH [8], Grasp [9] or LitmusRT [10] can analyze the practical operation of a real-time scheduling on a real system such as μC/OS-II and Linux.

Each tool provides a valuable aid for the analysis of real-time systems. However, it seemed that almost all of them focus on the analysis or design of a given scheduling: given my platform, or even my set of tasks, what will be the performance or how do I have to change my system to ensure its schedulability?

FORTAS implements some of these elements but often remains far less advanced than existing tools. However, it focuses on the possibility to compare and evaluate scheduling algorithms, whether based on a theoretical analysis of feasibility or on the simulation of scheduling, without necessarily focusing on a given platform or a specific set of tasks.

B. Our motivation for developing the FORTAS tool

Like for many researchers, the tools proposed by the community does not exactly correspond to our needs. Especially, we needed to evaluate and compare algorithms based on analytical tests and not simulations. We also needed to have a simple GUI to use this tool for teaching purposes. We certainly do not meet all the needs in this area but we simply want to offer and make our work available.

Thus, this tool was first developed for analytical testing and evaluation. The Java programming language was chosen for its development efficiency and proven interoperability. Each part of the program is conceived as a module and an effort of abstraction was given to each element. Thus, the GUI is completely interchangeable and can be redeveloped by anyone without coming to interfere with the core program. Similarly, a new algorithm, a new scheduling policy, a new way to sort tasks or processors, a new partitioning heuristic, a new criterion of comparison for graphs etc can be made by adding a simple Java file to the project without further changes.

To introduce the current possibilities of the tool, we identified four main axis which will be explained in the following sections. A basic GUI has been developed to quickly test some features and understand the possibilities.

III. Test a Mono/Multiprocessors Scheduling

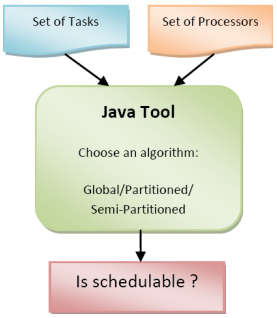


Fig. 1. Test a Scheduling

The first option is to test a scheduling algorithm. This part corresponds to analytically test if a set of tasks associated to a set of processors will be schedulable or not (see Fig. 1). Whatever the number of processors contained in the set, the tool should be able to act upon an algorithm and give a solution.

Based on the current state of the solutions, we have identified three different approaches for multiprocessors:

- The global scheduling, which consists in scheduling tasks in a single queue and allow jobs to migrate between processors, requires a global feasibility test in order to check the schedulability.
- With partitioned scheduling, we need to find a partitioning heuristic to assign tasks to processors and then to use a uniprocessor feasibility condition on each processor to decide on the schedulability of the task assigned to it. A sorting criterion for tasks and processors can be added to improve the performance of the approach.
- The semi-partitioned scheduling consists in partitioning the majority of the tasks, and allow a few to migrate between processors. In addition to partitioned scheduling, this approach needs a uniprocessor feasibility condition which takes into account the migrant tasks.

In order to obtain a modular and scalable tool, a scheduling algorithm has been split into several parts:

- A feasibility test is an interface which has to answer if a task added to a given processor is schedulable.
- A partitioning heuristic that defines how the processors should be checked in order to assign tasks.
- A criterion for sorting tasks or processors that defines the order in which they must be addressed.

A. Partitioning Heuristics

Used for partitioned and semi-partitioned approaches, partitioning heuristic was defined as an abstract class. This abstract object needs one function: according to a feasibility test, a set of processors and a task, it must return the processor able to schedule this task, if any.

Currently, four partitioning heuristics are proposed:

- First-Fit allocates the task to the first processor it fits into (according to the feasibility test). The process always starts from the first processor of the set.
- Next-Fit allocates the task to the first processor it fits into (according to the feasibility test). The process always starts from the last processor where a task has been assigned.
- Best-Fit allocates the task to the best processor it fits into so that it will minimize a quantity (according to the feasibility test, for example the utilization).
- Worst-Fit is similar to Best Fit but it selects the processor that maximizes the quantity defined in the feasibility test.

Modularity: A new partitioning heuristic can be added by deriving the abstract class. For information, the First-Fit heuristic is coded in about 10 lines.

B. Algorithm/Feasibility test

A scheduling algorithm is defined according to the multiprocessor approach used: global, partitioned or semi-partitioned scheduling. An abstract class defines the generic procedure for each approach:

- The global scheduling requires only a feasibility test on all tasks and processors.
- The partitioned scheduling sorts the tasks / processors based on criteria, then assigns them on processors according to the selected partitioning heuristic and to the uniprocessor feasibility test defined in the algorithm.
- The semi-partitioned scheduling offers several methods presented in the literature, which includes different ways to determine when and how to split tasks between

processors.

Modularity: For example, about 10 lines in a Java file are sufficient to define the partitioning algorithm which allows us to test any sort criterion of tasks and processors, any partitioning heuristic and which uses the uniprocessor feasibility test for preemptive Earliest Deadline First (EDF) scheduling based on the computation of the Load [11].

If we consider $\tau = \{\tau_1, \ldots, \tau_n\}$ a set of n sporadic tasks, $\tau_i(C_i, T_i, D_i)$ the i^{st} task where C_i is its worst-case execution time (WCET), T_i is its minimum inter-arrival time and D_i is its relative deadline, here are some feasibility tests currently available in the tool:

- EDF-LL [12]: the total utilization of the set $U_\tau \stackrel{\text{def}}{=} \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$.
- EDF-BHR [11]: $\text{Load}(\tau) \stackrel{\text{def}}{=} \sup_{t \geq 0} \frac{\text{DBF}(\tau,t)}{t} \leq 1$ with DBF (*Demand Bound Function*) represents the upper bound of the work load generated by all tasks with activation times and absolute deadlines in the interval $[0, t]$. The tool implements some optimizations to accelerate the calculation of the *Load* function such as the computation of the C-Space using the simplex algorithm proposed by George & al. in [13] or the QPA algorithm of Zhang & al. [14].
- DM-ABRTW [15]: Deadline Monotonic (DM) test based on the response-time analysis: $\forall \tau_i \in \tau$, $r_i \leq D_i$, where r_i is τ_i's *worst case response time*.
- RM-LL [12]: Rate Monotonic (RM) test based on the total utilization of the set $U_\tau \leq n(\sqrt[n]{2} - 1)$.

Here are some global and semi-partitioned scheduling algorithms currently available in the tool:

- RTA (Global) proposed by Bertogna & al. in [16]. It is a global feasibility test based on an iterative estimation of the worst case response time of each task for GlobalEDF schedulers.
- EDFWM (Semi-partitioned) proposed by Kato & al. in [17]. It splits migrants tasks in subtasks and defines a window during which a subtask should be executed on a processor.
- C=D (Semi-partitioned) proposed by Burns & al. in [18]. It splits migrants tasks in two parts: one with a C=D ($\tau_{i,1}(C, T_i, C)$, WCET equal to its deadline) and a second part with the remaining values ($\tau_{i,2}(C_i - C, T_i, D_i - C)$).
- EDF-RRJM (Semi-partitioned) proposed by George & al. in [4]. It uses Round Robin Job Migration to split migrants tasks and reduces the number of migration by using job migrations at job boundaries.

IV. View a Scheduling

The second option proposed is to allow the user to view the sequence of scheduling w.r.t time. This part is performed by an abstract object "Scheduler" which will proceed according to the rules defined by the scheduling, check deadline misses and record the jobs scheduled. A GUI proposes a graphical representation of the scheduling (see Fig. 2).

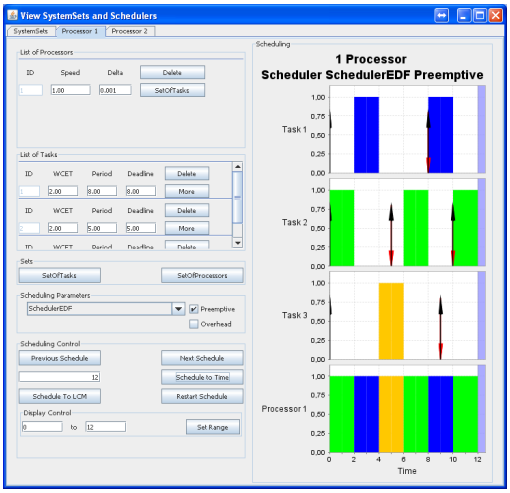


Fig. 2. GUI to display a scheduling

A. Available schedulers

Schedulers currently implemented are:

- PFair family (PF, PD^2) represents the global scheduling presented in [19]
- Arbitrary Priority Assignment chooses the active job with the highest predefined priority.
- Deadline Monotonic (DM) chooses the active job with the minimal relative deadline
- Rate Monotonic (RM) chooses the active job with the minimal period
- Earliest Deadline First (EDF) chooses the active job with the minimal absolute deadline
- Least Laxity First (LLF) chooses the active job with the minimal laxity

Each of these scheduler can then be used as mono or multiprocessors schedulers: one Java object EDF can represent the uniprocessor EDF scheduler or the $GlobalEDF$ scheduler according to the number of processors available.

Modularity: Add a new scheduling policy to the tool consists of adding an object that derives from the abstract class and only defines the function which chooses the job to be scheduled in the list of active jobs. The EDF scheduling, preemptive and non preemptive, for mono and multiprocessors, is thus a Java file of about 10 lines.

V. Generating Tasks and Sets Of Tasks

One of the challenges of a test tool for real-time scheduling is to offer a method of generating sets of tasks which give representative and reusable results for the most honest and consistent possible comparison.

We based our methods of generation of tasks and sets according to the article [20] and the UUnifast algorithm [21].

With a modular and abstract code, it is possible to use various methods of generation and various parameters such as type of task deadline or a specific probability distribution for the utilization of tasks. Sets are saved in a XML file to be loaded for others options of the tool.

A. Generating a Task

Here we present the procedure derived from [20]. UUnifast algorithm is also available. To generate a task, according to [20], several parameters are needed:

- The type of deadline, *IMPLICIT* (the deadline of each task equal its period), *CONSTRAINED* (the deadline of each task is less than or equal to its period) or *ARBITRARY* (the deadline of each task can be lower, equal or greater than its period).
- The probability distribution of the utilization of each task (such as uniform in the interval [0;1] or exponential of mean 0.5).
- The interval used to generate the values of periods and deadlines.

The generation procedure is as follows:

1) The period is generated following a uniform distribution in the defined interval.
2) The utilization of the task is generated according to the distribution selected.
3) The value of WCET is calculated based on the period and utilization of the task.
4) The value of the deadline is set to the period (*IMPLICIT*), uniformly selected between the WCET and period (*CONSTRAINED*) or uniformly selected between the WCET and the maximum value of the defined interval (*ARBITRARY*).

B. Generating Sets Of Tasks

To generate sets of tasks, several functions are available but the main procedure is also extracted from the article [20].
The following procedure needs a task generator (see section V-A), a minimum number of tasks, a maximum utilization of set of tasks and a number of sets to produce:

1) The minimum number of tasks is created based on the task generator; utilization of the set must not exceed the maximum utilization defined. This is the first set of tasks.
2) A new task is generated according to the same task generator. If it can be added to the previous set without exceeding the maximum defined utilization, it is added to create a new set. If not, return to the previous step.

These steps are repeated until the number of sets expected is reached.

VI. Edit/Run an Evaluation

This option uses the previous options (see sections III, IV and V) to automate the generation of results in order to compare various algorithms (see Fig. 3).

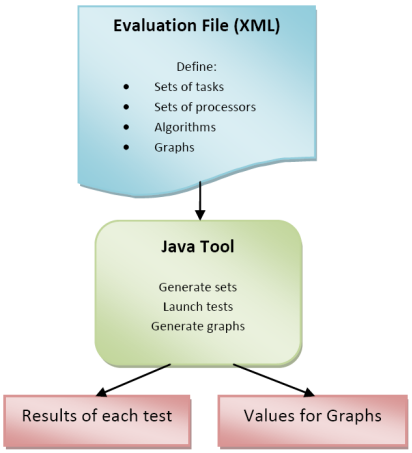


Fig. 3. Edit/Run an Evaluation

The definition of an evaluation is done in an XML file containing:

1) A list of types of sets of tasks. These sets of tasks can be defined by generation parameters according to section V-B (see section VI-A).
2) An equivalent list for sets of processors (see section VI-A).
3) A list of algorithms. For each one, we can define some settings: partitioning heuristics, criteria for sorting tasks, type of sets of tasks (previously defined in the XML at point 1) and the sets of processors to be considered (previously defined in the XML at point 2) (see section VI-B).
4) A list of graphs to be produced according to the results (see section VI-C).

A. Defining the sets

You could choose to use pre-existing sets of tasks or define generation parameters (see section V) and let the generator create the sets.

```
<EvaluationSetOfTasks name="Deadline_IMPLICIT__Distrib_UNIFORM"
        autoPath="true" path=".\SetOfTasks" fileName="setOfTasks.xml"
        deadline="IMPLICIT" distribution="UNIFORM" minU="2" maxU="4"
        nbMinTasks="5" number="10000" />
```

Fig. 4. Example to define a type of sets of tasks in the XML Evaluation file

Figure 4 defines that in the folder "./SetOfTasks/", a file "setOfTasks.xml" will be placed in an auto generated sub-folder "./SetOfTasks/IMPLICIT_UNIFORM/" and will contain 10000 sets of tasks with a utilization between 2 and 4, a minimum of 5 tasks for each set and each task will be generated with an "IMPLICIT" deadline and an "UNIFORM" distribution of utilization in the interval [0;1].

Figure 5 defines that in a folder "./SetOfProcessors/", a file "setOfProcessors4.xml" contains the definition of a set of processors with 4 homogeneous processors.

```
<EvaluationSetOfProcessors name="4_Processors_HOMOGENEOUS"
        autoPath="false" path=".\SetOfProcessors" fileName="setOfProcessors4.xml"
        nbProcessors="4" type="HOMOGENEOUS" />
```

Fig. 5. Example to define a type of sets of processors in the XML Evaluation file

B. Defining the scheduling algorithms

Then, you define algorithms to be tested. For each, indicate the name of the scheduling algorithm (corresponding to its class name), a file path defining the location where results will be stored and parameters such as the partitioning heuristics to consider, sets of tasks and processors to test and criteria for sorting tasks and processors.

```
<EvaluationAlgorithm name="4_EDFPartitionned" algoName="EDF_Load_P"
        path=".\Results" fileName="results.xml" >
 <Heuristic>FIRST_FIT</Heuristic>
 <Heuristic>WORST_FIT</Heuristic>
 <CriterionSortSetOfProcessors>PROCESSOR_NONE_ORDER
 </CriterionSortSetOfProcessors>
 <CriterionSortSetOfTasks>TASK_DENSITY_DECREASING_ORDER
 </CriterionSortSetOfTasks>
 <EvaluationSetOfTasks>Deadline_IMPLICIT__Distrib_UNIFORM
 </EvaluationSetOfTasks>
 <EvaluationSetOfProcessors>4_Processors_HOMOGENEOUS
 </EvaluationSetOfProcessors>
</EvaluationAlgorithm>
```

Fig. 6. Example to define an algorithm in the XML Evaluation file

Figure 6 defines that the algorithm "EDF_Load_P" (which correspond to the partitioning algorithm based on the feasibility test using the computation of the Load to EDF preemptive schedulers) will be tested on the previously defined sets of tasks "Deadline_IMPLICIT__Distrib_UNIFORM", without sorting processors and sorting tasks according to decreasing density. Heuristics "FIRST_FIT" and "WORST_FIT" will be tested following all possible combinations between all previous parameters.

The results will be stored automatically in files named "results.xml", in separate subfolders for each parameter in the main folder "./Results/".

C. Defining a graph

Finally, parameters for graphs can be defined. X-axis and Y-axis have to be selected according to a class name. For example, "GetUtilizationValue" returns the utilization of the set of tasks, "GetSuccessValue" retrieves in the result files if the set has been successfully scheduled by the algorithm.

Modularity: A new class placed in the correct package will automatically add a new possible value for axis in graphs.

By defining a curve name, it indicates what each curve must represent. For example, "GetAlgorithmCurveName" will generate a curve for each algorithm, while "GetHeuristicCurveName" will generate a curve for each partitioning heuristic found in the result files.

Modularity: To add a new type of curve, just add a Java file with a class derived from the abstract object "GetCurveName".

It is also possible to filter the results in order to focus only on some of the data. For example, the graph can concentrate on a particular type of deadline or on results for a 4-processor platform. It can consider only some algorithms, some heuristics or only sets of tasks in a particular range of utilization.

Modularity: Each of these parameters corresponds to "filter", it is possible to add a new filter to the tool by filing a Java file derived from the abstract class in the correct package.

```
<Graphs path="./Graphs/">
 <Graph name="MyGraph " Scale="1">
  <GetValueX name="GetUtilizationValue" />
  <GetValueY name="GetSuccessValue" />
  <GetCurveName name="GetAlgorithmCurveName" />
  <Deadlines>
    <Deadline>IMPLICIT</Deadline>
    <Deadline>CONSTRAINED</Deadline>
  </Deadlines>
  <Distributions>
    <Distribution>UNIFORM</Distribution>
  </Distributions>
  <NumbersOfProcessors>
    <NumberOfProcessors>4</NumberOfProcessors>
  </NumbersOfProcessors>
  <Filters>
    <Filter name="StatisticsHeuristicFilter">
      <ToKeep>FIRST_FIT</ToKeep>
    </Filter>
    <Filter name="StatisticsAlgorithmFilter">
      <ToKeep>EDF_Load_P</ToKeep>
      <ToKeep>DM_RT_P</ToKeep>
    </Filter>
    <Filter name="StatisticsUtilizationRangeFilter">
      <ToKeep>2</ToKeep>
      <ToKeep>4</ToKeep>
    </Filter>
  </Filters>
 </Graph>
</Graphs>
```

Fig. 7. Example to define a graph in the XML Evaluation file

Figure 7 creates a text file "MyGraph.txt" in the folder "./Graphs/" containing data which describe a graph with a X-axis representing the utilization of sets of tasks, Y-axis the success ratio. Each curve will be a different algorithm. We will focus on sets of tasks with "IMPLICIT" or "CONSTRAINED" deadlines, with a utilization generated with a "UNIFORM" distribution of probability. Only 4-processor platform will be checked and results from the "FIRST_FIT" heuristic. Both algorithms "EDF_Load_P" and "DM_RT_P" (partitioned scheduling algorithm based on the exact feasibility test on response time for a DM preemptive scheduler) will be taken into account. Finally, we are interesting only in sets of tasks with utilization in the range $[2; 4]$.

The graph produced with the example given in this paper is shown in figure 8. This figure is created using Gnuplot (http://www.gnuplot.info/) to interpret "MyGraph.txt".

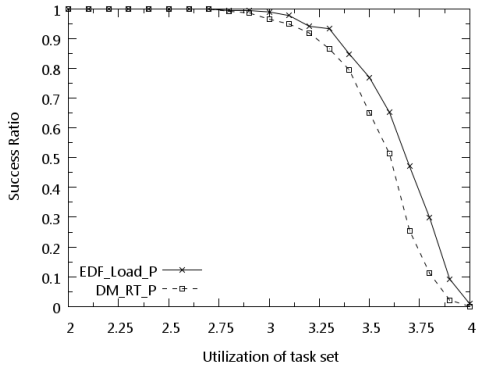


Fig. 8. Example of graph produced according to the example

D. Generating the evaluations

The evaluation file allows us automating the whole procedure: the generation of sets of tasks and the generation of graphs. Filters allows us to reuse some of the results and thus to resume the evaluations conducted previously.

However, this process can be time-consuming. Since the tool can also be used with a command-line, it allows us to run the computation, stop them at a predefined times and resume them later. It can then be used to spread the workload over multiple computers: the tool will generate a list of parameters corresponding to an XML Evaluation file; each parameter can be run on different computers and then assembled without recoveries problems.

VII. Conclusion and future work

The tool presented in this paper allows us to test if a set of tasks is schedulable on a set of processors according to a specific algorithm. You may view the sequence of scheduling in time to check that no deadline is missed. A procedure is also proposed to generate sets of tasks according to various parameters. Furthermore, the tool offers to automate the creation of evaluations of algorithms from beginning to end: generation of sets to test, computation of the results for all algorithms desired with a distribution of work on different computers and finally creation of graphs associated.

All these options can be improved by the user by defining itself new parameters, new algorithms, new axes for graphs etc. This is facilitated by a programming paradigm oriented to modules and abstract classes.

Version 0.5 of the tool can be downloaded from http://www.ece.fr/~courbin/Research/FORTAS/FORTASv0.5.zip. The first stable version will be released as open source licensing by the end of 2011. A complete documentation of the code will be made before this first stable version.

References

[1] I. Lupu, P. Courbin, L. George, and J. Goossens, "Multi-criteria evaluation of partitioning schemes for real-time systems," in *The 15th IEEE International Conference on Emerging Technologies and Factory Automation*, no. MF-001244. IEEE Computer Society Press, 2010.

[2] L. George and P. Courbin, *Reconfigurable Embedded Control Systems: Applications for Flexibility and Agility*. Hershey, PA, USA: IGI Global, 2011, ch. Reconfiguration of Uniprocessor Sporadic Real-Time Systems: The Sensitivity Approach.

[3] R. Davis, L. George, and P. Courbin, "Quantifying the sub-optimality of uniprocessor fixed priority non-pre-emptive scheduling," in *Proceedings of 18th International Conference on Real-Time and Network Systems, RTNS'10*, Toulouse, France, Nov. 2010. [Online]. Available: http://www-rocq.inria.fr/syndex/publications/pubs/rtns10/rtns10-2.pdf

[4] L. George, P. Courbin, and Y. Sorel, "Job vs portioned partitioning for the earliest deadline first semi-partitioned scheduling," *Elsevier Journal of Systems Architecture - Special issue on multiprocessor real-time scheduling*, to appear in 2011.

[5] M. G. Harbour, J. J. G. García, J. C. P. Gutiérrez, and J. M. D. Moyano, "Mast: Modeling and analysis suite for real time applications," in *In 13th Euromicro Conference on Real-Time Systems*, 2001, p. 125.

[6] F. Singhoff, J. Legrand, L. Nana, and L. Marcé, "Cheddar: a flexible real time scheduling framework," *Ada Lett.*, vol. XXIV, pp. 1–8, November 2004. [Online]. Available: http://doi.acm.org/10.1145/1046191.1032298

[7] Y. T. Richard Urunuela, Anne-Marie Deplanche, "Storm : A simulation tool for real-time multiprocessor scheduling evaluation," in *The 15th IEEE International Conference on Emerging Technologies and Factory Automation*, no. MF-000477. IEEE Computer Society Press, 2010.

[8] R. R. S. Kato and Y. Ishikawa, "A loadable real-time scheduler suite for multicore platforms," 2009. [Online]. Available: Available:http://www.contrib.andrew.cmu.edu/?shinpei/papers/techrep09.pdf

[9] M. Holenderski, M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Grasp: Tracing, visualizing and measuring the behavior of real-time systems," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, 2010.

[10] J. M. Calandrino, H. Leontyev, A. Block, U. C. Devi, and J. H. Anderson, "Litmusrt : A testbed for empirically comparing real-time multiprocessor schedulers," in *RTSS*, 2006, pp. 111–126.

[11] S. Baruah, R. Howell, and L. Rosier, "Algorithms and complexity concerning the preemptive scheduling of periodic real-time tasks on one processor," *Real-Time Systems*, vol. 2, pp. 301–324, 1990.

[12] L. C. Liu and W. Layland, "Scheduling algorithms for multiprogramming in a hard real time environment," *Journal of ACM*, vol. 20, no. 1, pp. 46–61, January 1973.

[13] L. George and J. Hermant, *Characterization of the Space of Feasible Worst-Case Execution Times for Earliest-Deadline-First scheduling*. Journal of Aerospace Computing, Information and Communication (JACIC), American Institute of Aeronautics and Astronautics (AIAA), November 2009, vol. Vol. 6, Num. 11.

[14] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with edf scheduling," *IEEE Transactions on Computers*, vol. 58, pp. 1250–1258, 2009.

[15] N. Audsley, A. Burns, M. Richardson, K. Tindell, and J. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, pp. 284–292, 1993.

[16] M. Bertogna and M. Cirinei, "Response-time analysis for globally scheduled symmetric multiprocessor platforms," in *RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 149–160.

[17] S. Kato, N. Yamasaki, and Y. Ishikawa, "Semi-partitioned scheduling of sporadic task systems on multiprocessors," in *ECRTS '09: Proceedings of the 2009 21st Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 249–258.

[18] A. Burns, R. Davis, P. Wang, and F. Zhang, "Partitioned edf scheduling for multiprocessors using a c=d scheme," in *Proceedings of 18th International Conference on Real-Time and Network Systems (RTNS)*, 2010, pp. 169–178.

[19] S. Baruah, J. Gehrke, and G. Plaxton, "Fast scheduling of periodic tasks on multiple resources," in *Proceedings of the International Parallel Processing Symposium*, vol. pp 280-288, Santa Barbara, California. IEEE Computer Society Press, April 1995.

[20] T. P. Baker, "A comparison of global and partitioned EDF schedulability tests for multiprocessors," in *International Conf. on Real-Time and Network Systems*, 2006, pp. 119–127.

[21] E. Bini and G. C. Buttazzo, "Biasing effects in schedulability measures," in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 196–203. [Online]. Available: http://portal.acm.org/citation.cfm?id=1009383.1009838

Hardware-Assisted Energy Consumption Evaluation Tool for Multi-core Embedded Systems

Shiao-Li Tsao, Jyun-Wei Lin, QuanChung Chen, Chen-Wei Huang, Chi-Neng Huang[+]
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan
[+]Information and Communications Research Laboratories, Industrial Technology Research Institute, Taiwan
sltsao@cs.nctu.edu.tw

Abstract—**Energy consumption evaluation is a critical issue in the design phase of system-on-chips (SoCs) and embedded systems. However, conventional approaches suffer from difficulties in providing fast and accurate estimation and evaluation results, especially for complicated multi-core embedded systems. In this paper, we propose and implement a hardware-assisted energy consumption evaluation tool for an embedded system. Our system, called reconfigurable hardware-assisted log profiler (REALprof), uses a programmable profiling hardware to monitor runtime programs, automatically record the energy related parameters of the system, and thus can estimate the system energy consumption based on the logs in the memory without introducing software profiling overhead. Our prototype works with a quad-core embedded system at 100 MHz on an FPGA platform and the experimental results demonstrate that the proposed profiling tool provides fast and fine-grained evaluation of energy consumption of a multi-core embedded system.**

Keywords-Energy Consumption, Energy Evaluation, Multi-core Embedded Systems

I. INTRODUCTION

Energy consumption is a critical issue for battery-operated embedded systems such as smart phones, mobile Internet devices, etc. The energy consumption evaluation tool is highly demanded in the design phase of system-on-chips (SoCs) and embedded systems. Conventional tools such as circuit-level power consumption simulation provide accurate estimation results but require long simulation time. This approach may not be able to apply to a complicated embedded system such as a multi-core embedded system running an operating system (OS) and embedded applications. On the other hand, high-level power consumption estimation tools based on hardware performance counters and registers can offer fast evaluation. However, the profiling software usually has to frequently sample the performance and energy status, i.e. to frequently access hardware counters and registers, to obtain accurate estimation results, and thus introduces considerable software profiling overhead.

The capacity and performance of reconfigurable hardware such as field programmable gate array (FPGA) grow rapidly. FPGA thus becomes a suitable platform for system-level emulation, design verification, and energy consumption evaluation. In this paper, we propose a hardware-assisted energy evaluation tool for embedded systems, called **re**configurable hardware-**a**ssisted **l**og **prof**iler (REALprof). The proposed tool composes of a profiling software and a flexible profiling hardware which can be easily integrated into the target SoC design on an FPGA platform. The profiling hardware is programmable and can be controlled by the profiling software dynamically. The profiling software can change the configurations of the profiling hardware such as monitoring parameters, sampling frequencies, etc. to obtain the energy consumption evaluation in different granularities. The embedded system then runs the system and application software, and the profiling hardware automatically records the energy consumption parameters in the memory without the involvement of the profiling software. The profiling software finally generates an energy consumption evaluation report based on the logs in the memory after a profiling process. The proposed tool offers fast, accurate and system-level energy consumption estimations so that system designers can identify energy consumption problems of the target system, and optimize their hardware and software designs.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 describes the methodology and architecture of our proposed energy evaluation tool. Section 4 describes the implementation, experimental environment and results, and a case study. Finally, Section 5 concludes this study.

II. RELATED WORK

Energy consumption evaluation tools for embedded systems can be categorized into measurement-based approach, circuit-level simulation and architecture-level simulation. Measurement-based approach measures the power consumption of an embedded system directly through a power meter [8]. The evaluation is fast and accurate, but the target hardware must be available and the platform must be specially designed to support hardware power measurements. Moreover, if a detailed and accurate evaluation is required, measurement-based approach may require expensive power measurement devices, and time synchronization between measurement devices and the target embedded system must be considered. The measurement-based approach is usually applied to the optimization during the system integration phase.

Energy consumption of an SoC is related to manufacturing technology, transistor characteristics, circuit architecture, and low-level hardware factors. To evaluate the energy consumption during the SoC design phase, gate-level and circuit-level simulations are usually adopted. For example, Synopsys HSPICE [6] is a circuit simulator which also supports the power estimation of an IC design, and Synopsys PrimeTime [7] is a cell-based design tool for gate-level power analysis. These tools are useful for hardware designers to estimate energy consumption in early design stage, but the simulation speed is extremely slow for a large scale integration circuit such as a multi-core embedded system running system and application software.

Architecture-level energy evaluation approach analyzes the architecture of a target system and constructs energy models for each component. Wattch [9] is one of the well-known energy evaluation tools based on this approach. It categorizes processor components into four categories: array structures, fully associative content-addressable memory, combinational logic and wires, and clocking. Power models of components are developed according to capacitance equations, and they are integrated with an architecture-level simulator such as SimpleScalar [5]. Architecture-level energy simulation [10][11] is faster than circuit-level simulation, but it still requires considerable simulation time for evaluating complicated embedded systems.

Energy consumption of a processor is closely related to some hardware activities. Therefore, Contreras et al. [12] proposed another architecture-level approach for Intel XScale processor. Embedded processor such as Intel XScale and ARM provides performance counters for monitoring hardware activities. They construct processor power models by using these available hardware performance counters. The tool thus can perform a quick energy evaluation based on the hardware performance counters. Because this approach is based on hardware monitoring at run time, the target hardware must be available, and the processor must support essential performance and/or energy counters [13].

Bhattacharjee et al. [3] proposed a FPGA-based power emulation approach. The concept is similar to Contreras' work. They add some component-specific event counters to the target processor and construct power models based on these counters. Because the FPGA is reconfigurable, this approach is more flexible than Contreras' approach. Although FPGA emulation speed is slower than the real application-specific integrated circuit (ASIC), it is much faster than circuit-level and architecture-level simulation. The energy evaluation approach proposed by Contreras and Bhattacharjee relies on hardware monitoring at run time. The profiling software has to sample and access performance and energy counters frequently to obtain accurate evaluation results. Considerable profiling software overhead is thus introduced.

Ghodrat et al. [4] further proposed a hybrid approach for system-level energy evaluation. Some of the components use software simulation and the other components use FPGA emulation. This approach reduces the simulation time and is more flexible than pure FPGA emulation. However, the communication and synchronization between software simulation and FPGA emulation are new challenges.

III. DESIGN OF THE PROPOSED REALPROF

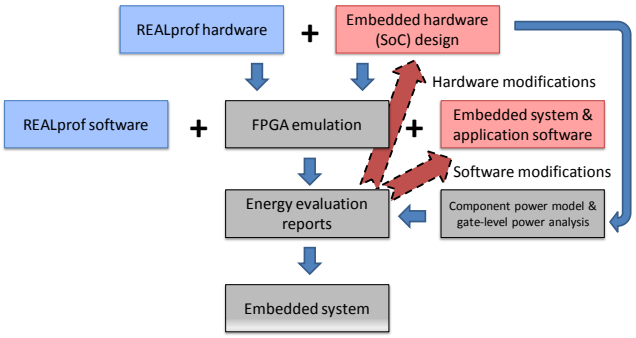


Figure 1. The proposed flow for evaluating and optimizing the energy consumption of an embedded system.

Conventional software profiling tools insert profiling instructions into the target software. This approach influences the behaviors of the original system and software, and also introduces additional software profiling overhead. To improve profiling accuracy and minimize the profiling overhead, the proposed REALprof uses flexible hardware to assist energy consumption evaluation of an embedded system without the involvement of the profiling software. Figure 1 illustrates our proposed flow for the development and energy consumption evaluation of an embedded system. In the design phase, an SoC which is composed of synthesizable soft intelligent properties can be constructed on an FPGA emulation platform. At this stage, we can use gate-level power analysis tools to obtain the power consumption of each hardware event such as a pipeline stall, cache miss, etc. on an SoC. These gate-level power analysis tools can calculate power dissipation based on the target design netlist, cell library power models, signal activities, and capacitance effects.

REALprof hardware is integrated into the target SoC, and it is managed by REALprof software. REALprof software can dynamically change the configurations of REALprof hardware such as sampling frequencies, number of monitoring parameters, etc. For a profiling process, embedded software including system and application software and REALprof profiling software run on the emulation platform. Based on the configurations, REALprof hardware automatically records the necessary hardware events in the memory when the embedded software is running. The performance logs are handled by the hardware so that the software behavior of the system remains unchanged during the profiling process. After the profiling process, REALprof software retrieves the hardware logs in the memory and generates an energy evaluation report. System designers can thus read the report, try to identify the design defects, and optimize the hardware and software designs. After a number of iterations of evaluation and optimization, the embedded system design can be confirmed and REALprof hardware and software can be removed from the final embedded system.

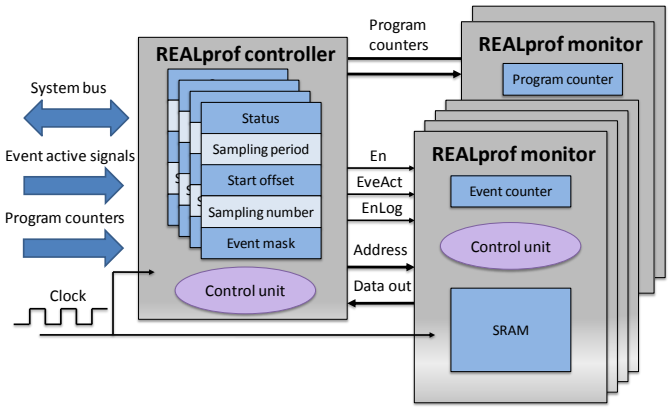


Figure 2. Architecture of REALprof hardware.

TABLE I. DESCRIPTION OF REGISTERS IN REALPROF HARDWARE.

Register	Description
Status	Control status of a REALprof monitor
Sampling Period	Sampling period of a REALprof monitor
Start Offset	Start time for profiling in number of delay cycles for a REALprof monitor
Sampling Number	Number of samples for a REALprof monitor
Event Mask	Enable/disable of a REALprof monitor

Figure 2 illustrates the architecture of REALprof hardware. REALprof hardware is designed as a bus slave component. Programmers can use memory-map I/O to access REALprof hardware. REALprof hardware consists of a hardware controller and a number of hardware monitors. REALprof controller contains five control registers for each REALprof monitor. TABLE I. lists the registers and their descriptions. Programmer can access these registers and configure the status, sampling period, start offset, and event mask for REALprof monitors individually. A number of event active signals are connected to REALprof controller, such as pipeline stall, cache miss, etc. Each event active signal is connected to "EveAct" signal of an individual REALprof monitor. REALprof monitor snoops the "EveAct" signal and increases the event counter when the event occurs. "En" signal is true when REALprof is active, and it indicates whether a REALprof monitor should snoop signals or not. "EnLog" signal is similar to a timer IRQ (Interrupt ReQuest) signal. The period of a timer trigger is determined by the sampling period register. When "EnLog" is triggered, a REALprof monitor stores the current value of the event counter and current time stamp to the SRAM and resets the event counter. After the procedure, REALprof controller increases the sampling number register which indicates how many samples are collected in the SRAM for a particular REALprof monitor. At the same time, the time stamp, processor identifier, and the program counter of the correspondent processor that triggers the event counter are also recorded in the memory by another REALprof monitor. The reason why the time stamp, processor identifier, and current program counter that associate with the event are also recorded is because REALprof software can associate the profiling results with a specific running program and a specific function of a program. Therefore, REALprof software could provide detail energy evaluation for not only a processor, but also a running program and a function in a running program.

We derive the energy consumption of an embedded system according to the gate-level power models and profiling results. Equation (1) shows our high-level energy model:

$$E = P_{idle} \cdot T_{total} + \sum_{i=1 \text{to } I} E^{i}_{event} \cdot C^{i}_{event} \quad (1)$$

Total energy consumption (E) consists of idle energy and active energy. Idle energy is the product of idle power (P_{idle}) and total execution time (T_{total}). Idle power reflects leakage power and power of hardware components which are always active. Active energy is contributed by all event energy. For an embedded system with I major power consumption events such as pipeline stall, cache miss, etc., the event energy is the summation of the product of energy of the i^{th} event (E^{i}_{event}) and counts of the i^{th} event (C^{i}_{event}). P_{idle} and E^{i}_{event} are obtained off-line by using gate-level power simulations which will be described in the next section. The event active count, C^{i}_{event}, and execution time, T_{total}, are from REALprof. Based on the design mentioned above, REALprof can provide performance evaluation through event counts and energy evaluation through event energy. It is useful for designers to optimize system and make a trade-off between performance and energy efficiency in the design stage.

Moreover, REALprof hardware can be dynamically configured by REALprof software. For example, system designers may want to see the overall energy consumption of an embedded system first. Then, they may want to see the power consumption during a specific period of execution or the power consumption of a specific program in details. Therefore, the system designer can configure the profiling starting time and duration, reduce or increase the sampling frequencies for monitoring parameters, reduce or increase the number of monitoring parameters, and can obtain energy evaluation of the embedded system in different granularities.

IV. IMPLEMENTATION

Figure 3 gives an overview of the prototype system. We established the MPSoC using GRLIB open source IP library [14] with the proposed REALprof. GRLIB includes LEON3 soft-core processor and abounding peripheral cores. LEON3 core is interfaced using AMBA 2.0 protocol and uses synthesizable VHDL model. LEON3 is a 32-bit soft-core processor based on SPARC V8 architecture. LEON3 has 7-stage pipeline with L1 Harvard architecture cache. Data cache of LEON3 supports snoop protocol so that it is possible to build a SMP system using LEON3 processor. Based on the GRLIB, we implemented a quad-core LEON3 multi-core processor on an FPGA board. Figure 3 also shows the detail of the target MPSoC design. It includes a LEON3 quad-core, abounding peripherals, and REALprof hardware. Four LEON3 processors snoop data on AHB (Advanced High-performance Bus). Configurations of each processor are shown in TABLE II. . REALprof hardware is a flexible hardware component and wrapped as an AHB slave. Therefore, it can be easily integrated into a target SoC design. Each processor connects

its program counter and event signals to REALprof hardware. We used DE3-340 [15] development board to construct the prototype emulation platform. The DE3-340 consists of an Altera Stratix III 340 FPGA, DDR2 SO-DIMM socket, and several peripheral components. We used Quartus II 8.0 design suite [16] to compile our design and downloaded the target design to the DE3 FPGA. We implement REALprof software as device drivers and utilities in SnapGear Linux distribution [17]. Multi-thread applications are integrated to SnapGear Linux package and use REALprof to perform performance and energy evaluation on the target multi-core emulation platform.

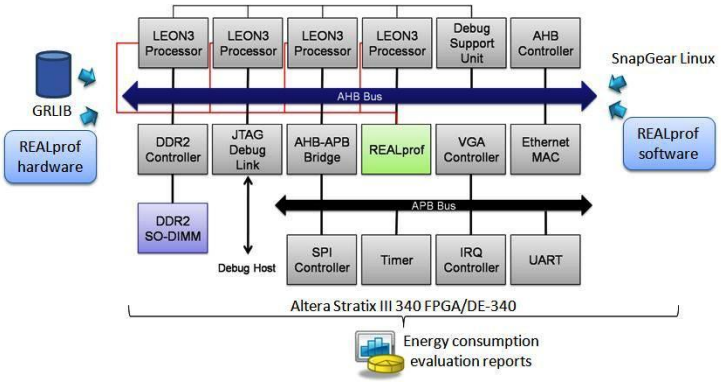


Figure 3. Overview of the prototype system with quad-core LEON3 and REALprof.

TABLE II. LEON3 PROCESSOR CONFIGURATIONS.

Integer Unit	8 register windows, 1-cycle load delay, SPARC V8 MUL/DIV support, 2-cycle multiplier latency, power-down mode support
Floating Point Unit	only netlist is available for FPGA
L1 Instruction Cache	4-way set-associative, total 16 KB, 32 bytes/line, LRU
L1 Data Cache	4-way set-associative, total 16 KB, 32 bytes/line, LRU, AHB fast snoop and separate snoop tags
MMU	Separate instruction/data TLB, fast write buffer, LRU, 32 entries for each instruction/data TLB

In order to evaluate the energy of LEON3 processor, we monitor a number of hardware events for each processor using REALprof. Table 3 lists these events. To obtain the power consumption of each event, we use gate-level power estimation tools. Figure 4 illustrates detail gate-level power analysis flow. We use Design Compiler [18] to synthesis the target design with Faraday cell-based design kit for UMC 90 nm 1P9M process. In typical condition (25˚C, 1.0 Volt), the target ASIC expects to operate at 400 MHz. After synthesis, we use ModelSim [19] to perform gate-level simulation for generated netlist and software. The generated waveform in VCD (Value Change Dump) format then feeds to PrimeTime PX for dynamic power calculation. PrimeTime PX produces a hierarchical power report and FSDB (Fast Signal DataBase) waveform according to design netlist, cell library, cell delay, and VCD waveform. Finally, we can construct event energy models. Table 3 lists event energy consumption of a LEON3 processor when the processor is finally produced as an ASIC according to the gate-level power analyses.

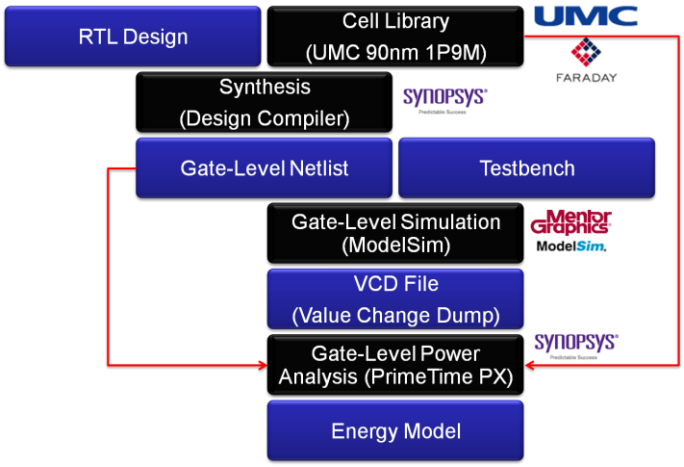


Figure 4. Flowchart of gate-level power analyses.

TABLE III. LEON3 PROCESSOR EVENTS AND THEIR ENERGY CONSUMPTION.

Component	Event	Energy/Power
Pipeline	Power down	16.1 mW
	Idle	49.2 mW
	Normal	58.2 mW
	Multiplication operation	32 pJ
	Division operation	218.75 pJ
	Instruction TLB miss	4.25 pJ
	Data TLB miss	4.25 pJ
Instruction cache	Power down	0.4084 mW
	Idle	49.1 mW
	Flush	16.403 nJ
	Hit	126.575 pJ
	Miss	123 pJ
Data cache	Idle	0.601 mW
	Flush	25.869 nJ
	Hit	314 pJ
	Miss	952.95 pJ
Register file	Idle	6.28 mW
	Single access	9.65 mW
	Double access	12.9 mW

Programmers set REALprof registers to start profiling before the target program execution and stop profiling after the execution or after a period of time. REALprof does not need software to sample hardware performance counters periodically. Therefore, the software overhead during the profiling process is significantly reduced. Figure 5 shows that the overhead of the conventional architecture-level evaluation based on hardware performance counters increases linearly with number of counters. The more information an architecture-level profiling tool gathers, the more overhead it suffers. On the contrary, our approach offers constant and negligible software overhead during a profiling process. Moreover, the behavior of the target program remains unchanged when REALprof is employed. Figure 6 shows that software overhead increases with the profiling frequency. As can be seen from the figure, it is difficult to perform profiling

below microsecond granularity using the conventional architecture-level approach.

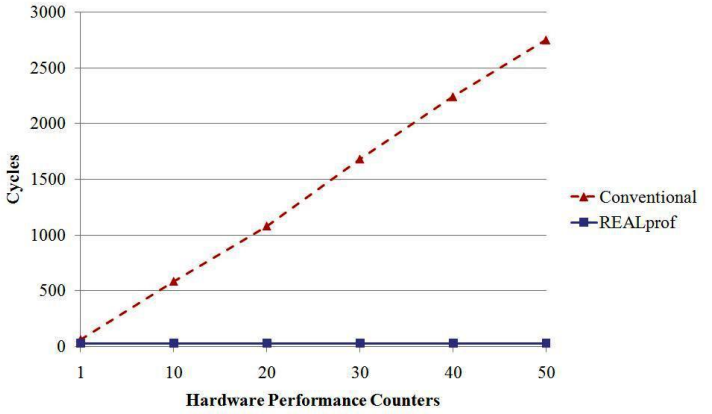


Figure 5. Overhead comparison between architecture-level simulations and REALprof under different number of hardware counters that tools refer

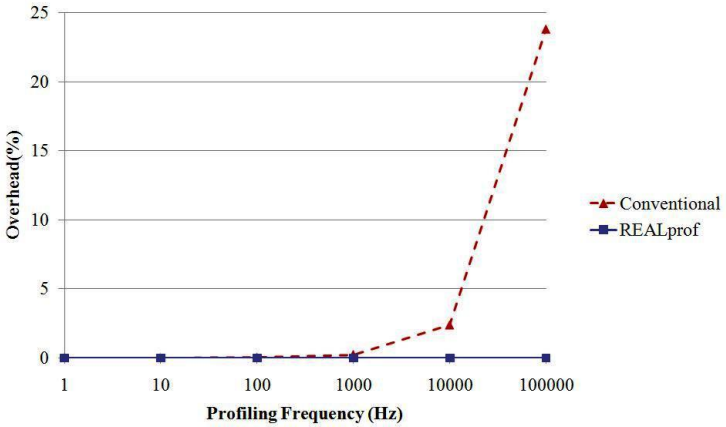


Figure 6. Overhead comparison between architecture-level simulations and REALprof under different profiling frequencies.

TABLE IV. compares our approach with other energy evaluation approaches. Our approach offers fast and fine-grained evaluation, and software overhead is negligible.

TABLE IV. COMPARISON OF PREVIOUS APPROACHES AND REALPROF.

	REALprof	Profiling based on hardware counters [12]	Architecture-level simulation [9]	Circuit-level simulation [6][7]
Method	Hardware emulation	directly execute	Software simulation	Software simulation
Speed	~ 100 MHz	real speed	KIPS ~ MIPS	extremely slow
Flexibility of profiling	Y	N	Y	Y
Profiling overhead	Negligible (less than 30 cycles)	Software profiling overhead	N/A	N/A
Profiling granularity	~30 cycles	~ millisecond	~microsecond	cycle
Profiling hardware resource	Hardware counters + RAM	Hardware counters	N	N
Program behavior while profiling	Remain unchanged	influenced	Remain unchanged	Remain unchanged
Operating system	Y	Y	usually N	N

Our approach uses reconfigurable hardware resources to record the profiling data. TABLE V. shows the resource usage of the prototype. For StratixIII 340 FPGA, the target LEON3 multi-core system with REALprof hardware only uses about 30% logic and 12% memory blocks. REALprof only uses about 2% logic and 4% memory blocks for 1 controller and 68 monitors.

TABLE V. STRATIXIII 340 FPGA RESOURCE USAGE.

	Combinational ALUTs	Logic registers	DSP block	Block memory bit	Number of components
LEON3 processor	12581	8451	4	343936	4
DDR2 controller	781	601	0	4096	1
peripherals	2600	1269	0	16384	N/A
REALprof	4826 (2%)	2091 (1%)	0 (0%)	557056 (4%)	1 controller 68 monitors
Total	58531 (22%)	37765 (14%)	16 (3%)	1953280 (12%)	

To demonstrate the profiling process on a multi-core embedded system, we use SPLASH-2 [20] FFT as the target multi-thread application program. Figure 7 illustrates the energy evaluation of events on the first CPU (CPU_0). As shown in figure, data cache dominates the variation of the energy consumption of a processor. This phenomenon matches the gate-level power analyses. Figure 8 illustrates the energy evaluation of each core. Designer can use the evaluation result to optimize the target hardware and software design. In this case, we can observe that SPLASH-2 FFT is a well parallelized multi-thread program.

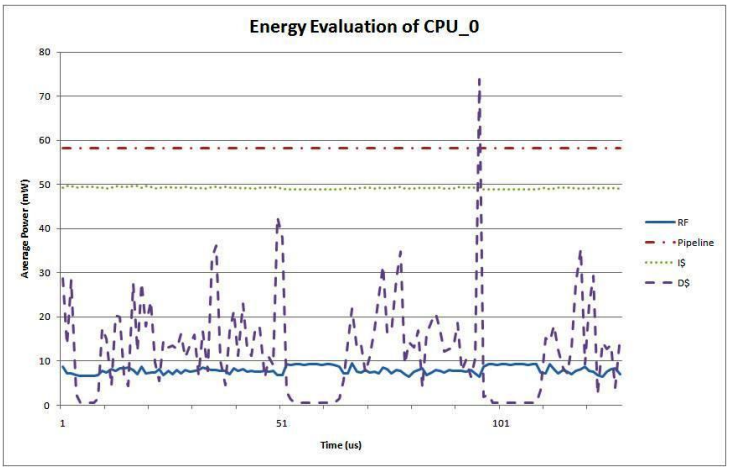


Figure 7. Energy evaluation of the events on the first CPU (CPU_0). (RF: total power of register file, Pipeline: total power of pipeline, I$: total power of instruction cache, D$: total power of data cache)

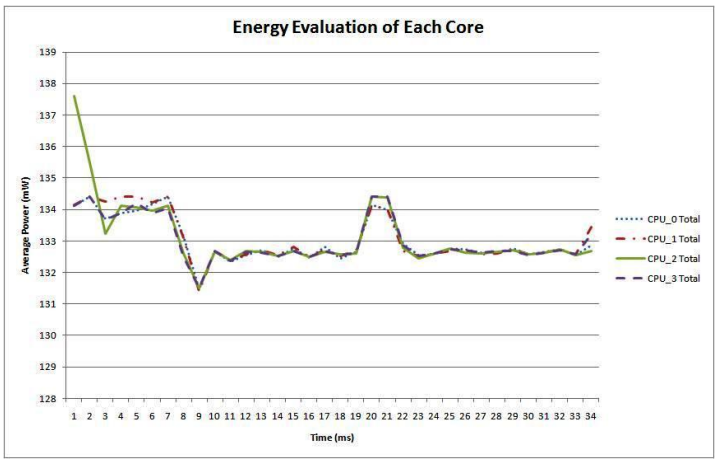


Figure 8. Energy evaluation of each core.

V. CONCLUSIONS

In this paper, we proposed a hardware-assisted energy evaluation tool - REALprof. REALprof uses flexible hardware to profile the energy consumption of the processors, programs, and functions without software involvement, and can provide energy evaluation in different granularities. The fast, accurate, and system-level energy evaluation can help system designers in optimizing hardware and software designs. The prototype system based on quad-core LEON3 MPSoC and Linux was implemented on an FPGA platform. Experimental results demonstrate that REALprof could provide fine-grained profiling results in sub-microsecond level and overhead can be negligible.

ACKNOWLEDGEMENT

The authors would like to thank Industrial Technology Research Institute, MediaTek Inc. and National Science Council of the Republic of China for financially supporting this research under Contract No. 100-2219-E-009-022-, 100-2220-E-009-038-, and 99-2220-E-009-045-.

REFERENCES

[1] L. Shannon and P. Chow, "Using Reconfigurability to Achieve Real-Time Profiling for Hardware/Software Codesign," Proceedings of the 12th International Symposium on Field Programmable Gate Arrays, pp. 190-199, Monterey, California, February 2004.

[2] J. G. Tong and M. A. S. Khalid, "Profiling Tools for FPGA-Based Embedded Systems: Survey and Quantitative Comparison," Journal of Computers, Vol. 3, No. 6, pp. 1-14, June 2008.

[3] A. Bhattacharjee, G. Contreras, and M. Martonosi, "Full-System Chip Multiprocessor Power Evaluations Using FPGA-Based Emulation," Proceedings of the 13th International Symposium on Low Power Electronics and Design, pp. 335-340, Bangalore, India, August 2008.

[4] M. A. Ghodrat, K. Lahiri, and A. Raghunathan, "Accelerating System-on-Chip Power Analysis Using Hybrid Power Estimation," Proceedings of the 44th Design Automation Conference, pp. 883-886, San Diego, California, June 2007.

[5] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An Infrastructure for Computer System Modeling," IEEE Computer, Vol. 35, No. 2, pp. 59-67, February 2002.

[6] Synopsys HSPICE,
http://www.synopsys.com/Tools/Verification/AMSVerification/CircuitSimulation/HSPICE/Pages/default.aspx

[7] Synopsys PrimeTime PX.
http://www.synopsys.com/Tools/Implementation/SignOff/Pages/PrimeTime.aspx

[8] D. Shin, H. Shim, Y. Joo, H.-S. Yun, J. Kim, and N. Chang, "Energy Monitoring Tool for Low-Power Embedded Programs," IEEE Design and Test of Computers, Vol. 19, No. 4, pp. 7-17, July 2002.

[9] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," Proceedings of the 27th International Symposium on Computer Architecture, pp. 83-94, Vancouver, British Columbia, Canada, June 2000.

[10] W. Ye, N. Vijaykrishnan, M. Kandemir, and M. J. Irwin, "The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool," Proceedings of the 37th Design Automation Conference, pp. 340-345, Los Angeles, California, June 2000.

[11] R. Ben Atitallah, S. Niar, and J.-L. Dekeyser, "MPSoC Power Estimation Framework at Transaction Level Modeling," Proceedings of the 19th International Conference on Microelectronics, pp.245-248, Cairo, Egypt, December 2007.

[12] G. Contreras and M. Martonosi, "Power Prediction for Intel XScale Processors Using Performance Monitoring Unit Events," Proceedings of the 10th International Symposium on Low Power Electronics and Design, pp. 221-226, San Diego, California, August 2005.

[13] G. Contreras, M. Martonosi, J. Peng, G.-Y. Lueh, and R. Ju, "The XTREM Power and Performance Simulator for the Intel XScale Core: Design and Experiences," ACM Transactions on Embedded Computing Systems, Vol. 6, No. 1, February 2007.

[14] J. Gaisler, E. Catovic, M. Isomäki, K. Glembo, and S. Habinc, GRLIB IP Library User's Manual Version 1.0.20, Aeroflex Gaisler, February 2009.

[15] ALTERA DE3 Development and Education Board User Manual, Terasic Technologies.

[16] Altera Quartus II.
http://www.altera.com/products/software/quartus-ii/subscription-edition

[17] D. Hellström, Manual: SnapGear Linux for LEON Version 1.38.0, March 2009.

[18] Synopsys Design Compiler.
http://www.synopsys.com/Tools/Implementation/RTLSynthesis/Pages/DesignCompilerGraphical.aspx

[19] Mentor Graphics ModelSim.
http://www.mentor.com/products/fv/modelsim

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," Proceedings of the 22nd International Symposium on Computer Architecture, pp. 24-36, Santa Margherita Ligure, Italy, June 1995.

Modelling real-time applications based on resource reservations

Laura Barros, César Cuevas, Patricia López Martínez, José María Drake, and Michael González Harbour
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
39005, Santander, Spain
{barrosl, cuevasce, lopezpa, drakej, mgh}@unican.es

Abstract—**The new MAST 2 specification for modelling and analysis of real-time systems, introduces two new classes named *VirtualSchedulableResource* and *VirtualCommunication Channel*. They are used for modelling the schedulable entities in real-time applications that are designed and executed relying on a resource reservation paradigm. These modelling elements bring together into one single object the two different views that are used to describe a virtual resource through its life cycle. In the initial phase of the application design, a virtual view of the modelling element is used. It describes the fraction of the processor or network bandwidth needed to satisfy the timing requirements of the application. The second view is used afterwards, during the application deployment phase. It describes the real scheduling parameters with which the processor or network scheduler must execute the threads or communication channels that implement the virtual resources in the physical platform. The negotiation process between the application and the resource reservation middleware, which is carried out to make the designed application compatible with the current workload of the platform, can be seen as the pursuit of a configuration that makes both views compatible. This paper describes these new modelling elements and some scenarios where they are used.**

Keywords: Real-time; Resource Reservation; Modelling; Virtual Resources

I. INTRODUCTION

Resource Reservations is a paradigm that is widely applied to the design and execution of hard real-time applications. Its basic principle [1][2][3] in processing resources consists in executing each application thread in a server, which has an assigned fraction of the processor capacity. If during the execution, the thread activity tries to exceed its assigned capacity, the server suspends it temporarily to avoid it. Similarly, for communication resources each message stream is assigned to a communication server that represents a fraction of the communication network bandwidth. The use of this paradigm provides three main advantages:

- *System robustness*: If a system is designed to be schedulable according to its specification, and one or more of its applications exceed their design specifications by requiring more processing capacity than expected, these applications would not meet their timing requirements but the rest of the system would not be affected.
- *Design simplicity*: Using the resource reservation paradigm, the design of the scheduling of an application is accomplished in two stages. First, the application is scheduled independently, based on a virtual execution platform. Afterwards, the virtual platform implementation is negotiated on the physical execution platform. Among other advantages, this allows deploying an application with hard real-time requirements on an open platform whose workload is unknown by the application designer.
- *Reusability of real-time legacy modules*: A legacy subsystem or a reusable software component that implements real-time services can include as metadata the virtual execution platform that describes the resources required to satisfy its timing requirements.

MAST [4] is an open source set of tools that enables modelling and performing timing analysis of real-time applications. MAST can be used to design real-time applications, representing the real-time behavior and requirements together with the design information, and allowing an automatic schedulability analysis. MAST is basically constituted by a modelling methodology, close to that proposed by the OMG's MARTE profile [5][6], and a suite of tools including worst-case response time schedulability analysis, calculation of blocking times, sensitivity analysis through the calculation of slack times, and optimized priority assignment techniques. Version 2 of MAST [7] has been formulated recently and it extends the modelling methodology to advanced real-time paradigms, like resource-reservation and partition-based scheduling. The MAST 2 tools are still under development but, being both MAST 1 and MAST 2 versions based on formal UML metamodels, it is possible to perform simple transformations using MDA strategies and take advantage of the tools available in the current MAST suite for developing systems based on these new paradigms.

The aim of this paper is to describe how MAST 2 can be used to cover the different phases followed in the development and execution of applications based on resource reservations. Section 2 describes the MAST extension that deals with this paradigm, whereas Section 3 describes how the extension is applied to develop real-time applications. The information provided by the different modelling elements is described through a simple example in Section 4. Finally, Section 5 summarizes some current work lines and conclusions.

This work has been funded by the EU under contracts FP7/NoE/214373 (ArtistDesign); and by the Spanish Ministry of Science and Technology under grant TIN2008-06766-C03-03 (RT-MODEL).

II. Modelling Elements For The Resource Reservation Paradigm

The temporal behavior model of a system is conceived in MAST (and in MARTE) as the superposition of two models: the reactive model and the resources model, shown in Figure 1. The reactive model describes, by means of *EndToEndFlow* elements (as e2efA and e2efB), three aspects: the set of steps that, ordered by the control flow, conform the responses to events executed in the application, the generation patterns of the *WorkloadEvents* (coming from the environment or from the clock) which trigger these responses (such as triggerA and triggerB), and the *TimingRequirements* that must be satisfied by the execution of the responses (as tr1 and tr2). The resources model describes the usage of processing or communication resources and of mutual exclusion resources, by the steps belonging to the different *EndToEndFlows*. In the case of passive resources (as a Mutex), for the description of the resource it is enough to specify the synchronization protocol (priority ceiling, priority inheritance, stack resource). However, in the active resources (such as processors and communication networks), the model specifies its *Scheduler* together with its policy. In addition, it is necessary to specify the set of *SchedulableResources*, which are schedulable entities in a processing resource, each of them characterized by its scheduling parameters.

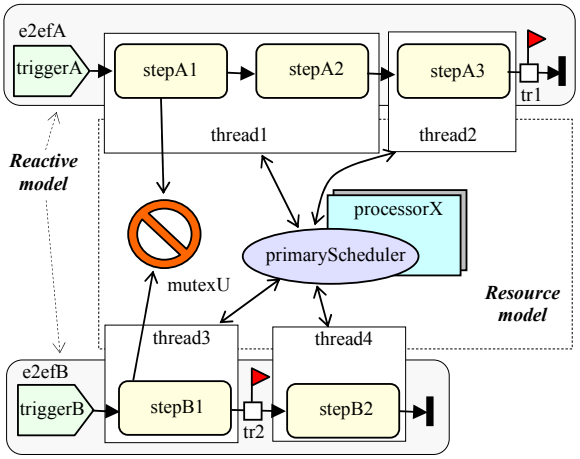


Figure 1: Root elements of the MAST models

The MAST 2 model extension that supports the resource reservation paradigm concerns only the *SchedulableResource* model elements. The new specialized elements decouple the reactive model from the resources model. This enables the design and analysis of the scheduling of an application using a virtual platform based upon resource reservation contracts, and without knowing the resources of the physical execution platform (Figure 2). Bearing in mind that in the development process of an application based on resource reservations the same instance of a *SchedulableResource* is going to be transformed from its virtual view to the real view, the virtual schedulable resource has been defined as inherited from the real schedulable resource, thus maintaining both views. The tool that processes the instance according to the stage of the design process, chooses the appropriate view.

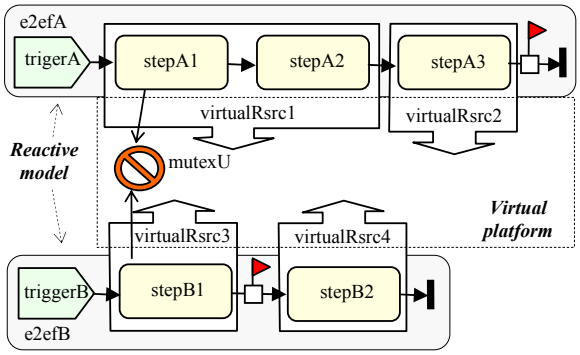


Figure 2: Application model in the virtual resource reservation view

The modelling elements defined in MAST 2 for representing virtual schedulable resources are *VirtualSchedulableResource* and *VirtualCommunicationChannel*. They extend the *Thread* and *Communication Channel* classes respectively, by adding a reference to the *VirtualResourceParams* or *VirtualComm ChannelParams* element that holds the information of the resource reservation contracts. Figure 3 shows the relations among the different modelling elements for the case of processing capacity reservation. For simplicity, the corresponding network elements are not shown.

It is important to notice that the inheritance relationship implies that a *VirtualSchedulableResource* is a *Thread*, so the tools that process it as a real schedulable resource will find within it the reference *SchedulingParameters* that characterizes it as a *Thread*. Likewise, the optional nature (0..1 multiplicity) of the *Scheduler* and *SchedulingParameters* references, makes it possible to create a *VirtualSchedulableResource* instance without any reference to the physical execution platform.

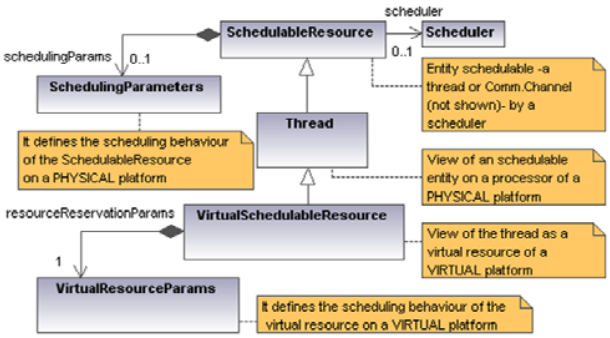


Figure 3: New classes in the resource reservation MAST model

The concrete classes that extend *ResourceReservationParams* define the behavior of the *VirtualSchedulableResource* (or of the *VirtualCommunication Channel*). This root class has been defined as abstract to include future types of virtual resources. Figure 4 shows the virtual resources currently defined, which follow the periodic replenishment strategy [11][12] for fixed priority scheduling: the periodic server [8], the deferrable server [9] and the sporadic server [10] models defined in the bibliography.

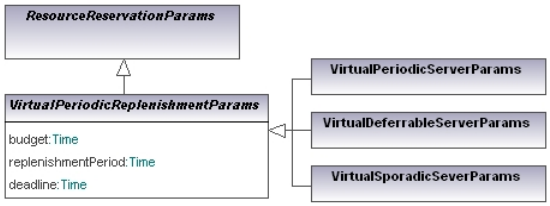


Figure 4: Resource reservation parameters classes in MAST 2

The attributes defining any type derived from *VirtualPeriodicReplenishmentParams* are the following:

- *Budget: Time* => Minimum execution capacity per server period.
- *Deadline: Time* => The server guarantees that a piece of work of size less than or equal to the budget and requested for a server with full capacity will be completed before the server's deadline.
- *Period: Time* => The period of the replenishment mechanism. The virtual resource will guarantee that every period, the part of the application running on it will get, if requested at the start of the period, at least the specified budget on the processing resource on which the associated schedulable resource is running.

The *VirtualCommChannelParams* class is similar, except for the *Budget* attribute, which is defined as the number of bits of transmission capacity.

The different server models vary in the granularity of the capacity replenishment and in how the platform responds when the application time is beyond the budget capacity. As an example, Figure 5 shows the worst-case response time of an activity of duration t_a in a virtual schedulable resource with *DeferrableServerParams* (Budget t_B, Deadline t_D and Period t_P). For this kind of virtual schedulable resource it is possible to evaluate the maximum response time t_x of an activity with duration t_a in the following manner:

$$If\ t_b \geq t_a \Rightarrow t_x < t_D - (t_B - t_a)$$

$$If\ t_b < t_a \Rightarrow \begin{cases} N_B = \left\lceil \frac{(t_a - t_b)}{t_B} \right\rceil \\ t_x < N_B t_R + t_D + t_a - t_b - N_B t_B \end{cases}$$

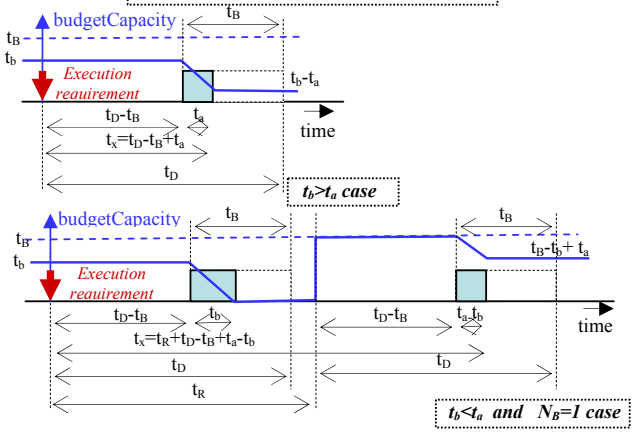


Figure 5: Worst case response time for the deferrable server

III. Supporting Real-Time Application Development

Figure 6 shows the four phases followed in the development of a real-time application based on the resource reservation paradigm: (1) The application is designed relying on a virtual platform. (2) The application is analysed to verify if it satisfies its timing requirements. (3) The instantiation of the virtual platform is negotiated with the execution platform. (4) The application is executed. The information and the models used in the process are also shown in the figure.

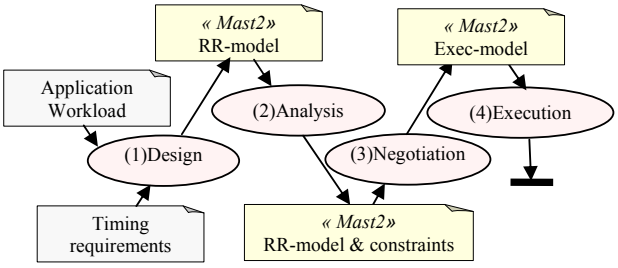


Figure 6: Phases in the design and execution of a real-time application

A simple example, called *ServoControl*, is used in this paper to illustrate the design process of a real-time system based on the resource reservation paradigm. It implements the controller of a servo engine, which is executed with a period of 10ms and a deadline of 5ms on a distributed platform composed of two processors, *Central* and *Remote*, interconnected by a CAN bus with a throughput of 1Mb/s. Figure 7 shows the reactive model of the application. It comprises only one *EndToEndFlow*, which involves seven *Step* elements, representing the activities executed in both processors and the transmission of messages through the bus.

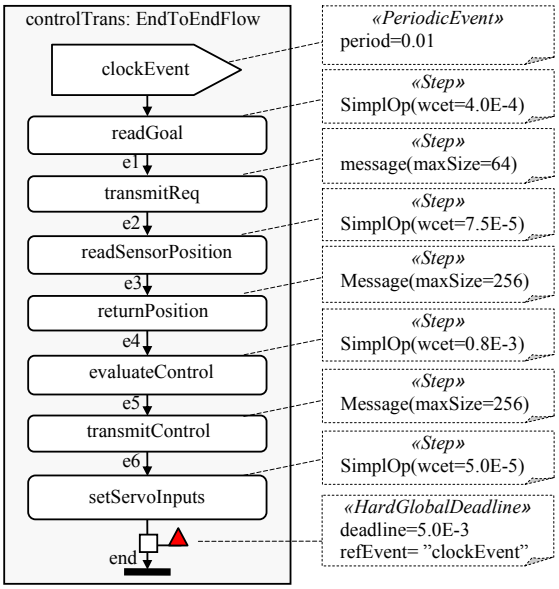


Figure 7: Reactive model of the ServoControl example

A. RT-Application specification and design.

The real-time application design consists in defining a virtual execution platform (i.e. a set of virtual resources), assigning to each step the virtual resource where it is executed, and according to this mapping, assigning values to the attributes of the virtual resources of the virtual platform. It is important to remark that this process can be done without knowing the physical platform where the application will be really executed.

Different design strategies may be applied to define the virtual platform. The simplest one consists in assigning an independent virtual resource per transaction and per processing node that take part in the application. In this case the replenishment period of each virtual resource is made equal to the minimum interarrival time of the transaction trigger, and the budget must be equal to the sum of the *wcet*s of the *Step*s executed in the same virtual resource. In this strategy the deadline attributes are kept indeterminate until the negotiation phase. The virtual platform that results from this design criterion for the *ServoControl* example is shown in Table I. The RR-Model (see Figure 6) that describes the application design is formulated as a MAST 2 model, which includes the reactive model (the "*controlTrans*" *EndToEndFlow* shown in Figure 7), and the virtual resources of Table I.

TABLE I. VIRTUAL PLATFORM AFTER DESIGN

VirtualResource	Assigned steps	R. Period	Budget	Deadline
VR_Central (vr1)	readGoal evaluateControl	10 ms	1.2 ms	?
VR_Bus (vr2)	transmitReq returnPosition transmitControl	10 ms	576 bits	?
VR_Remote (vr3)	readSensor setServos	10 ms	0.125 ms	?

When other design paradigms are used, the criteria to define the number of virtual resources and to assign steps to them may be different. Hence, if the *wcet* of the different steps executed in a processor has a large variability, or if some of its steps require access to a mutex with a high utilization, it can be advisable to use several virtual resources to schedule the steps of this processor. Likewise, if the application design is based on a component-based paradigm, the deployment of the components in the platform is unknown, so it is convenient to assign the virtual resources to each individual component in order to facilitate their later deployment. In this case, a virtual resource can be assigned for each invocation of the component services that is made in the different transactions. Hence, a strict ownership relation is kept between the component and its virtual resources.

B. RT-Application analysis

The analysis phase of a real-time application based on a resource reservation paradigm deals with establishing the relations among the virtual platform attributes that make the application satisfy its timing requirements. These requirements are formulated in the RR-model as *Timing_Requirement* modelling elements.

The design criterion described in the previous section guarantees that, when the execution of a step is required, there is enough budget to execute it without waiting for the next replenishment. Besides, its response time (t_x) can be delimited by the expression $t_x<t_D-(t_B\text{-wcet})$. In the case that the *EndToEndFlow* transaction is linear, for each timing requirement imposed on it, a restriction among the virtual resources attributes can be obtained. For example, in the *ServoControl* application, the restriction introduced by the timing requirement is the following:

$$vr1.t_D-(vr1.t_B\text{-readGoal.wcet})+ vr1.t_D-(vr1.t_B\text{-evaluateControl.wcet})+$$
$$+vr2.t_D-(vr2.t_B\text{-transmitReq.wcet})+ vr2.t_D-(vr2.t_B\text{-returnPosition.wcet})+$$
$$+vr2.t_D-(vr2.t_B\text{-transmitControl.wcet})+ vr3.t_D-(vr3.t_B\text{-readSensor.wcet})+$$
$$+vr3.t_D-(vr3.t_B\text{-setServos.wcet}) < t_{GD}$$

This expression leads to the following numerical solution:

$$2\ vr1.t_{D+}\ 3\ vr2.t_{D+}\ 2\ vr3.t_D \le 9.182\ ms$$

When the reactive model is more complex (e.g., when the control flow among the steps relies on fork, join, branch, or merge relations), the previous restrictions described as inequalities cannot be deduced directly, so the result of the analysis would be a set of concrete n-tuples of deadline attributes that make the application schedulable. These values can be obtained using the MAST 2 model and the MAST analysis tools.

The result of the analysis process of the real-time system is the RR-model&constraints model, which it is the result of adding to the previous RR-model the set of restrictions between the attributes of the virtual resources obtained from the analysis process. Again, it is important to notice that the analysis process is made without knowing neither the physical platform in which the application will be executed nor the other applications that will share the same physical platform.

C. RT-Application negotiation

The negotiation process is executed as a step previous to the deployment and execution of the application on the physical execution platform. It is an online process performed by the resource reservation service that must be provided by the execution platform. This negotiation requires knowing the RR-model&constraints of the application, the deployment plan, and the current workload of the physical platform. This process results in the assignment of values to the scheduling parameters of the threads and mutexes of the application that is being negotiated, and the reassessment of the threads and mutexes of the workload that was already executing, in order to adapt it to the new situation.

The deployment plan describes the assignment of the virtual resources of the application to the processing resources of the platform. Likewise, it includes the assignment of the kind of scheduling parameters (fixed-priority, EDF deadline, partition timetable, etc.) to the virtual resources, according to the scheduling policy applied to the scheduler of the processing resource assigned to them. The need to access this information reveals the importance of modelling both the virtual resource of the virtual platform and the schedulable resource of the final

resources model with a single element. Table II shows the resources model of the *ServoControl* application.

TABLE II. RESOURCE MODEL AFTER NEGOTIATION

VirtualResource/ Schedulable Rsrc	Assigned steps	R. Period/ priority	Budget/ scheduler	Deadline
VR_Central (vr1) /centrThr	readGoal evalControl	10 ms /14	1.2 ms /centrSch	2.51
VR_Bus (vr2) /commChannel	sendReq rtrnPosition sendControl	10 ms /146	576 bits /busSch	0.83
VR_Remote (vr3) /remiteThr	readSensors setServos	10 ms /22	0.125 ms /rmtSch	0.5

When the virtual platform established in the RR-model has all of its attributes assigned (i.e., in the case of complex reactive models), the negotiation process consists in building a timing behaviour model by gathering the application model and the current workload model, and executing a schedulability analysis. Different sets of values can be used until a schedulable solution is found.

However, when the result of the analysis phase is a set of restrictions among the virtual resources attributes (and not a set of concrete values), there is a higher negotiation capacity. In this case, the global model composed by the current workload of the platform plus the virtual platform model is partially specified, since the budgets and replenishment periods of the virtual resources have their definitive values assigned, but the deadlines do not have concrete values assigned yet. However, these deadlines must satisfy the set of restrictions obtained in the real-time analysis of the application. The negotiation task consists of looking for a set of values for these deadlines that not only satisfies the required restrictions, but also leads to a final schedulable workload including the prior application contracts together with the new one.

The online negotiation process requires a quick analysis of possible priority assignments and of the schedulability of the global model. The design strategy applied leads to global models composed of several, although very simple, elements. A typical model may be composed of hundreds of virtual resources, each of them with a periodic triggering pattern, with a period equal to the replenishment period, executing an activity whose *wcet* is equal to the budget, and with a deadline less than or equal to the period. If the platform relies on fixed-priority schedulers, the priority assignment criterion consists in assigning the maximum priority to the thread that implements the virtual resource with the minimum deadline, and using classical RMA for the shedulability analysis. The model is more complex if the activities of the virtual resources use mutual exclusion resources, although there are other known RMA techniques for these cases. In our case, a simplified version of the MAST analysis tool [4] is used. It makes it possible to analyse the system with hundreds of virtual resources in tenths of a second.

The negotiation process usually requires performing hundreds of schedulability analysis because the deadlines of the virtual platform have to be modified, keeping the imposed restrictions caused by the timing requirements of the application, until finding a schedulable global configuration.

In Table II, the results of the negotiation for a certain prior workload are shown. The values of the deadlines attributes are the output of this negotiation, and they are compatible with the set of restrictions imposed by the timing requirements of the application, and with the workload of the processing resources of the physical platform. Likewise, the scheduling parameters (priorities) of the threads of the physical platform with which the timing resources are implemented are also assigned.

D. Execution of the RT-Application

Once the virtual resources are instantiated in the physical execution platform (if the negotiation process has been successful), the execution of the application is launched using the resource reservation API of the middleware, binding the threads of the application with the existing virtual resources.

IV. CURRENT STATUS AND FUTURE LINES

The MAST 2 metamodel has been recently proposed. Therefore, updating the tools that form the complete modelling and analysis suite will take some time. Taking advantage of the fact that both MAST 2 and MAST 1 are defined as formal metamodels, MDA strategies have been used to develop the design and analysis tools that use the resource reservation paradigm. These tools process complete MAST 2 models, transforming them into other MAST 2 models that exclusively use modelling elements compatible with MAST 1, to whom the currently available MAST tools are applicable. Thus, the application model is unique and supports the whole resource reservation-based development cycle for real-time applications. However, in each stage, temporary models suitable to the current available MAST tools are generated by lightweight model-to-model (M2M) tools implemented with the ATL language.

As an example, we describe below the model transformation that leads to a model that allows the search of a solution by analysing the application using the MAST schedulability analysis tools. This approach is required when the application has a complex control flow and the timing analysis cannot be performed analytically according to the diagrams shown in Figure 5 and the equations derived from them.

The schedulability analysis is accomplished by transforming the MAST 2 model in such a way that each *VirtualSchedulable Resource* or *VirtualCommunication Channel* is replaced by a *Thread* or *CommunicationChannel* respectively, executed by an independent *ProcessingResource* where a given workload is also being executed. This load introduces contention that forces the worst-case response time in the activity scheduled by the virtual resource. Of course, this worst-case response time value must be compatible with the deadline and budget set in the virtual resource. Figure 8 shows the response time of an activity of *wcet* equal to the budget as a function of the period of the load activity. The *wcet* of the load activity is chosen as the maximum value that makes both activities (application and load) schedulable. We need to find a value of the load period that leads to a response time equal to the deadline of the virtual resource when the *wcet* of the activity is t_B. A load period of t_D (see Figure 8) is the only one

also causing a worst-case response time when the activity executed in the virtual resource has a duration $t_a<t_B$. This period is the one used in the model transformation. Figure 9 shows the modelling elements that form the analyzable model (compatible with MAST 1 tools) of the activity scheduled by the virtual schedulable resource *vrx*. The elements *vrxProc* and *vrxSch* are the processor and the scheduler where the *vrx* thread, which executes the activity of the virtual resource in the analysis model, is scheduled. *RxThLoad* is the higher-priority thread that executes the load activity modelled by the *vrxEtEF EndToEndFlow*.

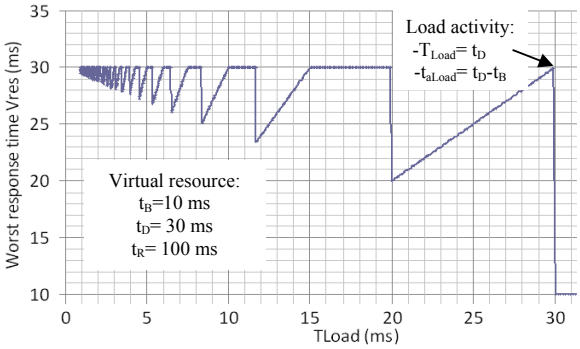


Figure 8: Selecting the load period for the virtual scheduling analysis

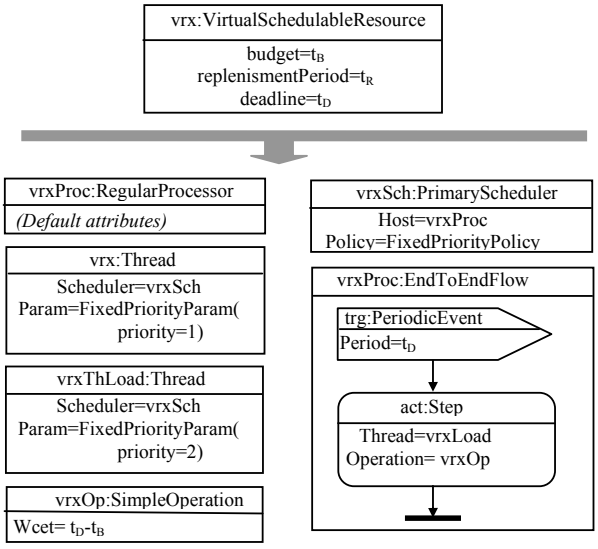


Figure 9: MAST 1 type model for the virtual scheduling analysis

Figure 10 shows the result of the schedulability analysis corresponding to the ServoControl example carried out in this work using the schedulability analysis tools currently available in MAST. Apart from verifying the application schedulability, it also assesses a set of worst-case response times that match those previously obtained from the equations in Section 3.

This paper demonstrates the capacity of the models conforming to MAST 2 to support in a uniform way the different phases of development of a real-time application based on a resource reservation paradigm. This contribution is a starting point towards updating the MAST tools in order to support the new advanced design paradigms for real-time systems covered by MAST 2.

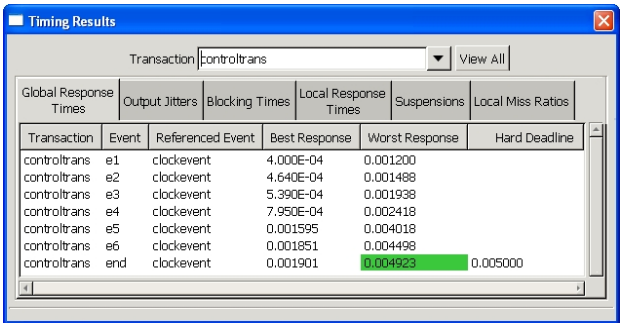


Figure 10: Results of the analysis of ServoControl using MAST 1 tools

The use of a formal UML metamodel to define MAST 2 allows it to cover in a unified manner the models corresponding to different design strategies for real-time systems. Likewise, MAST 2 represents a suitable environment to build a new generation of lightweight tools based on model transformation rules (instead of source code) that easily allow dealing with the new development paradigms for real-time systems recently appeared.

REFERENCES

[1] C. W. Mercer, S. Savage, and H. Tokuda, "Processor capacity reserves: an abstraction of managing processor usage", Proc. 4th Workshop on Workstation Operating Systems (WWOS-IV), 1993.

[2] C. W. Mercer, R. Rajkumar, and J. Zelenka, "Temporal protection in real-time operating systems", Proc. 11th IEEE Workshop on Real-Time Operating Systems and Software, 1994, pp. 79-83.

[3] R. Rajkumar, K.Juvva, A. Molano and S. Oikawa, "Resource kernels: A resource-centric approach to real-time and multimedia systems" Proc. SPIE/ACM Conf. on Multimedia Computing and Networking, 1998.

[4] M. González Harbour, J.J. Gutiérrez, J.C.Palencia and J.M.Drake, "MAST: Modeling and Analysis Suite for Real-Time Applications", Proc. 22nd. Euromicro Conf. Real-Time Systems (ECRTS 2001), 2001. MAST tool: http://mast.unican.es/

[5] Object Management Group. "UML Profile for Modeling and Analysis of Real-Time and Embedded systems (MARTE)" version 1.0, OMG doc. formal/2009-11-02, 2009.

[6] J.Medina and A. García Cuesta, "From composable design models to schedulability analysis with UML and UML profile for MARTE". Proc. of CRTS 2010. 3rd Workshop on pompositional Theory and Technology for Real-time Embedded Systems November 2010.

[7] C. Cuevas, J.M. Drake et all, "MAST 2 Metamodel" http://mast.unican.es/simmast/MAST_2_0_Metamodel.pdf.

[8] J.P.Lehoczky, L.Sha and J.K.Strosnider, "Enhanced aperiodic responsiveness in hard real-time enviroments",Proc. IEEE RTSS 1987.

[9] J. Strosnider, J. Lehoczky and L. Sha, "The Deferrable Server Algorithm for Enhanced Aperiodic Responsiveness in Hard Real-Time Environments", IEEE Transactions on Computers , 44 (1), January 1995.

[10] B. Sprunt, L. Sha, J. Lehoczky, "Aperiodic task scheduling for hard real-time systems", Journal of Real-Time Systems, vol 1, July 1989.

[11] S. Saewong, R. Rajkumary J.P. Lehoczky M.H. Klein:"Analysis of Hierarchical Fixed-Priority Scheduling" Proceedings of the 14 th Euromicro Conference on Real-Time Systems (ECRTS.02).

[12] R.I. Davis and A. Burns: "An Investigation into Server Parameter Selection for Hierarchical Fixed Priority Pre-emptive Systems" 16th International Conference on Real-Time and Network Systems (RTNS 2008

SimTrOS: A Heterogenous Abstraction Level Simulator for Multicore Synchronization in Real-Time Systems *

Jörn Schneider, Michael Bohn, Christian Eltges
Dept. of Computer Science
Trier University of Applied Sciences
Trier, Germany
{*j.schneider, m.bohn, c.eltges*}*@fh-trier.de*

***Abstract*—To provide a common ground for the comparison of real-time multicore synchronization protocols we developed a framework that supports heterogenous levels of abstraction for simulated functionality and simulated timing. Our intention is to make the simulator available to the real-time research community and industrial users. For the latter we initially focus on automotive real-time systems. This paper describes the simulation framework and the novel idea of heterogenous abstraction levels that lies at the heart of its design. Notwithstanding the clear focus, we believe that the simulator itself as well as the concept of heterogenous abstraction levels can be useful in a significantly broader way.**

***Keywords*-real-time, operating system, simulator, multicore, abstraction, synchronization**

I. Introduction

For real-time researchers it is important to demonstrate the benefits of novel approaches in comparison to previous ones. However, it can be a tedious job to achieve a comparison on equal terms, since this usually requires implementing concepts and evaluation infrastructures of other groups. Moreover, the transferability of the evaluation results to an industrial context is quite limited, as the original comparison ignores the caveats of a particular product context for good reasons. Unfortunately, important qualities (e. g. memory consumption, temporal implementation overhead, or energy demand) of a solution are often sensitive to the ignored factors, thus rendering too general results useless.

We believe that many valuable concepts of our community are due to the lack of scalable and transferable evaluations not appreciated in the way they would deserve and the adoption by industry is significantly smaller and slower than it could be. As it is infeasible to investigate each solution a priori in every potential industrial usage scenario, the comparison approaches should allow for a fast re-exploration considering a concrete product context.

We developed the simulation framework described in this paper to tackle these issues for multiprocessor resource locking protocols, especially in the context of automotive real-time systems. Notwithstanding this clear focus, we believe that the simulator core itself can be used for any timing evaluation of multicore real-time systems and moreover, that the novel idea of heterogenous abstraction levels that lies at the heart of its design can also be a key to fast re-exploration when investigating further runtime properties such as memory or energy consumption.

Efforts to migrate legacy applications to multicore systems, such as the one sketched in [1], which led to the development of the described simulator, require sound multicore synchronization mechanisms. Moreover, automotive industry identified in its AUTOSAR real-time operating system standard the need for resource locking protocols in future multicore systems [2]. However, none of the available approaches (e. g. [3], [4], [5]) has been chosen so far, nor was any suitable replacement incorporated into the standard.

Comparison studies such as the ones by Brandenburg, and Anderson [6] and by Lakshmanan, Niz, and Rajkumar [7] are of no help for the automotive industry as they come to diverging conclusions. One result is in favor of spin-based the other in favor of suspension based schemes. In our opinion it is evident that the question which multicore resource locking protocol is better can only be decided for a given characteristic of an application and a given implementation of the protocol in a particular operating system, e. g. AUTOSAR. Furthermore, we conjecture that the HW characteristics of automotive systems such as low processing power can have a significant effect on the outcome of this question.

As briefly described in [8] our SimTrOS simulator is designed to deliver detailed results about the suitability of different protocols and their implementation overhead for the specific characteristics of automotive applications. The design of the simulator was driven by the following central requirements:

1) Modelling of applications and operating system services, e. g. multicore resource locking protocols, shall be decoupled
2) The two simulation concerns functionality and temporal behavior shall be strictly separated
3) Any simulation result shall be deterministically reproducible, i. e. even if a system with race conditions is

*This work was partly supported by the German Federal Ministry of Education and Research (reference No. 17N1309)

simulated, the simulator shall produce the same output if it receives identical input. Otherwise incremental changes of the simulated system might become indistinguishable from random effects. Note that variability of the simulation can be achieved by using different seed values for pseudo random number generation.

II. Related Work

Since the core of the simulator is a discrete event simulation engine, it shares the basic characteristics of these types of simulators. We investigated several of them and found two crucial aspects that none of them addressed in the required combination:

Temporal Granularity: Essential timing issues determining the efficiency of multicore synchronization lie at the RTOS implementation level and even at HW level. A suitable simulator has to support the investigation of these properties, and many simulators do this.

Abstraction Level: The intended users of the simulator shall model the functionality of applications or resource locking protocols at convenient levels of abstraction.

The problem is that simulators that allow for fine grained timing investigation require the user to specify all the functional details at the corresponding low level of abstraction. To address this issue we developed the idea of a simulation framework with heterogenous abstraction levels as it is explained in Section III.

Although we found no simulator that is comparable in this respect, two simulators should be mentioned here. RTSSim [9] is a close match regarding other aspects. It uses a system model written in C-Code, where the actual simulation is running the compiled code, similar to our approach. The key differences are that we use an abstract language instead of C, support multicore, and consequently separate the two concerns simulated *timing* and *functionality*. The other simulator RTSim [10] is also similar and accepts definitions written as C-Code. The main difference to our solution is again, that our simulator differentiates strictly between *timing* and *functionality*.

III. Novelty of the Simulator

The main contribution of this paper is describing a simulation framework that consequently separates different levels of abstraction. It is designed to compare the performance of multicore resource locking protocols from a research or an industrial perspective. Researchers can utilize the simulator to quickly investigate crucial properties of their multicore synchronization approaches such as the question under which basic assumptions is one concept better than another. Practitioners can benchmark different multicore resource locking protocols against one another for a particular application and operating system context in a rapid prototyping fashion. The simulator is implemented in Haskell but requires no user knowledge about functional programming.

A major advantage of the simulator is that it provides a fast and easy way to investigate the same protocol with different underlying timing models and vice versa. This is achieved by the simulators ability to strictly separate the two concerns simulated functionality and simulated timing. In other words, the simulator works with heterogenous abstraction levels for model properties. The timing property can be changed without altering the functional model and the specification of the protocol algorithms can be modified while keeping the same basic timing model.

The benefit of separating the concerns functionality and timing can be illustrated by the following example. The performance of the ready queue management of the scheduler could make a significant difference as it contributes to the overhead of suspension-based schemes. In a normal simulation environment investigating the difference between algorithms maintaining a sorted or unsorted list of jobs would require to implement a complete RTOS scheduler in two versions. Our simulator allows to consider this by just specifying a different temporal behavior while the scheduler is described at a high level of abstraction with the same standard queuing algorithm in any case. Consider how much easier it is for an user of our simulator to evaluate different protocols and implementation flavors of the same protocol in comparison to the state-of-the-art simulation environments.

IV. System Model

The simulator distinguishes between *events* and *effects* of events. An event is a point in time and an effect is a state change. Note that each event has one assigned effect, that an effect can leave the current state unchanged, and that more than one event can happen at the same time.

Periodic and one time events are supported. Periodic events have two parameters: period (reoccurrence interval of events), and phase (time offset for the event sequence). One time events are periodic events with period = ∞.

Events are assigned to *locations* according to their origin. The current framework implements cores and the environment (anything outside of cores) as separate locations. Events from the environment are referred to as *external events* throughout the paper.

On top of the basic concepts of the simulator itself (i. e. events, effects, and locations) an arbitrary model of the simulated system can be established. So far we implemented an AUTOSAR compliant system model with its inherent restrictions. Note that these restrictions, such as a partitioned fixed priority scheduler, are by no means limitations of the simulator itself.

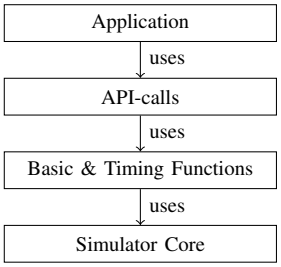


Figure 1. Layered architecture of the simulator

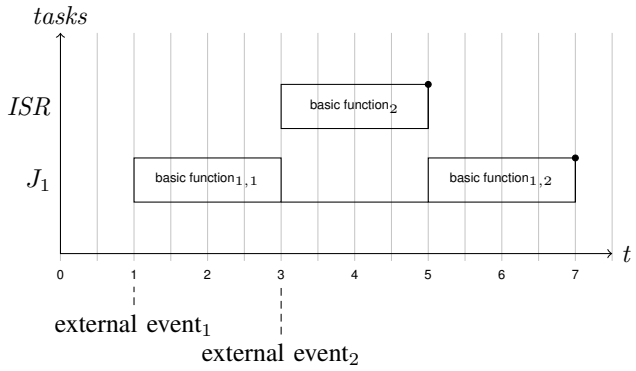


Figure 2. Events of a single job.

V. Architecture and Implementation

A. Architecture

The architecture of the simulation framework is depicted in figure 1. The top layer consists of tasks, interrupt service routines (ISRs), and external events. We denote this set of entities the *application set*.

The application layer uses functionality from the underlying API-call layer. The API-call layer provides two sets of functionality. The first set consists of operating system functionality of the simulated system, classically denoted as *system calls*. The second set of functionality are helper functions that are provided by the simulator.

As applications are composed of API-calls, API-calls are again composed of more primitive functions, the *basic functions*. Basic functions interface directly with the simulator core and only they can directly manipulate the system state of the simulator. Thus the basic functions are the foundation of the simulated functionality. The temporal behavior is specified separately by timing functions. By combining a basic function and a timing function the functional and temporal dimension of executing the simulated code can be freely defined.

The lowest layer consists of the simulator core that is responsible for the control of the simulation.

The first two layers are separated to ease the use by two developer groups: *Users* specify applications on top of predefined API-calls. *Implementors* create abstract implementations of real-time operating systems as API-calls for users to operate with. The distinction between these two groups is of course purely virtual, as one and the same person could be interested in both aspects.

Although the simulator is implemented in Haskell and running a simulation actually means executing compiled Haskell code, users and implementors use only a small subset of the Haskell language. The syntax of this abstract programming language is quite simple and requires no knowledge of the Haskell language.

B. Implementation of the Simulator Core

The simulator core is a discrete event simulation engine that selects one event after another and skips time periods for which no events occur. As the origin of an event is either a processor core or the environment, there are two cases where the system state of the simulator can be changed: finishing the execution of a basic function or the occurrence of an external event.

The simulator computes at which point in time the next event occurs by considering all modeled locations that might produce events. External events are explicitly defined, e. g. every 10 time units event x occurs. The finishing time of a basic function is calculated via the attached timing function. If the nearest future event is the only one that happens at this point in time the simulator selects it as next event to be processed. We first describe an example for these simple cases before considering the situation of two or more events at the same time.

1) Single Events: Consider the single core scenario shown in figure 2. We use the following notation

$$\sigma : SystemState$$

is the simulator internally used state of the system,

$$t : Time$$

is the simulated time,

$$bf_1, bf_2 : SystemState \to SystemState$$

are *basic function*$_1$ and *basic function*$_2$, and

$$tf_1, tf_2 : SystemState \to Time$$

are used to refer to the *timing function* attached to bf_1 and bf_2, respectively. Note that although it is possible to combine different timing functions with the same basic function at different calling points we refrain from using this ability in the examples of this paper.

The simulation starts with time $t = 0$. The next point in time where something happens is $t = 1$ where an external event occurs. The simulator advances t to 1 and derives the effect of the external event (by computing the attached

function of type $SystemState \rightarrow SystemState$). In this example, the event causes J_1 to be the new running job. J_1 executes bf_1.

To determine the next event the simulator calculates the nearest point in time when an event will happen. At time 3 an external event occurs and bf_1 would finish execution at time $t + tf_1(\sigma) = t + 4 = 5$. Since the external event occurs first, the simulator picks it and computes its effect after setting $t = 3$. As the event represents an interrupt request, the computed effect is starting an interrupt service routine (ISR). The ISR (consisting of bf_2) preempts bf_1. The simulator calculates tf_2 for the current system state and derives 2 time units as the execution time of bf_2. Thus the next time point for an event is $minimum(5, \infty)$ as no further external events are specified.

t is advanced to 5 where bf_2 finishes execution and the simulator computes the effect of this basic function on the system state, i. e. $bf_2(\sigma)$. The only effect of bf_2 is to return from the ISR, i. e. restoring the previously running job. bf_1 therefore continues execution. As this basic function already executed 2 time units, the simulator calculates the finishing time to $t + (tf_1(\sigma) - 2) = t + (4 - 2) = 7$. Note that tf_1 is applied to the current system state and thus could deliver a different execution time than beforehand, which is not the case in this example.

Thus the next (and last) event that occurs in the simulation happens at time 7 when bf_1 finishes execution. The effect of this basic function is applied to the system state. The simulator stops, since the next point in time where something happens is ∞.

2) Observations: As demonstrated in the example, effects of external events and basic functions are always applied atomically. This prevents race conditions in the simulator data structures and simplifies the implementation of basic functions. Furthermore, effects of events take zero time.

With the strict separation of basic functions and timing functions, different basic function implementations can be simulated without actually touching the implementation of effects and by only changing the timing function. It is also possible to simulate theoretical cases, where a basic function would take no time at all, e. g. scheduling without scheduler overhead.

Because the absolute time a basic function requires is newly computed whenever the simulator calculates the next event, it is possible that the time already executed on behalf of this basic function, is greater than the newly computed time. In this case the simulator finishes the basic function at the current point in time. Consider for example a basic function *getSpinlock*, which takes no time if the lock is free, but infinite time if the lock is blocked. If a job tries to acquire the lock, but the lock is not free, the job still accumulates calculation time. When the lock is released, the time needed to get the lock becomes 0 and the lock is granted immediately, but is not retroactively granted in the past.

3) Multiple Events at the same Time: Even in the special case where only one core is simulated, two events can happen at the same point in time. In order to fulfill requirement 3 (deterministic execution) the order in which the events are applied has to be the same for every execution of the simulation. Therefore external events occurring at the same time as the completion of a basic function are always considered to happen before the basic function finishes, and all external events have a strict order.

Now consider the case of a multicore system where several basic functions could finish simultaneously on different cores. To solve this problem, the implementor has to define a function that decides which effect of a basic function should be applied first. A simple decision function could apply the basic function from the core with the lowest ID first.

The decision function can inspect all possible next system states and return the new global system state. Setting a system state as the new global system state triggers a new calculation of the nearest effect in the simulator. In case there is more than one effect remaining for this point of time, the decision function is consulted again. Regardless of the decision taken, a log message is written, indicating that a potential race condition of the simulated system was encountered.

Simulating a multicore system is not really different from simulating a single core system. In the single core case the simulator has to calculate the next point in time from two locations, the environment and the one core. In the multicore case the simulator has to consider $n+1$ locations, where n is the number of cores that are simulated. Note that it is possible that more than one basic function finishes execution at the same time on one core, e. g. if two basic functions execute in zero time. The simulator resolves this automatically by the position of the basic function in the task program, i. e. a basic function that was called before another basic function on the same core always applies its effect first.

VI. Defining a simulation

In the following subsections we describe in more detail, how constructs from the three topmost layers are defined.

A. Defining applications

An example for a task definition in the simulator is depicted in listing 1. The most interesting property here is *taskProgram*. A task program defines the behavior of a task and is defined in the imperative language style mentioned before. The program here is strictly sequential, but the abstract task language also supports conditions and loops. A task program is mainly a series of calls to functions provided by the API-call layer.

The semantics of the abstract task programs depends on the API-call implementation. As in a concrete RTOS, the

application developer only sees the interface to the operating system and depends on a detailed description of its exact behavior. The example uses an AUTOSAR based semantics.

A useful function to abstract arbitrary computations is the API-call *time*. It consumes time without changing the state of the simulated system.

Listing 1. Task Example

```
task_i = autosarTask {
          taskPeriod   = 100,
          taskPhase    = 0,
          taskPriority = 1,
          taskName     = "task_i",
          taskCore     = 0,
          taskProgram = do {
              osGetResource "R1";
              time 33;
              osReleaseResource "R1";
              time 5;
              osTerminateTask;
          }
      }
```

The task description template is a wrapper around the primitives event and effect. The task definition in the example is a shortcut for the definition of an event with period 100 and phase 0. The effect of the task definition is then to start an ISR that activates the task on processor core 0.

The definition of an event is very similar to the definition of a task. An example for an event definition is depicted in listing 2 (in fact this example is an alternative way to activate a task). Like a task, an event has a period, phase and a name (but no priority and core ID). In the example, a unique event that occurs at time 70 is defined. The main distinction between a task and an event is the property *eventEffect*. Where a *taskProgram* consists of a series of statements, an *eventEffect* is more general and can be any Haskell function from $SystemState \rightarrow SystemState$.

Since we do not require Haskell knowledge from users, we provide primitive functions that should cover the most basic cases of system state manipulations due to events. In the example an ISR on core 1 is started with the helper function *startISR*. The second parameter can be an arbitrary, abstract program that defines the behavior of the ISR. Another typical case for the effect of an event is changing a variable in the system state. The helper function *updateGlobalVar* can be used to change the value of a global variable.

Listing 2. Event Example

```
event_j = event {
        eventPeriod = Infinity,
        eventPhase  = 70,
        eventName   = "event_j",
        eventEffect = startISR 1 ( do {
                        osActivateTask task3
                      }) -- interrupt on core 1
    }
```

B. Defining API-calls

API-calls are defined in a similar syntax as task programs. Listing 3 shows an example for the definition of *osTerminateTask*, which was used in listing 1. API-calls are composed of basic functions or other API-calls. Note that the function *schedule* is not a basic function, but a composite API-call. Composite API-calls are defined like normal API-calls. What distinguishes them from normal API-calls is that they are only used internally and are not intended to be used in task programs. To distinguish externally visible API-calls, i.e. system calls, from internal API-calls, we prefix the former with *os*. This is purely a convention and not mandatory.

Listing 3. API-call Example

```
osTerminateTask = do {
    setJobVar "state" Suspended; --suspend task
    schedule; --reschedule
}

schedule = do {
    j <- getHighestPriorityJob;
    setRunningJob j; --execute highest priority task
}
```

C. Defining basic functions

The layer below API-calls consists of two parts: functional behavior (basic functions) and temporal behavior (timing functions). The Basic function *getHighestPriorityJob*, used in Listing 3, searches the ready queue for the job with the highest priority and returns the corresponding job number as a result.

For the function *getHighestPriorityJob* one can imagine two implementation strategies: searching through an unsorted list or maintaining a priority ordered list of jobs. In the first case getting the highest priority job would take O(n) steps and in the second case it would take O(1) steps[1]. We can capture these two different timing behaviors by defining the timing functions depicted in Listing 4.

Listing 4. Timing Function Example

```
linearTime s = 10 + 5 * length (getL readyQueue s)

constTime s = 10
```

Note that timing functions can use the complete state of the simulated code to derive the proper execution time for each calling context. This feature is used in Listing 4 to consider the current length of the ready queue.

To combine a concrete pair of basic and timing functions the primitive *makeBasicFunction*, as shown in listing 5 is provided.

Listing 5. Basic Function Definition

```
getHighestPriorityJob = makeBasicFunction linearTime
    getHighestPriorityJobImpl
```

[1]Choosing one or the other implementation has of course an impact on the timing of the corresponding function *addJobToReadyQueue*. Adding a job to an unsorted list takes O(1) steps, adding a job to a sorted list takes at least O(log(n)) steps. Evaluating the impact of these kinds of decisions was the motivation for the development of the simulator.

D. Simulating a system

As described above, a system definition consists of various parts, which are possibly defined by different developers. Such a system, including application code, is compiled to a native executable in order to obtain a most efficient simulator. Since all definitions are pure Haskell code, we use the *Glasgow Haskell Compiler (GHC)* to create simulation executables.

The compiled executable accepts different options to control the simulation. It is possible to run a simulation interactively, simulating one step after another. In non-interactive mode a simulation is executed until completion or up to a specified time limit. During the simulation each call to a basic function and each occurrence of an external event is written to a log file.

VII. Conclusion

We presented the design of a novel simulator. The key idea of heterogenous abstraction levels allows to simulate a system with different timing parameters, without changing the functional behavior. This scheme was chosen to allow for a fast exploration of multicore real-time synchronization protocols in different implementation flavors for their employability in industrial automotive systems. We conjecture that the approach is equally suited to compare other mechanisms such as scheduling regarding their sensitivity to implementation overhead, e. g. context switches. Moreover, the idea of strict separation of timing and functionality is, as we believe, well suited to be used in many simulation approaches in the field of analysis of non-functional properties apart from timing.

The modular design of the simulator and its separation in application, i.e task sets and external events, and operating system layer, i. e. API-calls, is another key feature aimed at easy usage. We intend to provide the simulator as an open tool to the research community and to successively build up a repository of simulation code for real-time operating systems and applications.

We started implementing different multicore resource locking protocols on top of an AUTOSAR system model. The next step is providing better ways to examine simulation results such as a tool that extracts statistics from the produced XML output of the simulator, e. g. average or observed worst-case execution, or blocking times of jobs. For visualization of simulator runs we plan to leverage existing tools, e. g. translating the output of the simulator to the input format of the Grasp tool [11]. A more remote goal could be to generalize the principle of heterogenous abstraction levels for the simultaneous investigation of multiple non-functional run-time properties such as energy demand, or memory consumption in addition to timing.

References

[1] J. Schneider, M. Bohn, and R. Rößger, "Migration of automotive real-time software to multicore systems: First steps towards an automated solution," in *Proceedings Work-In-Progress Session of the 22th Euromicro Conference on Real-Time Systems*, ser. ECRTS'10, July 6–9 2010, pp. 37–40.

[2] *AUTOSAR Release 4.0 — Requirements on Multi-Core OS Architecture*, AUTOSAR Std. 408, Rev. 1, 11 2009, http://www.autosar.org/download/R4.0/AUTOSAR_SRS_MultiCoreOS.pdf.

[3] R. Rajkumar, L. Sha, and J. P. Lehoczky, "Real-time synchronization protocols for multiprocessors," in *IEEE Real-Time Systems Symposium*, 1988, pp. 259–269.

[4] P. Gai, G. Lipari, and M. D. Natale, "Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip," in *In Proceedings of the 22nd IEEE Real-Time Systems Symposium*. Society Press, 2001, pp. 73–83.

[5] A. Block, H. Leontyev, B. B. Brandenburg, and J. H. Anderson, "A flexible real-time locking protocol for multiprocessors," in *RTCSA '07: Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 47–56.

[6] B. B. Brandenburg and J. H. Anderson, "A comparison of the M-PCP, D-PCP, and FMLP on LITMUSRT," in *OPODIS '08: Proceedings of the 12th International Conference on Principles of Distributed Systems*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 105–124.

[7] K. Lakshmanan, D. d. Niz, and R. Rajkumar, "Coordinated task scheduling, allocation and synchronization on multiprocessors," in *RTSS '09: Proceedings of the 2009 30th IEEE Real-Time Systems Symposium*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 469–478.

[8] J. Schneider and C. Eltges, "Towards an evaluation infrastructure for automotive multicore real-time operating systems," in *Proceedings of the 4th international conference on Leveraging applications of formal methods, verification, and validation - Volume Part II*, ser. ISoLA'10. Springer-Verlag, October 18–20 2010, pp. 483–486.

[9] J. Kraft, "RTSSim - a simulation framework for complex embedded systems," Mälardalen University, Technical Report, March 2009. [Online]. Available: http://www.mrtc.mdh.se/publications/1629.pdf

[10] L. Palopoli, G. Lipari, L. Abeni, M. D. Natale, P. Ancilotti, and F. Conticelli, "A tool for simulation and fast prototyping of embedded control systems," in *LCTES/OM*, 2001, pp. 73–81.

[11] M. Holenderski, M. M. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Grasp: Tracing, visualizing and measuring the behavior of real-time systems," in *Proceedings of the 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, 2010.

Grasp: Visualizing the Behavior of Hierarchical Multiprocessor Real-Time Systems

Mike Holenderski, Reinder J. Bril and Johan J. Lukkien
Department of Mathematics and Computer Science
Technische Universiteit Eindhoven (TU/e)
Den Dolech 2, 5600 AZ Eindhoven, The Netherlands

***Abstract*—Trace visualization is a viable approach for gaining insight into the behavior of complex distributed real-time systems. Grasp is a versatile trace visualization toolset. This paper presents its unique visualization capabilities for hierarchical multiprocessor systems, including partitioned and global multiprocessor scheduling with migrating tasks and jobs, communication between jobs via shared memory and message passing, and hierarchical scheduling in combination with multiprocessor scheduling. Its flexible plugin infrastructure allows for easy extension with custom visualization and analysis techniques for automatic trace verification. Grasp is freely available on the web[1].**

I. Introduction

Modern real-time systems are becoming increasingly more complex, with many tasks executing concurrently on many processors, making it difficult to understand the system behavior. A popular trend in coping with the vast number of tasks and the resulting interferences between them is to hide tasks inside components and to integrate the system from those components. This approach requires hierarchical scheduling, which has been covered extensively in the literature for uniprocessor systems. Recently, the real-time literature has been investigating applying hierarchical scheduling to multiprocessor platforms. In this paper we address the problem of how to provide insight into complex interaction patterns between jobs executing in a hierarchical multiprocessor system.

Several approaches are available for tackling the complexity of modern software systems. Ideally, every system would be meticulously documented, providing a formal yet concise description of the emergent system behavior. However, this is a long and costly process without immediate effects (such as additional functionality) and is therefore not common in practice. Examples of poorly documented code and system designs are abundant. The description of the dynamic system behavior therefore needs to be extracted from existing systems. There are modeling and verification tools available, which rely on the developers analyzing the implementation and constructing its model. These tools then employ formal methods to verify the behavior of the extracted model against an abstract model. The state of the art modeling and verification techniques, however, are not scalable and therefore can be applied to verify only a small portion of the entire system.

[1]The work presented in this paper was supported in part by the European ITEA2-CANTATA project. The Grasp toolset together with example traces is available for Linux, Mac and Windows at http://www.win.tue.nl/~mholende/grasp.

Visualization tools offer an interesting alternative. Existing systems can be instrumented to generate runtime traces, which can then be analyzed by engineers and researchers, leveraging their expertise and human capacity to recognize patterns, to gain insight into the system behavior. The challenge here lies in presenting the information in an intuitive way, enabling the user to extract the essential properties of the analyzed system.

Grasp is a toolset for tracing and visualizing the behavior of complex real-time systems. Its main strength lies in providing many different visualizations for various real-time primitives and scheduling techniques in a consistent and intuitive way. Its flexible architecture allows to easily extend it with new visualization and analysis techniques.

We have been using Grasp extensively within our group during our research of embedded real-time systems and the development of various extensions of a commercial real-time operating system μC/OS-II, including a hierarchical scheduling framework and slot shifting. The usage of Grasp has also been reported in [1], [2], [3] where it was used to gain insights into new approaches for hierarchical scheduling in Linux and VxWorks operating systems.

Contributions

In this paper we build on top of our previous work presented in [4], and present Grasp's unique capabilities for visualizing the timing of job execution and communication in the context of:

- partitioned and global multiprocessor scheduling,
- migrating tasks and jobs,
- communication between jobs via shared memory and message passing,
- hierarchical scheduling in combination with multiprocessor scheduling.

Outline

Section II summarizes the related work, followed by an overview of the Grasp toolset in Section III. Grasp's support for multiprocessor scheduling is presented in Section IV and its support for hierarchical multiprocessor scheduling is presented in Section V. Concluding remarks and future work are presented in Section VI.

II. Related work

Existing visualization tools for real-time systems are specialized to visualize a fixed set of behaviors. For example,

the Tracealyzer [5] and TimeDoctor [6] are targeting only non-hierarchical uniprocessor systems. Making a step towards distributed systems is not trivial. Grasp, on the other hand, supports multiprocessor systems with two level virtualization.

There are several trace visualization tools which support the development of parallel programs on uniform parallel-processor platforms, such as VAMPIR [7], Paje [8], or Jedule [9]. They illustrate the execution of parallel jobs and communication between them, but they are limited to flat systems. To the best of our knowledge no visualization tools currently support the visualization of hierarchical scheduling in a uniprocessor or multiprocessor setting.

Traces accepted by most tools are lists of timed events, often in a binary format. Grasp, on the other hand, has adopted the idea of treating the trace as a script, allowing for large degree of flexibility, which was exploited during the development of Grasp's various visualization and analysis features.

There are several tracing tools available, mainly for the Linux platform, which generate traces. Examples include the Data Stream Kernel Interface (DSKI) [10], Ftrace [11], and Dtrace [12]. DSKI is a platform independent interface standard to support collection of a variety of performance data from the operating system internals. It has been implemented on Linux. Ftrace and Dtrace are integrated in many Linux distributions. They exhibit low performance overhead and low memory footprint. In order to leverage their popularity, we have implemented a converter from the sched_switch tracer output of Ftrace, allowing to use Grasp in many Linux and Unix environments.

III. Grasp overview

The Grasp toolset is composed of three entities: the Grasp Recorder, the Grasp Trace and the Grasp Player, as shown in Figure 1.

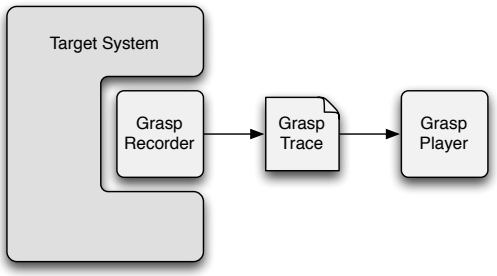


Fig. 1. Overview of the Grasp archictecture.

The Grasp Recorder is embedded in the target system and is responsible for generating a trace. The generated Grasp Trace contains the raw data from a particular system run. The Grasp Player reads in a trace and displays it in an intuitive way.

A. Grasp Recorder

The Grasp Recorder is implemented as a library providing functions to initialize the recorder, log events, and finalize the recorder. Calls to the event logging methods are inserted at several places inside the kernel to log common events, such as context switches, arrival of tasks, or server replenishment. The recorder also provides a function to log custom events, which programmers may call inside their applications.

Designing and implementing an instrumentation infrastructure which exhibits low performance and memory overheads can be a daunting task. Therefore, rather than designing a custom Grasp Recorder and integrating it within the target system, one can implement a converter for existing trace format, leveraging existing instrumentation and tracing tools, as we have done for the sched_switch tracer output of Ftrace.

B. Grasp Trace

The Grasp Trace is a Tcl [13] script. The decision for treating the Grasp Trace as a script results in large degree of flexibility. The Grasp Player basically provides a set of functions which can be called from within a Grasp Trace. A trace can therefore be a simple list of commands, but it can also be a complete system simulator, or anything in between. This allows to embed various extensions (or plugins) inside a trace, resulting in a self-contained trace which can be visualized by any Grasp Player, independent of the plugins it provides. It can also be used to reduce the size of very large traces, by automatically generating or factoring out common or repeating parts. Also, a trace may call methods in the player's public API to override its default settings, making sure that the trace is visualized as intended by its creator. The greatest benefit of the trace being a script, however, is the simple plugin infrastructure discussed in the next section.

A typical Grasp Trace event has the following structure:

`plot` *time event arguments*

which means that *event* has occurred at time *time*. The *arguments* parameter is a list and describes the instance of the *event*. Every *event* defines its own signature, i.e. the number and the semantics of the *arguments* which it accepts. Usually an event accepts a list of required arguments followed by a list of `-key` *value* pairs for optional arguments. In the remainder of this paper we will often ignore the `plot` *time* part, as it is common for many events. Also, we will ignore optional arguments for customizing the trace visualization, such as assigning names or colors to events.

There are several basic events for tracing job execution:

- `newTask` *task* creates a new task, where *task* is a new identifier used in later events.
- `jobArrived` *job task* indicates that *job* belonging to *task* has arrived, where *job* is a new identifier used in later events, and *task* is the identifier of a task created previously with `newTask`.
- `jobStarted` *job* indicates that *job* has started.
- `jobPreempted` *job* indicates that *job* has been preempted.
- `jobBlocked` *job* indicates that *job* has been blocked (e.g. trying to access a locked shared resource).
- `jobResumed` *job* indicates that *job* has been resumed.
- `jobCompleted` *job* indicates that *job* has completed.

An example trace is shown in Figure 2.

```
newTask task1 -priority 7 -name "Task 1"
newTask task2 -priority 8 -name "Task 2"
plot 5 jobArrived job2.1 task2
plot 5 jobResumed job2.1
plot 20 jobArrived job1.1 task1
plot 20 jobPreempted job2.1 -target job1.1
plot 20 jobResumed job1.1
plot 35 jobCompleted job1.1 -target job2.1
plot 35 jobResumed job2.1
plot 50 jobCompleted job2.1
```

Fig. 2. Example of a Grasp Trace.

C. Grasp Player

The Grasp Player is the main contribution of Grasp. It basically provides an execution environment for the script inside of a Grasp Trace. As the Grasp Player is also written in Tcl, its operation is very simple: it loads the definitions of all methods which can be called inside a trace, and then evaluates the trace script. Figure 3 shows an example of a trace of a video processing algorithm. The visualization correlates

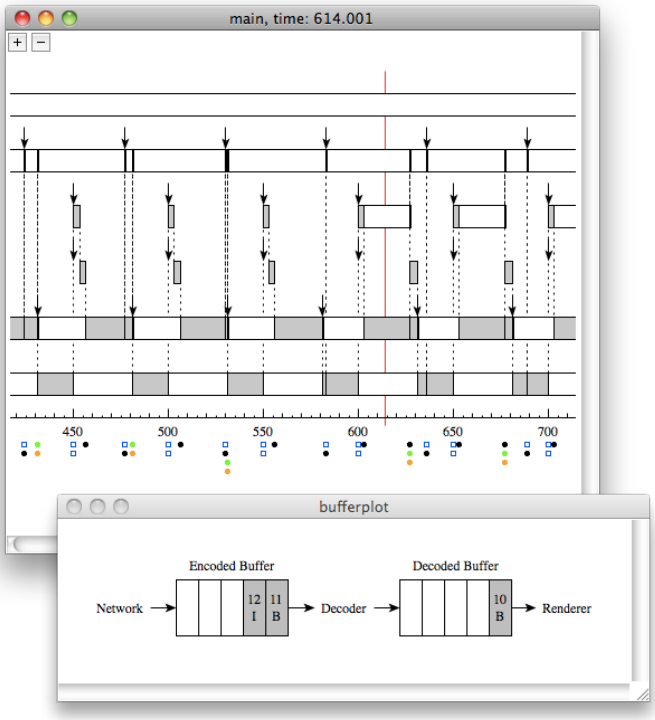


Fig. 3. Example illustrating a video processing application comprised of several tasks (including Network, Decoder and Renderer tasks) executing on a single processor and communicating individual frames of an MPEG video via two shared buffers. As the mouse cursor moves across the trace, the contents of the buffers changes. The figure shows the contents of the buffers at time 614, including the sequence number and the kind of the video frames.

the contents of the frame buffers with the system execution, allowing to inspect their content at different times in relation to the dynamic events occurring during runtime.

The Grasp Player comes with a powerful set of features, including the visualization of task execution in flat and hierarchical systems, uni- and multiprocessor scheduling, intervals in slot shifting, measurement of execution and response times, automatic verification of certain trace properties, command line interface, and exporting to postscript (useful for creating high quality figures for research articles, e.g. Figures 4,5,6, and 8).

Plugins: The Grasp Player provides a simple yet versatile infrastructure for extending it with custom visualization and analysis plugins. For example, the Grasp Recorder extension and Grasp Player visualization plugin for intervals in slot shifting was implemented by a student within hours, extending the budget visualization for servers in hierarchical scheduling.

A plugin has three interfaces at its disposal:

(i) A plugin can define and implement its own methods which can be called within a trace. The Buffer visualization in Figure 3 is an example of such a plugin. It defines methods for tracing the content of buffers via events for adding and removing messages from a buffer:

- `newBuffer` *buffer* creates a new buffer, where *buffer* is a new identifier used in later events.
- `bufferplot` *time* `write` *buffer message* indicates that *message* was added at *buffer*'s tail at time *time*.
- `bufferplot` *time* `read` *buffer* indicates that a message was removed from the *buffer*'s head at time *time*.

(ii) Alternatively, a plugin can register handlers for a set of virtual events, which are generated when the traced events are processed. The Grasp Player provides a method allowing a plugin to register a script which will be evaluated whenever a particular event occurs. For example, the Measurement plugin registers a handler for the `jobArrived` and `jobCompleted` events, to compute the response time of jobs.

(iii) The Grasp Player also provides a set of player events. For example, a plugin can register a script which will be called upon the `TimeChanged` event, which is generated when the mouse cursor is moved across the trace. This player event is used by the Buffer plugin to illustrate the buffer content at the time pointed to by the mouse cursor (e.g in Figure 3).

The simple plugin infrastructure is made possible by the Grasp Trace being a script. Other visualization tools rely on a "dispatch" method which is called for each event in the trace to dispatch the corresponding event handler. Extending such tools with new events requires to modify the dispatch method (or to limit the syntax of traced events). As the the Grasp Trace simply calls methods provided by the Grasp Player, there is no need for a dispatch method. Extending the Grasp Player with a plugin requires simply to place the plugin script inside of the plugins directory (which is automatically included when the player starts).

Automatic verification: The plugin infrastructure can be leveraged to implement various verification tools for automatically analyzing the system behavior in a trace. For example, the BudgetCheck plugin shows a warning when a server exceeds its budget, and the MutexCheck plugin verifies proper nesting of mutex locking events inside a trace.

For any given target system, if a particular behavior is expected, then a "test-suite" plugin may be implemented to

verify that for a specific scenario the target system satisfies the desired properties, e.g. after a maintenance activity.

IV. Multiprocessor visualization

In this section we present Grasps support for multiprocessor systems. The multiprocessor support is implemented by extending a subset of events for tracing job execution with an optional `-processor` argument.

Our goal was to support various concepts commonly found in multiprocessor scheduling. Grasp supports partitioned as well as global multiprocessor scheduling with task and job migration, and communication between jobs on shared and distributed memory platforms. In this section we discuss each of these features in more detail.

A. Creating a processor

Similar to other objects in a trace, such as tasks or servers, a processor needs to be created before it can be referred to in other trace events.

- `newProcessor` *processor* creates a new processor, where *processor* is an identifier which can be added to other trace events to support multiprocessor visualization.

B. Partitioned and global scheduling

In partitioned scheduling, each task is assigned to a particular processor and during runtime all of its jobs execute on that processor. In global scheduling, different jobs of the same task may execute on different processors.

Partitioned scheduling: When a task is created, it can be assigned to a particular processor:

- `newTask` *task* `-processor` *processor* creates a new *task* and assigns it to the *processor*.

All subsequent job events will be mapped to the *processor* (unless the processor argument is overridden, as discussed in the next section). Figure 4 shows an example of a trace on partitioned multiprocessor platform.

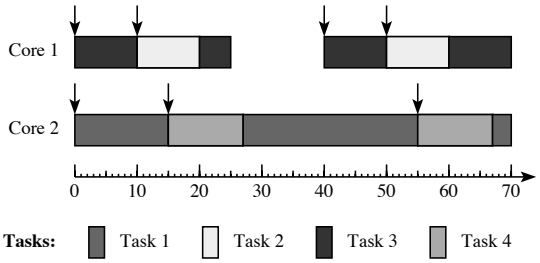


Fig. 4. Example illustrating the execution of five tasks on a partitioned multiprocessor platform consisting of two cores.

Figure 4 shows the system behavior in a collapsed view, where the execution of all tasks is collapsed on a single timeline. Alternatively, the Grasp Player also supports an expanded view, where each processor is shown in a separate window illustrating the interactions between the local tasks, as shown for a single processor in Figure 3.

Note that the Grasp Player provides many details upon a mouse click. For example, when the mouse is clicked on top of a downward pointing arrow, a message is shown telling which task has arrived. These features are difficult to visualize in a paper.

Global scheduling: In global scheduling we can distinguish between task and job migration (also referred to as restricted- and full-migration scheduling, respectively [14]). When only task migration is allowed, then tasks are allowed to migrate between processors, however, each job must execute on one processor. When job migration is allowed, then jobs may migrate between processors, i.e. they can halt on one processor and resume on another. Grasp supports both task and job migration by having the `jobArrived`, `jobStarted`, and `jobResumed` events accept an optional `-processor` argument. In a trace containing only task migration only the `jobArrived` event will specify the `-processor` argument. In a trace containing job migration also the `jobStarted` and `jobResumed` events will specify the `-processor` argument. Figure 5 illustrates job migration by having the first job of task 1 arrive at time 15 on core 2 and later at time 22 migrate to core 1.

```
newProcessor core1 -name "Core 1"
newProcessor core2 -name "Core 2"
newTask task1 -name "Task 1" -color orange3
...
plot 15 jobArrived job1.1 task1 -processor core2
plot 15 jobPreempted job4.1
plot 15 jobResumed job1.1
...
plot 22 jobPreempted job1.1 -processor core1
plot 22 jobPreempted job3.1
plot 22 jobResumed job1.1 -processor core1
plot 22 jobResumed job4.1
...
```

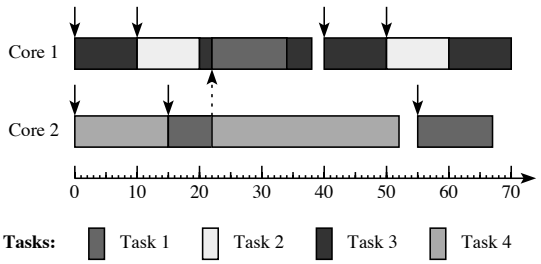


Fig. 5. Example showing a partial trace and the corresponding visualization, illustrating the migration of a job. At time 22 the first job o task 1 migrates from core 2 to core 1, indicated by the dashed arrow.

C. Communication between jobs

Depending on the memory architecture in a multiprocessor system, jobs can communicate via shared memory or via message passing.

Shared memory: When jobs executing on different processors communicate via shared memory, it is critical to maintain the data consistency of the shared data structures. A common approach is using mutexes. Grasp provides events for acquiring

and releasing a mutex, as shown in the example in Figure 6. The relevant events are:

- `jobAcquiredMutex` *job mutex* indicates that *job* has acquired *mutex*.
- `jobReleasedMutex` *job mutex* indicates that *job* has released *mutex*.

The arguments *job* and *mutex* are identifiers for a previously created job and mutex, respectively.

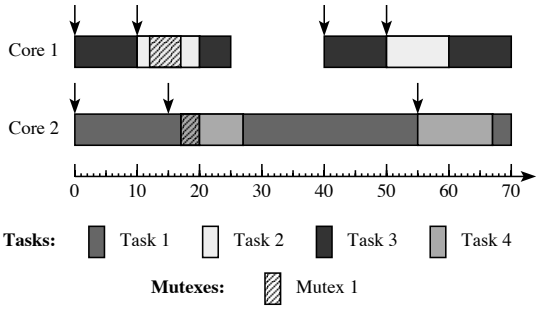


Fig. 6. Example showing tasks 2 and 4 using a mutex to communicate via shared memory.

Figure 6 shows an example of two tasks communicating via shared memory. At time 12 task 2 locks Mutex 1 guarding a shared memory location. When task 4 arrives at time 15 it finds the shared mutex in a locked state and is suspended. At time 17, when task 2 unlocks the mutex, task 4 is able to resume and lock Mutex 1 to read the data communicated from task 2.

Message passing: On a distributed memory platform jobs communicate via message passing. A popular example is the Message Passing Interface (MPI) [15]. We reuse the Buffer plugin [4] for this purpose. Depending on the communication paradigm (one to one, broadcast, multicast), we create the appropriate message buffers.

When the mouse cursor is dragged inside of the Grasp Player window, the contents of the buffers is animated, reflecting their state at the current time, indicated by the long vertical red line. Clicking on a buffer element reveals more message details (in case they were provided in the trace).

Figure 7 shows an example of tasks 1 and 2 communicating via message passing. At time 12, there are 2 messages A and B from task 2 inside of a message buffer, waiting for task 1 to read them.

D. Merging traces from different processors

A Grasp Trace can list the events in (nearly) any order. In a distributed multiprocessor system this allows to record traces on each processor individually, and then to simply concatenate the traces to form a single system trace, without the need for interweaving them.

Notice that the traced event times are local to the processor where the events occur. This means that in a multiprocessor setting, for the events on different processors to align properly, Grasp relies on the time being synchronized between the processors. We plan to alleviate this restriction in future work.

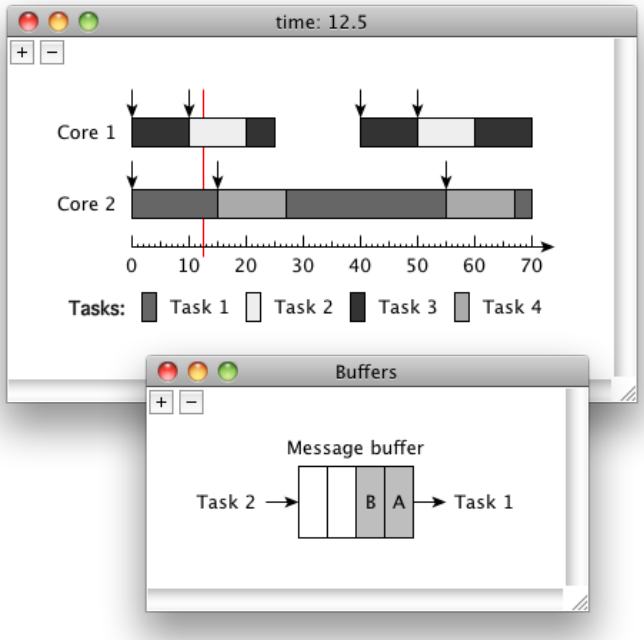


Fig. 7. Example showing tasks 1 and 2 using a buffer to communicate via message passing.

V. Hierarchical multiprocessor visualization

In [4] we have introduced Grasp's support for hierarchical scheduling in uniprocesor systems. The events for tracing the budget of a server are:

- `serverReplenished` *server budget* indicates that *server*'s remaining budget was replenished to *budget*.
- `serverResumed` *server* indicates that a task has started consuming *server*'s budget.
- `serverPreempted` *server* indicates that a task has stopped consuming *server*'s budget.
- `serverDepleted` *server budget* indicates that *server*'s remaining budget has been depleted.

In this section we elaborate on the combination of hierarchical and multiprocessor scheduling.

Using the standard hierarchical scheduling support, the Grasp Player is not aware of the task-to-server mapping, nor of the desired behavior of particular server types (such as periodic-idling or deferrable server). The hierarchical scheduling events pertain only to the replenishment, depletion and consumption of server's budget. The target system is responsible for generating the correct behavior. However, the Grasp Player can be easily extended with a verification plugin, making sure that the server behavior is according to its specification, e.g. that only tasks assigned to the server consume its budget, or that a periodic idling server always idles its budget away.

The fact that Grasp is not aware of the mapping between servers and tasks allows to easily trace systems where tasks consume budgets from *several* servers, and systems where a server is serving its budget to *several* tasks executing at

the same time on different processors [16]. The latter is accomplished by allowing several `serverResumed` events to occur in a trace without a corresponding `serverPreempted` event in between. This provides a very simple way for tracing budget consumption in a multiprocessor setting: whenever a task assigned to a budget is resumed on any processor, the corresponding server is also resumed. Similarly, a server is preempted whenever a task consuming its budget is preempted on any processor.

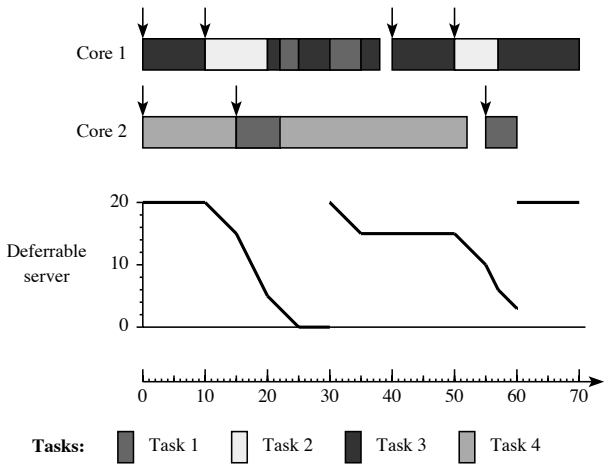


Fig. 8. Example showing a trace visualization of a hierarchical multiprocessor system, where a deferrable server with period 30 and capacity 20 is serving its budget to tasks 1 and 2.

Figure 8 shows an example visualization of such a system, where a deferrable server with period 30 and capacity 20 is serving its budget to tasks 1 and 2. Tasks 3 and 4 are not bound to any server. At time 10, when task 2 arrives, it starts consuming server's budget. At time 15, when task 1 arrives, it also starts consuming server's budget. The budget is consumed at twice the rate until task 2 completes at time 20.

VI. Conclusions

Grasp is a visualization toolset aiming to provide insight into the behavior of complex real-time systems. In this paper we have presented its unique features for visualizing hierarchical multiprocessors scheduling. It provides various visualizations for partitioned and global multiprocessor scheduling with migrating tasks and jobs, communication between jobs via shared memory and message passing, and hierarchical scheduling in combination with multiprocessor scheduling.

Future work

Currently, Grasp relies on time synchronization between the processors for proper alignment of tasks and servers executing on different processors. This is sufficient for multicore platforms which have synchronized clocks between their cores. However, in a more general distributed setting, where the time skews may become significant, the events on different processors may be shifted in time. As future work, we would like to address such systems by for example introducing synchronization events allowing the Grasp Player to synchronize the traces from different processors, e.g. using the approach in [17].

References

[1] M. Åsberg, J. Kraft, T. Nolte, and S. Kato, "A loadable task execution recorder for linux," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2010.

[2] M. Åsberg, T. Nolte, and S. Kato, "Towards hierarchical scheduling in linux/multi-core platform," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2010.

[3] M. van den Heuvel, R. Bril, and J. Lukkien, "Protocol-transparent resource sharing in hierarchically scheduled real-time systems," in *IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, September 2010.

[4] M. Holenderski, M. M. H. P. van den Heuvel, R. J. Bril, and J. J. Lukkien, "Grasp: Tracing, visualizing and measuring the behavior of real-time systems," in *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*, July 2010.

[5] M. I. Mughal and R. Javed, "Recording of scheduling and communication events on telecom systems," Master's thesis, Mälardalen University, 2008.

[6] TimeDoctor. http://sourceforge.net/projects/timedoctor/.

[7] W. E. Nagel, A. Arnold, M. Weber, H. C. Hoppe, and K. Solchenbach, "Vampir: Visualization and analysis of MPI resources," *Supercomputer*, vol. 12, pp. 69–80, 1996.

[8] J. C. D. Kergommeaux, B. D. O. Stein, and M. S. Martin, "Paje: An extensible environment for visualizing multi-threaded program executions," *LNCS 1900*, 2000.

[9] S. Hunold, R. Hoffmann, and F. Suter, "Jedule: A tool for visualizing schedules of parallel applications," in *International Conference on Parallel Processing Workshops (ICPPW)*, 2010, pp. 169–178.

[10] B. Buchanan, D. Niehaus, S. Sheth, and Y. Wijata, "The data stream kernel interface," University Of Kansas, Tech. Rep. ITTC-FY98-TR11510-04, June 1998.

[11] Ftrace. https://rt.wiki.kernel.org/index.php/ftrace.

[12] Dtrace. http://www.sun.com/bigadmin/content/dtrace/.

[13] B. Welch, K. Jones, and J. Hobbs, *Practical Programming in Tcl and Tk*. Prentice Hall, 2003.

[14] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah, *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004, ch. A Categorization of Real-Time Multiprocessor Scheduling Problems and Algorithms, pp. 30–1 – 30–19.

[15] P. Pacheco, *Parallel Programming with MPI*. Morgan Kaufmann, 1996.

[16] I. Shin, A. Easwaran, and I. Lee, "Hierarchical scheduling framework for virtual clustering of multiprocessors," in *Euromicro Conference on Real-Time Systems (ECRTS)*, July 2008, pp. 181 –190.

[17] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," *SIGOPS Operating Systems Review*, vol. 36, pp. 147–163, December 2002.

Modeling Real-Time Networks with MAST2

Michael González Harbour, J. Javier Gutiérrez, J. María Drake, Patricia López, and J. Carlos Palencia

Computers and Real-Time Group, Universidad de Cantabria, 39005-Santander, SPAIN
{mgh, gutierjj, drakej, lopezpa, palencij}@unican.es

Abstract[1]

Switched networks have an increasingly important role in real-time communications. The IEEE ethernet standards have defined prioritized traffic (802.1p) and other QoS mechanisms (802.1q). The Avionics Full-Duplex Switched Ethernet (AFDX) standard defines a hard real-time network based on switched ethernet. In the process of defining the new MAST2 model, the network elements have been enhanced to include switches and routers. This paper introduces the schedulability model that will enable an automatic schedulability analysis of an application using switched networks.

1. Introduction

MAST (Modeling and Analysis Suite for Real-Time Applications) [2][4] defines a model to describe the timing behavior of real-time systems designed to be analyzable via schedulability analysis techniques. MAST also provides an open-source set of tools to perform schedulability analysis or other timing analysis, with the goal of assessing whether the system will be able to meet its timing requirements, and, via sensitivity analysis, how far or close is the system from meeting its timing requirements.

The model defined in MAST is very similar to the model defined in the standardized UML profile for real-time embedded systems called MARTE [5]. A new enhanced model is currently being defined as a project called MAST2, trying to incorporate new modelling elements that can be found in real systems. It is expected that the ideas introduced in MAST2 will contribute to the future evolution of the MARTE standard.

Some of the new elements being defined in MAST2 are network switches and routers. Switched networks are being used increasingly to build real-time systems, as new network switches incorporate the real-time mechanisms being defined in standards such as IEEE 802.1p with prioritized traffic, 802.1q with various QoS mechanisms [6], or the Avionics Full-Duplex Switched Ethernet (AFDX) [1] that defines a hard real-time network based on switched ethernet.

1. This work has been funded in part by the Spanish Ministry of Science and Technology under grant number TIN2008-06766-C03-03 (RT-MODEL), and by the European Commission under project FP7/NoE/214373 (ArtistDesign).

This paper introduces the model elements required to add network switches and routers into the MAST model. These elements will allow an automatic schedulability analysis of applications using switched networks.

The paper is organized as follows. In Section 2 we present a general overview of the MAST2 model, and we focus on the network modelling elements in Section 3. The new elements introduced to model network switches are presented in Section 4, and similarly in Section 5 for network routers. Section 6 introduces the new modelling elements for AFDX networks and switches together with a simple example using these elements. Finally, Section 7 gives our conclusions.

2. Overview of the MAST2 Model

A real-time system is modelled in MAST2 using different independent views for describing the execution platform, the software modules and messages exchanged through the networks, the concurrent architecture, and the workload and flow of events for a particular configuration of the application. This independence among the various elements of the model is ideal for building a full model through the composition of partial models developed independently.

The execution platform view contains *Processing_Resources* such as *Processors* and *Networks*, together with their *Schedulers* and associated *Scheduling_Policy* elements. Each of these elements contains attributes that describe their timing behavior including overheads such as context switching, interrupt service, or system timers. *Processors* have a relative speed and *Networks* have a bandwidth expressed in bits per time unit. MAST2 also defines elements to describe clock synchronization mechanisms.

The operations view contains the elements that describe the usage of the processing resources, through *Operations*. *Code_Operations* model the execution of sequential code with a given execution time distribution, while *Message_Operations* represent data of a given size that is sent through a network. The *Code_Operation* abstract class is specialized with *Simple_Operation* and *Composite_Operation*, while *Message_Operation* is specialized with *Message* and *Composite_Message*.

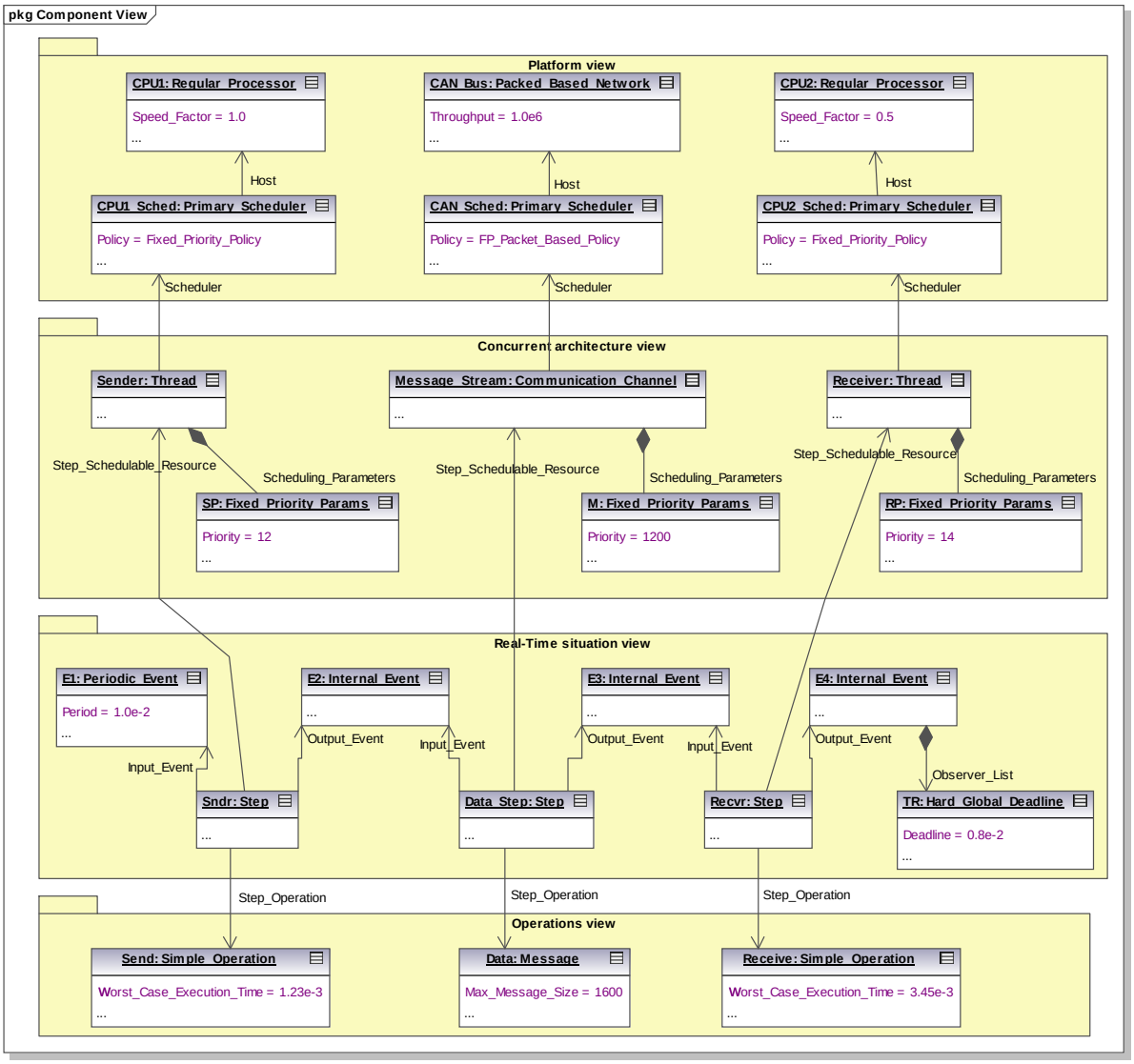


Figure 1. MAST2 elements of the simple distributed example

The concurrent architecture view contains the *Schedulable_Resources* that represent schedulable entities in a *Processing_Resource*. The *Thread* models a unit of concurrent execution in a processor (a task, single-threaded process, thread, or interrupt service routine) while the *Communication_Channel* models a message stream transmitted through a network with specific scheduling parameters. *Schedulable_Resources* have a reference to the *Scheduler* where they are scheduled, and the associated *Scheduling_Parameters* that are used by the scheduler to arbitrate access to the processing resource.

The real-time situation view represents a particular mode of operation of a real-time system. It is modelled as a set of concurrent *End_To_End_Flows* (previously called transactions) that compete for the resources offered by the platform. Each end-to-end flow is activated from one or more *Workload_Events*, and contains a sequence of *Steps* that are executed in the system. Each *Step* represents the execution of an *Operation* (*Code_Operation* for a processor or *Message_Operation* for a network) by a *Schedulable_Resource*. When a *Step* finishes its execution it generates an *Internal_Event* that may, in turn, activate other *Steps*. Special *Event_Handlers* exist in the model to

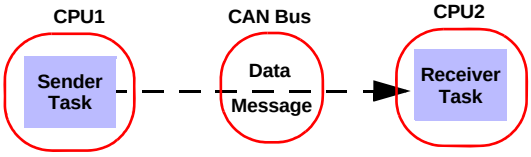


Figure 2. Example with a simple distributed application

handle events in special ways. *Internal_Events* may have *Observer*s associated with them , and rep resenting timing requirements or other requirements that must be observed.

We will use the example with a simple distributed application depicted in Figure 2 t o show the MAST2 elements related to the networks. The application has a periodic task sending a message through a CAN bus to a second task executing in a different processor, and its model is shown in Figure 1. In the real-time situation view we can see th e single end-to-end flow consisting of three steps joined through the corresponding events, with the first event defining the periodicity, and the last one with a hard end-to-end deadline.

3. Networks

A *Network* is a ki nd of *Processing_Resource* that models a communication system specialized in the transmission of messages among processors or net work switches.

Figure 3 show s the class diag ram of t he networks in MAST2. The main element th at represents a real-time network is the *Packet_Based_Network*, which models a network that uses some kind of real-time protocol based on non-preemptible packets for sending mes sages. This element can represent a networ k that supports priorities in its standard protocol, such as th e CAN bus, and other networks with no priorities such as the point-to-point ethernet links tha t are used to connect CPUs to/from network switches.

Other priority-based networks may need more complex models. For instance, the *RTEP_Network* represents a network that uses the RTEP protocol [3], which is a token-passing protocol with prioritized messages, that uses a two-phase mechanism to send each information packet: a priority arbitration phase and the transmission phase.

Network switches were n ot part of the original MAST model, neither are defined in the MA RTE standard. We will now introduce modelling elements to represent the network switches and the cont ention inside them. To support the A FDX networks we wi ll need to introduce a special *AFDX_Switch* and the *AFDX_Link*, as a new kind of network. Both will be described in Section 6.

4. Network Switches

A network switch is a communications subsystem that is capable of establishing simu ltaneous connections between a number of input networks and output networks. These connections are establ ished on a per-message basis. In a store and forward switch each message arriving at an input port is stored in the switch's internal memory. As t he message carries information on its dest ination port, the switch forwards this message to the output port, or to the specified output ports in case of a multicas t or broadcast message. The switch is capable of managing multiple such connections simultaneously, through replicated hardware. This ability allows a full usage of the available network bandwidth, depending on the traffic source and destination addresses.

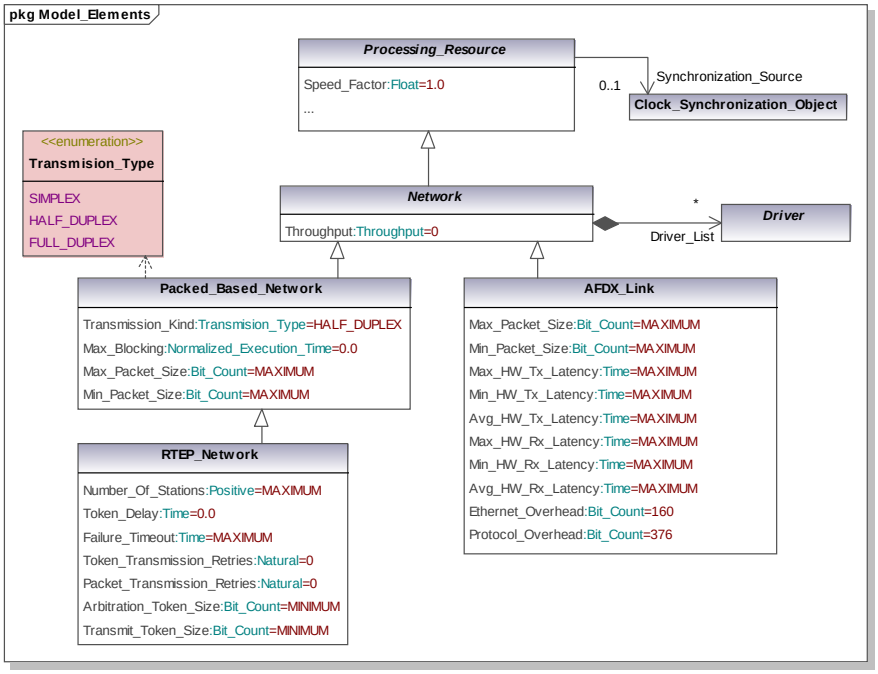


Figure 3. Network modelling elements

Contention may occur in the switch when several messages need to be forwarded to the same output port concurrently. The switch usually manages a queue of messages addressed at a specific output port. Some switches manage this queue in FIFO order, and others may use information contained in the message to prioritize the queue.

New modelling elements are introduced in MAST2 to model the timing effects introduced by network switches, and in particular the worst-case behavior of the contention in the output ports.

The *Network_Switch* (Figure 4) is an abstract *Processing_Resource* that models a communication system specialized in the transfer of messages among networks. It is capable of delivering messages arriving at an input port to one or more output ports.

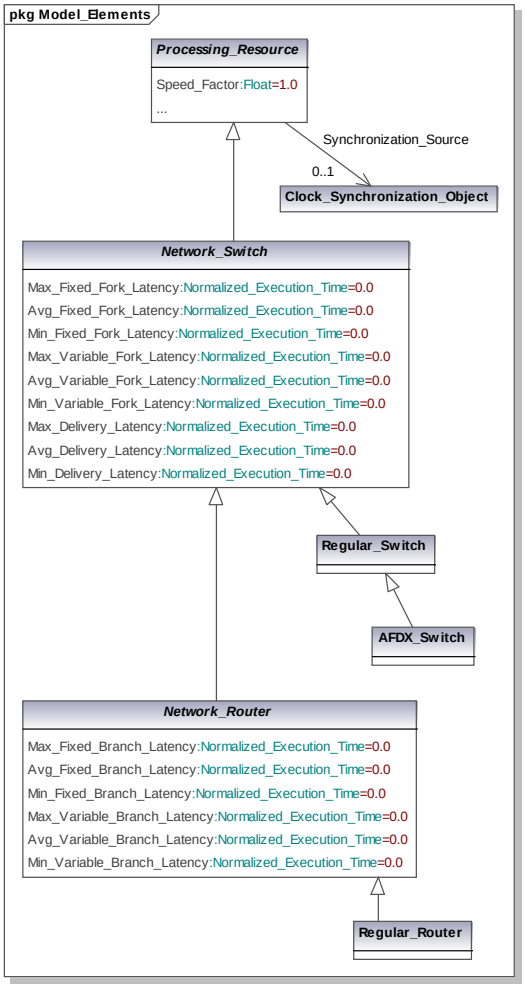


Figure 4. Network switches and routers

The delivery of a message inside a switch is specified through new special-purpose *Message_Event_Handlers*, described in Figure 5. The *Message_Delivery* element represents a simple port-to-port delivery, while the *Message_Fork* handler enables modelling multicast messages. The destination port or ports are implied in the message stream.

The *Message_Delivery* may contain scheduling parameters to define the priority used to resolve the contention at the output port queue in the switch.

The attributes of the *Network_Switch* allow calculating the overheads associated with the message delivery. Max/Min/Average overheads are defined for the delivery of a message to a single destination port, and for the delivery to multiple destinations through a *Message_Fork* handler. This latter overhead has a fixed part and a variable part that depends on the number of output ports.

A *Regular_Switch* is a concrete and simple switch in which the output queues are assumed to work according to the policy associated with the scheduling parameters of the message delivery elements allocated to the switch. For instance, priority-based in case of fixed-priority parameters, or FIFO when no scheduling parameters are specified. The switch is assumed to have routing information preconfigured, so there is no need for additional network traffic originated by the switch.

MAST2 does not require defining the network topology, since it is implicitly defined in the end-to-end flows.

5. Network Routers

A *Network_Router* is an abstract specialization of the *Network_Switch* that models a communication system specialized in routing messages among networks. In addition to the capabilities of a switch, the router is able to route a message through a destination port that may be dynamically obtained.

Routers support a new *Message_Branch* event handler, which is capable of delivering a message to one output port chosen form of a set of possible outputs. The overhead model for the *Message_Branch* handlers is similar to the model for message fork. It contains a fixed latency and a variable latency that depends on the number of possible output ports.

The class *Regular_Router* is a concrete and simple router in which the output queues are assumed to work according to the policy associated with the scheduling parameters of the allocated message delivery elements.

6. Modelling AFDX Networks

AFDX (Avionics Full Duplex Switched Ethernet) is a communications network defined in the ARINC-644, Part

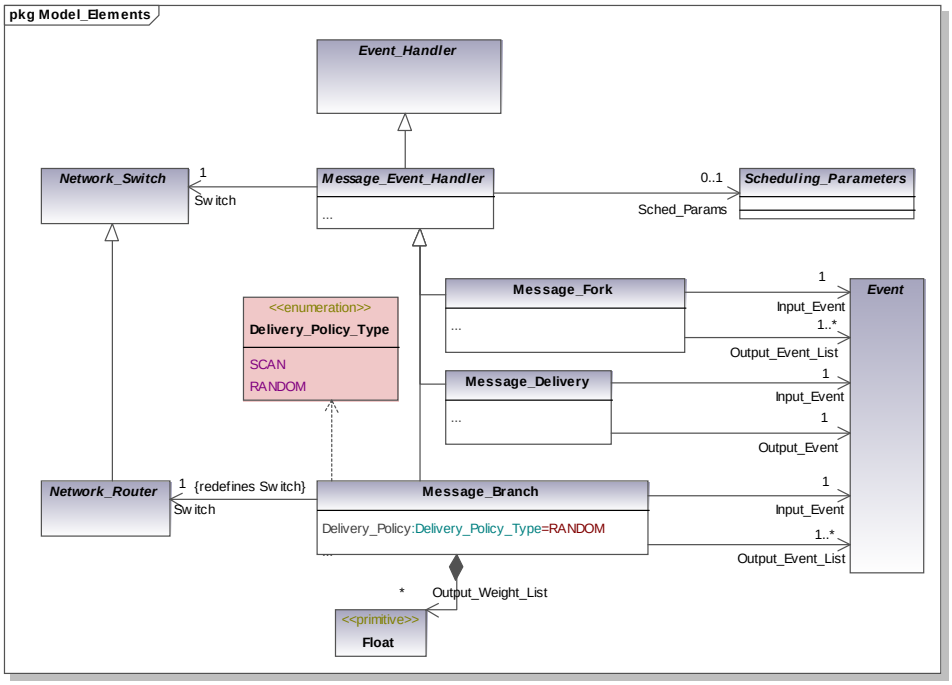


Figure 5. Message event handlers

7 standard [1] and based on the use of point to point full duplex ethernet links and special purpose switches. The routing of messages is preconfigured so that there is no delay in the discovery of routing addresses through network protocols that could interfere with the transmission of application messages.

The AFDX network provides traffic regulation at the sending end via *Virtual Links*, defined as conceptual communication objects to establish a logical unidirectional connection from one source end system to one or more destination end systems having a dedicated maximum bandwidth. Each virtual link (VL) is characterized by two parameters used for traffic regulation: the largest Ethernet frame (*Lmax*) in bytes, and the Bandwidth Allocation Gap (*BAG*), a power of the value 2 in the range [1,128]. The *BAG* represents the minimum interval in milliseconds between Ethernet frames transmitted on the VL.

Each virtual link has a FIFO queue for all the fragmented packets to be transmitted through it. This queue introduces contention that needs to be modelled and analyzed to determine real-time schedulability.

The AFDX network is modelled in MAST2 through two new elements: The *AFDX_Link* network, and the *AFDX_Switch*.

An *AFDX_Link* (see Figure 3) represents a link between a processor and/or a switch, allowing full duplex communications. When a message originated in a processor traverses this link, it uses the new *AFDX_Policy* scheduling policy. The *Schedulable_Resources* that may be used on this link need to specify as *Scheduling_Parameters* an object of the *AFDX_Virtual_Link* class, which describes the corresponding *Lmax* and *BAG* parameters as it is shown in Figure 6.

An *AFDX_Switch* is a concrete and simple switch working according to the AFDX specification. The output queues in the switch have two priorities. The priority may be assigned in the model by specifying it in the scheduling parameters of the *Message_Delivery* operation executed inside the switch.

Figure 7 shows a partial view of the example described in Figure 2 and Figure 1, focussing only on the network part, and assuming that instead of a CAN Bus we are using

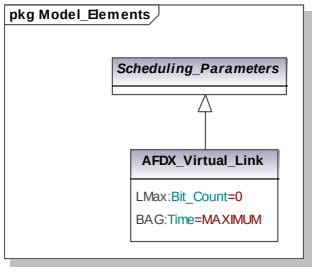


Figure 6. Virtual Link Scheduling Parameters

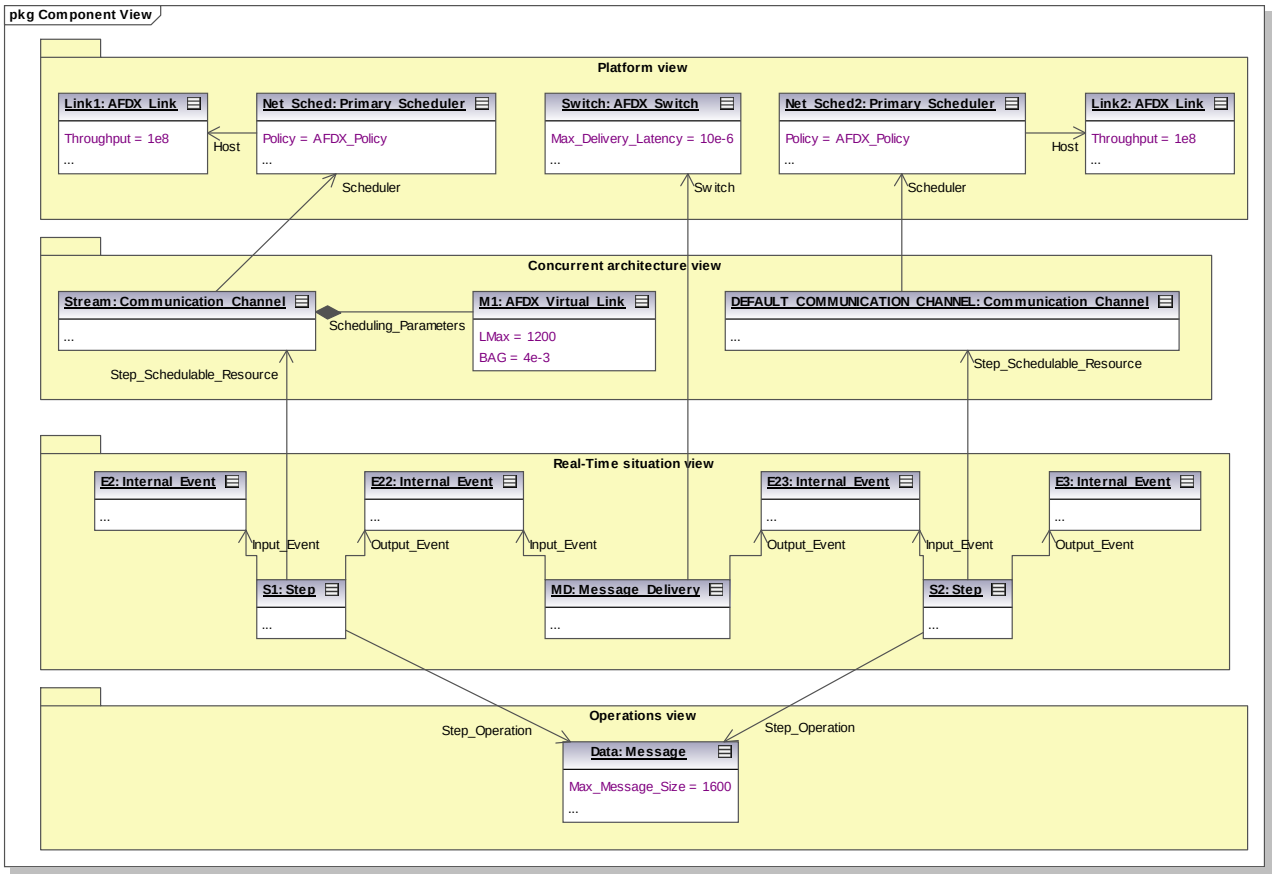


Figure 7. Partial view of an end-to-end flow using an AFDX switch

an AFDX network with a switch and two AFDX links communicating each CPU with the switch. No priority is specified.

7. Conclusions

In this paper we propose new modelling elements to support network switches and routers and AFDX real-time networks in the timing models used to assess the schedulability of real-time distributed applications. These elements, together with their associated analysis techniques are being implemented in MAST2, and will be proposed for a future version of the MARTE UML profile for real-time embedded systems.

References

[1] Airlines Electronic Engineering Committee, Aeronautical Radio INC., "ARINC Specification 664P7: Aircraft Data Network, Part 7 - Avionics Full Duplex Switched Ethernet (AFDX) Network," June 27, 2005.

[2] M. González Harbour, J.J. Gutiérrez García, J.C. Palencia Gutiérrez, and J.M. Drake Moyano. "MAST: Modeling and Analysis Suite for Real Time Applications". Proceedings of 13th Euromicro Conference on Real-Time Systems, Delft, The Netherlands, IEEE Computer Society Press, pp. 125-134, June 2001.

[3] J.M. Martínez, and M. González Harbour. "RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet". Proc. of the 10th International Conference on Reliable Software Technologies - Ada-Europe 2005, York (UK), in LNCS, Vol. 3555, Springer (2005).

[4] MAST: Modelling and Analysis Suite for Real-Time Systems. Home page: `http://mast.unican.es`

[5] Object Management Group. "A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems". MARTE specification version 1.0 (formal/2009-11-02).

[6] Xu, Lei. "Industry Network QoS Approach Based on New Ethernet Standards". Proceedings of 2009 4th International Conference on Computer Science & Education.

Continuous Constant-Memory Monitoring of Embedded Software Timing

Johan Kraft and Thomas Nolte
School of Innovation, Design and Technology
Mälardalen University
Västerås, Sweden
{johan.kraft, thomas.nolte}@mdh.se

***Abstract*—A method is presented for generating statistical models of timing data continuously over very long monitoring sessions. This method is intended for memory-efficient runtime modeling of timing properties in embedded software systems, such as execution times or inter-arrival times, but is a quite generic method that should be applicable for other purposes and domains as well. Specifically, we intend to use this method as a component in automatic generation of simulation models for probabilistic timing analysis of complex embedded software systems. Given a stream of data as input, this method gradually builds up a statistical model capturing the approximate distribution of the data. The method uses a modest and fixed amount of on-target RAM, decided by the desired accuracy of the model, and allows for long monitoring sessions covering billions of data points. The paper presents the motivation, algorithm, a prototype implementation and evaluation using real execution time data from an ARM7 microcontroller.**

I. Introduction

If developers of embedded software systems were able to predict runtime properties related to timing and resource usage before implementing new software designs, they could identify and avoid unsuitable software designs at an early stage and thereby also avoiding the associated costs and delays as a result of having to redesign the software when the system verification (hopefully) detects the problems. Predicting task response times is possible for many systems using established methods for schedulability and response time analysis [1], [2], [3] or tools for formal verification using model checking, such as UPPAAL [4] and KRONOS [5]. However, there are embedded systems in industry today which have real-time and performance requirements, but hardly can be analyzed using such methods. An example is the industrial robot control system IRC 5 developed by ABB Robotics which contains some 3 million lines of code and 50-100 tasks. Model checking tools do not come near to scaling to systems of this size. The existing analytical methods for schedulability or response-time analysis are not suitable for the IRC 5 system, due to the complex runtime behavior which does not comply with the system assumptions required to use these methods. In the IRC 5 system, tasks trigger each other using asynchronous messages and the contents of the messages often have a major impact on the temporal behavior of the receiver. The telecom domain poses similar even greater analysis challenges, for instance the radio base stations and radio network controllers developed by Ericsson. Such systems are event triggered, massively parallel, contain many millions lines of code, thousands of processes and also has several layer of fault tolerance which together gives very high system complexity. An alternative for analysis of complex embedded systems is to use simulation-based timing analysis [6]. This approach is more related to testing than formal verification as it is only possible to show the presence of potential errors, not their absence. However, simulation has the advantage of not posing restrictions regarding the design of the software system. Moreover, simulation allows for analysis of any measurable run-time property, in contrast to the analytical methods for schedulability and response time analysis [1], [2], [3] which have strict assumptions and focus on specific properties. Since a simulation model can be much more abstract than the modeled system, and since a PC is typically much faster than embedded hardware, many thousands of simulations can often be performed in short time where each simulation is given more or less random variations in execution times and other parameters. This way, many scenarios are explored which otherwise might be hard to generate on a real system. Note that the type of simulation in focus is not cycle-accurate low-level simulation like, e.g., Simics from Wind River, which are much slower than normal execution. In the perspective of analyzing an existing software system, simulation does not require manual modeling, since a simulation model typically is implemented using common programming languages and can be automatically extracted from the system implementation, either using a combination of program analysis and runtime measurements [6] or using runtime measurements [7] alone. This paper focuses on a specific problem in enabling automatic model generation: how to obtain execution time data for simulation models from recordings, without storing each data point individually. This is specifically in the context of the RTSSim simulation framework presented in Section II. The specific contribution of this paper is the proposed algorithm and a brief evaluation on real timing data, with diagrams visualizing the results. The algorithm has been implemented in an offline trace visualization tool, which feeds the algorithm one data point at a time, like when used in runtime monitoring. A proper implementation on an embedded system is planned in future work. As the session length is only limited by counter overflows, for 32-bit CPUs a session may cover at least $2^{32}-1$ (4.294.967.295) observations of each specific property, since

each interval may hold up to $2^{32} - 1$ data points. If assuming a data production rate of 100 Hz per measured property, this solution allows for at least 30 years of continous monitoring before wrapping occurs. This is in practice unlimited.

This solution is presented in Section III, and we present an implementation and evaluation of the method in Section IV. Section V puts this paper in perspective of related work, and Section VI concludes the paper and outlines future work.

II. RTSSim and Tracealyzer

RTSSim [6] is a simulation environment which emulates a generic real-time operating system on a PC. It works as a "sandbox" with respect to timing. All time-triggered events in RTSSim are driven by an integer simulation clock incremented by explicit *Execute* calls, using timing data recorded from the modeled system. The timing of the modeled system is thereby preserved in the simulation even though it runs on a PC instead of the embedded hardware. The simulated timing is however approximate, it is not guaranteed to be 100 % identical to the real timing when executing the code on the intended hardware, since the simulation model abstracts from the details of instruction-level timing. It instead describes the execution times between relevant program points, in a probabilistic manner. RTSSim focuses on the timing and resource usage of tasks, i.e., threads in a real-time operating system. An RTSSim task is a C function which is registered in the RTSSim scheduler together with attributes such as priority. An RTSSim task contains at least one *Execute* statement, which models CPU time usage. A simulation model can be specified at different abstraction levels. In one extreme is the most basic RTSSim task containing a single Execute statement only, i.e., only execution time is specified, no behavior. In the other extreme is to include the full source code of the analyzed software system, extended with Execute statements. Our intention of the RTSSim framework is however to use it together with a *model extraction* tool, which produces an abstraction of the implementation focusing on behavior of relevance for timing and resource usage. The tasks of an RTSSim simulation model are by default scheduled using fixed priority preemptive scheduling, and are assumed to share a single CPU core. In future work we intend to extend RTSSim to support parallel systems as well. RTSSim can produce two kinds of output, a detailed simulation trace focusing on the scheduling and other logged events, and a text file with selected statistics on task timing, such as highest response time observed for the selected task. The simulation trace is produced using a *trace recorder* described in Kraft et al. [8], which outputs a simulation trace for the *Tracealyzer* tool, including task scheduling, task communication and synchronization events. Tracealyzer is trace visualization tool for studying scheduling, resource usage and interaction of tasks and interrupts in embedded software. This tool originated in a research project back in 2004, performed in collaboration with ABB Robotics. They have used it since then in every industrial robot, and the Tracealyzer is now product in the author Johan Kraft's company Percepio AB[1]. Tracealyzer is today sold by Quadros Systems, Inc. as RTXCview and is the official trace tool for the real-time operating system RTXC Quadros. A description of the trace recorder used in RTXC Quadros is found in Kraft et al [8].

III. Statistical Runtime Modeling

Classic theoretical distributions are often hard to fit to timing data from software systems, since the distributions often are discontinuous and complex. Even linear code may result in discontinuities due to hardware effects such as cache misses, and if the measurements cover code with conditional jumps that is another cause of discontinuities in the resulting distribution. The most exact method of logging timing data from a program is to store every observation separately, i.e., an empirical distribution. However, over a long monitoring session this would consume vast amounts of RAM. Another approach is to only store watermarks (lowest and highest value) and use them as bound for a uniform distribution, but this way the shape of the distribution is lost which may result in incorrect simulations.

We suggest to use an established statistical approach known as *stratification*, where the data points are divided into intervals and where each such interval has a counter attribute, which is incremented at each matching data point. The individual observations are attributed to an interval (or used to create a new interval), and then forgotten. Each interval represents a uniform distribution, so the shape of the distribution within each interval still is lost, but given enough intervals this may give a sufficiently accurate statistical model, using a modest amount of memory.

The amount of RAM needed for a particular measurement is $3wi$, where i is the number of intervals used and w is the width of the variables, measured in bytes. In many cases $w = 2$ is sufficient, otherwise $w = 4$ allows for over four billion observations per interval. The constant (3) corresponds to the number of properties stored per interval, i.e., lower bound, upper bound and count. When using this simulation model for replay of timing data in a simulation, a two step sampling approach is used. At first the interval is selected, with the probability indicated in its count property, i.e., the number of observations in the interval divided by the total number of intervals. Once the interval has been selected, its bounds are used as a uniform distribution from which the final value is sampled.

A. Modeling Algorithm

The proposed algorithm takes as input a sequence of integer values, and produces as output a set of intervals, where each interval I has:

- $I.min$: the lower interval bound
- $I.max$: the upper interval bound, and
- $I.count$: the number of inputs x observed matching the interval, i.e., where $I.min \leq x \leq I.max$.

[1] http://www.percepio.se

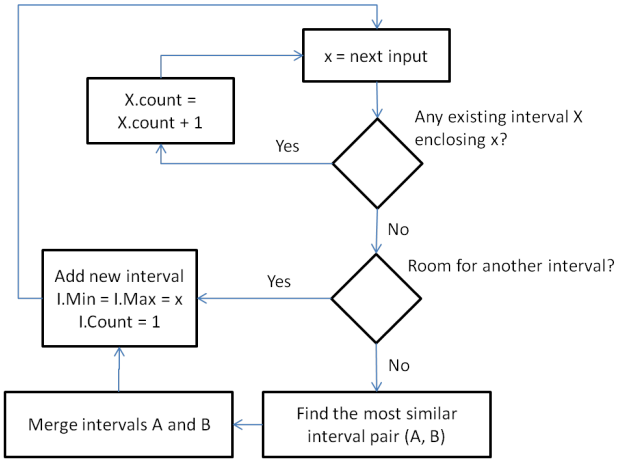


Fig. 1. The Modeling Algorithm

If there is no existing interval that match a specific input x and the maximum number of intervals has not yet been reached, a new interval I is created where

$$I.min = I.max = x$$

$$I.count = 1$$

If the maximum number of intervals has been reached, an analysis is performed in order to find and merge the two most similar adjacent intervals, in order to make room for a new interval according to above. The overall algorithm is illustrated in Figure 1. Initially there are no intervals and the first input thus result in the first interval created. Most early inputs will produce new intervals (until the maximum number of intervals have been reached), unless a specific input value is repeated. The merging of intervals will cause them to grow, which increases the probability of later inputs falling within the bounds of the interval. When merging two intervals A and B, they are replaced by a new interval C where

$$C.min = Min(A.min, B.min)$$

$$C.max = Max(A.max, B.max)$$

$$C.count = A.count + B.count$$

Only neighbor intervals are merged and intervals never overlap. The crucial aspect of this algorithm is how to select what intervals to join. We propose the following selection heuristics: Calculate a fitness value for each interval apart from the first, indicating its suitability for a merge with the lower neighbor interval. Then find the interval with best fitness value and merge it, i.e., with its lower neighbor. We suggest to base the fitness value on three properties:

- Proximity
- Density
- Count

Proximity is the absolute distance between the intervals of the candidate pair, i.e., the size of the gap in between the intervals. The smaller the gap, the more suitable merge. Density indicates the number of observations in relation to the width of the interval. A "spike", where many observations have been made in a narrow band, should not be merged with a "plateau" with lower density, i.e., where observations are more sparse. This is indicated by a density measure

$$I.density = I.count/(I.max - I.min)$$

The density fitness value of two merge candidate intervals A and B, a value between 0 and 1, indicates the similarity in density. The closer to 1, the better fitness for merge. The density fitness is calculated by

$$Min(A.density, B.density)/Max(A.density, B.density)$$

Finally, Count is the absolute number of observations accounted to the interval, i.e., $I.count$. If this is very low, the density becomes very sensitive to random variations in $I.count$. We therefore skip the density comparison if the count is below a certain threshold for either of the two intervals, we have used 5 as minimum in the prototype evaluation. Only if both intervals have at least 5 data points is the density property used, otherwise the fitness is based on proximity only. The density property is the dominant one when used, the proximity only matters in case the densities of two pairs are identical. To relax this a bit, we represent the density using an 8-bit integer in order to allow minor differences in actual density. We do not claim this to be the optimal merge selection heuristic, but the overall method proposed is however quite generic and can be used with different heuristics. In future work we plan evaluate the accuracy of this approach in greater depth and probably develop improved methods for selecting what intervals to merge.

B. Suggested Implementation

When implemented in an embedded system, we envision a solution where the intervals are stored in a linked list and kept sorted using insertion sort. The input is read from a circular RAM buffer which the measurements probes (code instrumentation) writes to. The modeling algorithm runs periodically on a low priority thread. This means that the modeling computations do not interfere with higher priority processes, but may cause loss of data if the system is under high load over a longer period of time. This might however be acceptable, since it is easaly detectable and can be compensated by increasing the rate of the modeling task, the size of the buffer, or both. The buffering will cause some additional RAM usage, but should typically only need to hold a few thousand observations. Consider a system with 100 tasks, each producing on average 100 observations per second. Assuming 4 bytes per observation, this would require about 40 KB for one second of execution. The buffer can however be eliminated by processing each data point when captured, i.e., the code instrumentation would update the statistical model directly, without buffering. This will cause CPU overhead at higher priority levels, thereby decreasing system performance, but might be an option for systems with severely limited amount

of RAM and where a few percent lower system performance can be tolerated.

IV. Evaluation

An implementation of the algorithm has been made within the Tracealyzer tool; a trace visualization tool developed by Johan Kraft in his company Percepio AB. Note that the proposed algorithm has not yet been implemented in the embedded software recorder, but has been added as a prototype feature in the Tracealyzer tool itself, i.e., in the offline analysis of the recorded trace. However, the prototype implementation works in the same way as intended when integrated in an embedded software recorder; it gradually constructs and updates the statistical model for each data point given as input.

In our prototype implementation, Tracealyzer has also been extended to visualize the resulting timing models (i.e., the intervals) together with a plot of the raw data. Figure 2 show four diagrams generated using the prototype implementation, on a specific task using different configurations of the algorithm; 2, 4, 6 or 8 intervals allowed. The horizontal axis represents the value domain and the vertical axis the relative frequency of the different values. The rectangles represent the resulting intervals of the generated timing model, i.e., the output of the modeling algorithm, while the thin vertical bars (some are very short) represent the individual data points used as input, i.e., the recorded execution times of the analyzed task. The height of the interval rectangles correspond to the relative number of matching data points, and the absolute number of matching inputs of the interval (the count property) is shown on the top of the rectangle.

The execution time data used in the evaluation comes from a demo trace for RTXCview provided by Quadros Systems, Inc, which has been recorded on a development board with a microcontroller using the ARM7 core. The four diagrams of Figure 2 illustrate how the interval merging is affected by the interval limit. The most significant gap is around 120 μs, when allowing only two intervals, only this gap survives the merging of intervals. When allowing four intervals, a second major gap is recognized at around 135 μs, which is divided in two parts by an interval of size 1. The reason this single data point is not merged with the large interval to the right, is since it was not forced to by the interval limit, as four intervals were allowed. When increasing to six and eight intervals, additional larger gaps are recognized as the merging process is less aggressive. Figure 3 show diagrams from two other tasks from the same trace file (TMR0ISR is actually an interrupt service routine). In this case, four intervals are allowed in both cases, a relatively aggressive setting but which represents the distributions fairly well. In future work we plan to evaluate the accuracy of this method by running simulations using the statistical model as base for generating execution times and compare the resulting distribution with the real distribution used as input for the modeling. Such a comparison can be made using established statistical methods, such as the two-sample Kolmogorov-Smirnoff test [9], or KS test for short. A good overview of this test can be found at the U.S. NIST website [10]. The KS test is non-parametric and makes no assumptions on the underlying distribution of the data, which is important for this type of data. As a basic verification of the correctness of the implemented algorithm, traces have been generated using RTSSim, with a known distribution, and given as input to the algorithm. The resulting diagrams are shown by Figure 4. The distribution of the input data in the first case, for the task TEST, is uniformly distributed in two intervals, 30% between 200–300 μs and 70% between 400–450 μs. If allowing for two intervals only, the generated intervals fits exactly. The right diagram of Figure 3 shows a similar experiment using a different reference distribution.

V. Related Work

Several methods have been proposed for stochastic representation of task execution time. Bernat et al. [11] use Execution Profiles (EPs) to represent execution times of sections of a task. The EPs can then be combined to Joint Execution Profiles (JEPs) which represents the execution time of a whole task. This work resulted in the founding of Rapita Systems, Ltd. [12] and their RapiTime tool for execution time profiling. They use a hardware recorder device, the RTBx, to log execution times, which uses a large hard drive to store very long traces of execution time data. This is however fairly large and expensive equipment which hardly can be deployed in the field.

Hansen et al. [13] presented an approach for probabilistic estimation of the worst-case execution time (WCET) of tasks using extreme value theory, based on earlier work by Edgar and Burns [14]. They divide the sample data into blocks, take the maximum of each block (an independent random variable) and use these maximas to derive a Gumbel distribution, which essentially have a skewed bell-shape. Such an approach could possibly be combined with the statistical model proposed in this paper; instead of modeling the intervals as uniform distributions, we could fit the data within each interval to a theoretical distribution such as the Gumbel distribution. If this is possible in a memory efficient manner during continuous runtime monitoring remains to be investigated in future work.

RTSSim is not claimed a novel contribution conceptually, several similar simulation frameworks have been proposed. The most similar ones are Virtual Time, from Rapita Systems, Ltd. [12], ARTISST [15] and DRTSS [16]. An earlier result is STRESS [17], which inspired our first simulator implementation ART-ML back in 2002.

Nolte et al. [18] outlined how execution-time profiles can be used for probabilistic timing analysis in general and in the context of component-based software engineering. The idea was to let components keep track of their own execution times and they highlighted the problem of how to generate execution time profiles, which is effectively solved by this paper.

Kraft et al. [19] (at the time named Andersson) made a case study of practical application of simulation-based timing analysis in collaboration with ABB Robotics, and studied in particular how to model execution time data measured in runtime. A concept of *instance equivalence classes*, or IECs,

Fig. 2. COMODRV execution times, using 2, 4, 6 and 8 intervals

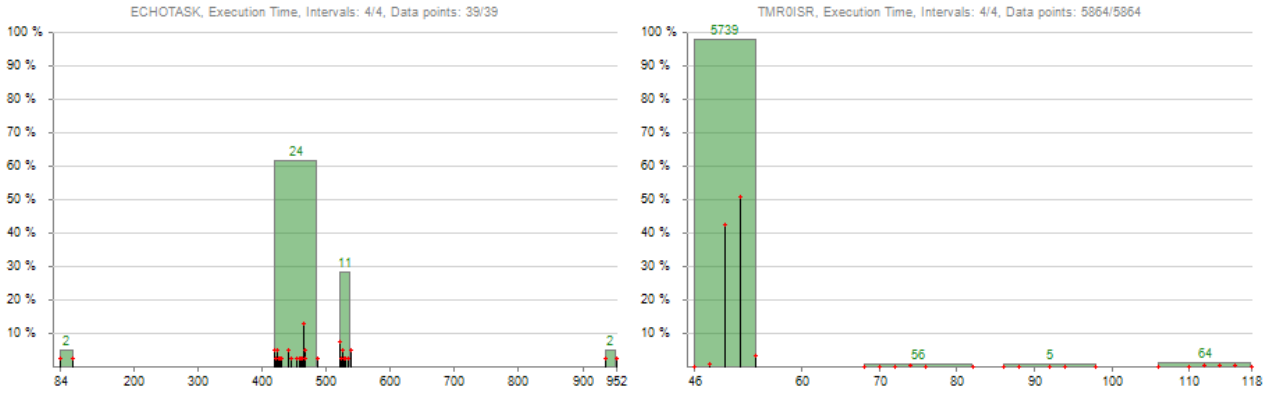


Fig. 3. ECHOTASK and TMR0ISR execution times, using 4 intervals

was introduced which is very similar to the statistical model proposed in this paper, and an algorithm was presented for offline identification of IECs. This was however not designed for use during continuous monitoring, but was intended for offline analysis of scheduling trace files.

VI. Conclusions

We have presented an approach for constant-memory monitoring of embedded systems, intended for continuous profiling of execution times and other runtime properties, which allows for profiling of systems in live operation over extended periods of time, possibly for years, using no hardware tracing equipment but only onboard software mechanisms. Due to the modest memory requirements, this approach is feasible even for microcontrollers with limited amounts of RAM. We have presented a prototype implementation of the algorithm and an evaluation which indicates that the method performs satisfactorily, although improvements in the selection heuristics most likely are possible. In future work we intend to implement,

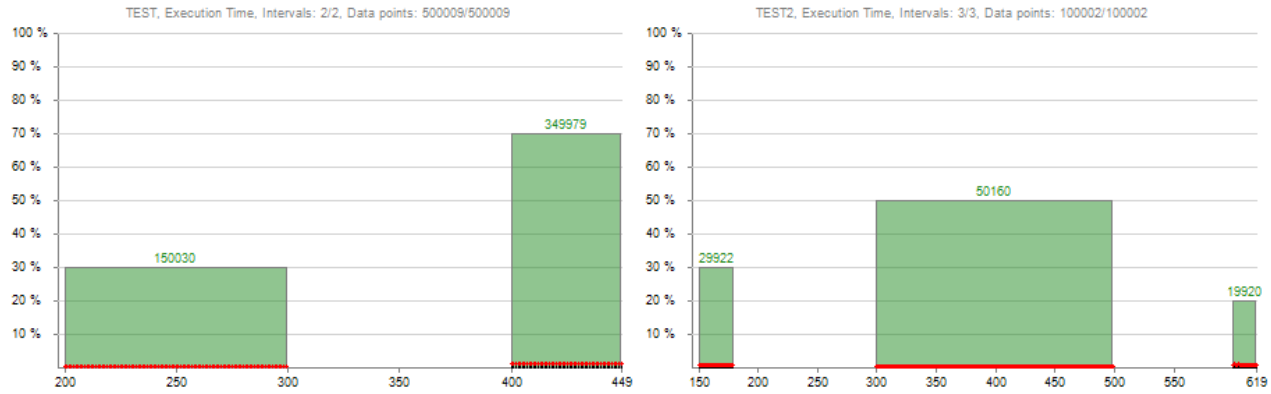


Fig. 4. Verification cases from RTSSim

evaluate and refine the modeling method using industrial cases; the ABB Robotics control system and the Ericsson telecom platform CPP.

ACKNOWLEDGMENT

This work is supported by the Artemis-JU project *CHESS*, performed in collaboration with Ericsson AB and ENEA AB, and the Swedish Foundation for Strategic Research.

REFERENCES

[1] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in hard-real-time environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, 1973.

[2] M. Joseph and P. K. Pandya, "Finding Response Times in a Real-Time System," *The Computer Journal*, vol. 29, no. 5, pp. 390–395, 1986.

[3] N. C. Audsley, A. Burns, R. I. Davis, K. W. Tindell, and A. J. Wellings, "Fixed priority pre-emptive scheduling: An historical perspective," *Real-Time Systems Journal*, vol. 8, no. 2/3, pp. 173–198, 1995.

[4] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi, "Uppaal – a tool suite for automatic verification of real-time systems," in *4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.

[5] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, "Kronos: A Model-Checking Tool for Real-Time Systems," in *10th Intl. Conf. on Computer Aided Verification*, vol. 1427, 1998, pp. 546–550.

[6] J. Kraft, "Enabling timing analysis of complex embedded software systems," Ph.D. dissertation, Mälardalen University Press, August 2010.

[7] J. G. Huselius, J. Andersson, H. Hansson, and S. Punnekkat, "Automatic generation and validation of models of legacy software," in *12th IEEE Intl. Conf. on Embedded and Real-Time Computing Systems and Applications (RTCSA'06)*, Aug. 2006, pp. 342–349.

[8] J. Kraft, A. Wall, and H. Kienle, "Trace recording for embedded systems: Lessons learned from five industrial projects," in *Proceedings of the First International Conference on Runtime Verification (RV 2010)*. Springer-Verlag (Lecture Notes in Computer Science), November 2010.

[9] I. M. Chakravarti, J. Roy, and R. G. Laha, *Handbook of methods of applied statistics*. Wiley, New York, 1967.

[10] "U.S. National Institute of Standards and Technology (NIST) e-Handbook of Statistical Methods, Section 1.3.5.16: Kolmogorov-Smirnov Goodness-of-Fit," http://www.itl.nist.gov/div898/handbook.

[11] G. Bernat, A. Colin, and S. M. Petters, "WCET analysis of probabilistic hard real-time systems," in *23rd IEEE Real-Time Systems Symposium (RTSS'02)*, Dec. 2002, pp. 289–300.

[12] "Rapita Systems, Ltd." http://www.rapitasystems.com.

[13] J. Hansen, S. Hissam, and G. Moreno, "Statistical-based WCET estimation and validation," in *9th Intl. Workshop on Worst-Case Execution Time Analysis (WCET'09)*, Jun. 2009, pp. 123–133.

[14] S. Edgar and A. Burns, "Statistical analysis of wcet for scheduling," in *Proceedings of the 22nd IEEE Real-Time Systems Symposium*, ser. RTSS '01. Washington, DC, USA: IEEE Computer Society, 2001, pp. 215–. [Online]. Available: http://portal.acm.org/citation.cfm?id=882482.883802

[15] D. Decotigny and I. Puaut, "Artisst: An extensible and modular simulation tool for real-time systems," in *Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Washington D.C.*, 2001.

[16] M. Storch and J.-S. Liu, "DRTSS: A Simulation Framework for Complex Real-Time Systems," in *Proceedings of the 2nd IEEE Real-Time Technology and Applications Symposium (RTAS'96)*. Dept. of Computer Science, Illinois Univ., Urbana, IL, USA, 1996.

[17] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "STRESS: A Simulator for Hard Real-Time Systems," *Software-Practice and Experience*, vol. 24, no. 6, pp. 543–564, June 1994.

[18] T. Nolte, A. Möller, and M. Nolin, "Using components to facilitate stochastic schedulability analysis," in *WIP at 24th IEEE Real-Time Systems Symposium (RTSS'03)*, Dec. 2003, pp. 7–10.

[19] A. Wall, J. Andersson, J. Neander, C. Norström, and M. Lembke, "Introducing Temporal Analyzability Late in the Lifecycle of Complex Real-Time Systems," in *9th Intl. Conf. on Real-Time and Embedded Computing Systems and Applications (RTCSA'03)*, 2003.