

PARAD Repository: On the Capitalization of the Performance Analysis Process for AADL Designs

Thanh Dat Nguyen, Yassine Ouhammou^(✉), and Emmanuel Grolleau

LIAS Laboratory, ISAE-ENSMA and University of Poitiers, Futuroscope, France
{[@ensma.fr](mailto:thanh-dat.nguyen,yassine.ouhammou,grolleau)},
<https://www.lias-lab.fr>

Abstract. In this paper, we focus on RTES (real-time embedded systems) designs expressed in AADL (Architecture and Analysis Design Language) and we propose a model-based approach to improve the way that designers check and analyze the performance of their system designs by capitalizing the analysis process. Our approach is based on proposing customized repositories of models using formal AADL-compliant query and constraint languages in order to orient designers to choose the most suitable analysis models and tests. Furthermore, this work is also dedicated to research teams to share their researches and prototypes, in order to enhance the (re-)usability of the real-time performance analysis tests.

1 Introduction

Critical real-time embedded systems (RTES) are used in many domains (such as avionics, nuclear and automotive) where the development life-cycle takes months up to several years. Hence, RTES designs need to be analyzed at an early phase of the life-cycle in order to check if all the requirements are met, including temporal requirements (e.g. deadlines, end-to-end delays, etc.). For that, numerous analysis tests have been proposed, based on the scheduling theory, dedicated to different system behaviours and architectures. However, one of the main difficulties that system designers face is to choose the suitable analysis test helping to validate and/or to dimension their designs properly. Also, improving reuse in complex systems like real-time systems is increasingly recognized as primordial as it contributes to paring down costs for the engineering and shortening the development time. In order to analyze the temporal behavior of a critical RTES and to ensure its correctness, a set of quantitative performance tests (e.g., schedulability tests) has to be applied to analysis models by using algebraic methods.

1.1 Context

RTES are composed of a set of interacting tasks sharing communication resources, generating several messages, executing on a set of hardware components and are arbitrated by scheduling algorithms and network protocols.

To study the temporal correctness of a system, a schedulability test takes into account all of these elements. Architectures of RTES have been sharply impacted by the technology evolution in terms of hardware and software components (mutli-core processors, cache memories, avionics networks, hierarchical processes, mixed criticality, etc.) [11, 22, 25, 28]. The performance analysis community follows actively this evolution by proposing numerous tests that match the variety of RTES architectures due to the critical aspect of RTES. The result of a performance analysis test can never be reliable unless the system’s architecture fits accurately with the analysis model of the applied test. That is, choosing a non-suitable test may conduct to a wrong optimistic result (e.g. the calculated response-time of a task can be less than the time required for its execution in practical cases), or it can lead to a very pessimistic result generating a system over-sizing (usage of non-required processors and wires).

Once upon a time, the design (modeling and performance analysis) of RTES was several come-and-go flows between designers and analysts. Nowadays, Model-Driven Engineering (MDE) [24] gains in terms of popularity and becomes used during the design process of RTES. Thanks to MDE settings (modeling, transformation, code generation) the design of RTES becomes a model-based process tool-chain integrating both modeling and performance analysis phases and shortening the time-to-market. Indeed, a set of design languages have been proposed like UML-MARTE [16] and AADL [4] to help designers to get a pivot centric-model that can support several kinds of functional and non-functional analyses (energy, time, safety, etc.). The timing performance analysis community has also taken benefits from the MDE facilities. Hence, numerous implementations of analysis tests and models have been produced as commercial and academic analysis tools (such as Rt-Druid [23], MAST [17]). Each tool supports one or several kinds of RTES architectures, and provides a set of analysis tests in order to help designers to conclude about the schedulability of the system under design.

1.2 Problem Statement

Nowadays, we witness a gap between the real-time scheduling research theory and its utilization in the industry [20]. Performing analysis tests, as it is currently used, is still driven by the real-time designers experience. Therefore, determining what type of analysis tests to use for a given architecture may be difficult. This is due to several reasons. (i) By analyzing the literature¹, we realized the presence of a “jungle” of analysis tests whose analysis models are sometimes well defined in the scientific papers, but sometimes they are drowned in paper discussions, or even left implicit. For example, the survey presented in [28] enumerates a dozen of analysis models, which are only dedicated to simple systems with independent tasks, uni-processor architecture and fixed priority scheduling algorithms. Therefore, to know if a performance analysis test of a specific system

¹ We have analyzed papers published in the main real-time system conferences like: RTSS (rtss.org), ECRTS (ecrts.eit.uni-kl.de) and RTAS (rtas.org).

architecture exists is time-consuming. (ii) Moreover, the existing analysis tools are more dedicated to automatize the test calculation than orienting designers to the right tests that match their needs, and the analysis models of the computed tests are often explicit and need a deep knowledge of both modeling and scheduling theory, which is uncommon.

In the light of these mentioned problems, the current situation is penalizing in terms of reusing and finding easily appropriate analysis tests. It also deepens the gap between the tests presented by the researches community and their utilization in the industry.

1.3 Paper Contribution

In this research, we are interested on RTES designs based on AADL standard design language. We propose a collaborative framework called PARAD (Performance Analysis Repository for AADL Designs)². This framework is dedicated to construct performance analysis repositories playing the role of “decision supports”. Thanks to our proposition, designers can be helped during the analysis phase in order (i) to detect the analysis situation corresponding to the system design and (ii) to choose the most suitable analysis tests. Our proposition also aims at enhancing the applicability of the real-time performance analysis theory. Indeed, this work can be used as showcase and teaching-aid allowing the research teams to show their results (e.g. analysis models, tests, prototypes) and to share them with other teams among the RTES community.

1.4 Paper Outline

The remainder of the paper is organized as follows. Section 2 introduces real-time advances in terms of tests, tools and design languages, and presents a running example which will be used to motivate our contribution and to highlight its relevance. Section 3 is devoted to present formally the concepts of PARAD. Section 4 shows the proof of concept through a case study. Finally, Sect. 5 summarizes and concludes this article.

2 Background and Work Positioning

The development life-cycle of RTES can span over years. Indeed, any wrong choice during the design phase can impact sharply the time-to-market. Therefore, the analysis should be carried out at an early-stage of the design phase. In practice, a RTES design has to satisfy many constraints, including the temporal ones. That means, threads have to respect their deadlines, the execution order, end-to-end delays, etc. To ensure the satisfaction of these constraints in a system design, we use temporal performance tests.

² <https://forge.lias-lab.fr/projects/parad>.

2.1 Real-Time Concerns

The RTES analysis research community is very active and has been offering various analysis tests since the 1970's (like worst-case response time analysis, time demand analysis, simulation, etc.) [8, 25, 28]. The provided analyses are not only schedulability tests (which allow to conclude if the system is schedulable or not) but they also address other aspects such as the dimensioning, the sensitivity analysis and the quality of service, etc. The inputs of each test represent an analysis model (a.k.a. workload model) which is an abstract representation of the system being under design. Moreover, each test is characterized as a sufficient test, a necessary test, or both of them (i.e., exact test). A test is defined to be *sufficient* if all of the task-sets that are deemed schedulable according to the test are in fact schedulable. A test can also be referred to as *necessary* if failure of the test will indeed lead to a deadline miss at some points during the execution of the system. Schedulability test that is both *sufficient* and *necessary* is labeled as *exact*, then it is in some sense optimal. The result provided by each test can help to conclude about the temporal satisfaction of the RTES.

Actually, during the progress from the modeling phase to the analysis phase, designers should choose between two main pathways to analyze their designs. (i) The first one is related to the use of analysis tools that are already integrated to a model-based tool chain. In other words, a “push-button” action can be enough to analyze the design. However, the designer cannot know if the chosen test perfectly matches the design characteristics. That is, since a “push-button” action is enough to analyze the design, the analysis result may be optimistic (which means wrong because of the criticality of RTES) or oversized (thus, it can be costly in terms of equipments and wiring). Moreover, in case of analysis failures, designers may not be informed about the failure reasons. Hence, designers could not know if the problem is due to a wrong choice of the analysis techniques or due to the analysis tool itself which may not support the architecture and the specificities of the system under design. (ii) The second pathway is to ask an expert. This pathway is safer but also more expensive. During the development process, an expert has to repetitively examine architectures provided by designers till finding out the configuration that meets the temporal requirements. Unfortunately, this work-flow is not capitalized, hence the idea behind this paper.

2.2 Motivating and Running Example

In this section, we present a toy example for initiating the discussion. We consider a software architecture of a RTES consisting of four periodic independent tasks. Each task is characterized by a set of temporal properties (See Table 1): the worst-case execution time (WCET), the deadline, the period, the release-time and the priority.

The four tasks are preemptive and scheduled referring to their priorities. They are executed on a uni-processor hardware architecture. Task1's priority is higher than Task2's priority, which is higher than Task3's priority, which is higher than Task4's priority. After launching the analysis process through two

Table 1. Example of the task-set configuration of a RTES

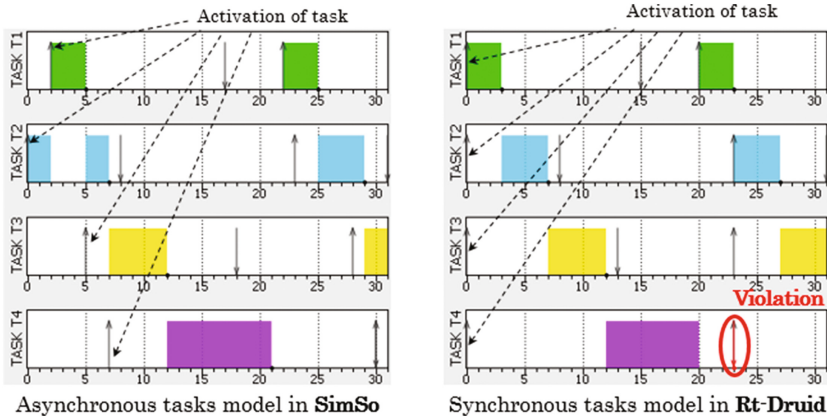
Task	WCET	Deadline	Period	Release time	Priority
Task1	3 ms	15 ms	20 ms	2 ms	4
Task2	4 ms	8 ms	23 ms	0 ms	3
Task3	5 ms	13 ms	23 ms	5 ms	2
Task4	9 ms	23 ms	23 ms	7 ms	1

Table 2. Worst-case response times (WCRT) provided by the tools

Task	WCRT (SimSo)	WCRT (Rt-Druid)
Task1	3 ms	3 ms
Task2	7 ms	7 ms
Task3	8 ms	12 ms
Task4	14 ms	33 ms

different analysis tools (Rt-Druid [23] and Simso [27]), we have obtained two different results for the same input architecture as shown in Table 2.

Discussion: The difference is not related to a wrong implementation of the analysis methods, but to the choice of the analysis models. The result in the second column of Table 2 is carried out as a simulation by Simso [27] so that the tasks release-times are taken into consideration. Whereas, the analysis model chosen via Rt-Druid (whose result is presented in the third column) ignores the release-times and considers that all tasks are released at the same time [14]. Figure 1 presents differences of behaviors related to the analysis models considered by Simso and RT-Druid. Indeed, the result provided by Rt-Druid are not wrong,

**Fig. 1.** Analysis models of SimSo and RT-Druid

but the analysis model used via the Rt-Druid tool represents the worst-case behavior that can never be produced by the system under-analysis. Consequently, the test that corresponds to this analysis model leads to pessimistic results that are not close to the practical case of the system under analysis.

2.3 AADL in a Nutshell

Although the approach that we propose in this paper can be generalized to be used with any prescriptive design language, in this paper, we focus on AADL designs. Then, we propose a brief presentation of the language to have a self-content paper. AADL (**A**rchitecture **A**nalysis and **D**esign **L**anguage) [4] is a domain specific language dedicated to design software and hardware architectures of real-time embedded systems. AADL provides components helping to define hardware (such as processors and buses) and software concerns (such as threads and data). Interactions between components are expressed through their interfaces (i.e., ports, bus access, etc.). In addition, AADL provides a set of properties which can be easily extended. These properties make AADL designs become pivot model-centric since they enable to apply different analyses and settings (e.g., formal methods, schedulability analysis, energy consumption).

Modeling the example with AADL. Figure 2 represents the AADL design that corresponds to the running example (presented previously in Sect. 2.2). To ease

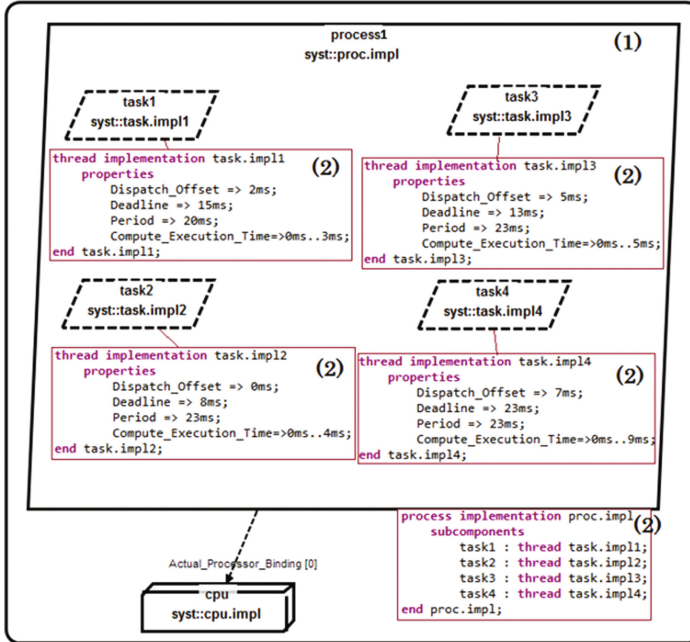


Fig. 2. Running example expressed in AADL language

the understanding and to give sufficient details we have mixed graphical and textual syntaxes. While Parts (1) give an overview of the architecture thanks to the graphical syntax, Parts (2) give more temporal details thanks to the AADL textual syntax.

2.4 Related Work

Recently, several works have coped with the problem of helping designers to analyze the temporal performance of their AADL architectures. Peres et al. [21] have proposed the usage of techniques based on model checking. This approach requires the transformation of AADL designs to another formalism (e.g., petri-nets or timed automata). Moreover, the utilization of this technique seems to be complex since it suffers from the problem of the combination explosion. Gaudel et al. [12] implemented through Cheddar tool [2] an extraction of information from AADL models. That requires a set of ad-hoc information as design patterns that must be mentioned to support the analysis of AADL models and which is recognized only by the Cheddar tool. In addition, the recognition of the design patterns is based only on the architectural model, so it does not consider the behavior of the modeled system. Ouhammou et al. [19] suggested an example of model transformation from AADL to *MoSaRT Language* dedicated to the schedulability analysis. However, this approach requires to be familiar with MoSaRT language. Moreover, the transformation is not always equivalent. There are some concepts in one language which cannot be described in others languages or there are some concepts which can be transformed to different concepts in other languages depending on the context of the transformation. Brau et al. [9] proposed to construct an analysis repository to deduce all feasible pathways (a sequence of successive analyses) for achieving a predefined goal. However, authors do not focus on how pertinent and accurate the application of an analysis to a system, while it is an important factor.

Generally, we can classify all these works into two categories. Those which use the design patterns (piece of design-solution for conventional problem in design) and those that are based on the model transformation. However both categories contains hard-coded solutions. Also the information that helps designers to understand and to justify the choice of a test instead of others still implicit.

In this article, we adapt the solution proposed in [19] to support the systems designed in AADL. We have implemented this approach into a framework named **Performance Analysis Repository for AADL Designs** (PARAD). We propose two usages of our framework. The first usage consists in helping the AADL designers to choose appropriate tests for their systems. The choice is accompanied by sufficient explanation about the system under design and how it matches characteristics of the proposed test (if it exists). The second usage consists in helping analysts and researchers to share their knowledge by proposing repositories with relevant information making the repository users (especially designers) sufficiently autonomous during the analysis phase.

3 PARAD Approach and Its Fundamentals

This section is devoted to present our model-based contribution. The PARAD framework is mainly based on a description language allowing to instantiate repositories with contents based on analysis models, analysis tests and their characteristics. First, we present an overview of PARAD framework approach by highlighting briefly different capabilities. Secondly, we present the relevant foundation elements of PARAD and their roles. Our contribution is based on the facilities of model-driven engineering settings.

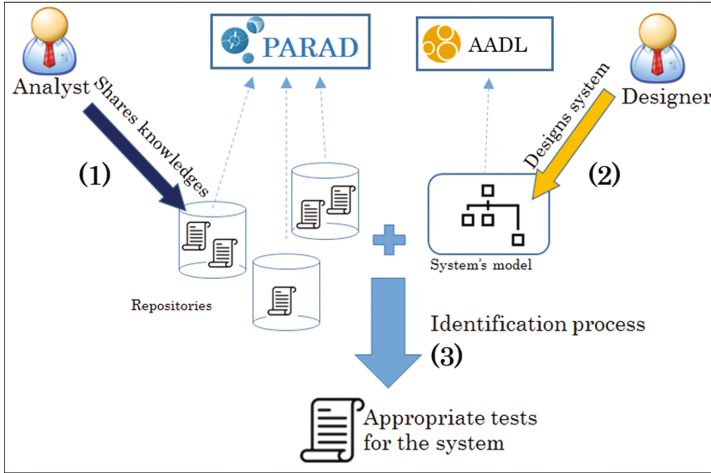


Fig. 3. Overview of Parad's utilization

3.1 PARAD Overview

Figure 3 shows an overview of the PARAD framework. PARAD offers two mechanisms. (i) The first one consists in creating analysis repositories, it is dedicated to analysts and researchers (process (1) in Fig. 3). An analysis repository can be made by one or many analysts. It represents the knowledge and expertise that analysts would like to share with other collaborators. (ii) The second mechanism is the identification process, dedicated to designers (process (3) in Fig. 3). Indeed, designers can apply their systems architectures, expressed in AADL (process (2) in Fig. 3), to a chosen analysis repository (provided by an expert analyst) in order to be assisted during the analysis phase. In the sequel, we will detail those two mechanisms.

3.2 Core Concepts of PARAD

The approach behind PARAD is based on a set of concepts related to the performance analysis theory. Figure 4 shows the principal excerpt of PARAD's meta-model. In the following we present the relevant concepts of PARAD.

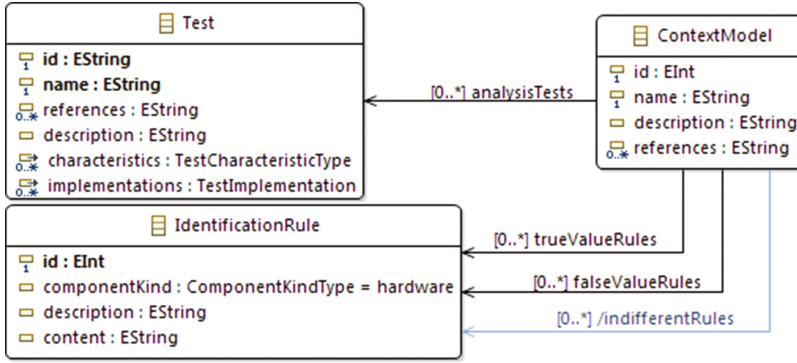
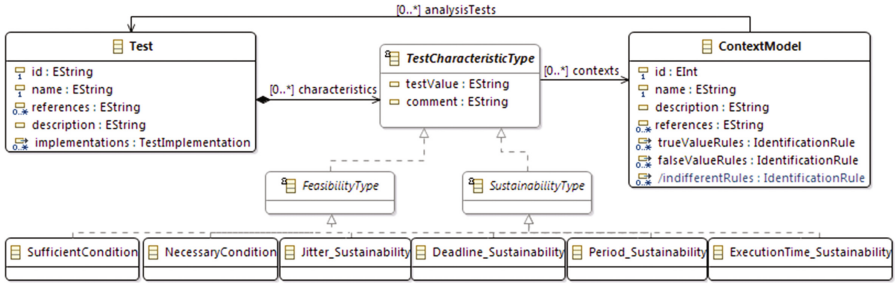


Fig. 4. Excerpt of PARAD metamodel

Fig. 5. Excerpt of PARAD metamodel: *Test*, *Context* and their relationship

- The *IdentificationRule* is the cornerstone notion of PARAD. An *IdentificationRule* represents an assumption on the system (e.g., ‘System has only periodic tasks’, ‘System has mono-processor architecture’, etc.). The value of each *IdentificationRule* is not always true or false but depends on the studied system. Due to evaluated values of these *IdentificationRules* on a system, we can determine the analysis situation of this system.
- *ContextModel* (or *Context* in short) represents the analysis situation of system, that let us know about the analysis model simulating the system and what analyses can be applied. In fact, a *ContextModel* is a set of hypotheses on the system. For example, Liu&Layland context [15] is based on these following hypotheses: ‘System has mono-processor architecture’, ‘All tasks are independent’, ‘All tasks are characterised by worst-case execution time and period’, etc. The hypotheses that constitute *ContextModel* are modelled by *IdentificationRule* with an expected value on the system: *trueValueRule*, *falseValueRule* and *indifferentValueRule*. *trueValueRule* represents the *IdentificationRule* that the system have to satisfy, *falseValueRule* represents the *IdentificationRule* that the system must not satisfy. It sometimes happens that a *ContextModel* can match a system whether the system satisfies an

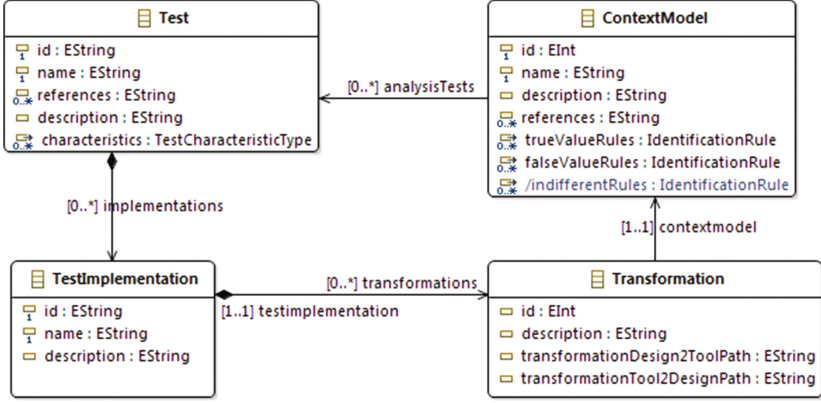


Fig. 6. Excerpt of PARAD metamodel: *TestImplementation* and *Transformation*

IdentificationRule or not, for example: Liu&Layland context [15] does not care whether system’s tasks are concrete or not. We qualify this type of *IdentificationRule* as *indifferentValueRule*.

- *Test* represents the analysis technique which allows to conclude about the schedulability of systems. Since *ContextModel* represents the analysis situation of systems, in it, a number of *Tests* is applicable. To a specific *ContextModel*, a *Test* may be exact or sufficient or necessary (as we have already explained in Sect. 2.1). Tests are also characterised by the sustainability aspect [6]. In fact, analysis tests often consider the worst-case values (e.g. worst-case execution time, minimal period, maximal release jitter). During the execution of the system, the task parameters are always better than those considered in analysis, so the validity of analysis should retain. It is an important quality of an analysis. An analysis test with respect to a system is called sustainable, if the system, deemed valid by the test, remains valid even after changing tasks parameters: (Fig. 5) decreasing execution time (ExecutionTime.Sustainability), increasing period or inter-arrival times (Period.Sustainability), decreasing jitter (Jitter.Sustainability) and increasing relative deadline (Deadline.Sustainability).
- *TestImplementation* (Fig. 6) represents implementation of an analysis technique in a tool allowing to perform this test. *Transformation* represents a technique of assimilating concepts between the studied model and the input formalism of the associated tool. Each instance of *Transformation* enables to define the location of a program (executable files, call of web services,...) which can be used to automatize the transformation process.

Since we aim at examining AADL designs to detect the corresponding analysis contexts by evaluating *IdentificationRules*, they have to be based on a formalism. Therefore, we have opted for OCL and LUTE as languages to express these rules. Users can choose the language that they are more familiar with.

Properties

Id: 2

Component Kind: ☐ hardware ☒ software

Description: All tasks are concrete

Content: ComponentInstance.allInstances()
 ->select(t : ComponentInstance|t.category.toString() = 'thread')
 ->forAll(t:ComponentInstance|t.ownedPropertyAssociation.property.name
 ->includes('Dispatch_Offset'));

Fig. 7. Identification rule: “All tasks must have the Offset property” in OCL’s syntax

- OCL (**O**bject **C**onstraint **L**anguage [18]) is Ecore-compliant. While the AADL metamodel conforms to Ecore [1], the content of *IdentificationRules* can be written in OCL’s syntax. Thanks to OCL’s checker, these rules can be evaluated. Figure 7 presents an identification rule expressed in OCL.
- LUTE is a constraint language that allows to query AADL models and therefore helps designers to check model structures and system requirements. It is composed of different functions to query the components hierarchy as well as their features (ports, connections, etc.) and properties [3]. The following listing shows an example of LUTE query which represents the same identification rule as the one of Fig. 7.

```

1      theorem Offset_Defined
2          foreach t in Thread_Set do
3              check Property_Exists(t, "Dispatch_Offset");
4      end;
```

Listing 1.1. An identification rule expressed in LUTE’s syntax

3.3 PARAD Identification Process

The *Identification Process* is the way PARAD repositories seek the context of a system. It is composed of 2 steps: The first one is to evaluate all Identification rules (of the chosen repository) on the system and the second one compares results obtained in step 1 with the expected values of contexts of the repository to find out the appropriate contexts. Figure 8 presents an example sketching the identification process. We notice that the context model that corresponds to the system is the one which is defined by the same way as the evaluation result (obtained in step 1). In other words, the corresponding context is the one whose *trueValueRules* must be all evaluated as true on the system and whose *falseValueRules* must be evaluated as false on the system. Note that, a system can match more than one context. Moreover, the identification process can be called several times throughout an iterative design process.

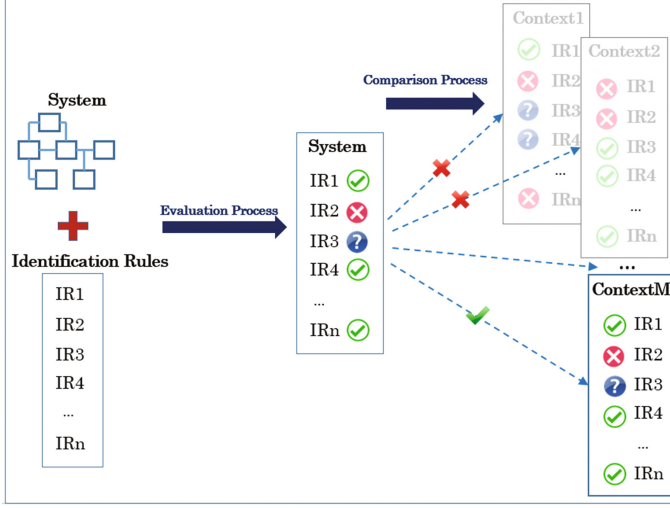


Fig. 8. Steps of Identification process

4 Proof of Concept

We introduce a simple case study to highlight the utilization of PARAD for choosing appropriate analyses for RTES and to demonstrate the possibility of combining several tests. We prepared a sample analysis repository, which is fulfilled with a list of conventional analysis tests. Thanks to the analysis repository, we can find the appropriate tests for a given system. Also, the results produced by an analysis test for a system make possible to apply other tests for the system. That enables to combine a number of test into a an incremental analysis-chain. The detail is presented hereafter.

4.1 PARAD Repository

We propose an *Analysis Repository* - instance of PARAD with the following schedulability analysis tests: *Response Time Analysis* - RTA [14], *Holistic Analysis* [29], *Request Bound Function* - RBF [7, 13], *Rate Monotonic schedulability test* [26], *Audsley Priorities Assignment* - OPA [5]. These tests are already implemented in many academic and industrial third-party tools.

We also defined 4 contexts for tests: (Ctx1) Liu&Layland for tasks with pre-assigned priority - constrained deadline - fixed priority scheduler, (Ctx2) Periodic tasks model with arbitrary deadline in distributed system, (Ctx3) Liu&Layland for tasks with constrained deadline - EDF (Earliest Deadline First) scheduler, (Ctx4) Liu&Layland for tasks with offset - arbitrary deadline - fixed priority scheduler. The *Response Time Analysis* is sufficient, sustainable in Ctx1. Although the *Holistic Analysis* is a generic test, it can be applied in both Ctx1 and Ctx2. The application of *Holistic Analysis* in Ctx1 is identical to *Response*

Time Analysis, so we only added it to Ctx2 to deal with distributed system architecture. *Holistic Analysis* is sufficient and sustainable in Ctx2. The *Request Bound Function* is exact and non-sustainable in Ctx3. *Rate Monotonic schedulability test* is sufficient test in Ctx1 and Ctx4. And last *Audsley Priorities Assignment* is exact, non-sustainable in both Ctx1 and Ctx4. We introduce several *IdentificationRules* to identify these contexts. The content of each rule and its expected evaluation result in each context are presented in Table 3. To simplify the comprehension, we only choose 20 relevant rules for each context.

Table 3. Description of *ContextModel* in term of *IdentificationRule*

Contexts	Ctx1	Ctx2	Ctx3	Ctx4
IR1: All tasks must have predefined offset	U	U	U	✓
IR2: All tasks must have predefined execution time	✓	✓	✓	✓
IR3: All tasks must have predefined deadline	✓	✓	✓	✓
IR4: All tasks must have predefined period	✓	✓	✓	✓
IR5: All tasks must have predefined priority	✓	✓	U	U
IR6: All tasks must have predefined scheduling protocol	✓	✓	✓	✓
IR7: Mono-processor architecture	✓	✗	✓	✓
IR8: Earliest deadline first policy	✗	✗	✓	✗
IR9: Deadline monotonic policy	U	✓	✗	✗
IR10: Rate monotonic policy	U	✓	✗	✗
IR11: No predefined scheduling policy	U	U	U	U
IR12: All tasks must be concrete	U	U	U	✓
IR13: Tasks must be non-concrete	U	U	U	✗
IR14: All tasks must be periodic	✓	✓	✓	✓
IR15: All tasks must be synchronously activated	U	U	U	U
IR16: All tasks must be asynchronously activated	U	U	U	U
IR17: All tasks must be independent	✓	U	✓	✓
IR18: Deadline of all tasks must be implicit	U	U	U	U
IR19: Deadline of all tasks must be constrained	✓	U	✓	U
IR20: Deadline of all tasks must be arbitrary	✗	✓	✗	U

– ✓: rule whose evaluated value should be true in the context.

– ✗: rule whose evaluated value should be false in the context.

– U: regarding to the context, evaluation value of this rule is not important so it can be true or false.

4.2 Analysis Process Using PARAD as a Decision Support

System's description: we consider the same system presented in the motivating example (Sect. 2.2), but without any preassigned priority.

Context Characteristics	Context Models	Analysis Tests
Context 4		
Rules which should be true	- 1 - 2 - 3 - 4 - 6 - 7 - 12 - 14 - 17 - 20	
Rules which should be false	- 10 - 13 - 8 - 9	
Rules which do not have any impact	- 5 - 11 - 15 - 16 - 18 - 19	
Description	Liu&Layland for tasks with offset, arbitrary deadline and fixed priority scheduler	
References	N.C.Audsley "Optimal priority assignment and feasibility of static priority tasks with arbitrary start time	
Corresponding Tests		
Test's id:	3	
Test's name:	Audsley's Priorities Assignment (OPA_1)	
Test's characteristic:	Sufficient Necessary	
Description	Priority Assignment Algorithm of Audsley based on simulation, this algorithm is at the same time a scheduling polic *****	
Test's id:	5	
Test's name:	Rate Monotonic analysis (RM1)	
Test's characteristic:	Sufficient	
Description	A sufficient test based on comparison between processor utilisation factor and bound value *****	

Fig. 9. Appropriate found contexts

We applied the constructed PARAD repository to the system expressed in AADL to find out the appropriate tests. The result is presented in Fig. 9. We notice that two contexts match the studied system, hence we have two tests are available at this stage: Audsley's Priorities Assignment (OPA) and the Rate Monotonic schedulability test (RM1). RM1 is sufficient in this context and OPA is exact in this context, so we choose OPA for the first analysis. The result of OPA is not only schedulability of system but also priority of tasks under which the system is schedulable (if the test is succeed). Part (1) in Fig. 11 sketches the obtained result of OPA. The system is schedulable by the calculated task's priority configuration. We assign the calculated priorities to tasks in the system and retry detection process. Due to priorities assigned, Response Time Analysis (RTA1) is also available (Fig. 10). Once again we apply RTA1 and get the result displayed in part (2) of Fig. 11. The provided result shows the worst-case response time of each task of the studied system. All tasks respect their deadlines except the last one. Note that RTA is a sufficient test in this context so the system remains schedulable under the priorities calculated by OPA.

4.3 Learned Lessons and Discussion

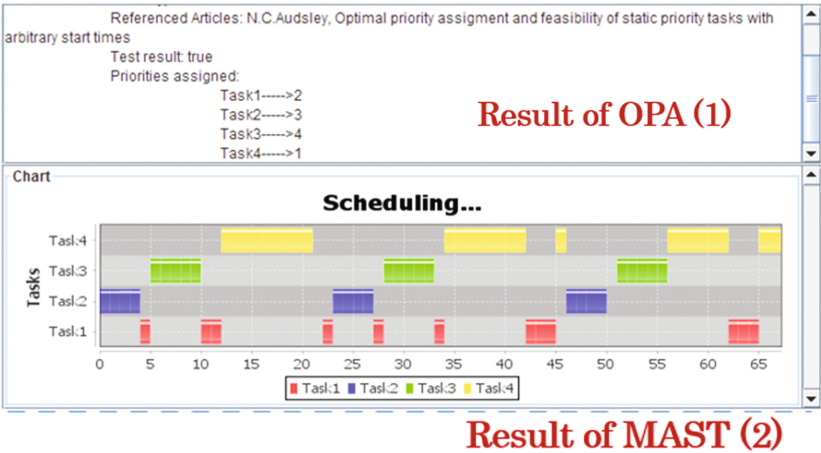
The case study has stressed usefulness of our approach to find appropriate analyses for a system model. Our approach has been used in two industrial projects with avionics partners [10, 30]. Therefore, we have realized some points to improve:

Context Characteristics	Context Models	Analysis Tests
Context 1		
Rules which should be true	- 2 - 3 - 4 - 5 - 6 - 7 - 14 - 17 - 19	
Rules which should be false	- 20 - 8	
Rules which do not have any impact	- 1 - 11 - 12 - 13 - 18 - 9 - 10	
Description	Liu&Layland for tasks with pre-assigned priority, constrained deadline, fixed priority	
References	K. Tindell, "An Extendible Approache for Analysing Fixed Priority Hard Real-Time Ta	
Corresponding Tests		
Test's id:	3	
Test's name:	Audsley's Priorities Assignment (OPA_1)	
Test's characteristic:	Sufficient	
Test's characteristic:	Necessary	
Description	Priority Assignment Algorithm of Audsley based on simulation, this algorithm is at the same time.	

Test's id:	4	
Test's name:	Response Time Analysis (RTA_1)	
Test's characteristic:	Sufficient	
Description	Calculate the response time of independent tasks deadline constrained with priority pre-assigned	

Test's id:	5	
Test's name:	Rate Monotonic analysis (RM1)	

Fig. 10. Detection result



Transaction	Event	Referenced Event	Worst Response	Hard Deadline
task1	deadline_monitor_task1	activator_task1	7.000	15.000
task2	deadline_monitor_task2	activator_task2	4.000	8.000
task3	deadline_monitor_task3	activator_task3	12.000	13.000
task4	deadline_monitor_task4	activator_task4	33.000	23.000

Fig. 11. Result of case study

1. The more *IdentificationRules* are defined, the more accurate is the context detection. Then, in practice, we provide a dozen of *IdentificationRules* to constitute analysis repositories.
2. Often, *IdentificationRules* are not independent. Their semantics are implicitly related. For example, the rule ‘All tasks must be concrete’ implies that ‘All tasks must have Offset property’. So, we plan to explicit and model the relations between *IdentificationRules*.
3. In case of an existing repository with a huge number of *IdentificationRules*, the extension can be difficult since it can lead to some duplicates or semantic conflicts. We plan a verification process of the repository content by using formal methods.
4. Basically, all rules are inspired from the scheduling theory and are expressed in natural language. We interpret them using query languages (like OCL or LUTE), but the interpretation process is not easy and not reversible, we cannot retrieve easily the rule semantic from its interpretation in OCL or LUTE. Therefore, we plan to create a domain specific language which is close to natural language (humain-readable) and understandable by the machine to express the *IdentificationRules*.

5 Conclusion

The main difficulty that faces a RTES designer is the lack of an advisor during the performance analysis phase. The PARAD framework represents a solution for this problem by defining a way allowing a collaboration between analysts and designers thanks to the analysis repositories. Moreover, PARAD can be used to launch a set of detection processes to find out the most suitable performance tests for a system design. PARAD is not a hard-coded solution which eases its usage and extension. Through PARAD, we can automatize as much as possible the analysis. We are working on the relationships between analysis contexts to facilitate their instantiation in a repository, based on existing ones. We are also working on the relationships between identification rules to detect contradictory rules and complementary ones.

Acknowledgements. This work is co-funded through the Waruna project by the French Ministry of the Economy, Finances and Industry, and by the PIA CORAC Panda project.

References

1. Eclipse modeling framework. <https://www.eclipse.org/modeling/emf/>. Accessed 20 Feb 2017
2. The cheddar project: a GPL real-time scheduling analyzer (2015). <http://beru.univ-brest.fr/~singhoff/cheddar/>. Accessed 11 Feb 2015
3. Constraint language for AADL: LUTE (2016). <http://www.aadl.info/aadl/osate/osate-doc/osate-plugins/lute.html>

4. AADL. Architecture analysis and design language. <http://www.aadl.info/>
5. Audsley, N.C.: Optimal Priority Assignment and Feasibility of Static Priority Tasks with Arbitrary Start Times. CiteSeer (1991)
6. Baruah, S., Burns, A.: Sustainable scheduling analysis. In: 27th IEEE International Real-Time Systems Symposium, RTSS 2006, pp. 159–168. IEEE (2006)
7. Baruah, S.K., Rosier, L.E., Howell, R.R.: Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-Time Syst.* **2**(4), 301–324 (1990)
8. Bini, E., Di Natale, M., Buttazzo, G.: Sensitivity analysis for fixed-priority real-time systems. *Real-Time Syst.* **39**(1–3), 5–30 (2008)
9. Brau, G., Hugues, J., Navet, N.: A contract-based approach to support goal-driven analysis. In: ISORC, pp. 236–243. IEEE (2015)
10. CORAC.Le conseil pour la recherche aéronautique civile. <http://aerorecherchecorac.com/>
11. Davis, R.I., Burns, A.: A survey of hard real-time scheduling for multiprocessor systems. *ACM Comput. Surv. (CSUR)* **43**(4), 35 (2011)
12. Gaudel, V., Singhoff, F., Plantec, A., Rubini, S., Dissaux, P., Legrand, J.: An ada design pattern recognition tool for AADL performance analysis. In: SIGAda, pp. 61–68 (2011)
13. Jeffay, K., Stone, D.: Accounting for interrupt handling costs in dynamic priority task systems. In: 1993 Proceedings of the Real-Time Systems Symposium, pp. 212–221. IEEE (1993)
14. Joseph, M., Pandya, P.: Finding response times in a real-time system. *Comput. J.* **29**(5), 390–395 (1986)
15. Liu, C.L., Layland, J.W.: Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM (JACM)* **20**(1), 46–61 (1973)
16. MARTE. Modeling and analysis of real-time and embedded systems. <http://www.omg.org/omgmarte/>. Accessed 20 Feb 2017
17. MAST. Modeling and analysis suite for real-time applications. <http://mast.unican.es/>. Accessed 20 Feb 2017
18. Object constraint language proposed by object management group (omg). <http://www.omg.org/spec/OCL/>. Accessed 20 Feb 2017
19. Ouhammou, Y., Grolleau, E., Richard, M., Richard, P.: Towards a model-based approach guiding the scheduling analysis of real-time systems design. In: WATERS (2014)
20. Ouhammou, Y., Grolleau, E., Richard, P., Richard, M.: Reducing the gap between design and scheduling. In: 20th RTNS, pp. 21–30 (2012)
21. Peres, F., Hladik, P.-E., Vernadat, F.: Specification and verification of real-time systems using pola. *Int. J. Crit. Comput.-Based Syst.* **2**(3–4), 332–351 (2011)
22. Rouhifar, M., Ravanmehr, R.: A survey on scheduling approaches for hard real-time systems. *Int. J. Comput. Appl.* **131**(17), 41–48 (2015)
23. RT-Druid. <http://www.evidence.eu.com/products/rt-druid.html>
24. Schmidt, D.C.: Guest editor’s introduction: model-driven engineering. *IEEE Comput.* **39**(2), 25–31 (2006)
25. Sha, L., Abdelzaher, T., Årzén, K.-E., Cervin, A., Baker, T., Burns, A., Buttazzo, G., Caccamo, M., Lehoczky, J., Mok, A.K.: Real time scheduling theory: a historical perspective. *Real-Time Syst.* **28**(2–3), 101–155 (2004)
26. Sha, L., Klein, M.H., Goodenough, J.B.: Rate monotonic analysis for real-time systems. In: van Tilborg, A.M., Koob, G.M. (eds.) *Foundations of Real-Time Computing: Scheduling and Resource Management*, pp. 129–155. Springer, New York (1991)

27. Simso. Simulation of multiprocessor scheduling with overheads. <http://projects.laas.fr/simso/>. Accessed 20 Feb 2017
28. Stigge, M., Yi, W.: Graph-based models for real-time workload: a survey. *Real-Time Syst.* **51**(5), 602–636 (2015)
29. Tindell, K., Clark, J.: Holistic schedulability analysis for distributed hard real-time systems. *Microprocess. Microprogram.* **40**(2–3), 117–134 (1994)
30. Waruna. Atelier de modélisation et de vérification de propriétés temporelles. <http://www.waruna-projet.fr/>

Software Architecture

11th European Conference, ECSA 2017, Canterbury,
UK, September 11-15, 2017, Proceedings

Lopes, A.; de Lemos, R. (Eds.)

2017, XII, 217 p. 50 illus., Softcover

ISBN: 978-3-319-65830-8